

NRC Publications Archive **Archives des publications du CNRC**

Software for the recording and playback of UDP sockets: user's and reference manual

Johnston, Daniel F.

For the publisher's version, please access the DOI link below./ Pour consulter la version de l'éditeur, utilisez le lien DOI ci-dessous.

Publisher's version / Version de l'éditeur:

<https://doi.org/10.4224/21272477>

IMTI ITFI; no. IMTI-TR-011, 2002-10-16

NRC Publications Archive Record / Notice des Archives des publications du CNRC :

<https://nrc-publications.canada.ca/eng/view/object/?id=03b6734d-c186-41a3-a462-2afc5fb0777c>

<https://publications-cnrc.canada.ca/fra/voir/objet/?id=03b6734d-c186-41a3-a462-2afc5fb0777c>

Access and use of this website and the material on it are subject to the Terms and Conditions set forth at

<https://nrc-publications.canada.ca/eng/copyright>

READ THESE TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS WEBSITE.

L'accès à ce site Web et l'utilisation de son contenu sont assujettis aux conditions présentées dans le site

<https://publications-cnrc.canada.ca/fra/droits>

LISEZ CES CONDITIONS ATTENTIVEMENT AVANT D'UTILISER CE SITE WEB.

Questions? Contact the NRC Publications Archive team at

PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca. If you wish to email the authors directly, please see the first page of the publication for their contact information.

Vous avez des questions? Nous pouvons vous aider. Pour communiquer directement avec un auteur, consultez la première page de la revue dans laquelle son article a été publié afin de trouver ses coordonnées. Si vous n'arrivez pas à les repérer, communiquez avec nous à PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca.



**INTEGRATED MANUFACTURING
TECHNOLOGIES INSTITUTE**

**INTSTITUT DES TECHNOLOGIES
DE FABRICATION INTÉGRÉE**

Pages: 162

**REPORT
RAPPORT**

Date: 2002 October 16

Fig.

SAP Project #

Diag. 19

de Projet de SAP 55-W99

For

Unclassified

Pour IMTI

(Classification)

IMTI-TR-011 (2002/10)

**Software for the Recording and Playback of
UDP Sockets – User's and Reference Manual**

Submitted by

Présenté par Daniel F. Johnston

First Author

Auteur Premier Daniel F. Johnston



Approved

Approuvé Dr. Gian Vascotto
Director

THIS REPORT MAY NOT BE PUBLISHED WHOLLY
OR IN PART WITHOUT THE WRITTEN CONSENT
OF THE INTEGRATED MANUFACTURING
TECHNOLOGIES INSTITUTE

CE RAPPORT NE DOIT PAS ÊTRE REPRODUIT, NI EN
ENTIER NI EN PARTIE SANS UNE AUTORISATION
ÉCRITE DE L'INSTITUT DES TECHNOLOGIES DE
FABRICATION INTÉGRÉE

b20084365 B

Abstract

Two software programs have been created to facilitate the creation and testing of applications that make use of TCP/IP UDP sockets for communication. These programs provide flexible recording of socket messages, and subsequent playback of all or part, with speed control. The document describes how to build and use the two programs, and also contains all the source code and design information for those who wish make improvements.

Contents

1	Introduction	1
1.1	General comments	1
1.2	Software documentation chapters	2
1.3	Appendices	3
2	General Description	5
2.1	Why these programs are needed	5
2.2	Computer communication with UDP sockets	7
2.3	Recording TCP/IP UDP sockets	8
2.4	Playback of pre-recorded TCP/IP UDP sockets	10
2.5	Status of the programs and work planned	12
3	Design, Build and Install	13
3.1	General design for GUI control of socket recording	13
3.2	How to compile the software	18
3.3	Suggestions on where and how to install the software	20
3.4	Other design and build considerations	21
4	Documentation for the socket record and playback	23
5	Record/Play Sockets: Application Module Index	25
5.1	Record/Play Sockets: Application Library Interface Modules	25
6	Record/Play Sockets: Application Compound Index	27
6.1	Record/Play Sockets: Application Library Interface Compound List	27

7 Record/Play Sockets: Application File Index	29
7.1 Record/Play Sockets: Application Library Interface File List	29
8 Record/Play Sockets: Application Page Index	31
8.1 Record/Play Sockets: Application Library Interface Related Pages . . .	31
9 Record/Play Sockets: Application Module Documentation	33
9.1 All user functions common to both socket programs	33
9.2 User functions only for socket playback operation	39
9.3 User functions only for socket record operation	46
10 Record/Play Sockets: Application Class Documentation	53
10.1 _globalMemory Struct Reference	53
11 Record/Play Sockets: Application File Documentation	59
11.1 callbacks-c.c File Reference	59
11.2 callbacks-c.c File Reference	63
11.3 common_callbacks.c File Reference	66
11.4 common_support.c File Reference	69
11.5 creation-c.c File Reference	71
11.6 creation-c.c File Reference	74
11.7 main-c.c File Reference	77
11.8 main-c.c File Reference	80
11.9 myInterface.h File Reference	83
11.10mySocket.h File Reference	86
11.11play_support.c File Reference	88
11.12record_support.c File Reference	90
11.13recording.c File Reference	91
11.14sending.c File Reference	93
12 Record/Play Sockets: Application Page Documentation	95
12.1 Todo List	95
13 Software Index	97

CONTENTS

iii

A Common Source Code	103
B Record Program Source Code	105
C Playback Program Source Code	107
D Other Source Code	109

List of Figures

2.1 Motion capture	6
2.2 Socket recording - main window	8
2.3 Socket playback - main window	10
3.1 Two processes with signals	14
3.2 Builder Xcessory version 5.0	17
3.3 Source code directory structure	19
11.1 Include dependency graph for play_callbacks-c.c	59
11.2 Include dependency graph for record_callbacks-c.c	63
11.3 Include dependency graph for common_callbacks.c	66
11.4 Include dependency graph for common_support.c	69
11.5 Include dependency graph for play_creation-c.c	72
11.6 Include dependency graph for record_creation-c.c	75
11.7 Include dependency graph for play_main-c.c	77
11.8 Include dependency graph for record_main-c.c	80
11.9 Include dependency graph for myInterface.h	83
11.10 Include dependency graph for myInterface.h	83
11.11 Include dependency graph for mySocket.h	86
11.12 Include dependency graph for recording.c	91
11.13 Include dependency graph for sending.c	93

Chapter 1

Introduction

1.1 General comments

Many software programs send or receive computer network messages. These messages are used to copy data, provide information about user interaction, and to implement control between software programs on the same or different computers. An important sub-set of socket messages is the 'UDP' or 'connection less' variety. Since messages of this type do not require an active 'send and acknowledge' interaction, they are simpler to implement in software and can transfer data much faster. For these reasons UDP sockets are often used to communicate information about user interaction and program control. The potential for lost messages that is inherent with UDP sockets can be almost ignored when using modern computer networks.

It takes two programs running simultaneously to transfer any computer network message. i.e. one to send and the other to receive. But what if the receiving program is not available? What if the hardware to support the sending program is off line? What if you are trying to debug a program which sends or receive socket messages? What if you are interested in only a small part of the full message stream?

These are some of the reasons for the two programs described here. One of the two programs will record any valid stream of TCP/IP UDP sockets in a file. This record will also include the time delay between messages. The second program is able to read this data file and send all or part of the socket stream, with timing intact. The sending program can direct the socket messages to either any valid network destination or to another file.

1.2 Software documentation chapters

The documentation for the socket record and playback programs is provided in the following chapters:

1. Introduction:

An introduction to UDP socket recording and playback, i.e. this chapter.

2. Overview:

Background and a description of the UDP recording and playback software: Why were they created? Where can they be used? This overview would be useful to anyone who needs a brief description of the software's features and applications.

3. Design and Build:

A description of the design features for the two programs, found in this chapter, would help anyone who wants to improve the code, or the description would even help someone to understand the remaining chapters and source code better. This chapter then describes how to build the two programs from source code and suggests how these can be installed in your computer.

4. Documentation for the source code of the software:

This chapter is an introduction into the reference manual for all the source code used in the development of the two software tools. In the chapters that follow, all structures and variables are listed and described, all functions listed, and a cross reference of all parameter and function names is given. These chapters are for **developers** of the source code, not for **users**. They are provided for those who wish to extend the functionality of the two applications.

All the functions available for the developer are listed in chapter 5. The functions are grouped, and each is provided with a brief description of use. All input parameters for each function are described. The return value for each function (if any) is also described.

The remaining, special chapters provided for reference to the software are listed below.

5. Module Index. A list of all the modules defined for the software and the page where the description of those modules starts. A 'module' in the context of this software documentation is a collection of functions which are related by the interaction they provide.

6. Compound Index. A list of any structures defined for the software and the page numbers of their detailed descriptions.

7. File Index. A list of all the source files used to create the two computer programs, and the page number where the general descriptions for these files are provided.

8. Page Index. A list of pages of related documentation for the software. In this case, it is a reference to the pages of any remaining 'to do' items, i.e. changes that the developer has suggested be done to the source files.
9. Module Documentation. The groups of software functions (modules) and the detailed description of each function in each group.
10. Class Documentation. The detailed description of each structure defined for the software, and a description of each member of each structure.
11. File documentation. All the source files, and the general comments and descriptions found in these files. Functions that are defined in these source files are listed, but they are all contained in one of the 'modules' already described, so their detailed descriptions are given there.
12. Page Documentation. Documentation for related pages, i.e. the actual list of 'to do' items.
13. Index. All the structure, function, variable, names used in the source code for the two socket record/playback programs, and the page number where the documentation on that item is to be found.

1.3 Appendixes

Additional information is provided in the appendixes;

- A Common Code: The source code listings for the socket common code
- B Record Code: The source code listings for the socket record program
- C Playback Code: The source code listings for the socket playback program
- D Other Code: The source code listings for other utility functions

All application source code files for both programs at the time this report was generated are reproduced here. These appendices thus provide a permanent record of the code in its current form and version.

Chapter 2

General Description

This chapter describes the background for the two UDP socket programs; why they were created, what they can do, how they are expected to be used. This chapter should be read first by anyone who plans to use these programs.

The following is a table-of-contents for the topics covered in this chapter:

- Why these programs are needed
- Computer communication with UDP sockets
- Recording TCP/IP UDP sockets
- Playback of pre-recorded TCP/IP UDP sockets
- Status of the programs and work planned

2.1 Why these programs are needed

In the spring of 2002 the NRC's Virtual Environment Technologies Centre (VETC) was starting to make fuller use of their Motion Capture System for 'real-time' reproduction of human and machine movement. The plan was to use the capture system to control the position of humanoid and other devices in a virtual environment, or even multiple virtual environments. It was quickly realized that it was difficult and time consuming to get a volunteer to dress up with all the markers needed for the position tracking. It was almost impossible to position all the markers exactly where they were in the previous session. And, the different size of volunteer and different marker position meant that the existing computer models could not be re-used without taking considerable time to adjust them. Tracking system computer software could store and replay measurement sessions, but the overhead of storing data from long sessions does degrade the performance of the tracker. Sessions could be replayed only when the tracking system was

connected, and a session could only be replayed in totality, i.e. you could not select a part of particular interest. These were all serious impediments for all the testing that would be required for software development.



Figure 2.1: Motion capture

Some mechanism was required to record the data sent by the tracking system so that it could be used even when the tracking system was disconnected. The data sent by the tracker is in the form of TCP/IP UDP sockets. It was necessary to record a sequence or 'stream' of socket messages on computers other than the Motion Capture system.

A computer program was created to record any valid stream of UDP socket messages and store them in a file for later use. This program is fully described in a section below, but its main function is to record the socket data and the time delay between messages so that messages can be sent, and re-sent, from the recording computer to any other device at a later time.

A second computer program, also described below, was created to manage the socket message playback from the data file. This program can read quickly through the data file, ignoring the delay time, and then play back, with correct timing, any selected portion of this data stream.

Now the development of the software for tracker controlled models in virtual environments can proceed. The two programs are not restricted only to this software development project, but can be used for testing and development of other applications such as heavy equipment simulators. Selected playback of message streams would allow, for example, the operator of a finished and working simulator to replay any specific portions of a trainee's session for closer examination or to highlight problems that need to be corrected.

Because the two programs are (potentially) of general interest to software developers, this report was created. It contains all the information a programmer would need to

use, recreate or improve on the socket recording and playback programs.

2.2 Computer communication with UDP sockets

'Sockets' are used for communicating between computer systems. More specifically, TCP/IP sockets are the de-facto standard method for communicating between computers and computer equipment - from the small dedicated devices to the largest system. One important sub-class of TCP/IP sockets is the 'connection-less' variety, also known as "datagram" or UDP. Connection-less sockets are 'one way' messages that do not need any reply or acknowledgement. Therefore they have much less overhead and are much faster. Because they are not acknowledged and repeated if lost, there is a chance of missed messages or messages out of order. However, on modern computer network hardware the probability of these errors is very low.

Fast UDP sockets are used for communication of status, of control, or of fast changing data. Usually the contents of these messages are designed to **not** depend on previous messages, so that one lost message will have little effect on the operation. It would matter very little, for example, if the receiver of messages about a steering wheel position got several messages about angles of 25.6 and 25.7 degrees, and missed one message with a value of 25.65.

All computer communication assumes that there is a 'sender' and a 'receiver' for each message, i.e. the purpose of the messages is to establish communication between two system. With datagram sockets, it is possible for one system to send messages before the second system is available or ready to listen for them. In this case the UDP messages would be lost. Once the messages are sent, it may be difficult or even impossible to reproduce them. Wouldn't it be useful to have some program that could record this stream of messages for later analysis or replay?

2.3 Recording TCP/IP UDP sockets

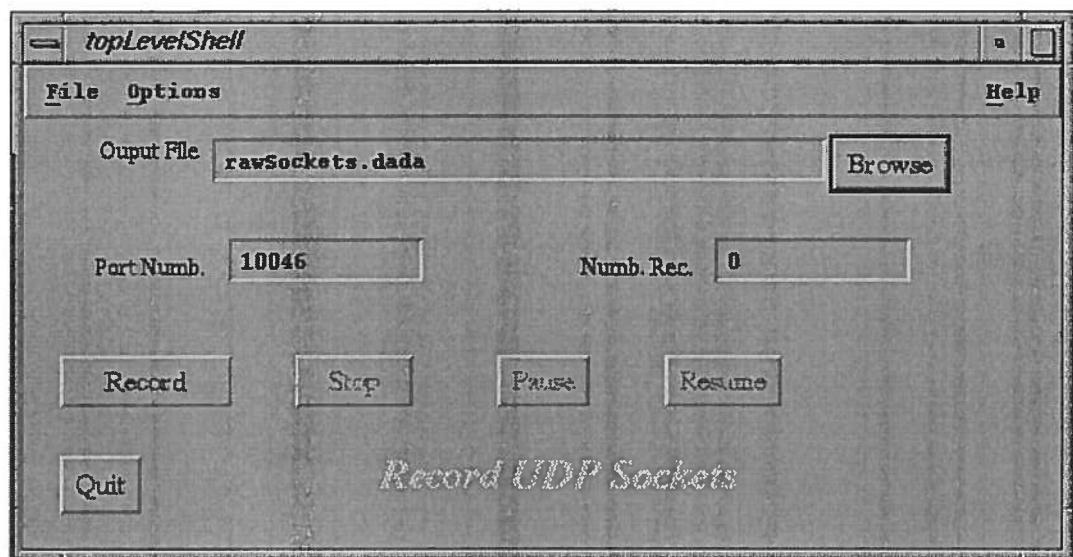


Figure 2.2: Socket recording - main window

The program to record socket messages is actually two computer process, as described in the next chapter, but the buttons and menus of the user interface hide this complexity from the user. The program must listen to sockets and save them in a file, so the two main functions of the recording software are to specify the TCP/IP port number that selects the messages, and to use a 'file browser' to identify the file name and location where the socket messages will be stored.

The program does more than the basics just mentioned. It is possible to specify the two required parameters, i.e. socket port number and a file name, on the same command line used to start the program. If you make an error when adding these command line arguments, the program will print a short message about the correct format. If you do not supply one or both of these arguments, the program will set default values.

You can change the socket port number to use by selecting the existing number with a mouse click and editing or replacing it. You can change the output file name the same way, or you can select the "Browse" button to bring up a file selection dialog to look in any directory for an existing file name. Both of these parameters can be changed at any time before the user selects the "Record" button.

Selecting "Record" will cause the program to open the specified output file and start looking for UDP socket messages on the current port number. The time delay before the first message, and the delay between subsequent messages, is stored in the output file along with the socket message itself. If the user selects "Pause" then the program

will stop reading and saving messages, but the time interval will continue to increase. "Resume" will start the storing of messages again. When the "Stop" button is selected the program stops reading socket messages and closes the output file. The number of messages stored in this file will be counted and displayed as the "Numb. Rec." value.

The program is terminated with the "Quit" button. Any recording will cease, and any open output file will be closed.

All the button options are also available via a pull down menu under the heading "File". These work exactly the same as the buttons on the main display. Another pull down menu is labeled "Options", but it currently only contains a toggle button to switch a 'debug' mode on and off. When the 'debug' mode is on, the program will print lots of useful messages during the operation of the buttons and recording of messages. These messages help you verify the correct operation of the software, but they will also slow down the recording of socket messages.

Users of the program can get a general message about the program from the "Help" menu. One option is "About" which gives the program name, author and date. More general help is available from the "Usage" option.

2.4 Playback of pre-recorded TCP/IP UDP sockets

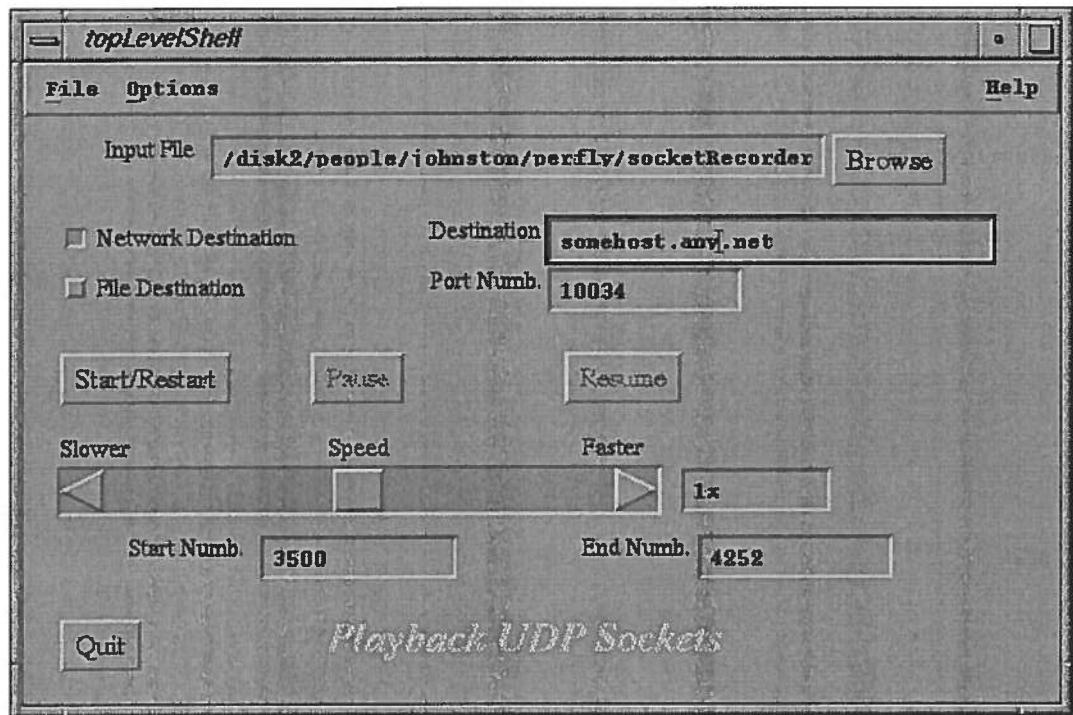


Figure 2.3: Socket playback - main window

Pre-recorded socket messages are played back using a second software program. This program needs to know what file will contain the messages to be sent. It also need to know where to send the messages, but the program provides two options for this. Messages can be sent over a computer network (as the original recorded messages were) or sent to another computer data file. The advantage of the latter option is that selected portions of a large message stream can be broken into a number of specific message sets. And, since all inter-message delay is ignored for file output, this recording of sub-sections of a large message stream is very fast.

You can specify the input file name and the network address, including port number, on the same command line used to start the software. If you set the port number parameter to be '0', the destination parameter is taken to be the file name where the (subset of) messages will be sent. Any error in these parameters will cause the program to print a short message about the correct parameter options.

You can change any parameter from the main window of the program. Select the "Browse" button to search directories for the data file which contains the messages you

want to send. You can also use the mouse and keyboard in the input file name area to edit this value. When you press the 'Enter' key after editing the file name, or after you select a new file with the browser window, the program will count the number of messages in the file and display it in the "End Numb" display. If the file you selected does not contain socket messages in the expected format, the program will 'beep' to indicate that the file name is not valid.

Use the mouse to switch between a network or a file destination. When network mode is in effect, you can edit the "Destination" value to supply a valid TCP/IP network address. You will also need to supply a valid TCP/IP port number for a complete network destination. In file mode, the destination is any valid file name for data output and the port number is ignored.

Two number areas are also shown. Whenever the input data file is changed, the "Start Numb" field will be reset to '0' and the "End Numb" field will be set to the total number of messages in the selected input file. Edit these two values to select a subset of the data. The program will not allow you to set the values less than zero or more than the number of messages in the input file.

Press the "Start/Rewind" button to begin sending the messages from the input file to the output destination. If the destination is a file, then the portion of the input file designated by the start and end numbers will be very quickly written, the output file closed and the program will be ready to send/save another portion of the input file.

If the destination mode is 'network', then pressing the "Start/Rewind" button will cause this button to be disabled and the "Pause" button will be enabled. Very quickly the program will read the file until the 'start' message number, and then will send each message with the stored inter-message delay until the 'end' message number is reached. Then the "Pause" button will be disabled and the "Start/Rewind" button re-enabled for the next transmission.

During a transmission, i.e. while the "Pause" button is enabled, the user can press this 'Pause' button to stop the transmission of data. It will be resumed from the current message number when the user selects the "Resume" button, or it can then be re-started from the beginning with the "Start/Rewind" button.

The "Speed" slider will allow the user to slow down or speed up the transmission of data to a network destination. The selected data will be sent with an inter-message delay that is 1/4 to 4 times the delay recorded in the original data file.

All program values, including the speed setting, are recognized only when the "Start/Rewind" button is pressed. Changing the speed while transmitting, for example, will have no affect.

Use the "Quit" button to stop all transmission, close all files, and exit the program.

Pull-down menus on the main program display can also be used to control the program. In the "File" menu you will find options that duplicate all the buttons on the main display. In the "Options" menu you will find toggles to switch from network to file destination, and a 'Debug' mode toggle that will cause the program to print additional

messages about its status and operation while the user controls the program. Program operation will be slower when these extra messages are being printed.

As for the socket record program, there is a "Help" menu option as well. You can get a general message about the program from the "About" selection, i.e. a dialog will appear which gives the program name, author and date. More general help is available from the "Usage" option.

2.5 Status of the programs and work planned

- The two programs have been build and tested on SGI UNIX computers. They should also run without any changes on any Linux computer, but this has not been tested yet. The programs should build and run on any computer which has development libraries for TCP/IP and Motif.
- The software will be released to other IMTI staff for their comments on the code and documentation. These people, and their projects, will also serve as 'alpha' testing situations for the software.
- Source code 'bug' fixing will continue with the results from this 'alpha' testing.

This chapter describes the general design methodology for the TCP/IP UDP socket recording and playback software. It should help any programmer understand how the software works, how the source code is structured, and how to make changes to the code if necessary. A description of how to build the programs from source code is then included. This chapter should be read by anyone planning to build the programs from source or modify the source code.

Chapter 3

Design, Build and Install

This chapter describes the general design methodology for the TCP/IP UDP socket recording and playback software. It should help any programmer understand how the software works, how the source code is structured, and how to make changes to the code if necessary. A description of how to build the programs from source code is then included. This chapter should be read by anyone planning to build the programs from source or modify the source code.

The following is a table-of-contents for this chapter:

- General design for GUI control of socket recording
 - Design considerations common to both programs
 - Design considerations specific to recording
 - Design considerations specific to playback
 - User interface building with a tool
 - Documenting source code with 'doxygen'
- How to compile the software
- Suggestions on where and how to install the software
- Other design and build considerations

3.1 General design for GUI control of socket recording

The functions of receiving (recording) or transmission (playback) of UDP sockets on any UNIX computer are quite simple - perhaps too simple from a computer security point of view. A short, one page source code will do it, and examples can be found in many "Introduction to Computer Network Programming" manuals. Such examples

work, are simple to write, but require some expertise to use. Most users find it easier to use programs which have a more modern Graphical User Interface (GUI).

Two program were created that perform the socket recording and playback, and which have a user interface designed in the Motif style. The following sub-sections discuss some of the key design ideas used in both of these programs.

3.1.1 Design considerations common to both programs

It is not easy to create a program to, for example, record socket messages and also allow the user to interact with a GUI. The program would have to respond in 'real time' to all sockets messages no matter when they arrived, and also respond in 'real time' to the user's keyboard and mouse selection of buttons and pull-down menus. A single program cannot do both. Fortunately UNIX supports several solutions - one of which is multitasking. You can create two independent processes on the computer using the 'fork' function.

These two processes have to communicate with each other. One process is receiving control information from the user's interaction with the GUI. It must use this input to control the operation of the second process. The second process is concentrating on the sending or receiving of socket messages, but it must let the first process know when it is done its work. There are also several UNIX solutions for this communication problem. The two programs described here use the simplest form - "signals".

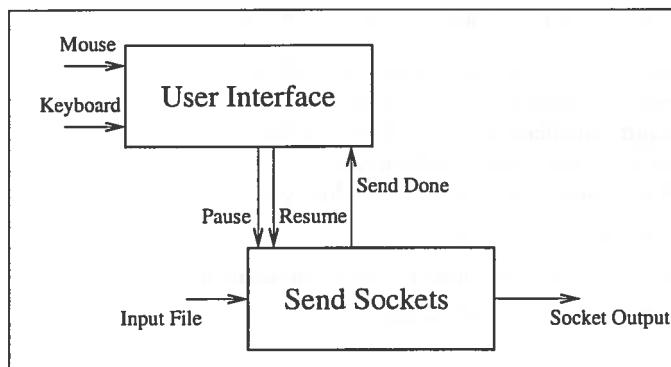


Figure 3.1: Two processes with signals

Consider the situation of 'pause' and 'resume'. When the user interface (via the "Start" button) creates the second process for socket input or output, it will condition the second process to accept a standard UNIX 'signal'. The socket process will respond to this signal by immediately interrupting its work and executing a short function which simply puts the socket process to sleep. The design philosophy used in these two programs

is to have the "Pause" button cause this signal to be sent, which will force the second process to immediately go to sleep. When the user interface responds to the button to "Resume", it will send another signal which has a default behavior that causes the second process to continue. The socket process returns from the sleep command, and resumes its work where it left off.

Another signal used by both socket programs is sent by the second process when it is finished its work, i.e. when the second process terminates. This signal will cause the user interface program to reset the state of the buttons, to close files, to count messages, and other such 'house cleaning'.

Another design concept used in the two programs relates to the reading of messages in the file. Both programs make use of the fact that just reading the file, and counting the number of messages in it, is very fast and can be done whenever required. The file is closed, re-opened for reading, the number of messages is counted, and the file is closed again. If necessary it will re-opened again for reading (playback). All these file manipulation are very fast and make the software design much simpler than would be required to manipulate internal file pointers.

Both programs must open a socket and send/receive messages from this, both are multitasking programs with similar signals and responses, and both programs use a computer generated GUI interface with similar button and menus. Therefore, there are many identical functions required in both programs. These have been grouped as the "common" functions and have been placed in a separate directory. Any improvements to these functions will improve both programs.

3.1.2 Design considerations specific to recording

Recording is the simpler of the two socket message functions, so the user interface has fewer buttons and options. Also simpler is the design of the software. It is actually a stripped copy of the socket playback program - which was created first.

One difference in design is the fact that the second process would never terminate on its own. Unless there is an error creating or reading from the socket, the second process will continue forever waiting for a message to store. So we need a good way to kill it. The default behavior of the child process to the standard SIGABRT signal was replaced with a function which would stop the process and still send the desired signal back to the main program to cause button state to be reset and files closed.

3.1.3 Design considerations specific to playback

One unique feature of the playback socket messages program is the option to output a (sub)set of messages to a file. Several features are needed to support this. There is

extra logic in decoding the command line arguments, there are toggle buttons to select the destination mode, and there is code in the sending function to write the message and time delay value back to a file without actually waiting for the delay period.

Another important feature is the concept of a starting and ending message number to select a subset of the total message stream. This is very useful for editing long message sequences. It is invaluable for testing of the programs which will make use of these socket messages.

To implement the start and end numbers required a couple of new fields on the user interface. Since these could be changed by the user at any time, the values for these numbers are read only when the user starts sending socket messages. The values of all parameters are stored in the global memory structure before the software 'forks' the second process. Both processes share a copy of these updated values, and they can be used in the socket sending routine to define the limits for the transmitting loop.

Another parameter stored before the program 'forks' the child process is the 'speed' multiplier. It is very useful to be able to send a complex socket stream much slower than the original, or to speed over a very slow moving sequence of messages. The speed multiplier value is multiplied by the original inter-message delay to calculate a new delay value before the next message.

3.1.4 User interface building with a tool

The user interface for both programs was build using the software tool **Builder Xcessory** version 5.0 from ICS Incorporated. Using a library of user interface components, anyone can quickly create the buttons and menus required for a new program. Buttons and be moved, removed, resized, and recolored very easily using the controls of this builder tool. You do the same for all other displays associated with the interface, such as the supplied file browser, help dialogs, and error messages.

Once the windows are prepared, you go into more detail for each widget of the user interface. You can specify the 'scope' of the widget (private or global) and supply the name for the callback function for each widget that generates events. You can even use some pre-defined functions to cause dialogs to appear and disappear, or to exit the application.

When all additions and changes are complete, you ask the builder software to generate the computer code that would build the interface you want. The two socket applications use the Motif style of user interface and the 'C' programming language. Motif with 'C++' and 'Java' interface designs and code are also supported by the builder tool.

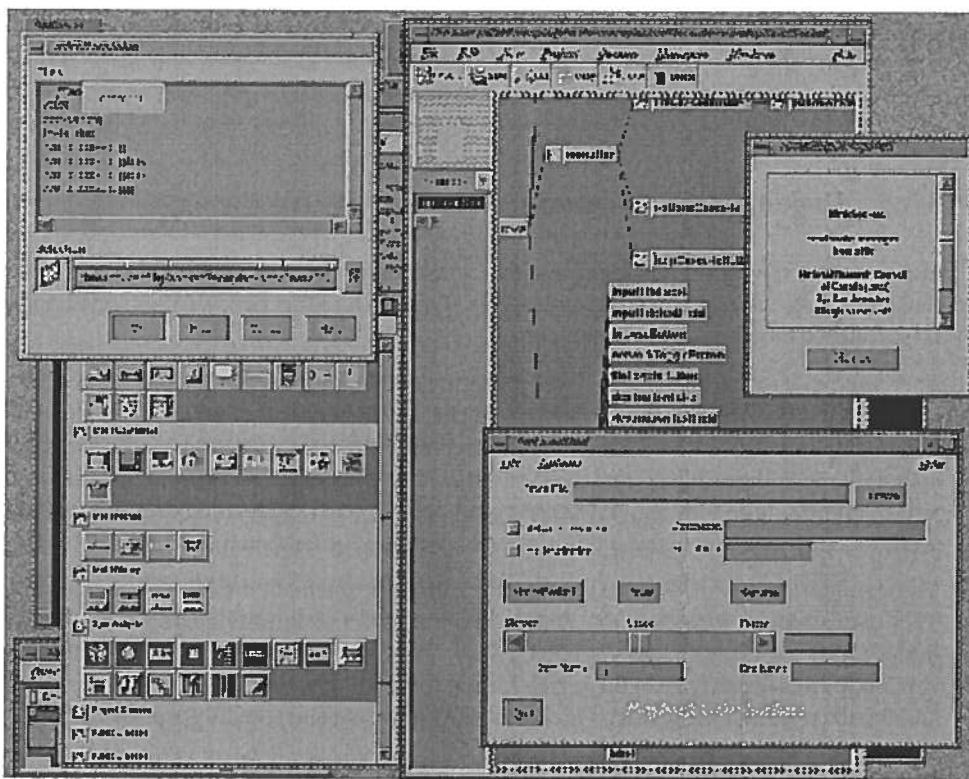


Figure 3.2: Builder Xcessory version 5.0

The computer generated code includes several '.c' source files and one '.h' include file. They can be recognized because they have the form "name-c . ext". The tool also creates a simple 'makefile', which is used to compile and link the application, and a few other files which are useful in a complex X11 environment.

You then add your code to these source files. If you need to make a change to the interface design, you will need to re-generate the source files. Builder Xcessory provides special comment lines in its source files, and if you place your code only between these lines the new computer generated code will not overwrite your work. There are some limitations to this technique.

First, you have only limited control over the style of the computer generated source files. The naming of functions and variables, and even of the source files themselves can only be changed with a great deal of time to modify all the default actions of the builder. It is much easier to live with the result, i.e. accept the code as-is and assume the builder wrote good code.

Second, you cannot use other source code documentation tools to describe some of this computer generated software or the changes you add to the computer generated

source file. There is no place to include the special comments used by documentation software, i.e. there is no place that will not be overwritten by the next code generation.

3.1.5 Documenting source code with 'doxygen'

Documentation of software is never easy. So, when a tool is available that claims to make this task easier it is worth a close look. One such tool is doxygen, created and freely supplied by Dimitri van Heesch.

The core philosophy of this tool is to place the documentation as close to the source code as possible. Then, in theory, when you change the source you will also update the documentation. Documentation of files, functions, variables, structures and classes is added by the author of the source code as special comments. The documentation tool will read the source files and use these comments to generate detailed descriptions of all the code. It will create these descriptions in several formats such as `html` and `rtf`.

You can then add your own text to complete the documentation of your program(s). This entire report, including the detailed source code descriptions and index found in the following chapters, was created by the doxygen tool. Actually it created two versions of this report; one in the `html` format for reading with standard web browsers, and the other in `latex` format which was used to create and format the printed copy.

3.2 How to compile the software

The socket record and playback software can be, in theory, installed on any computer that supports the 'C' programming language, TCP/IP sockets, and the X11/Motif user interface. Currently all varieties of UNIX will satisfy these requirements, including all Linux variations. You should also be able to meet the requirements on a computer with a MS-Windows operating system and an optional Motif emulation package. The documentation tool is already available for all these platforms. Building the software on a UNIX computer will be described.

To change the appearance of either or both GUI interfaces will require access to a licensed copy of the interface builder tool. In this case it must be a copy of 'Builder Xcessory version 5.0' or newer. It should be possible for a knowledgeable programmer to make changes to the interface just by editing the source files provided. An interface builder tool will just make this faster and easier.

Standard UNIX utility programs are used to build and rebuild the two programs. You will need a 'C' compiler program to turn the source code text into the digital files the computer need to execute. Most UNIX computer systems have a 'C' compiler, but if there is not one available, the public domain ("Free") GNU compiler will work fine.

You will make extensive use of the 'make' utility. In its simplest form this standard UNIX utility will create new versions of the program whenever it detects that one of the source files has been changed. It is possible to build both programs from source code, and completely re-create the documentation in several formats, all by typing the command 'make' in the top-most directory for the program. Other common options that can be used in any of the source directories, are 'make debug' to create versions of the programs with extra information used by UNIX program development tools, 'make clean' which will delete most of the object files and 'core' files left from failed program tests, and 'make clobber' which will also 'clean' the directories but also remove all existing versions of the programs, all generated documentation, and any computer created temporary files and directories.

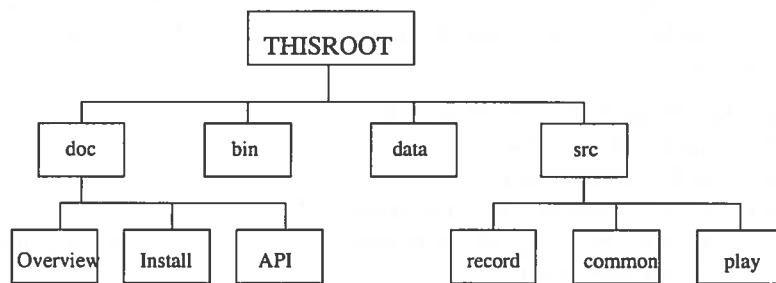


Figure 3.3: Source code directory structure

The highest level directory for the two socket programs will contain four subdirectories and two important files. The first important file is the 'Makefile' which is used to supply instructions to the 'make' utility. Almost every directory in the structure will contain a 'Makefile'. This top-level one is configured to run the 'Makefile' in all the subdirectories. The second file found in this directory is 'setup_env' which contains a short set of definitions used by the computer to rebuild the programs, in this case the definitions are needed only to rebuild the documentation, i.e. a key definition found in this file is to define the current, top level directory as 'THISROOT' and this definition is used in document source files. Use the command 'source setup_env' in this directory before you try to update the documentation.

Two subdirectories of the top directory are mostly used to hold files created by something else. The 'bin' directory will hold a copy of the socket record and playback function, along with their help files and images, after these programs have been rebuilt from the source code. This directory also contains a few utility programs used to create listings of the source and to change the 'copyright' message in each source file. The 'data' directory can be used by the socket recording program to save files for later playback.

All program source code is found under the 'src' directory. There are three sub-directories;

- the source code used by both programs in 'common',
- the source code used only for the record sockets program, and
- the source code used only for the playback sockets program.

All the computer generated files, the 'C' source code, and the 'Makefiles' need to (re)build the programs are found here. Just type 'make' in any directory to insure that the two programs you will use are based on the most recent version of the source files. You should 'make' the common directory first.

Documentation, using the 'doxygen' tool, is controlled and stored under the 'doc' directory. Again, 'Makefiles' are used to control the (re)building of the documentation from the source code and the author supplied text files. The 'doc' directory and sub-directories also contain a number of files used by the 'doxygen' documentation tool, but these will not be described in this document. Refer to the documentation for the 'doxygen' tool if necessary.

The text for the introduction, program description and this description of programs design and building, is found as files in these directories with the ".doc" extension. These are basically text files that are also scanned by 'doxygen' to create the formatted documentation you are now reading. One subdirectory, 'API' contains the extra instructions used to control how the grouping and documentation of the "Application Programming Interface" for the programs is created.

If you do not have a copy of the documentation software tool, and cannot install one on your computer for some reason, then you should **not** delete the generated documentation that was supplied with the program. Since the tool is free, this situation should not occur. You could also copy the source for the two socket programs to another computer that does have the required documentation tools and rebuild the documentation there, even if the programs cannot be compiled on this second computer.

3.3 Suggestions on where and how to install the software

By default, the two socket programs will be created in the directories where their source code exists, and then each copied to the 'bin' directory along with their respective help file and images. Anyone can run these programs by specifying the full directory path to the 'bin' subdirectory along with the name of the program.

It is much simpler to run the programs if they are copied to a directory already searched by the system when the user issues a command. On UNIX systems this is called "placing the programs in a directory that is already in your 'path'". Use the UNIX command 'echo \$PATH' to see what directories are already searched. A common location is a 'bin' subdirectory of the computer user's top (or 'home') directory. Then the name of the program is sufficient to run the program for this user no matter where he/she is on

the computer.

Other standard locations in the 'path' will make the programs available to all users of the computer system. A common approach is to use the directory '/usr/local/bin'. You will probably need to ask your system manager to copy the program files to this read-only directory. Remember to copy the two help files to this same directory, as well as the help file images in a sub-directory called 'images'.

3.4 Other design and build considerations

- See the 'todo' section of this documentation.

Chapter 4

Documentation for the socket record and playback

The information in the following chapters was created automatically from the source files for the two socket record and playback programs. A documentation 'tool' was used for this, i.e. the software doxygen created by Dimitri van Heesch. See the WWW site <http://www.stack.nl/dimitri/doxygen/index.html> for complete information (and a copy!) of this software. Being computer generated, the format and the titles of the chapters and sections cannot be easily controlled, so this chapter is used to provide an introduction.

The remaining chapters form a reference manual on all the functions, variables, and structures used in the two software programs. All the information is derived directly from the source files for these programs. Being a reference document, these chapters are not meant to be read sequentially but rather accessed randomly as the need arises, i.e. to refresh your memory when needing to use, modify, maintain or extend functions in the software. The index chapter (10) for the software reference will help with this random access. Some of the most useful (to the developer) references are:

- Module Documentation. All functions are separated into groups and described.
- Class Documentation. All structures and their members described.
- Page Documentation. Todo List, i.e. list of things 'to do', changes suggested for the source files and to extend program functionality.
- Index. Functions, structures, members, variables with page numbers for their documentation

Chapter 5

Record/Play Sockets: Application Module Index

5.1 Record/Play Sockets: Application Library Interface Modules

Here is a list of all modules:

All user functions common to both socket programs	33
User functions only for socket playback operation	39
User functions only for socket record operation	46

Chapter 6

Record/Play Sockets: Application Compound Index

6.1 Record/Play Sockets: Application Library Interface Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

-globalMemory (This is the structure to declare all the global variables required for the main program of the Motif GUI interface, and for the child process we spawn to do the actual recording/sending of messages)	53
--	----

Record/Play Sockets: Application Compound Index

Chapter 7

Record/Play Sockets: Application File Index

7.1 Record/Play Sockets: Application Library Interface File List

Here is a list of all documented files with brief descriptions:

common_callbacks.c	66
common_support.c	69
myInterface.h	83
mySocket.h	86
play/callbacks-c.c	59
play/creation-c.c	71
play/main-c.c	77
play_support.c	88
sending.c	93
record/callbacks-c.c	63
record/creation-c.c	74
record/main-c.c	80
record_support.c	90
recording.c	91

Chapter 8

Record/Play Sockets: Application Page Index

8.1 Record/Play Sockets: Application Library Interface Related Pages

Here is a list of all related documentation pages:

Todo List	95
----------------------------	----

Chapter 9

Record/Play Sockets: Application Module Documentation

9.1 All user functions common to both socket programs

All the common functions for socket record and playback are listed here.

Files

- file common_callbacks.c
- file common_support.c
- file myInterface.h
- file mySocket.h

Functions

- void debugToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void aboutHelpCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void aboutDismissCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void usageHelpCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void browseButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)

- void cancelFileDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void AppendString (char **return_text,int *length,char *text)
- Boolean XmStringToString (XmString xmstr,char **return_text)
- int getNumbMessages (FILE *fileD)
- int readNumberMessages (char *fileName)

9.1.1 Detailed Description

All the common functions for socket record and playback are listed here.

Since both programs use a similar style of Grapical User Interface, and since both programs share a common design for multitasking and interprocess communication, they both share a number of functions and definitions. These were collected and placed in a separate directory so that there would be no duplication of source code, i.e. one fix to any function would correct both programs.

9.1.2 Function Documentation

9.1.2.1 void AppendString (*char ** return_text, int * length, char * text*) [static]

This function is declared 'static' so that it's scope is local to this source file (i.e. only functions in this source file can call it). The function will append a character array to another, allocating more memory for the string as required.

Parameters:

return_text is the pointer to the character string array which is either NULL (no existing string) or a pointer to an existing string. This pointer will point to the new or amended string on return from this function.

length is the length of the existing character array, and on return it will be equal to the total new length of the string.

text is the array of characters to be appended to the existing(?) string

Definition at line 43 of file common.support.c.

Referenced by XmStringToString().

9.1.2.2 Boolean XmStringToString (XmString *xmstr*, char ** *return_text*)

This function will try to convert a Motif 'string' value to the more general character array string used in 'C' programs. The success of the conversion is set as the return from the function. The character array is returned as one of the function arguments.

Parameters:

xmstr is the Motif XmString variable to be converted.

return_text is the character array that will hold the converted text

Returns:

true or false, i.e. whether a conversion to character array was possible or not

Definition at line 75 of file common_support.c.

Referenced by okFileDialogCallback(), and okFilePlayDialogCallback().

9.1.2.3 void aboutDismissCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'dismiss' button on the 'about' help message. It will simply remove this pop up help message from the display.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 90 of file common_callbacks.c.

9.1.2.4 void aboutHelpCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'about' button on the 'Help' pull down menu. It will cause a small dialog to show with the name of the program, a very brief description, and the name of the author.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 73 of file common_callbacks.c.

9.1.2.5 void browseButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'browse' button on user interface or the 'open' option in the 'File' pull down menu. It will cause a file selection dialog to be visible.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 135 of file common_callbacks.c.

9.1.2.6 void cancelFileDialogCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface puts up a file selection dialog and the user decides to cancel it. It is a 'null' event except we need to remove the file selection dialog from the display.

Parameters:

w the widget that caused the event and this callback. The push button (cancel) on the file selection dialog in this case.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 154 of file common_callbacks.c.

9.1.2.7 void debugToggleCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'enable debug mode' toggle buttons. It will use the current widget state to set the global debug mode flag.

Parameters:

w the widget that caused the event and this callback. We need this to read the current state of this widget (otherwise the toggle button would also have to be global in scope)

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 53 of file common_callbacks.c.

9.1.2.8 int getNumbMessages (FILE **fileD*)

This function will be called to quickly read the entire file and count the number of messages in it. This number can be used to set values on the user interface. The file is defined by the (already open) file descriptor.

Parameters:

fileD is the file descriptor for the (already opened) file which contains the socket messages and delays.

Returns:

the number of messages in the file, or 0 if there was an error reading the file or an invalid file.

Definition at line 148 of file common_support.c.

Referenced by readNumberMessages(), and resetReadButtons().

9.1.2.9 int readNumberMessages (char **fileName*)

This function will be called to quickly read the entire file and count the number of messages in it. This number can be used to set values on the user interface. The file is defined by a character file name. After the number of messages is determined, the file is closed and re-open in order to place the file pointer back to the beginning.

Parameters:

fileName is the name of a file which (we hope) contains socket messages and delay values.

Returns:

the number of messages in the file, or 0 if there was an error reading the file or an invalid file.

Definition at line 194 of file common_support.c.

Referenced by okFilePlayDialogCallback(), shared(), and startButtonCallback().

9.1.2.10 void usageHelpCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'usage' button on the 'Help' pull down menu. It will cause the software to start Netscape with a pointer to a descriptive help file, complete with images and links. The help file selected is defined by the current application.

Todo:

We need to add code so that the two different socket program help files can be stored in some directory other than the same directory as the program executables. We should provide a mechanism to change the name of the html document used for help.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to client data defined for the callback, in this case the name of the file to open

call_data the standard event information provided by X11-Motif

Definition at line 114 of file common_callbacks.c.

9.2 User functions only for socket playback operation

All the functions required only for socket playback are listed here.

Files

- file play/callbacks-c.c
- file play/creation-c.c
- file play/main-c.c
- file play_support.c
- file sending.c

Functions

- void fileToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void networkToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void speedSliderCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void okFilePlayDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void pausePlayButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void resumePlayButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void startButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void usageSend (void)
- int createSendSocket (char *address,int port)
- int sendSocketMessage (int sock,char *buffer,int sizeBuffer)
- void resetSendButtons ()
- void sendProcessPause ()
- void sendProcessDone ()
- int sendMessages (FILE *fileD,FILE *outFileD)

9.2.1 Detailed Description

All the functions required only for socket playback are listed here.

Many of these are very similar to function required for the record program so you must be careful when examining the name of the function or variables. Some functions have

exactly the same name and parameters, but these were generated by the GUI builder tool, and the names are set by that tool and cannot be modified.

9.2.2 Function Documentation

9.2.2.1 int createSendSocket (char * *address*, int *port*)

This function will create a socket. It uses the supplied network address and port number to create a socket that can write data to that address.

Parameters:

address is the TCP/IP network address of the network node we want to send data to. It can be in a name format, or an address in the IP 'dot' address form.

port is the integer port number to use for the new socket

Returns:

an error indication (if < 0) or the integer value of the socket

Definition at line 50 of file sending.c.

Referenced by startButtonCallback().

9.2.2.2 void fileToggleCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects one of the 'enable file output' toggle buttons. It will make sure that all the file buttons are enabled, and that all the network buttons are disabled.

Parameters:

w the widget that caused the event and this callback. It could be one of two, but we don't care which in this case.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 97 of file play/callbacks-c.c.

**9.2.2.3 void networkToggleCallback (Widget *w*, XtPointer *client_data*,
XtPointer *call_data*)**

This callback is the function that will be called when the Motif user interface selects one of the 'enable network output' toggle buttons. It will make sure that all the network buttons are enabled, and that all the file buttons are disabled.

Parameters:

- w* the widget that caused the event and this callback. It could be one of two, but we don't care which in this case.
- client_data* is a pointer to any (optional) client data defined for the callback (none in this case)
- call_data* the standard event information provided by X11-Motif

Definition at line 120 of file play/callbacks-c.c.

**9.2.2.4 void okFileDialogCallback (Widget *w*, XtPointer *client_data*,
XtPointer *call_data*)**

This callback is the function that will be called when the Motif user interface puts up a file selection dialog and the user selects a file and presses the 'OK' button. We try to check the validity of the new file name by trying to read how many socket messages it contains. If the file is invalid, the user interface will just 'beep' as the user tries to 'ok' the selection of an improper file.

Parameters:

- w* the widget that caused the event and this callback. The push button (ok) on the file selection dialog in this case.
- client_data* is a pointer to any (optional) client data defined for the callback (none in this case)
- call_data* the standard event information provided by X11-Motif

Definition at line 206 of file play/callbacks-c.c.

**9.2.2.5 void pausePlayButtonCallback (Widget *w*, XtPointer *client_data*,
XtPointer *call_data*)**

This callback is the function that will be called when the Motif user interface decides to respond to the user activating one of the 'Pause' push buttons. Since this button can only be active when the program is actually sending, we know that there is a sub-process in existence and that it is running. We want to stop the sub-process, and change the state of active buttons in the interface to allow the user to start it up again.

Parameters:

- w* the widget that caused the event and this callback. It could be one of several widgets in this case, and we don't care which one.
- client_data* is a pointer to any (optional) client data defined for the callback (none in this case)
- call_data* the standard event information provided by X11-Motif

Definition at line 264 of file play/callbacks-c.c.

9.2.2.6 void resetSendButtons ()

This function will reset the start, pause and resume buttons to their default state. This state is the start button enabled, and the others disabled.

Definition at line 109 of file sending.c.

9.2.2.7 void resumePlayButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating the 'Resume' push buttons. Since this button can only be active when the program is sending but in a paused state, we know that there is a sub-process in existence and that it is not running. We want to tell the sub-process to continue, and change the state of active buttons in the interface to show this change in the sub-process.

Parameters:

- w* the widget that caused the event and this callback.
- client_data* is a pointer to any (optional) client data defined for the callback (none in this case)
- call_data* the standard event information provided by X11-Motif

Definition at line 292 of file play/callbacks-c.c.

9.2.2.8 int sendMessages (FILE * *fileD*, FILE * *outFileD*)

This function will be called by the child (sending) process and the process will spend most of its time here in this loop. The child process will loop until all the file of messages has been sent, or it can be paused by a 'signal' from the main program. When the file is done, the child process will trigger a signal to the main process to let it know the file is finished.

Parameters:

- fileD* is the file descriptor for the (already opened) file which contains the socket messages and delays.

outFileD is the file descriptor of the (already opened) output file used to save the (subset?) of the messages if the destination selected is not an network address

Returns:

an error indication (< 0) if there are sending problems

Definition at line 210 of file sending.c.

Referenced by startButtonCallback().

9.2.2.9 void sendProcessDone ()

This function will be called by the child (sending) process when it is done, i.e. when the complete file of socket messages has been sent. This signal is used to queue an event for the user interface to switch the button states as required by this change in program state.

Definition at line 164 of file sending.c.

9.2.2.10 void sendProcessPause ()

This function will be called by the child (sending) process in response to a 'pause' signal. The function will make sure that the process ID is correct and, if it is, it will cause the process to 'pause'. The sending process will remain in this state until another signal changes it.

Definition at line 129 of file sending.c.

9.2.2.11 int sendSocketMessage (int *sock*, char * *buffer*, int *sizeBuffer*)

This function will send a message via an existing socket.

Parameters:

sock is the integer socket to use for sending

buffer is an array of bytes to be sent via the socket. Buffer contents do not have to be only ASCII characters.

sizeBuffer is the integer length of the user supplied message

Returns:

an error indication (if < 0) or the number of characters sent

Definition at line 89 of file sending.c.

9.2.2.12 void speedSliderCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface changed the 'speed' slider on the user interface. It will read the new value from the slider, and compute a speed multiplier to show in the associated text area. This multiplier will also be saved in the shared global structure.

Todo:

We should modify the slider to provide a wider range of speed multiplier values, including one which could implement a single step (single message) mode.

Parameters:

w the widget that caused the event and this callback. We ask the slider widget for its current value.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 147 of file play/callbacks-c.c.

9.2.2.13 void startButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating the 'Start' push button. This is the most complex of the callbacks, because we read the current values for most of the user-modifiable fields, create the socket or output file, re-open the input file, read the start and end message values, and 'fork' a subprocess to send the socket messages. If all this works, then we change the state of the user interface buttons and we are done.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 322 of file play/callbacks-c.c.

9.2.2.14 void usageSend (void)

This function will be called when there is an error with the user supplied command line arguments for the socket playback program. It will simply print out the list of standard and optional command arguments, and then finish.

Definition at line 32 of file play_support.c.

Referenced by shared().

9.3 User functions only for socket record operation

All the functions required only for socket recording are listed here.

Files

- file record/callbacks-c.c
- file record/creation-c.c
- file record/main-c.c
- file record_support.c
- file recording.c

Functions

- void recordButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void stopButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void pauseRecButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void resumeButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void okFileDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void usageRecord (void)
- int createReadSocket (int port)
- void resetReadButtons ()
- void readProcessPause ()
- void readProcessStop ()
- void readProcessDone ()
- int readMessages (FILE *fileD)
- void quitButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)

9.3.1 Detailed Description

All the functions required only for socket recording are listed here.

Many of these are very similar to function required for the playback program so you must be careful when examining the name of the function or variables. Some functions have exactly the same name and parameters, but these were generated by the GUI builder tool, and the names are set by that tool and cannot be modified.

9.3.2 Function Documentation

9.3.2.1 int createReadSocket (int *port*)

This function will create a socket. It uses the supplied port number to create a socket that can read messages on the local host.

Parameters:

port is the integer port number to use for the new socket

Returns:

an error indication (if < 0) or the integer value of the socket

Definition at line 44 of file recording.c.

Referenced by recordButtonCallback().

9.3.2.2 void okFileDialogCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface puts up a file selection dialog and the user selects a file and presses the 'OK' button. We try to check the validity of the new file name by trying to read how many socket messages it contains. If the file is invalid, the user interface will just 'beep' as the user tries to 'ok' the selection of an improper file.

Parameters:

w the widget that caused the event and this callback. The push button (ok) on the file selection dialog in this case.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 274 of file record/callbacks-c.c.

9.3.2.3 void pauseRecButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating one of the 'Pause' push buttons. Since this button can only be active when the program is actually reading, we know that there is a sub-process in existence and that it is running. We want to stop the sub-process, and change

the state of active buttons in the interface to allow the user to start it up again. We also set the current value of the message counter into the text field on the display.

Parameters:

w the widget that caused the event and this callback. It could be one of several widgets in this case, and we don't care which one.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 214 of file record/callbacks-c.c.

9.3.2.4 void quitButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface selects the 'quit' button on the user interface or in the 'File' pull down menu. It will close the sub-process (if any) and then shut down the main program.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 172 of file common.callbacks.c.

9.3.2.5 int readMessages (FILE **fileD*)

This function will be called by the child (reading) process, and this process will spend most of its time here in this loop. The child process will loop until killed (stopped) by the main program. When the file is done, the child process will trigger a signal to the main process to let it know the child process is finished.

Parameters:

sock is the open socket we use for reading messages

fileD is the file descriptor for the (already opened) file into which we will store the socket messages and delays.

Returns:

an error indication (< 0) if there are sending problems

Definition at line 223 of file recording.c.

Referenced by recordButtonCallback().

9.3.2.6 void readProcessDone ()

This function will be called by the child (reading)process when this child process is done, i.e. when terminated by an error or the user. This signal handler is used to queue an event for the user interface to switch the button states as required for the user interface.

Definition at line 179 of file recording.c.

9.3.2.7 void readProcessPause ()

This function will be called by the child (reading) process in response to a 'pause' signal. The function will make sure that the process ID is correct, and if it is it will cause the process to 'pause'. The reading process will remain in this state until another signal changes it.

Definition at line 124 of file recording.c.

9.3.2.8 void readProcessStop ()

This function will be called by the child (reading) process in response to a 'stop' signal. The function will make sure that the process ID is correct and, if it is, it will cause the process to exit. This should trigger the 'child terminated' function in the main process.

Definition at line 160 of file recording.c.

9.3.2.9 void recordButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating the 'Record' push buttons. This is the most complex of the callbacks, because we read the current values for most of the user-modifiable fields, create the socket, re-open the output file, and 'fork' a subprocess to read the socket messages. If all this works, then we change the state of the user interface buttons and we are done.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 99 of file record/callbacks-c.c.

9.3.2.10 void resetReadButtons ()

This function will be called only if the recording child process is terminated. We will reset the record, stop, pause and resume buttons to their default state. This state is the 'start' button enabled, and the others disabled. It will also close the output file and write the number of messages stored into the user interface.

Definition at line 89 of file recording.c.

9.3.2.11 void resumeButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating the 'Resume' push buttons. Since this button can only be active when the program is reading but in a paused state, we know that there is a sub-process in existence and that it is not running. We want to tell the sub-process to continue, and change the state of active buttons in the interface to show this change in the sub-process.

Parameters:

w the widget that caused the event and this callback.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 243 of file record/callbacks-c.c.

9.3.2.12 void stopButtonCallback (Widget *w*, XtPointer *client_data*, XtPointer *call_data*)

This callback is the function that will be called when the Motif user interface decides to respond to the user activating one of the 'Stop' push buttons. Since this button can only be active when the program is actually reading, we know that there is a sub-process in existence and that it is running. We want to terminate the sub-process, and change the state of active buttons in the interface to reflect this. We also set the current value of the message counter into the text field on the display.

Parameters:

w the widget that caused the event and this callback. It could be one of several widgets in this case, and we don't care which one.

client_data is a pointer to any (optional) client data defined for the callback (none in this case)

call_data the standard event information provided by X11-Motif

Definition at line 186 of file record/callbacks-c.c.

9.3.2.13 void usageRecord (void)

This function will be called when there is an error with the user supplied command line arguments for the socket recording application. It will simply print out the list of standard and optional command arguments, and then finish.

Definition at line 28 of file record_support.c.

Referenced by shared().

Chapter 10

Record/Play Sockets: Application Class Documentation

10.1 _globalMemory Struct Reference

This is the structure to declare all the global variables required for the main program of the Motif GUI interface, and for the child process we spawn to do the actual recording/sending of messages.

```
#include <myInterface.h>
```

Public Attributes

- char inputFileName [MAX_NAME_TEXT_SIZE]
the character array used to hold the name of the input file.
- char outputFileName [MAX_NAME_TEXT_SIZE]
the character array used to hold the name of the output file.
- char address [MAX_NAME_TEXT_SIZE]
the network address to send messages to, or output file name.
- char portNumbText [MAX_NUMBER_TEXT_SIZE]

port number (as a text string) to send/listen to.

- int portNumb
port number (as an integer) to send/listen messages to.
- FILE* fileDescrip
the input or output socket file, as a file descriptor.
- int messageNumber
the number of messages currently stored in the file.
- int startMesNumb
the message number in file to start sending.
- int endMesNumb
the message number in file to stop sending.
- int socketNumb
the socket we create when sending/reading via the network.
- float speedFactor
the rate we send, ie faster/slower than recorded.
- int globalDebugMode
when this flag is set, print extra debug statements.
- int childProcess
the process id of the sending/reading sub-process.
- XtAppContext mainAppContext
the application context of main process.

10.1.1 Detailed Description

This is the structure to declare all the global variables required for the main program of the Motif GUI interface, and for the child process we spawn to do the actual recording/sending of messages.

We use the same structure for both record and playback for simplicity, and with a small cost of extra memory requirement. The members of this structure are related mostly by their common requirement for global scope.

Remarks:

This requirement for global scope would disappear if the code was re-written into the C++ computer language.

Definition at line 64 of file myInterface.h.

10.1.2 Member Data Documentation**10.1.2.1 char _globalMemory::address[MAX_NAME_TEXT_SIZE]**

the network address to send messages to, or output file name.

Definition at line 71 of file myInterface.h.

10.1.2.2 int _globalMemory::childProcess

the process id of the sending/reading sub-process.

Definition at line 91 of file myInterface.h.

10.1.2.3 int _globalMemory::endMesNumb

the message number in file to stop sending.

Definition at line 83 of file myInterface.h.

10.1.2.4 FILE * _globalMemory::fileDescrip

the input or output socket file, as a file descriptor.

Definition at line 77 of file myInterface.h.

10.1.2.5 int _globalMemory::globalDebugMode

when this flag is set, print extra debug statements.

Definition at line 89 of file myInterface.h.

10.1.2.6 char _globalMemory::inputFileName[MAX_NAME_TEXT_SIZE]

the character array used to hold the name of the input file.

Definition at line 67 of file myInterface.h.

10.1.2.7 XtApplicationContext _globalMemory::mainAppContext

the application context of main process.

Definition at line 93 of file myInterface.h.

10.1.2.8 int _globalMemory::messageNumber

the number of messages currently stored in the file.

Definition at line 79 of file myInterface.h.

10.1.2.9 char _globalMemory::outputFileName[MAX_NAME_TEXT_SIZE]

the character array used to hold the name of the output file.

Definition at line 69 of file myInterface.h.

10.1.2.10 int _globalMemory::portNumb

port number (as an integer) to send/listen messages to.

Definition at line 75 of file myInterface.h.

10.1.2.11 char _globalMemory::portNumbText[MAX_NUMBER_TEXT_SIZE]

port number (as a text string) to send/listen to.

Definition at line 73 of file myInterface.h.

10.1.2.12 int _globalMemory::socketNumb

the socket we create when sending/reading via the network.

Definition at line 85 of file myInterface.h.

10.1.2.13 float _globalMemory::speedFactor

the rate we send, ie faster/slower than recorded.

Definition at line 87 of file myInterface.h.

10.1.2.14 int _globalMemory::startMesNumb

the message number in file to start sending.

Definition at line 81 of file myInterface.h.

The documentation for this struct was generated from the following file:

- myInterface.h

Chapter 11

Record/Play Sockets: Application File Documentation

11.1 callbacks-c.c File Reference

```
#include <Xm/Xm.h>
#include <Xm/TextF.h>
#include <signal.h>
#include "creation-c.h"
#include "../common/myInterface.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

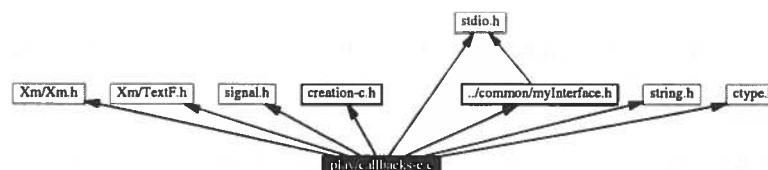


Figure 11.1: Include dependency graph for `play_callbacks-c.c`

Defines

- #define PROTOTYPE(p) ()
- #define ARGLIST(p) p
- #define ARG(a,b) a b;
- #define GRA(a,b) a b;
- #define UARG(a,b) a b;
- #define GRAU(a,b) a b;

Functions

- Widget BxFindTopShell PROTOTYPE ((Widget))
- WidgetList BxWidgetIdsFromNames PROTOTYPE ((Widget,char *,char *))
- void fileToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void networkToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void speedSliderCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void okFileDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void pausePlayButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void resumePlayButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void startButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)

Variables

- globalMemory shared
the shared, global variable structure.

11.1.1 Detailed Description

In this source file are the definitions for the callback routines that will be called by Motif and X11 when the user clicks on push buttons, toggle buttons and pull down menu options for the 'playback sockets' application.

Some callback function are simple enough to fully define here. But, many require a call to some of the support and sending functions defined in other source files. A few will actually start other process or send signals, and the support functions will define what happens when these signals are processed.

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):

Daniel F. Johnston

Since:

April 2002

Id:

callbacks-c.c,v 1.10 2002/08/20 15:37:01 johnston Exp

Definition in file play/callbacks-c.c.

11.1.2 Define Documentation**11.1.2.1 #define ARG(a, b) a b;**

Definition at line 32 of file play/callbacks-c.c.

11.1.2.2 #define ARGLIST(p) p

Definition at line 31 of file play/callbacks-c.c.

11.1.2.3 #define GRA(a, b) a b;

Definition at line 33 of file play/callbacks-c.c.

11.1.2.4 #define GRAU(a, b) a b;

Definition at line 35 of file play/callbacks-c.c.

11.1.2.5 #define PROTOTYPE(p) 0

Definition at line 30 of file play/callbacks-c.c.

11.1.2.6 #define UARG(a, b) a b;

Definition at line 34 of file play/callbacks-c.c.

11.1.3 Variable Documentation

11.1.3.1 globalMemory shared

the shared, global variable structure.

Definition at line 88 of file record/main-c.c.

11.2 callbacks-c.c File Reference

```
#include <Xm/Xm.h>
#include <Xm/TextF.h>
#include <signal.h>
#include "creation-c.h"
#include "../common/myInterface.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

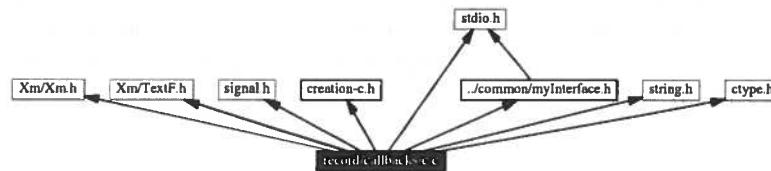


Figure 11.2: Include dependency graph for record_callbacks.c.c

Defines

- #define PROTOTYPE(p) ()
- #define ARGLIST(p) p
- #define ARG(a,b) a b;
- #define GRA(a,b) a b;
- #define UARG(a,b) a b;
- #define GRAU(a,b) a b;

Functions

- Widget BxFindTopShell PROTOTYPE ((Widget))
- WidgetList BxWidgetIdsFromNames PROTOTYPE ((Widget,char *,char *))
- void recordButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void stopButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void pauseRecButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)

- void resumeButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void okFileDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)

Variables

- globalMemory shared
the shared, global variable structure.

11.2.1 Detailed Description

In this source file are the definitions for the callback routines that will be called by Motif and X11 when the user clicks on push buttons, toggle buttons and pull down menu options for the 'record sockets' application.

Some callback function are simple enough to fully define here. But, many require a call to some of the support and sending functions defined in other source files. A few will actually start other process or send signals, and the support functions will define what happens when these signals are processed.

```
Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

April 2002

Id:

callbacks-c.c,v 1.7 2002/08/20 15:36:40 johnston Exp

Definition in file record/callbacks-c.c.

11.2.2 Define Documentation

11.2.2.1 #define ARG(a, b) a b;

Definition at line 32 of file record/callbacks-c.c.

11.2.2.2 #define ARGLIST(p) p

Definition at line 31 of file record/callbacks-c.c.

11.2.2.3 #define GRA(a, b) a b;

Definition at line 33 of file record/callbacks-c.c.

11.2.2.4 #define GRAU(a, b) a b;

Definition at line 35 of file record/callbacks-c.c.

11.2.2.5 #define PROTOTYPE(p) ()

Definition at line 30 of file record/callbacks-c.c.

11.2.2.6 #define UARG(a, b) a b;

Definition at line 34 of file record/callbacks-c.c.

11.2.3 Variable Documentation**11.2.3.1 globalMemory shared**

the shared, global variable structure.

Definition at line 82 of file record/callbacks-c.c.

11.3 common_callbacks.c File Reference

```
#include <Xm/Xm.h>
#include <Xm/TextF.h>
#include <signal.h>
#include "mySocket.h"
#include "myInterface.h"
```

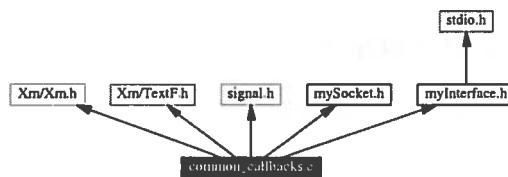


Figure 11.3: Include dependency graph for common_callbacks.c

Functions

- void debugToggleCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void aboutHelpCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void aboutDismissCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void usageHelpCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void browseButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void cancelFileDialogCallback (Widget w,XtPointer client_data,XtPointer call_data)
- void quitButtonCallback (Widget w,XtPointer client_data,XtPointer call_data)

Variables

- globalMemory shared
the shared, global variable structure.
- Widget inputFileSelectionBox
the global reference to the file browser window.

- Widget aboutHelpBulletinBoard
the global reference to the help display window.

11.3.1 Detailed Description

In this source file are the definitions for the callback routines that will be called by Motif and X11 when the user clicks on push buttons, toggle buttons and pull down menu options for the 'record and playback sockets' application.

These routines are the same for both applications and are thus defined in this 'common' source file.

```
Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

August 2002

Id:

common_callbacks.c,v 1.5 2002/10/16 18:05:35 johnston Exp

Definition in file common_callbacks.c.

11.3.2 Variable Documentation

11.3.2.1 Widget aboutHelpBulletinBoard

the global reference to the help display window.

Definition at line 37 of file common_callbacks.c.

11.3.2.2 Widget inputFileSelectionBox

the global reference to the file browser window.

Definition at line 35 of file common_callbacks.c.

11.3.2.3 globalMemory shared

the shared, global variable structure.

Definition at line 33 of file common_callbacks.c.

11.4 common_support.c File Reference

```
#include <Xm/Xm.h>
#include "mySocket.h"
#include "myInterface.h"
```

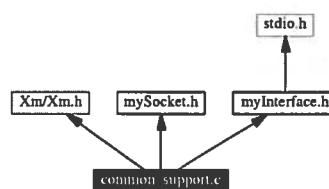


Figure 11.4: Include dependency graph for common_support.c

Defines

- #define XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG XmSTRING_COMPONENT_CHARSET

Functions

- void AppendString (char **return_text,int *length,char *text)
- Boolean XmStringToString (XmString xmstr,char **return_text)
- int getNumbMessages (FILE *fileD)
- int readNumberMessages (char *fileName)

11.4.1 Detailed Description

This file contains the support function needed for both applications. We have defined motif string manipulation and functions to read the number of standard socket messages in a file.

```
Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

May 2002

Id:

common_support.c,v 1.5 2002/10/16 18:05:35 johnston Exp

Definition in file common_support.c.

11.4.2 Define Documentation

11.4.2.1 #define XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG XmSTRING_COMPONENT_CHARSET

Definition at line 58 of file common_support.c.

11.5 creation-c.c File Reference

```
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/DialogS.h>
#include <Xm/MwmUtil.h>
#include <Xm/BulletinB.h>
#include <Xm/ScrolledW.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/FileSB.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/Separator.h>
#include <Xm/ToggleB.h>
#include <Xm/Form.h>
#include <Xm/TextF.h>
#include <Xm/ScrollBar.h>
#include "creation-c.h"
```

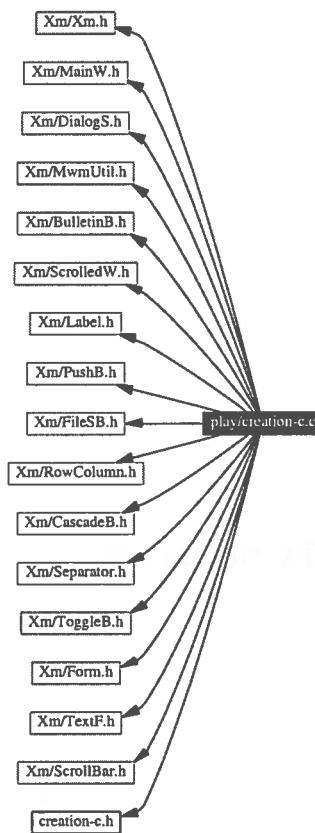


Figure 11.5: Include dependency graph for `play_creation.c.c`

Functions

- `void RegisterBxConverters (XtApplicationContext)`
- `XtPointer BX_CONVERT (Widget,char *,char *,int,Boolean *)`
- `XtPointer BX_DOUBLE (double)`
- `XtPointer BX_SINGLE (float)`
- `void BX_MENU_POST (Widget,XtPointer,XEvent *,Boolean *)`
- `Pixmap XPM_PIXMAP (Widget,char **)`
- `void BX_SET_BACKGROUND_COLOR (Widget,ArgList,Cardinal *,Pixel)`
- `Widget CreatemainWindow (Widget parent)`

11.5.1 Detailed Description

In this source file is the code generated automatically by the user interface builder software. It will use this source again, preserving user supplied code between the special brackets, when the software is used to modify the interface.

```
Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

July 2002

Id:

creation-c.c,v 1.9 2002/10/16 18:20:27 johnston Exp

Definition in file play/creation-c.c.

11.5.2 Define Documentation

11.5.2.1 #define DECLARE_BX_GLOBALS

Definition at line 60 of file play/creation-c.c.

11.5.3 Function Documentation

11.5.3.1 Widget CreatemainWindow (Widget *parent*)

Definition at line 107 of file play/creation-c.c.

11.6 creation-c.c File Reference

```
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/DialogS.h>
#include <Xm/MwmUtil.h>
#include <Xm/BulletinB.h>
#include <Xm/ScrolledW.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/FileSB.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/Separator.h>
#include <Xm/ToggleB.h>
#include <Xm/Form.h>
#include <Xm/TextF.h>
#include "creation-c.h"
```

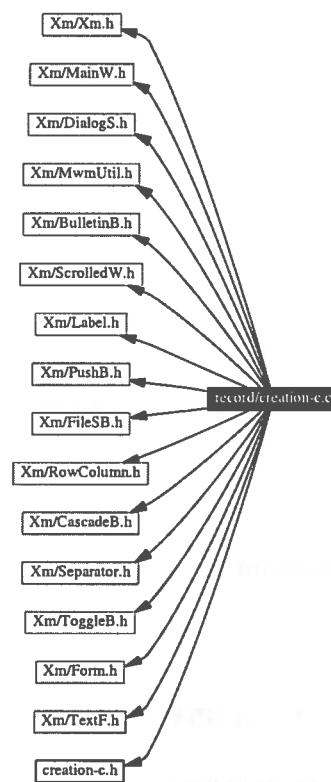


Figure 11.6: Include dependency graph for record_creation-c.c

Functions

- `void RegisterBxConverters (XtApplicationContext)`
- `XtPointer BX_CONVERT (Widget,char *,char *,int,Boolean *)`
- `XtPointer BX_DOUBLE (double)`
- `XtPointer BX_SINGLE (float)`
- `void BX_MENU_POST (Widget,XtPointer,XEvent *,Boolean *)`
- `Pixmap XPM_PIXMAP (Widget,char **)`
- `void BX_SET_BACKGROUND_COLOR (Widget,ArgList,Cardinal *,Pixel)`
- `Widget CreatemainWindow (Widget parent)`

11.6.1 Detailed Description

In this source file is the source code generated automatically by the user interface builder software. It will use this source

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):
Daniel F. Johnston

Since:
July 2002

Id:
creation-c.c,v 1.6 2002/08/19 19:55:02 johnston Exp

Definition in file record/creation-c.c.

11.6.2 Define Documentation

11.6.2.1 #define DECLARE_BX_GLOBALS

Definition at line 57 of file record/creation-c.c.

11.6.3 Function Documentation

11.6.3.1 Widget Createmainwindow (Widget *parent*)

Definition at line 101 of file record/creation-c.c.

11.7 main-c.c File Reference

```
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/MwmUtil.h>
#include "creation-c.h"
#include "../common/mySocket.h"
#include "../common/myInterface.h"
```

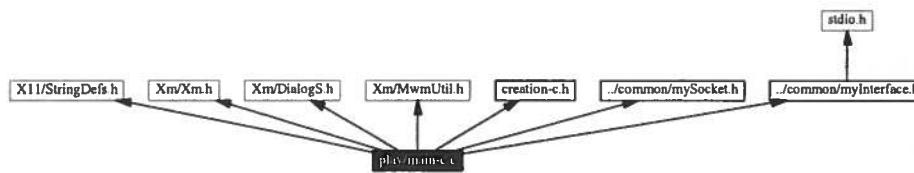


Figure 11.7: Include dependency graph for play_main-c.c

Defines

- #define BX_APP_CLASS "BuilderProduct"

Functions

- void RegisterBxConverters (XtApplicationContext)
- XtPointer BX_CONVERT (Widget,char *,char *,int,Boolean *)
- XtPointer BX_DOUBLE (double)
- XtPointer BX_SINGLE (float)
- void BX_MENU_POST (Widget,XtPointer,XEvent *,Boolean *)
- Pixmap XPM_PIXMAP (Widget,char **)
- void BX_SET_BACKGROUND_COLOR (Widget,ArgList,Cardinal *,Pixel)
- int main (int argc,char **argv)

Variables

- globalMemory shared

the block of user defined globals shared by this interface.

11.7.1 Detailed Description

This file is mostly created by the user interface builder tool Builder Xcessory version 5.0. There are only a few places where we can add our input.

The purpose of this file is to create the user interface, set some of the initial variables and user interface widget states, and then loop waiting for (and responding to) events.

It is the response to the events, mainly in the callbacks source file that will define the logic of the program.

```
Copyright (c) 2002  
All content of this file is copyrighted to the  
National Research Council of Canada.  
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

April 2002

Id:

main-c.c,v 1.6 2002/08/19 18:46:55 johnston Exp

Definition in file play/main-c.c.

11.7.2 Define Documentation

11.7.2.1 #define BX_APP_CLASS "BuilderProduct"

Definition at line 90 of file play/main-c.c.

11.7.3 Function Documentation

11.7.3.1 int main (int argc, char ** argv)

Definition at line 92 of file play/main-c.c.

11.7.4 Variable Documentation

11.7.4.1 globalMemory shared

the block of user defined globals shared by this interface.

Definition at line 84 of file play/main-c.c.

11.8 main-c.c File Reference

```
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/MwmUtil.h>
#include "creation-c.h"
#include "../common/mySocket.h"
#include "../common/myInterface.h"
```

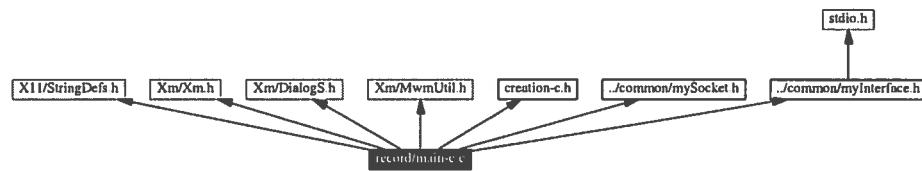


Figure 11.8: Include dependency graph for record_main-c.c

Defines

- #define BX_APP_CLASS "BuilderProduct"

Functions

- void RegisterBxConverters (XtApplicationContext)
- XtPointer BX_CONVERT (Widget,char *,char *,int,Boolean *)
- XtPointer BX_DOUBLE (double)
- XtPointer BX_SINGLE (float)
- void BX_MENU_POST (Widget,XtPointer,XEvent *,Boolean *)
- Pixmap XPM_PIXMAP (Widget,char **)
- void BX_SET_BACKGROUND_COLOR (Widget,ArgList,Cardinal *,Pixel)
- int main (int argc,char **argv)

Variables

- globalMemory shared

the block of user defined globals shared by this interface.

11.8.1 Detailed Description

This file is mostly created by the user interface builder tool Builder Xcessory version 5.0. There are only a few places where we can add our input.

The purpose of this file is to create the user interface, set some of the initial variables and user interface widget states, and then loop waiting for (and responding to) events.

It is the response to the events, mainly in the callbacks source file that will define the logic of the program.

Todo:

We should add logic to the main program to support '-h' or '-help' as a valid command line argument. It would simply cause the "usage" message to be written.

```
Copyright (c) 2002  
All content of this file is copyrighted to the  
National Research Council of Canada.  
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

August 2002

Id:

main-c.c,v 1.6 2002/08/19 18:46:03 johnston Exp

Definition in file record/main-c.c.

11.8.2 Define Documentation

11.8.2.1 #define BX_APP_CLASS "BuilderProduct"

Definition at line 94 of file record/main-c.c.

11.8.3 Function Documentation

11.8.3.1 int main (int *argc*, char ** *argv*)

Definition at line 96 of file record/main-c.c.

11.8.4 Variable Documentation**11.8.4.1 globalMemory shared**

the block of user defined globals shared by this interface.

Definition at line 88 of file record/main-c.c.

11.9 myInterface.h File Reference

```
#include <stdio.h>
```

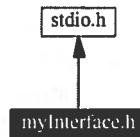


Figure 11.9: Include dependency graph for myInterface.h

This graph shows which files directly or indirectly include this file:

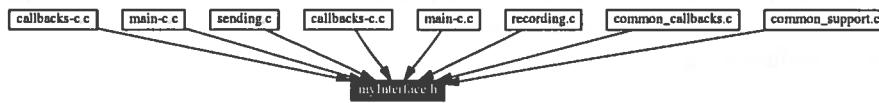


Figure 11.10: Include dependency graph for myInterface.h

Compounds

- struct _globalMemory

Defines

- #define MAX_NAME_TEXT_SIZE 124

Motif GUI interface defined constants This constant will specify the maximum size of general text and filename input as read from command line arguments and large text fields.

- #define MAX_NUMBER_TEXT_SIZE 20

Motif GUI interface defined constants This constant will specify the biggest number of text digits we will deal with from small text fields.

Typedefs

- typedef struct _globalMemory globalMemory

This is the structure to declare all the global variables required for the main program of the Motif GUI interface, and for the child process we spawn to do the actual recording/sending of messages.

11.9.1 Detailed Description

This file will contain a number of things important to the Motif GUI interfaces created for the 'Mimic Sockets' or the 'record and playback Sockets' applications. It contains some defines that set maximum sizes for the arrays, and it describes the structure that contains all the global memory of the application. Some of the parameters of the global structure are only required for one application or the other, but are all defined here for simplicity. (We accept the small cost of the extra memory that must be allocated.)

Note: The structure is defined here in an attempt to simplify the source code. You could create all the elements as individual global variables, and include reference to them as needed. But, I find that the source code becomes complicated to read. It, i.e. the common definition for the globals, also makes the source code more consistent with the set of globals created by the user interface builder tool for some elements of the user interface widgets.

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):

Daniel F. Johnston

Since:

April 2002

Id:

myInterface.h,v 1.6 2002/10/16 18:21:43 johnston Exp

Definition in file myInterface.h.

11.9.2 Define Documentation

11.9.2.1 #define MAX_NAME_TEXT_SIZE 124

Motif GUI interface defined constants This constant will specify the maximum size of general text and filename input as read from command line arguments and large text fields.

Definition at line 43 of file myInterface.h.

11.9.2.2 #define MAX_NUMBER_TEXT_SIZE 20

Motif GUI interface defined constants This constant will specify the biggest number of text digits we will deal with from small text fields.

Definition at line 50 of file myInterface.h.

11.9.3 Typedef Documentation

11.9.3.1 typedef struct _globalMemory globalMemory

This is the structure to declare all the global variables required for the main program of the Motif GUI interface, and for the child process we spawn to do the actual recording/sending of messages.

We use the same structure for both record and playback for simplicity, and with a small cost of extra memory requirement. The members of this structure are related mostly by their common requirement for global scope.

Remarks:

This requirement for global scope would disappear if the code was re-written into the C++ computer language.

11.10 mySocket.h File Reference

This graph shows which files directly or indirectly include this file:

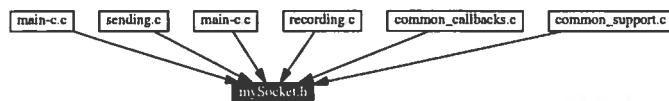


Figure 11.11: Include dependency graph for mySocket.h

Defines

- `#define LOCAL_ADDRESS "localhost"`
the default value for the address of the local computer.
- `#define DEFAULT_PORT_NUMBER 10025`
the default value we will use for a network port number.
- `#define MAX_MESSAGE_SIZE 1024`
the biggest socket message we will deal with.

11.10.1 Detailed Description

This file contains some standard values (defines) that we use for creating TCP/IP network communication applications.

```

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
  
```

Author(s):

Daniel F. Johnston

Since:

April 2002

Id:

mySocket.h,v 1.3 2002/08/19 18:46:19 johnston Exp

Definition in file mySocket.h.

11.10.2 Define Documentation

11.10.2.1 #define DEFAULT_PORT_NUMBER 10025

the default value we will use for a network port number.

Definition at line 24 of file mySocket.h.

11.10.2.2 #define LOCAL_ADDRESS "localhost"

the default value for the address of the local computer.

Definition at line 21 of file mySocket.h.

11.10.2.3 #define MAX_MESSAGE_SIZE 1024

the biggest socket message we will deal with.

Definition at line 27 of file mySocket.h.

11.11 play_support.c File Reference

Functions

- void usageSend (void)

Variables

- unsigned int frameNumber
a counter of where we are in reading file.

11.11.1 Detailed Description

This file contains the support function needed for the application. We have usage printing function and motif string manipulation

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):

Daniel F. Johnston

Since:

May 2002

Id:

play_support.c,v 1.3 2002/08/19 18:46:55 johnston Exp

Definition in file play_support.c.

11.11.2 Variable Documentation

11.11.2.1 unsigned int frameNumber

a counter of where we are in reading file.

Definition at line 22 of file play_support.c.

11.12 record_support.c File Reference

Functions

- void usageRecord (void)

11.12.1 Detailed Description

This file contains the support function needed for the recording application. We have a usage printing function.

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):

Daniel F. Johnston

Since:

May 2002

Id:

record_support.c,v 1.4 2002/10/16 18:22:18 johnston Exp

Definition in file record_support.c.

11.13 recording.c File Reference

```
#include <Xm/Xm.h>
#include "creation-c.h"
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <time.h>
#include "../common/mySocket.h"
#include "../common/myInterface.h"
```

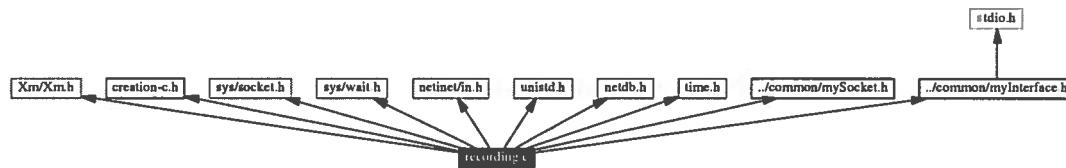


Figure 11.12: Include dependency graph for recording.c

Functions

- int createReadSocket (int port)
- void resetReadButtons ()
- void readProcessPause ()
- void readProcessStop ()
- void readProcessDone ()
- int readMessages (FILE *fileD)

Variables

- char buf [MAX_MESSAGE_SIZE]
the byte array that will hold a UDP socket message before storing.

11.13.1 Detailed Description

This file contains the support functions needed for creating the socket and reading socket messages. We also provide functions for controlling the sending process, i.e. pause/resume, sending done, etc.

Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.

Author(s):

Daniel F. Johnston

Since:

Aug 2002

Id:

recording.c,v 1.7 2002/10/16 18:13:43 johnston Exp

Definition in file recording.c.

11.13.2 Variable Documentation

11.13.2.1 char buf[MAX_MESSAGE_SIZE]

the byte array that will hold a UDP socket message before storing.

Definition at line 34 of file recording.c.

11.14 sending.c File Reference

```
#include <Xm/Xm.h>
#include "creation-c.h"
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <time.h>
#include "../common/mySocket.h"
#include "../common/myInterface.h"
```

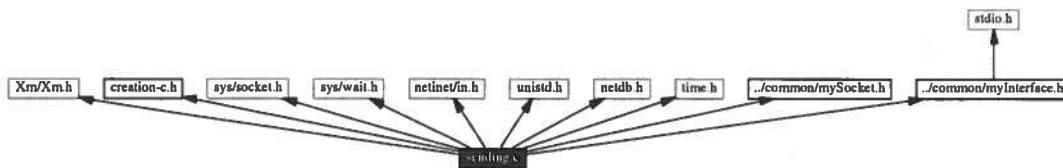


Figure 11.13: Include dependency graph for sending.c

Functions

- int createSendSocket (char *address,int port)
- int sendSocketMessage (int sock,char *buffer,int sizeBuffer)
- void resetSendButtons ()
- void sendProcessPause ()
- void sendProcessDone ()
- int sendMessages (FILE *fileD,FILE *outFileD)

Variables

- char sendBuf [MAX_MESSAGE_SIZE]
the byte array that will hold a UDP socket message for sending.

- struct sockaddr_in server
the socket address structure for the destination (if defined).

11.14.1 Detailed Description

This file contains the support functions needed for creating the socket and sending socket messages. We also provide functions for controlling the sending process, i.e. speed, pause/resume, sending done, etc.

```
Copyright (c) 2002
All content of this file is copyrighted to the
National Research Council of Canada.
All rights reserved.
```

Author(s):

Daniel F. Johnston

Since:

May 2002

Id:

sending.c,v 1.10 2002/10/16 18:10:30 johnston Exp

Definition in file sending.c.

11.14.2 Variable Documentation

11.14.2.1 char sendBuf[MAX_MESSAGE_SIZE]

the byte array that will hold a UDP socket message for sending.

Definition at line 34 of file sending.c.

11.14.2.2 struct sockaddr_in server

the socket address structure for the destination (if defined).

Definition at line 36 of file sending.c.

Chapter 12

Record/Play Sockets: Application Page Documentation

12.1 Todo List

file main-c.c We should add logic to the main program to support '-h' or '-help' as a valid command line argument. It would simply cause the "usage" message to be written.

member ::speedSliderCallback(Widget w,XtPointer client_data,XtPointer call_data)
We should modify the slider to provide a wider range of speed multiplier values, including one which could implement a single step (single message) mode.

member ::usageHelpCallback(Widget w,XtPointer client_data,XtPointer call_data)
We need to add code so that the two different socket program help files can be stored in some directory other than the same directory as the program executables.

We should provide a mechanism to change the name of the html document used for help.

2.1 Record/Play Sockets
2.1.1 Record Sockets
2.1.1.1 Record Sockets Overview
2.1.1.2 Record Sockets API
2.1.1.3 Record Sockets Examples
2.1.2 Play Sockets
2.1.2.1 Play Sockets Overview
2.1.2.2 Play Sockets API
2.1.2.3 Play Sockets Examples

Chapter 13

Software Index

Index

-globalMemory, 53
 address, 55
 childProcess, 55
 endMesNumb, 55
 fileDescrip, 55
 globalDebugMode, 55
 inputFileName, 55
 mainAppContext, 56
 messageNumber, 56
 outputFileName, 56
 portNumb, 56
 portNumbText, 56
 socketNumb, 56
 speedFactor, 56
 startMesNumb, 56

aboutDismissCallback
 mod_common, 35
aboutHelpBulletinBoard
 common_callbacks.c, 67
aboutHelpCallback
 mod_common, 35
address
 -globalMemory, 55
All user functions common to both
 socket programs, 33
AppendString
 mod_common, 34
ARG
 play/callbacks-c.c, 61
 record/callbacks-c.c, 64
ARGLIST
 play/callbacks-c.c, 61
 record/callbacks-c.c, 64

browseButtonCallback
 mod_common, 35

buf
 recording.c, 92

BX_APP_CLASS
 play/main-c.c, 78
 record/main-c.c, 81

BX_CONVERT
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

BX_DOUBLE
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

BX_MENU_POST
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

BX_SET_BACKGROUND_COLOR
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

BX_SINGLE
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

callbacks-c.c, 59, 63
cancelFileDialogCallback
 mod_common, 36
childProcess
 -globalMemory, 55
common_callbacks.c, 66

aboutHelpBulletinBoard, 67
inputFileSelectionBox, 67
shared, 67
common_support.c, 69
 XmSTRING_COMPONENT_-
 FONTLIST_ELEMENT_-
 TAG, 70
CreatemainWindow
 play/creation-c.c, 73
 record/creation-c.c, 76
createReadSocket
 mod_record, 47
createSendSocket
 mod_play, 40
creation-c.c, 71, 74

debugToggleCallback
 mod_common, 36
DECLARE_BX_GLOBALS
 play/creation-c.c, 73
 record/creation-c.c, 76
DEFAULT_PORT_NUMBER
 mySocket.h, 87

endMesNumb
 _globalMemory, 55

fileDescrip
 _globalMemory, 55
fileToggleCallback
 mod_play, 40
frameNumber
 play_support.c, 88

getNumbMessages
 mod_common, 37
globalDebugMode
 _globalMemory, 55
globalMemory
 myInterface.h, 85
GRA
 play/callbacks-c.c, 61
 record/callbacks-c.c, 65
GRAU
 play/callbacks-c.c, 61
 record/callbacks-c.c, 65

inputFileName
 _globalMemory, 55
inputFileSelectionBox
 common_callbacks.c, 67

LOCAL_ADDRESS
 mySocket.h, 87

main
 play/main-c.c, 78
 record/main-c.c, 81
main-c.c, 77, 80
mainApplicationContext
 _globalMemory, 56
MAX_MESSAGE_SIZE
 mySocket.h, 87
MAX_NAME_TEXT_SIZE
 myInterface.h, 84
MAX_NUMBER_TEXT_SIZE
 myInterface.h, 85
messageNumber
 _globalMemory, 56
mod_common
 aboutDismissCallback, 35
 aboutHelpCallback, 35
 AppendString, 34
 browseButtonCallback, 35
 cancelFileDialogCallback, 36
 debugToggleCallback, 36
 getNumbMessages, 37
 readNumberMessages, 37
 usageHelpCallback, 37
 XmStringToString, 34
mod_play
 createSendSocket, 40
 fileToggleCallback, 40
 networkToggleCallback, 40
 okFilePlayDialogCallback, 41
 pausePlayButtonCallback, 41
 resetSendButtons, 42
 resumePlayButtonCallback, 42
 sendMessage, 42
 sendProcessDone, 43
 sendProcessPause, 43
 sendSocketMessage, 43
 speedSliderCallback, 43

startButtonCallback, 44
 usageSend, 44
mod_record
 createReadSocket, 47
 okFileDialogCallback, 47
 pauseRecButtonCallback, 47
 quitButtonCallback, 48
 readMessages, 48
 readProcessDone, 48
 readProcessPause, 49
 readProcessStop, 49
 recordButtonCallback, 49
 resetReadButtons, 49
 resumeButtonCallback, 50
 stopButtonCallback, 50
 usageRecord, 50
myInterface.h, 83
 globalMemory, 85
 MAX_NAME_TEXT_SIZE, 84
 MAX_NUMBER_TEXT_SIZE,
 85
mySocket.h, 86
 DEFAULT_PORT_NUMBER, 87
 LOCAL_ADDRESS, 87
 MAX_MESSAGE_SIZE, 87

networkToggleCallback
 mod_play, 40

okFileDialogCallback
 mod_record, 47
okFilePlayDialogCallback
 mod_play, 41
outputFileName
 _globalMemory, 56

pausePlayButtonCallback
 mod_play, 41
pauseRecButtonCallback
 mod_record, 47
play/callbacks-c.c
 ARG, 61
 ARGLIST, 61
 GRA, 61
 GRAU, 61
 PROTOTYPE, 60, 61

shared, 62
UARG, 61
play/creation-c.c
 BX_CONVERT, 72
 BX_DOUBLE, 72
 BX_MENU_POST, 72
 BX_SET_BACKGROUND_-
 COLOR, 72
 BX_SINGLE, 72
 CreatemainWindow, 73
 DECLARE_BX_GLOBALS, 73
 RegisterBxConverters, 72
 XPM_PIXMAP, 72

play/main-c.c
 BX_APP_CLASS, 78
 BX_CONVERT, 77
 BX_DOUBLE, 77
 BX_MENU_POST, 77
 BX_SET_BACKGROUND_-
 COLOR, 77
 BX_SINGLE, 77
 main, 78
 RegisterBxConverters, 77
 shared, 79
 XPM_PIXMAP, 77

play_support.c, 88
 frameNumber, 88
portNumb
 _globalMemory, 56
portNumbText
 _globalMemory, 56
PROTOTYPE
 play/callbacks-c.c, 60, 61
 record/callbacks-c.c, 63, 65

quitButtonCallback
 mod_record, 48

readMessages
 mod_record, 48
readNumberMessages
 mod_common, 37
readProcessDone
 mod_record, 48
readProcessPause
 mod_record, 49

readProcessStop
 mod_record, 49

record/callbacks-c.c
 ARG, 64
 ARGLIST, 64
 GRA, 65
 GRAU, 65
 PROTOTYPE, 63, 65
 shared, 65
 UARG, 65

record/creation-c.c
 BX_CONVERT, 75
 BX_DOUBLE, 75
 BX_MENU_POST, 75
 BX_SET_BACKGROUND_-
 COLOR, 75
 BX_SINGLE, 75
 CreatemainWindow, 76
 DECLARE_BX_GLOBALS, 76
 RegisterBxConverters, 75
 XPM_PIXMAP, 75

record/main-c.c
 BX_APP_CLASS, 81
 BX_CONVERT, 80
 BX_DOUBLE, 80
 BX_MENU_POST, 80
 BX_SET_BACKGROUND_-
 COLOR, 80
 BX_SINGLE, 80
 main, 81
 RegisterBxConverters, 80
 shared, 82
 XPM_PIXMAP, 80

record_support.c, 90

recordButtonCallback
 mod_record, 49

recording.c, 91
 buf, 92

RegisterBxConverters
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

resetReadButtons
 mod_record, 49

resetSendButtons

 mod_play, 42

resumeButtonCallback
 mod_record, 50

resumePlayButtonCallback
 mod_play, 42

sendBuf
 sending.c, 94

sending.c, 93
 sendBuf, 94
 server, 94

sendMessages
 mod_play, 42

sendProcessDone
 mod_play, 43

sendProcessPause
 mod_play, 43

sendSocketMessage
 mod_play, 43

server
 sending.c, 94

shared
 common_callbacks.c, 67
 play/callbacks-c.c, 62
 play/main-c.c, 79
 record/callbacks-c.c, 65
 record/main-c.c, 82

socketNumb
 _globalMemory, 56

speedFactor
 _globalMemory, 56

speedSliderCallback
 mod_play, 43

startButtonCallback
 mod_play, 44

startMesNumb
 _globalMemory, 56

stopButtonCallback
 mod_record, 50

UARG
 play/callbacks-c.c, 61
 record/callbacks-c.c, 65

usageHelpCallback
 mod_common, 37

usageRecord

mod_record, 50
usageSend
 mod_play, 44
User functions only for socket play-back operation, 39
User functions only for socket record operation, 46

XmSTRING_COMPONENT_-
 FONTLIST_ELEMENT_-
 TAG
 common_support.c, 70
XmStringToString
 mod_common, 34
XPM_PIXMAP
 play/creation-c.c, 72
 play/main-c.c, 77
 record/creation-c.c, 75
 record/main-c.c, 80

Appendix A

Common Source Code

10/16/02
15:30:22

common_callbacks.c

1

```
1  /**
2   * File common_callbacks.c
3   * \ingroup mod_common
4   *
5   * In this source file are the definitions for the callback routines
6   * that will be called by Motif and X11 when the user clicks on
7   * push buttons, toggle buttons and pull down menu options for
8   * the 'record and playback sockets' application.
9   *
10  * These routines are the same for both applications and are thus
11  * defined in this 'common' source file.
12  */
13
14  Copyright (C) 2002
15  All content of this file is copyrighted to the
16  National Research Council of Canada.
17  All rights reserved.
18  Vendorcode
19  Author Daniel F. Johnston
20  Vince August 2002
21
22  File: common_callbacks.c,v 1.5 2002/10/16 10:05:35 Johnston Exp $
23  */
24
25  #include <xm/xm.h>
26  #include <xm/TextP.h>
27  #include <signal.h>
28  #include "mySocket.h"
29  #include "myInterface.h"
30
31  /**
32   * the shared, global variable structure
33  extern GlobalMemory shared;
34  /**
35  extern Widget inputFileSelection;
36  /**
37  extern Widget aboutHelpBulletinBoard;
38
39  /**
40  /**
41  /**
42  /**
43  /**
44  /**
45  /**
46  /**
47  /**
48  /**
49  /**
50  /**
51  /**
52  /**
53  /**
54  /**
55  /**
56  /**
57  /**
58  /**
59  */
60
61  /**
62  /**
63  /**
64  /**
65  /**
66
```

10/16/02
15:30:22

common_callbacks.c

2

```
67  \param w the widget that caused the event and this callback.
68  \param client_data is a pointer to any (optional) client data
69  defined for the callback (none in this case)
70  \param call_data the standard event information provided by X11-Motif
71  */
72  void
73  aboutHelpCallback( Widget w, XtPointer client_data, XtPointer call_data )
74  {
75      /* show the small message dialog */
76      XtManageChild(aboutHelpBulletinBoard);
77  }
78
79  /**
80  \ingroup mod_common
81  This callback is the function that will be called when the Motif user
82  interface selects the 'dimiss', button on the 'about' help message. It
83  will simply remove this pop up help message from the display.
84  \param w the widget that caused the event and this callback.
85  \param client_data is a pointer to any (optional) client data
86  defined for the callback (none in this case)
87  \param call_data the standard event information provided by X11-Motif
88  */
89  void
90  aboutDimissCallback( Widget w, XtPointer client_data, XtPointer call_data )
91  {
92      /* remove the file selection dialog */
93      XtUnmanageChild(aboutHelpBulletinBoard);
94
95  /**
96  \ingroup mod_common
97  This callback is the function that will be called when the Motif user
98  interface selects the 'usage' button on the 'Help' pull down menu. It
99  will cause the software to start Netscape with a pointer to a
100 descriptive help file, complete with images and links. The
101 help file selected is defined by the current application.
102 (todo) We need to add code so that the two different
103 socket program help files can be stored in some directory
104 other than the same directory as the program executables.
105 (todo) We should provide a mechanism to change the name
106 of the html document used for help.
107 \param w the widget that caused the event and this callback.
108 \param client_data is a pointer to client data
109 defined for the callback, in this case the name of the file to open
110 \param call_data the standard event information provided by X11-Motif
111 */
112 void
113 usageHelpCallback( Widget w, XtPointer client_data, XtPointer call_data )
114 {
115     char commandline[MAX_NAME_TEXT_SIZE];
116     strcpy(commandline,"netscape ");
117     strcat(commandline,(char *)client_data);
118     strcat(commandline," & ");
119     strcat(commandline,(char *)call_data);
120     system(commandline);
121
122 /**
123 \ingroup mod_common
124 This callback is the function that will be called when the Motif user
125 interface selects the 'file', pull down menu. It will cause a file selection
126 option in the 'file' menu to be visible.
127 \param w the widget that caused the event and this callback.
128 \param client_data is a pointer to any (optional) client data
129 defined for the callback (none in this case)
130 \param call_data the standard event information provided by X11-Motif
131
132 /**
133 \ingroup mod_common
134 This callback is the function that will be called when the Motif user
135 interface selects the 'about' button on the 'Help' pull down menu. It
136 will cause a small dialog to show with the name of the program, a very
137 brief description, and the name of the author.
138
139
```

10/16/02
15:30:22

common_callbacks.c

3

```
133 */
134 void browseButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
135 {
136     /* show the file selection dialog */
137     XmManageChild(inputFileSelectionBox);
138 }
139 */
140 */
141 /**
142 \ingroup mod_common
143 This callback is the function that will be called when the Motif user
144 interface puts up a file selection dialog and the user decides to
145 cancel it. It is a 'null' event except we need to remove the
146 file selection dialog from the display
147 (param w the widget that caused the event and this callback. The push
148 button (cancel) on the file selection dialog in this case).
149 (param client_data is a pointer to any (optional) client data
150 defined for the callback (none in this case))
151 (param call_data the standard event information provided by X11-Motif
152 */
153 void cancelFileDialogCallback( Widget w, XtPointer client_data, XtPointer call_data)
154 {
155     /*
156     /* remove the file selection dialog */
157     XmManageChild(inputFileSelectionBox);
158 }
159 */
160 /**
161 \ingroup mod_record
162 This callback is the function that will be called when the Motif user
163 interface selects the 'quit' button on the user interface or in the
164 'file' pull down menu. It will close the sub-process (if any) and
165 then shut down the main program.
166 (param w the widget that caused the event and this callback.
167 (param client_data is a pointer to any (optional) client data
168 defined for the callback (none in this case))
169 (param call_data the standard event information provided by X11-Motif
170 */
171 void quitButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
172 {
173     if( shared.childProcess != 0 )
174         /* kill the current child process if any */
175         kill( shared.childProcess, SIGTERM );
176
177     /* we close the input file, close the socket, and exit the program */
178     if( shared.fileName[0] != '\0' )
179         fclose( shared.fileName );
180
181     /* close down the socket */
182     close( shared.socketHandle );
183
184     exit(1);
185 }
186
187
```

10/16/02
15:30:32

common_support.c

1 10/16/02
2 15:30:32

```
1  /* file common_support.c
2   \ingroup mod_common
3
4 This file contains the support function needed for both
5 applications. We have defined motif string manipulation
6 and functions to read the number of standard socket messages
7 in a file.
8 */
9
10 \code
11 Copyright (c) 2002
12 All content of this file is copyrighted to the
13 National Research Council of Canada.
14 All rights reserved.
15 \endcode
16 \author Daniel P. Johnston
17 \since May 2002
18 $Id: common_support.c,v 1.5 2002/10/16 18:05:35 johnston Exp $
19
20 */
21 #include <xm/xm.h>
22 #include "mySocket.h"
23 #include "myInterface.h"
24
25 /**
26 \ingroup mod_common
27 This function is declared 'static', so that it's scope is local to
28 this source file (i.e. only functions in
29 this source file can call it). The
30 function will append a character array to another, allocating
31 more memory for the string as required.
32 Param return_text is the pointer to the character string array
33 which is either NULL (no existing string) or a pointer to an
34 existing string. This pointer will point to the new or amended
35 string on return from this function.
36 Param length is the length of the existing character array, and on
37 return it will be equal to the total new length of the string.
38 Param text is the array of characters to be appended to the existing (*)
39 string.
40
41 static void
42 AppendingString(char **return_text, int *length, char *text)
43 {
44     *length += strlen(text);
45     if (*return_text == (char *)NULL)
46     {
47         *return_text = (char *)xtMalloc(*length + 1, sizeof(char));
48     }
49     else
50     {
51         *return_text = (char *)xtRealloc(*return_text, *length + 1);
52         *return_text += strlen(*return_text);
53     }
54     strcat(*return_text, text);
55 }
```

This function will try to convert a Motif 'string' value to the

more general character array string used in 'C' programs. The success of this conversion.

is set as the return from the function. The character array

common_support.c

1 2 10/16/02
2 2 15:30:32

```
67 is returned as one of the function arguments.
68 param smatr is the Motif String variable to be converted.
69 param return_text is the character array that will hold the converted
70 text.
71 \return true or false, i.e. whether a conversion to character array
72 was possible or not.
73 */
74 Boolean
75 XmStringToString(XmString smatr, char **return_text)
76 {
77     Boolean
78     XmStringContext
79     char
80     XmStringCharSet
81     XmStringDirection
82     int
83     XmStringComponentType
84     XmStringComponentType
85     unsigned short
86     unsigned char
87     length = 0;
88     *return_text = NULL;
89     if (!xmStringInitContext(&context, smatr)) return (False);
90     while (1done)
91     {
92         type = xmStringGetStartComponent(&context, &start, &tag, &direction,
93                                         &unknownTag, &unknownDir, &unknownVal);
94         switch( type )
95         {
96             case XmSTRING_COMPONENT_TEXT:
97                 case XmSTRING_COMPONENT_LOCAL_TEXT:
98                     case XmSTRING_COMPONENT_PORTLIST_ELEMENT_TEXT:
99                         AppendString(return_text, start, length, text);
100                        AppendString(return_text, start, length, "\0");
101                        XtFree(text);
102                    break;
103                case XmSTRING_COMPONENT_SEPARATOR:
104                    AppendString(return_text, start, length, "\n");
105                    break;
106                case XmSTRING_COMPONENT_USER_DEFINED:
107                    /* IGNORED */
108                    break;
109                case XmSTRING_COMPONENT_PORTLIST_ELEMENT_TAG:
110                    XtFree((char *)tag);
111                    break;
112                case XmSTRING_COMPONENT_DIRECTION:
113                    break;
114                case XmSTRING_COMPONENT_UNDEFINED:
115                    XtFree((char *)unknownVal);
116                    break;
117                case XmSTRING_COMPONENT_END:
118                    default:
119                    done = True;
120                }
121            }
122        XmStringFreeContext(&context);
123        return (true);
124    }
```

10/16/02
15:30:32

common_support.c

3
15:30:32

```
133 }
134 /* read the number of messages in the file */
135 /*\ingroup mod_common
136 */
137 /**
138 * This function will be called to quickly read the entire file and
139 * count the number of messages in it. This number can be used
140 * to set values on the user interface. The file is defined by the
141 * (already open) file descriptor. The file is defined by the
142 * which contains the socket messages and delays.
143 * \param fileID is the descriptor for the (already opened) file
144 * \return the number of messages in the file, or 0 if there was an
145 * error reading the file or an invalid file.
146 */
147 int
148 getNbMessages( FILE *fileID )
149 {
150     long dummy;
151     int length;
152     char buffer[MAX_MESSAGE_SIZE];
153     int itemRead;
154     int count;
155
156     count = 0;
157
158     /* read the first part of the delay from the file */
159     itemRead = fread( &dummy, sizeof(long), 1, fileID );
160
161     if( itemRead != 1 )
162         return count;
163
164     /* read the second part of the delay from the file */
165     itemRead = fread( &dummy, sizeof(long), 1, fileID );
166
167     if( itemRead != 1 )
168         return count;
169     if( itemRead != 1 )
170         return count;
171
172     /* read the size of the message from the file */
173     itemRead = fread( &length, sizeof(int), 1, fileID );
174
175     if( itemRead != 1 )
176         return count;
177
178     /* read the number of messages in a user supplied file name */
179 /*\ingroup mod_common
180 */
181 /**
182 * This function will be called to quickly read the entire file and
183 * count the number of messages in it. This number can be used
184 * to set values on the user interface. The file is defined by a character
185 * file name. After the number of messages is determined, the file
186 * is closed and re-open in order to place the file pointer back to
187 * the beginning.
188 * \param filename is the name of a file which (we hope) contains socket
189 * messages and delay values.
190 * \return the number of messages in the file, or 0 if there was an
191 * error reading the file or an invalid file.
192 */
193 int
194 readNbMessages( char *filename )
195 {
196     extern GlobalMemory shared;
197     int count;
198 }
```

4
15:30:32

common_support.c

4

```
199 /* close the current file (if one open), open and size the new */
200 if( strlen( shared.inpFileName ) != 0 ,
201     fclose( shared.fileDescriptor );
202
203 if( ( shared.fileDescriptor = fopen( filename, "r" ) ) )
204 {
205     printf( "\n%s(support) Cannot open file '%s'\n", filename );
206     count = 0;
207
208     /* if there is a valid input file, get the number of messages in it */
209     count = getNbMessages( shared.fileDescriptor );
210
211     /* reset the file read pointer to the start of the file */
212     fseek( shared.fileDescriptor,
213           shared.fileDescriptor = fopen( filename, "r" );
214
215     if( shared.globalMode == 1 )
216         if( shared.globalMode == 1 )
217             printf( "File message count = %d\n", count );
218
219     return count;
220 }
```

10/16/02
15:30:39

myInterface.h

10/16/02
15:30:39

1

myInterface.h

2

```
1  /*
2   * File myInterface.h
3   * \ingroup mod_common
4   *
5   * This file will contain a number of things important to the
6   * Motif GUI interfaces created for the Minic Sockets - or the
7   * record and playback sockets, applications. It contains some defines
8   * that set maximum sizes for the arrays, and it describes the
9   * structure that contains all the global memory of the global structure
10  * application. Some of the parameters of the global structure
11  * are only required for one application or the other, but are
12  * all defined here for simplicity. (We accept the small cost
13  * of the extra memory that must be allocated.)
14  *
15  * Note: The structure is defined here in an attempt to simplify
16  * the source code. You could create all the elements as individual
17  * global variables, and include references to them as needed. But, I
18  * find that the source code becomes complicated to read. It, i.e.,
19  * the common definition for the globals, also makes the source code
20  * more consistent with the set of globals created by the user interface
21  * builder tool for some elements of the user interface widgets.
22  */
23
24  Copyright (c) 2002
25  All content of this file is copyrighted to the
26  National Research Council of Canada.
27
28  All rights reserved.
29
30  \endcode
31  \author Daniel F. Johnston
32  \since April 2002
33
34  $Id: myInterface.h,v 1.6 2002/10/16 18:21:43 johnston Exp $
35
36  #include <stdio.h>
37
38  /*
39   * Motif GUI interface defined constants
40   * This constant will specify the maximum size of general
41   * text and filename input as read from command line
42   * arguments and large text fields.
43  #define MAX_NAME_TEXT_SIZE 144
44
45  /*
46   * Motif GUI interface defined constants
47   * This constant will specify the biggest number of text
48   * digits we will deal with from small text fields.
49  */
50  #define MAX_NUMBER_TEXT_SIZE 20
51
52  /*
53  This is the structure to declare all the global variables required for the
54  main program of the Motif GUI interface, and for the child
55  process we spawn to do the actual recording/reading of messages.
56  We use the same structure for both record and playback for simplicity,
57  and with a small cost of extra memory requirement. The
58  members of this structure are related mostly by their
59  common requirement for global scope.
60  \remark This requirement for global scope would disappear if
61  the code was re-written into the C++ computer language.
62
63  {
64  \typedef struct _globalMemory
65
66  // the character array used to hold the name of the input file
```


10/16/02
15:30:48

mySocket.h

1

```
1  /*
2   * \file mySocket.h
3   * \ingroup mod_common
4   */
5   This file contains some standard values (defines)
6   that we use for creating TCP/IP network communication
7   applications.
8
9  \code
10 Copyright (c) 2002
11 All content of this file is copyrighted to the
12 National Research Council of Canada.
13 All rights reserved.
14 \endcode
15 \author Daniel F. Johnston
16 \since April 2002
17 $Id: mySocket.h,v 1.3 2002/08/19 18:46:19 johnston Exp $
18 */
19 // the default value for the address of the local computer
20 #define LOCAL_ADDRESS "localhost"
21
22 // the default value we will use for a network port number
23 #define DEFAULT_PORT_NUMBER 10025
24
25 // the biggest socket message we will deal with
26 #define MAX_MESSAGE_SIZE 1024
27
28
```

Appendix B

Record Program Source Code

10/16/02
15:35:25

callbacks-c.c

1 10/16/02

15:35:25

```
1 /* NAME: This file is appended to at file generation time.
2  * Edits can be made throughout the file
3  */
4 /*
5  * Generated by the ICS Builder Accessory (X1).
6  *
7  * Builder Xcessory Version 5.0
8  * Code Generator Xcessory 5.0 (05/22/98)
9  */
10 /*
11 #include <Xm/Xm.h>
12 #include <Xm/TextF.h>
13 #include <Xm/FormF.h>
14 #include <Xm/Label.h>
15 #include <Xm/Create.h>
16 #include ".../common/myinterface.h"
17 */
18 /* Standard includes for builtins.
19 */
20 #include <stdio.h>
21 #include <string.h>
22 #include <ctype.h>
23 /*
24 * Macros to make code look nicer between ANSI and K&R.
25 */
26 #ifndef ARGDEF
27 /*
28 #endif ARGDEF
29 #ifne DEFUNIONPrototypes == 0
30 #define PROTOTYPE(p) \
31 #define ARGDEF(s) \
32 #define ARG(a, b) \
33 #define GRA(a, b) \
34 #define UARG(a, b) \
35 #define GRAB(a, b) \
36 #else \
37 #define PROTOTYPE(p) \
38 #define ARGDEF(s) \
39 #define ARG(a, b) \
40 #define GRA(a, b) \
41 #ifndef _CPLURPLUS \
42 #define UARG(a, b) \
43 #define GRAB(a, b) \
44 #else \
45 #define UARG(a, b) \
46 #define GRAB(a, b) \
47 #endif \
48 #endif \
49 #endif \
50 */
51 Widget XmFindOrShell PROTOTYPE((Widget));
52 WidgetList Xmidgetsfromnames PROTOTYPE((Widget, char*, char*));
53 */
54 /**
55 * File record/callbacks-c.c
56 */
57 /*
58 In this source file are the definitions for the callback routines
59 that will be called by Motif and X11 when the user clicks on
60 push buttons, toggle buttons and pull down menu options for
61 the 'record sockets' application.
62 Some callback functions are simple enough to fully define here.
63 But, many require a call to some of the support and sending
64 functions defined in other source files. A few will actually
65 start other processes or send signals, and the support functions
66
```

1 callbacks-c.c

2 10/16/02

15:35:25

```
67 will define what happens when these signals are processed.
68 */
69 \endcode
70 Copyright (c) 2002
71 All content of this file is copyrighted to the
72 National Research Council of Canada.
73 All rights reserved.
74 \endcode
75 \author Daniel F. Johnston
76 \since April 2002
77 $Id: callbacks-c.c,v 1.7 2002/08/20 15:36:40 Johnston Exp $
78 */
79 /**
80 /**
81 // the shared, global variable structure
82 extern GlobalMemory shared;
83 */
84 /**
85 \ingroup mod_record
86 This callback is the function that will be called when the Motif user
87 interface decides to respond to the user activating the 'Record'
88 push button. This is the most complex of the callbacks, because we
89 read the current values for most of the user-modifiable fields,
90 create the socket, re-open the output file, and fork a subprocess to
91 read the socket messages. If all this works, then we change the
92 state of the user interface buttons and we are done.
93 \param w the widget that caused the event and this callback.
94 \param client_data is a pointer to any (optional) client data
95 defined for the callback (none in this case)
96 \param call_data the standard event information provided by X11-Motif
97 */
98 void
99 recordButtonCallback(Widget w, XtPointer client_data, XtPointer call_data)
100 {
101     char tapBuf[MAX_STRING_TEXT_SIZE];
102     /* read all the fields on the user interface to make sure we have
103     * the most up-to-date values for each option.
104     */
105     /* get the output file name */
106     strcpy(shared.outputFileName, XmTextFieldGetString( outputTitleTextField ));
107     /* the socket port number we are listening to */
108     strcpy(tcpPort, XmTextFieldGetString( portTextField ));
109     /* the socket port number */
110     if( socketNumber < 0 )
111         socketNumber = 1;
112     XmTextDisplay(w, 100 );
113     return;
114 }
115 */
116 /**
117 * create socket */
118 shared.socketNum = createReadSocket( shared.portNum );
119 if( shared.socketNum < 0 )
120     XmTextDisplay(w, 100 );
121 }
122 */
123 */
124 */
125 /**
126 * open file at beginning and reset message counter */
127 shared.filereDescriptor = fopen( shared.outputFileName, "w" );
128 /* (read) the frame counter */
129 shared.messageNumber = 0;
130 /* write the current value of the counter into the text field display */
131 XmTextFieldSetString( numRecTextField, "0" );
132 */


```

10/16/02
15:35:25

callbacks-C.C

3 15:35:25

```
133 /* if there is a suspended child process already, terminate it first */
134 if( shared.childProcess != 0 )
135     kill( shared.childProcess, SIGKILL );
136
137 /* kill the current (suspended) child process */
138 /* we need to know when the (new) recording process is finished */
139 signal( SIGCHLD, readProcessHandle );
140
141 /* fork a subprocess to run the long recording procedure */
142 switch( fork( shared.childProcess = fork() ) )
143 {
144     case 0: /* child process */
145         /* we want the child process to respond to a pause/resume control */
146         /* signal( SIGSTOP, readProcessPause ); */
147         /* we want the child process to respond to a stop control */
148         signal( SIGKILL, readProcessStop );
149         readMessages( shared.fileDescriptor );
150         /* when the record process is terminated, exit with signal set */
151         exit(255);
152     case -1:
153         XBell( XtDisplay(w), 100 );
154         print("record messages failed\n");
155         return;
156
157     /* the main (main) program continues... */
158     if( shared.globalDebugMode == 1 )
159         print("Read process started: pid %d \n", shared.childProcess);
160
161     /* set prohibiton state - disable the start button, enable the stop/pause */
162     XSetSensitive( recordPushButton, False );
163     XSetSensitive( stopPushButton, True );
164     XSetSensitive( pausePushButton, True );
165     XSetSensitive( resumePushButton, False );
166 }
167
168 /* */
169 /*ingroup mod_record
170 this callback is the function that will be called when the Motif user
171 interface decides to respond to the user activating one of the 'Stop',
172 'Pause' buttons. Since this button can only be active when the program
173 is actually reading, we know that there is a sub-process in existence
174 and that it is running. We want to terminate the sub-process, and
175 change the state of active buttons in the interface to reflect this.
176 We also set the current value of the message counter into the text
177 field on the display.
178 Xparam w the widget that caused the event and this callback. It
179 could be one of several widgets in this case, and we don't care
180 which one.
181 Xparam client_data is a pointer to any (optional) client data
182 defined for the callback (none in this case)
183 Xparam call_data the standard event information provided by X11-Motif
184 */
185 void
186 stopButtonCallback( Widget w, XtPointer client_data, XtPointer call_data )
187 {
188     union signal_sval
189     {
190         /* abort the current running child process */
191         if( sigqueue( shared.childProcess, SIGABRT, sval ) < 0 )
192     }
193     print("Stop cannot queue signal to child\n");
194 }
195
196 /* */
197 /*ingroup mod_record
198 this callback is the function that will be called when the Motif user
199 */

200 /* if( shared.globalDebugMode == 1 )
201     if( sigqueue( shared.childProcess, SIGSTOP, sval ) < 0 )
202 */
203
204 /*ingroup mod_record
205 this callback is the function that will be called when the Motif user
206 interface puts up a file selection dialog and the user selects a file
207 and presses the 'OK' button. We try to check the validity of the
208 new file name by trying to read how many socket messages it
```

4 10/16/02

callbacks-C.C

4

```
199 interface decides to respond to the user activating one of the 'Pause',
200 'stop' buttons. Since this button can only be active when the program
201 is actually reading, we know that there is a sub-process in existence
202 and that it is running. We want to stop the sub-process, and change
203 the state of active buttons in the interface to allow the user to
204 start it up again. We also set the current value of the
205 counter into the text field on the display.
206 Xparam w the widget that caused the event and this callback. It
207 could be one of several widgets in this case, and we don't care
208 which one.
209 Xparam client_data is a pointer to any (optional) client data
210 defined for the callback (none in this case)
211 Xparam call_data the standard event information provided by X11-Motif
212 */
213 void
214 pauseRecordCallback( Widget w, XtPointer client_data, XtPointer call_data )
215 {
216     union signal_sval
217     {
218         /* disable the pause button, enable the record, stop and resume */
219         XSetSensitive( recordPushButton, True );
220         XSetSensitive( stopPushButton, True );
221         XSetSensitive( pausePushButton, False );
222         XSetSensitive( resumePushButton, True );
223         sval.level.intc = 1;
224         if( sigqueue( shared.childProcess, SIGSTOP, sval ) < 0 )
225             print("pause: cannot queue signal to child\n");
226     }
227     /* */
228     /*ingroup mod_record
229     this callback is the function that will be called when the Motif user
230     interface decides to respond to the user activating the 'Resume'
231     'push' buttons. Since this button can only be active when the program
232     is reading but in a paused state, we know that there is a sub-
233     process in existence and that it is not running. We want to tell the
234     sub-process to continue, and change the state of active buttons in
235     the interface to show this change in the sub-process.
236     Xparam w the widget that caused the event and this callback.
237     Xparam client_data is a pointer to any (optional) client data
238     defined for the callback (none in this case)
239     Xparam call_data the standard event information provided by X11-Motif
240     */
241     void
242     resumeRecordCallback( Widget w, XtPointer client_data, XtPointer call_data )
243     {
244         union signal_sval
245         {
246             /* disable the resume button, reenable the pause */
247             XSetSensitive( recordPushButton, False );
248             XSetSensitive( stopPushButton, True );
249             XSetSensitive( pausePushButton, True );
250             XSetSensitive( resumePushButton, False );
251             /* start recording messages into current position in file */
252             sval.level.intc = 1;
253             if( sigqueue( shared.childProcess, SIGCONT, sval ) < 0 )
254                 print("resume: cannot queue signal to child\n");
255     }
256     /* */
257     /*ingroup mod_record
258     this callback is the function that will be called when the Motif user
259     interface puts up a file selection dialog and the user selects a file
260     and presses the 'OK' button. We try to check the validity of the
261     new file name by trying to read how many socket messages it
```

10/16/02
15:35:25

Callbacks-C.C

5

```
contains. If the file is invalid, the user interface will just
265 'beep' as the user tries to ok, the selection of an incorrect file.
266 \param w the widget that caused the event and this callback. The push
267 button (ok) on the file selection dialog in this case.
268 \param client_data is a pointer to any (optional) client data
269 defined for the callback (none in this case)
270 \param call_data the standard event information provided by X11-Motif
271 */
272 void
273 orfileDialogCallback( Widget w, XtPointer client_data, XtPointer call_data)
274 {
275     char    *file;
276     currentdirectory[MAX_NAME_TEXT_SIZE];
277     char    result;
278     int;
279     XmFileSelectionCallbackStruct *fab = 
280     (XmFileSelectionCallbackStruct *)call_data;
281     if( XmStringGetString(fab->value, &file) )
282     {
283         /* copy the filename selected, but don't copy the directory part */
284         getwd( currentDirectory );
285         /* if the first part of the selected file is the same, cut it off */
286         /* if the first part of the selected file is the same, cut it off */
287         result = strcmp( file, currentDirectory, strlen(currentDirectory));
288         if( result == 0 )
289         {
290             sharedOutputFilename[0] = '.'; /* substitute for current dir */
291             strcpy( sharedOutputFilename+1, file+strlen(currentDirectory) );
292         }
293         else
294             /* better copy the whole directory */
295             strcpy( sharedOutputFilename, file );
296             XmTextFieldsGetString( outputFileChooserField, sharedOutputFilename );
297             XtFree((char *)file);
298         }
299     else
300         printf("Invalid Selection\n");
301     /* remove the file selection dialog */
302     XtUnmanageChild(inputFileSelectionBox);
303 }
304 }
```


10/16/02
15:35:35

creation-c.c

10/16/02
15:35:35

creation-c.c

2

```
1 /*  
2  * README: Portions of this file are merged at file generation.  
3  * time. Edits can be made "only" in between specified code blocks, look  
4  * for keywords <begin user code> and <end user code>.  
5  */  
6 /*  
7  * Generated by the ICS Builder Xcessory (X).  
8  *  
9  * Builder Xcessory Version 5.0 (05/22/98)  
10 * Code Generator Xcessory 5.0 (05/22/98)  
11 *  
12 */  
13 /*  
14 /* Begin user code block <file_comments> */  
15 /*  
16 /* End user code block <file_comments> */  
17 /*  
18 \ingroup mod_record  
19 In this source file is the source code generated automatically  
20 by the user interface builder software. It will use this source  
21 code.  
22 Copyright (c) 2002  
23 All content of this file is copyrighted to the  
24 National Research Council of Canada.  
25 All rights reserved.  
26  
27 \endcode  
28 \author Daniel F. Johnston  
29 \since July 2002  
30  
31 #id: creation-c.c,v 1.6 2002/08/19 19:55:02 johnston Exp $  
32 /*  
33 /* End user code block <file_comments> */  
34 #include <Xm/Xm.h>  
35 #include <Xm/Rainbow.h>  
36 #include <Xm/Dialogs.h>  
37 #include <Xm/ScrolledW.h>  
38 #include <Xm/SheetW.h>  
39 #include <Xm/Bulletin.h>  
40 #include <Xm/ScrolledW.h>  
41 #include <Xm/Label.h>  
42 #include <Xm/PushB.h>  
43 #include <Xm/FileS.h>  
44 #include <Xm/ErrorComm.h>  
45 #include <Xm/Cascade.h>  
46 #include <Xm/Separator.h>  
47 #include <Xm/Form.h>  
48 #include <Xm/PopUp.h>  
49 #include <Xm/TestF.h>  
50 /*  
51 /* Global declarations are now stored in the header file.  
52 */  
53 /*  
54 * If DECLARE_XC_GLOBALS is defined then this header file  
55 * declares the globals, otherwise it just externs them.  
56 */  
57 #define DECLARE_XC_GLOBALS  
58 /*  
59 /* Globally included information.  
60 */  
61 /*  
62 */  
63 /*  
64 /* Common constant and pixmap declarations.  
65 */  
66 */
```

```
67 #include "creation-c.h"  
68 /*  
69 /* Convenience functions from utilities file.  
70 */  
71 extern void RegisterXConverters(XAppContext);  
72 extern XtPointer XK_CONVERT(Widget, char *, int, Boolean);  
73 extern XtPointer XK_DOUBLE(double);  
74 extern XtPointer XK_FLOAT(float);  
75 extern XtPointer XK_INTEGER(int);  
76 extern XtPointer XK_PIXEL(Widget, char **);  
77 extern Pixmap Xpm_PIXMAP(Widget, Boolean *, Boolean, Arglist, Cardinal, Pixel);  
78 extern void XK_SET_BACKGROUND_COLOR(Widget, Arglist, Cardinal, Pixel);  
79  
80 /*  
81 /* Declarations for callbacks and handlers.  
82 */  
83 #ifndef DYNAMIC_SHOULD_GENERATE_MHS  
84 extern void aboutHelpCallback(Widget, XcPointer, XcPointer);  
85 extern void okHelpCallback(Widget, XcPointer, XcPointer);  
86 extern void cancelFileHelpCallback(Widget, XcPointer, XcPointer);  
87 extern void broadcastButtonCallback(Widget, XcPointer, XcPointer);  
88 extern void recordButtonCallback(Widget, XcPointer, XcPointer);  
89 extern void acceptButtonCallback(Widget, XcPointer, XcPointer);  
90 extern void passAcceptButtonCallback(Widget, XcPointer, XcPointer);  
91 extern void zoomAcceptButtonCallback(Widget, XcPointer, XcPointer);  
92 extern void quitButtonCallback(Widget, XcPointer, XcPointer);  
93 extern void debugPopUpListCallback(Widget, XcPointer, XcPointer);  
94 extern void usageHelpCallback(Widget, XcPointer, XcPointer);  
95 extern void aboutHelpCallback(Widget, XcPointer, XcPointer);  
96 /*endif  
97 */  
98 /* Create the mainWindow hierarchy of widgets.  
99 */  
100 Widget CreateMainWindow(Widget parent)  
101 {  
102     Cardinal ac = 0;  
103     Arg args[256];  
104     Cardinal cdc = 0;  
105     Boolean argok = False;  
106     mainWindow;  
107     Widget aboutHelpDialogShell;  
108     Widget aboutHelpLabel;  
109     Widget aboutUsLabel;  
110     Widget selectFileDialog;  
111     Widget menuBar;  
112     Widget fileCaseAndButton;  
113     Widget pullDownMenu;  
114     Widget openFilePushButton;  
115     Widget recordFilePushButton;  
116     Widget separator1;  
117     Widget separator2;  
118     Widget pasteFilePushButton;  
119     Widget resumeFilePushButton;  
120     Widget separator3;  
121     Widget quitFilePushButton;  
122     Widget optionsCascadeButton;  
123     Widget pullDownMenu1;  
124     Widget helpCascadeButton;  
125     Widget pullDownMenu2;  
126     Widget usageHelpPushbutton;  
127     Widget aboutHelpPushbutton;  
128     Widget form;  
129     Widget numberRecordsLabel;  
130     Label label;  
131     Widget quitPushButton;  
132 }
```

10/16/02
15:35:35

creation-c.C

10/16/02
15:35:35

creation-c.C

4

```
Widget portLabel;
Widget browseButton;
Widget outputPushButton;
Widget filePushButton;
/*
 * Register the converters for the widgets.
 */
133 XInitializeConverters(XWidgetContext (parent));
134 XInitializeWidgetClass (WidgetClass) xmMainWindowWidgetClass;
135 XInitializeWidgetClass (WidgetClass) xmMainShellWidgetClass;
136 XInitializeWidgetClass (WidgetClass) xmMainTitleBarWidgetClass;
137 XInitializeWidgetClass (WidgetClass) xmScrollbarWidgetClass;
138 /* Register the converters for the widgets.
139 XInitializeConverters (XWidgetContext (parent));
140 XInitializeWidgetClass (WidgetClass) xmMainMenuBarWidgetClass;
141 XInitializeWidgetClass (WidgetClass) xmMainMenuItemWidgetClass;
142 XInitializeWidgetClass (WidgetClass) xmMainSeparatorWidgetClass;
143 XInitializeWidgetClass (WidgetClass) xmMainTextWidgetClass;
144 XInitializeWidgetClass (WidgetClass) xmMainTextEntryWidgetClass;
145 XInitializeWidgetClass (WidgetClass) xmMainTextTextWidgetClass;
146 XInitializeWidgetClass (WidgetClass) xmMainTextTextEntryWidgetClass;
147 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextWidgetClass;
148 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextEntryWidgetClass;
149 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextWidgetClass;
150 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextEntryWidgetClass;
151 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextTextWidgetClass;
152 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextTextEntryWidgetClass;
153 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextTextTextWidgetClass;
154 XInitializeWidgetClass (WidgetClass) xmMainTextTextTextTextTextTextEntryWidgetClass;
155 ac = 0;
156 XSetArg(args[ac], XmNlabelString, "main");
157 XSetArg(args[ac], XmNx, 52);
158 XSetArg(args[ac], XmNy, 156);
159 XSetArg(args[ac], XmWidth, 619);
160 XSetArg(args[ac], XmHeight, 290);
161 mainWindow = XmCreateMainWindow (parent,
162 "mainWindow",
163 "mainWindow",
164 ac);
165 ac = 0;
166 XSetArg(args[ac], XmWidth, 619);
167 XSetArg(args[ac], XmHeight, 28);
168 XSetArg(args[ac], XmCreateMenuBar, mainWindow,
169 "menuBar",
170 args,
171 ac);
172 XManageChild (menuBar);
173 ac = 0;
174 XSetArg(args[ac], XmNlabelString, "File");
175 XSetArg(args[ac], XmNlabelString, "Edit");
176 XSetArg(args[ac], XmNlabelString, "View");
177 XSetArg(args[ac], XmNlabelString, "Help");
178 XCreateCascadeButton (menuBar,
179 XmCreateCascadeButton, 0, args);
180 XSetArg(args[ac], XmWidth, 20);
181 XSetArg(args[ac], XmHeight, 20);
182 XSetArg(args[ac], XmWidth, 43);
183 XSetArg(args[ac], XmHeight, 24);
184 fillCascadeButton = XmCreatePushbutton (menuBar,
185 "fillCascadeButton",
186 args,
187 ac);
188 XManageChild (fileCascadeButton);
189 ac = 0;
190 XSetArg(args[ac], XmNx, 0);
191 XSetArg(args[ac], XmNy, 0);
192 XSetArg(args[ac], XmWidth, 0);
193 XSetArg(args[ac], XmHeight, 76);
194 XSetArg(args[ac], XmWidth, 156);
195 pullDownMenu = XmCreatePullDownMenu (fileParent (fileCascadeButton),
196 "pullDownMenu",
197 args,
198 ac);
```

```
199 ac = 0;
200 XSetArg(args[ac], XmN mnemonic, 'O');
201 XSetArg(args[ac], XmLabelString,
202 XmCreatePushbutton, "Open...", "Open");
203 XmCreatePushbutton, 0, args);
204 if (argsok) ac++;
205 openFilePushButton = XmCreatePushButton (pullDownMenu,
206 "openFilePushButton",
207 args,
208 ac);
209 XManageChild (openFilePushButton), XmActivateCallback, browseButtonCallback, (XtPointer
210 er);
211 ac = 0;
212 separator0 = XmCreateSeparator (pullDownMenu,
213 separator0,
214 args,
215 ac);
216 XManageChild (separator0);
217 ac = 0;
218 XSetArg(args[ac], XmN mnemonic, 'B');
219 XSetArg(args[ac], XmLabelString, "Record");
220 XSetArg(args[ac], XmN record,
221 XmCreatePushbutton, "Record...", "Record");
222 XmCreatePushbutton, 0, args);
223 if (argsok) ac++;
224 recordFilePushButton = XmCreatePushButton (pullDownMenu,
225 "recordFilePushButton",
226 args,
227 ac);
228 XManageChild (recordFilePushButton),
229 XmAddCallback (recordFilePushButton, XmActivateCallback, recordButtonCallback,
230 inter);
231 ac = 0;
232 XSetArg(args[ac], XmN mnemonic, 'S');
233 XSetArg(args[ac], XmLabelString,
234 XmCreatePushbutton, "Stop...", "Stop");
235 XmCreatePushbutton, 0, args);
236 if (argsok) ac++;
237 stopFilePushButton = XmCreatePushButton (pullDownMenu,
238 args,
239 ac);
240 XManageChild (stopFilePushButton),
241 XmAddCallback (stopFilePushButton, XmActivateCallback, stopButtonCallback,
242 inter);
243 ac = 0;
244 XSetArg(args[ac], XmN mnemonic, 'P');
245 XSetArg(args[ac], XmLabelString,
246 XmCreatePushbutton, "Pause...", "Pause");
247 XmCreatePushbutton, 0, args);
248 pauseFilePushButton = XmCreatePushButton (pullDownMenu,
249 args,
250 ac);
251 XManageChild (pauseFilePushButton);
252 XmAddCallback (pauseFilePushButton, XmActivateCallback, pauseButtonCallback,
253 inter);
254 ac = 0;
255 XSetArg(args[ac], XmN mnemonic, 'R');
256 XSetArg(args[ac], XmLabelString,
257 XmCreatePushbutton, "Rename...", "Rename");
258 XmCreatePushbutton, 0, args);
259 if (argsok) ac++;
260 renameFilePushButton = XmCreatePushButton (pullDownMenu,
```

10/16/02
15:35:35

creation-c.c

5
10/16/02
15:35:35

```
"resumeFilePushButton",
    args,
    ac);
263    XtManageChild(resumeFilePushButton, XmActivateCallback, resumeButtonCallback, (XtPointer)
264    XtAddCallback(resumeFilePushButton, XmActivateCallback, resumeButtonCallback, resumeButtonCallback,
265    (XtPointer)0),
266    ac = 0;
267    separator2 = XmCreateSeparator(pullDownMenu,
268    "separator2",
269    args,
270    ac);
271    XtManageChild(separator2);
272    ac = 0;
273    args = XmStringCreate(args[ac], XmNemonic, 'Q');
274    XtSetArg(args[ac], XmNlabelString, "Quit",
275    XmCONVERT(menuBar, "Quit",
276    XmStringCreate(args[ac], XmNlabelString, "Quit",
277    XmCONVERT(pullDownMenu, "pullDownMenu", 0, XmPopUp, "Quit",
278    XmStringCreate(0, XmPopUp, "Quit",
279    quitFilePushButton = XmCreatePushButton(pullDownMenu,
280    "quitFilePushButton",
281    args,
282    ac));
283    XtManageChild(quitFilePushButton, XmActivateCallback, quitButtonCallback, (XtPointer)
284    XtAddCallback(quitFilePushButton, XmActivateCallback, XmActivateCallback, quitButtonCallback,
285    (XtPointer)0),
286    ac = 0;
287    XmSetArg(args[ac], XmSubMenuItem, pullDownMenu, args, ac);
288    XmSetValues(fileCascadeButton, args, ac);
289    ac = 0;
290    args = XmStringCreate("0", XmNemonic, 'O');
291    XtSetArg(args[ac], XmLabelString,
292    XmCONVERT(menuBar, "Options",
293    XmStringCreate(0, XmPopUp, "Options",
294    XmStringCreate(args[ac], XmNlabelString, "Options",
295    XmSetArg(args[ac], XmNlabelString, "Options",
296    XmStringCreate(args[ac], XmNlabelString, "Options",
297    XmStringCreate(args[ac], XmNlabelString, "Options",
298    XmStringCreate(args[ac], XmNlabelString, "Options",
299    optionsCascadeButton = XmCreateCascadeButton(optionsCascadeButton,
300    "optionCascadeButton",
301    args,
302    ac));
303    XtManageChild(optionsCascadeButton,
304    ac = 0;
305    args = XmStringCreate("0", XmNemonic, 'A');
306    XmSetArg(args[ac], XmLabelString,
307    XmCONVERT(pullDownMenu, "about",
308    XmStringCreate(args[ac], XmNlabelString, "about",
309    XmStringCreate(args[ac], XmNlabelString, "about",
310    pullDownMenu = XmCreatePullDownMenu(XtParent(optionsCascadeButton,
311    "pullDownMenu"),
312    args,
313    ac));
314    ac = 0;
315    args = XmStringCreate("debug mode",
316    XmLabelString,
317    XmCONVERT(pullDownMenu, "debug mode",
318    debugToggleButton = XmCreateToggleButton(debugModeButton,
319    "debugModeButton",
320    args,
321    ac);
322    XtManageChild(debugModeButton, XmValueChangedCallback, debugModeChangedCallback, (XtPointer)
323    XtAddCallback(debugModeButton, XmValueChangedCallback, debugModeChangedCallback, debugModeChangedCallback,
324    (XtPointer)
```

```
nter0);
325    ac = 0;
326    XtSetArg(args[ac], XmSubMenuItem, pullDownMenu1, args, ac);
327    XtSetValues(optionCascadeButton, args, ac);
328    ac = 0;
329    XtSetArg(args[ac], XmAlign, XmALIGNMENT_BEGINNING);
330    XtSetArg(args[ac], XmNlabelString, "Help");
331    XtSetArg(args[ac], XmNlabelString, "Help");
332    XtSetArg(args[ac], XmNlabelString, "Help");
333    XtSetArg(args[ac], XmNlabelString, "Help");
334    BX_CONVERT(menuBar, "Help");
335    XmStringCreate(0, XmPopUp, "Help");
336    XtSetArg(args[ac], XmNlabelString, "Help");
337    XtSetArg(args[ac], XmNlabelString, "Help");
338    XtSetArg(args[ac], XmNlabelString, "Help");
339    XtSetArg(args[ac], XmNlabelString, "Help");
340    XmCreateCascadeButton(helpCascadeButton, "helpCascadeButton", 50);
341    args,
342    ac);
343    XtManageChild(helpCascadeButton);
344    ac = 0;
345    args,
346    ac = 0;
347    XmStringCreate(args[ac], XmNlabelString, "About");
348    XtSetArg(args[ac], XmNlabelString, "About");
349    XmStringCreate(0, XmPopUp, "About");
350    XtSetArg(args[ac], XmNlabelString, "About");
351    XmCreatePullDownMenu(XtParent(helpCascadeButton),
352    "pullDownMenu2",
353    args,
354    ac);
355    ac = 0;
356    args,
357    XtSetArg(args[ac], XmNlabelString, "Usage");
358    XtSetArg(args[ac], XmNlabelString, "Usage");
359    XmStringCreate(0, XmPopUp, "Usage");
360    XtSetArg(args[ac], XmNlabelString, "Usage");
361    XmStringCreate(0, XmPopUp, "Usage");
362    usageHelpPushButton = XmCreatePushButton(pullDownMenu2,
363    "usageHelpPushButton",
364    args,
365    ac);
366    XtManageChild(usageHelpPushButton, usageHelpPushButton,
367    "usageHelpPushButton");
368    ac = 0;
369    args,
370    XtSetArg(args[ac], XmNlabelString, "About");
371    XmStringCreate(0, XmPopUp, "About");
372    aboutHelpPushButton = XmCreatePushButton(pullDownMenu2,
373    "aboutHelpPushButton",
374    args,
375    ac);
376    XtManageChild(aboutHelpPushButton, aboutHelpPushButton,
377    "aboutHelpPushButton");
378    XtAddCallback(aboutHelpPushButton, XmActivateCallback, aboutHelpCallback, (XtPointer)
379    0);
380    ac = 0;
381    XtSetArg(args[ac], XmSubMenuItem, pullDownMenu1);
382    XtSetValues(helpCascadeButton, args, ac);
383    ac = 0;
384    XtSetArg(args[ac], XmNlabelString, "About");
385    XmStringCreate(0, XmPopUp, "About");
386    XtSetArg(args[ac], XmNlabelString, "About");
387    XtSetArg(args[ac], XmNlabelString, "About");
```

creation-c.c

6

10/16/02
15:35:35

creation-c.c

7 15:35:35

```

388 form = XmCreateForm(mainWindow,
389     "Form",
390     args,
391     ac);
392 XtManageChild(form);
393 ac = 0;
394 XtSetArg(args[ac], XmFONTLIST, "Form", 200); ac++;
395 XmConvArg(args[ac], XmFONTLIST, "Form", 200); ac++;
396 XmConvArg(args[ac], XmLABELSTRING,
397     XmFontList, 0, fargok); if (fargok) ac++;
398 XmConvArg(args[ac], XmLABELSTRING,
399     "Stop", 0, fargok); if (fargok) ac++;
400 XmConvString(0, fargok); if (fargok) ac++;
401 XtSetArg(args[ac], XmNlabelString, "Stop", 160); ac++;
402 XtSetArg(args[ac], XmNheight, 160); ac++;
403 XtSetArg(args[ac], XmNlabelString, "Quit", 140); ac++;
404 XtSetArg(args[ac], XmNheight, 35); ac++;
405 stopPushButton = XmCreatePushButton(form,
406     "stopPushButton",
407     args,
408     ac);
409 XtManageChild(stopPushButton);
410 XtAddCallback(stopPushButton, XmactivateCallback, stopButtonCallback, (XtPointer)0);
411 ac = 0;
412 no = 0;
413 XtSetArg(args[ac], XmEDITABLE, False); ac++;
414 XtSetArg(args[ac], XmNlabelString,
415     "Edit", 410); ac++;
416 XtSetArg(args[ac], XmNlabelString,
417     "Text", 120); ac++;
418 numRecTextField = XmCreateTextfield(form,
419     "numRecTextfield",
420     args,
421     ac);
422 XtManageChild(numRecTextField);
423 ac = 0;
424 XtSetArg(args[ac], XmFONTLIST,
425     XmConvArg(args[ac], XmFONTLIST, "Form", 140); ac++;
426 XmFontList, 0, fargok); if (fargok) ac++;
427 XtSetArg(args[ac], XmLABELSTRING,
428     "Record", 30); ac++;
429 XmConvArg(args[ac], XmHORIZONTAL, "Record", 50);
430 XmConvArg(args[ac], XmHORIZONTAL, "Record", 50);
431 XtSetArg(args[ac], XmNlabelString,
432     "Record", 30); ac++;
433 XtSetArg(args[ac], XmNlabelString,
434     "Record", 70); ac++;
435 numberRecordLabel = XmCreateLabel(form,
436     "numberRecordLabel",
437     args,
438     ac);
439 XtManageChild(numberRecordLabel);
440 ac = 0;
441 no = 0;
442 XtSetArg(args[ac], XmNALIGNMENT, XmTOPCENTER_CENTER); ac++;
443 XtSetArg(args[ac], XmNSENSITIVE, False); ac++;
444 XtSetArg(args[ac], XmFONTLIST,
445     XmConvArg(args[ac], XmFONTLIST, "Form", 240); ac++);
446 XmFontList, 0, fargok); if (fargok) ac++;
447 XtSetArg(args[ac], XmNforeground, "#000000");
448 XmConvArg(args[ac], XmNforeground, "#FFFFFF");
449 XmPixel, 0, fargok); if (fargok) ac++;
450 XtSetArg(args[ac], XmLABELSTRING,
451     XmConvArg(args[ac], XmFONTLIST, "Record UP Books",
452     XmNlabelString, 0, fargok); if (fargok) ac++;
453 XmSetArg(args[ac], XmNlabelString, "Books", 200); ac++;
454 XmConvArg(args[ac], XmNlabelString, "Books", 200); ac++;
455 XmSelArg(args[ac], XmNlabelString, "Books", 230); ac++;
456 XmSelArg(args[ac], XmNheight, 25); ac++;
457 label = XmCreateLabel(form,
458     "label",
459     args,
460     ac);
461 XmManageChild(label);
462 ac = 0;
463 XmSelArg(args[ac], XmNfontList,
464     XmConvArg(args[ac], XmNfontList,
465     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++;
466 XmConvArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
467 XtSetArg(args[ac], XmNlabelString,
468     XmConvArg(args[ac], XmNlabelString, "Quit", 20); ac++);
469 XmSelArg(args[ac], XmNlabelString, "Quit", 20); ac++;
470 XmSelArg(args[ac], XmNlabelString, "Stop", 20); ac++;
471 XmSelArg(args[ac], XmNlabelString, "Stop", 20); ac++;
472 XmSelArg(args[ac], XmNlabelString, "Record", 52); ac++;
473 quitPushButton = XmCreatePushButton(form,
474     "quitPushButton",
475     args,
476     ac);
477 XtManageChild(quitPushButton);
478 XtAddCallback(quitPushButton, XmactivateCallback, quitButtonCallback, (XtPointer)0);
479 ac = 0;
480 XmSelArg(args[ac], XmNfontList,
481     XmConvArg(args[ac], XmNfontList,
482     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++;
483 XmConvArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
484 XtSetArg(args[ac], XmNlabelString,
485     XmConvArg(args[ac], XmNlabelString, "Record", 30); ac++);
486 XmConvArg(args[ac], XmNlabelString, "Record", 30); ac++;
487 XtSetArg(args[ac], XmNlabelString, "Push", 380); ac++;
488 XmSelArg(args[ac], XmNlabelString, "Push", 140); ac++;
489 XmSelArg(args[ac], XmNlabelString, "Push", 140); ac++;
490 XmSelArg(args[ac], XmNlabelString, "Push", 74); ac++;
491 XmSelArg(args[ac], XmNlabelString, "Push", 74); ac++;
492 resumePushButton = XmCreatePushButton(form,
493     "resumePushButton",
494     args,
495     ac);
496 XtManageChild(resumePushButton);
497 XtAddCallback(resumePushButton, XmactivateCallback, resumeButtonCallback, (XtPointer)0);
498 ac = 0;
499 XmSelArg(args[ac], XmNfontList,
500     XmConvArg(args[ac], XmNfontList,
501     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++);
502 XmConvArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
503 XtSetArg(args[ac], XmNlabelString,
504     XmConvArg(args[ac], XmNlabelString, "Pause", 200); ac++);
505 XmSelArg(args[ac], XmNlabelString, "Pause", 200); ac++;
506 XmSelArg(args[ac], XmNlabelString, "Pause", 140); ac++;
507 XmSelArg(args[ac], XmNlabelString, "Pause", 140); ac++;
508 XmSelArg(args[ac], XmNlabelString, "Pause", 59); ac++;
509 XmSelArg(args[ac], XmNlabelString, "Pause", 59); ac++;
510 pausePushButton = XmCreatePushButton(form,
511     "pausePushButton",
512     args,
513     ac);
514 XtManageChild(pausePushButton);
515 XtAddCallback(pausePushButton, XmactivateCallback, pauseButtonCallback, (XtPointer)x);

```

creation-c.c

8

```

453 XmSelArg(args[ac], XmNfontList,
454     XmConvArg(args[ac], XmNfontList,
455     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++);
456 XmSelArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
457 label = XmCreateLabel(form,
458     "label",
459     args,
460     ac);
461 XmManageChild(label);
462 ac = 0;
463 XmSelArg(args[ac], XmNfontList,
464     XmConvArg(args[ac], XmNfontList,
465     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++);
466 XmSelArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
467 XtSetArg(args[ac], XmNlabelString,
468     XmConvArg(args[ac], XmNlabelString, "Quit", 20); ac++);
469 XmSelArg(args[ac], XmNlabelString, "Quit", 20); ac++;
470 XmSelArg(args[ac], XmNlabelString, "Stop", 20); ac++;
471 XmSelArg(args[ac], XmNlabelString, "Stop", 20); ac++;
472 XmSelArg(args[ac], XmNlabelString, "Record", 52); ac++;
473 quitPushButton = XmCreatePushButton(form,
474     "quitPushButton",
475     args,
476     ac);
477 XtManageChild(quitPushButton);
478 XtAddCallback(quitPushButton, XmactivateCallback, quitButtonCallback, (XtPointer)0);
479 ac = 0;
480 XmSelArg(args[ac], XmNfontList,
481     XmConvArg(args[ac], XmNfontList,
482     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++);
483 XmConvArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
484 XtSetArg(args[ac], XmNlabelString,
485     XmConvArg(args[ac], XmNlabelString, "Record", 30); ac++);
486 XmConvArg(args[ac], XmNlabelString, "Record", 30); ac++;
487 XtSetArg(args[ac], XmNlabelString, "Push", 380); ac++;
488 XmSelArg(args[ac], XmNlabelString, "Push", 140); ac++;
489 XmSelArg(args[ac], XmNlabelString, "Push", 140); ac++;
490 XmSelArg(args[ac], XmNlabelString, "Push", 74); ac++;
491 XmSelArg(args[ac], XmNlabelString, "Push", 74); ac++;
492 resumePushButton = XmCreatePushButton(form,
493     "resumePushButton",
494     args,
495     ac);
496 XtManageChild(resumePushButton);
497 XtAddCallback(resumePushButton, XmactivateCallback, resumeButtonCallback, (XtPointer)0);
498 ac = 0;
499 XmSelArg(args[ac], XmNfontList,
500     XmConvArg(args[ac], XmNfontList,
501     "XmConvArg(args[ac], XmNfontList, "Form", 160); ac++);
502 XmConvArg(args[ac], XmNfontList, 0, fargok); if (fargok) ac++;
503 XtSetArg(args[ac], XmNlabelString,
504     XmConvArg(args[ac], XmNlabelString, "Pause", 200); ac++);
505 XmSelArg(args[ac], XmNlabelString, "Pause", 200); ac++;
506 XmSelArg(args[ac], XmNlabelString, "Pause", 140); ac++;
507 XmSelArg(args[ac], XmNlabelString, "Pause", 140); ac++;
508 XmSelArg(args[ac], XmNlabelString, "Pause", 59); ac++;
509 XmSelArg(args[ac], XmNlabelString, "Pause", 59); ac++;
510 pausePushButton = XmCreatePushButton(form,
511     "pausePushButton",
512     args,
513     ac);
514 XtManageChild(pausePushButton);
515 XtAddCallback(pausePushButton, XmactivateCallback, pauseButtonCallback, (XtPointer)x);

```

10/16/02
15:35:35

creation-c.C

9
10/16/02
15:35:35

creation-c.C

10

```

516     ac = 0;
517     XtSetArg((arg[1]), XmNfontList,
518             "XtConvertForm, \"-*-times-medium-r-*-*-160-*-*-*-*iso8859-1",
519             XmFontList, 0, &arg0); if ((arg0) ac++);
520     XtSetArg((arg[1]), XmNlabelString,
521             "XtConvertForm, \"Record\"", ac++);
522     XtSetArg((arg[1]), XmNheight, 30);
523     XtSetArg((arg[1]), XmNwidth, 20); ac++;
524     XtSetArg((arg[1]), XmNlabel, "Record"); if ((arg0) ac++);
525     XtSetArg((arg[1]), XmNlabelString, "Record");
526     XtSetArg((arg[1]), XmNwidth, 106); ac++;
527     XtSetArg((arg[1]), XmNheight, 35); ac++;
528     recordPushbutton = XmCreatePushButton(form,
529                                         "recordPushbutton",
530                                         args,
531                                         ac);
532     XtManageChild(recordPushbutton, xmactivateCallback, recordButtonCallback, (XtPointer)
533                                         0);
534     ac = 0;
535     XtSetArg((arg[1]), XmNlabelString, "120"); ac++;
536     XtSetArg((arg[1]), XmNlabel, "120"); ac++;
537     XtSetArg((arg[1]), XmNwidth, 70); ac++;
538     XtSetArg((arg[1]), XmNlabelString, "120");
539     XtSetArg((arg[1]), XmNheight, 120); ac++;
540     XtSetArg((arg[1]), XmNwidth, 30); ac++;
541     portTextfield = XmCreateTextfield(form,
542                                         "portTextfield",
543                                         args,
544                                         ac);
545     XtManageChild(portTextfield);
546     ac = 0;
547     XtSetArg((arg[1]), XmNfontList,
548             "XtConvertForm, \"-*-times-medium-r-*-*-160-*-*-*-*iso8859-1",
549             XmFontList, 0, &arg0); if ((arg0) ac++);
550     XtSetArg((arg[1]), XmNlabelString,
551             "XtConvertForm, \"Port Name\"", ac++);
552     XtSetArg((arg[1]), XmNlabel, "Port Name"); if ((arg0) ac++);
553     XtSetArg((arg[1]), XmNlabelString, "Port Name");
554     XtSetArg((arg[1]), XmNwidth, 40); ac++;
555     XtSetArg((arg[1]), XmNlabelString, "80");
556     XtSetArg((arg[1]), XmNwidth, 70); ac++;
557     portLabel = XmCreateLabel(form,
558                               "portLabel",
559                               args,
560                               ac);
561     XtManageChild(portLabel);
562     ac = 0;
563     XtSetArg((arg[1]), XmNfontList,
564             "XtConvertForm, \"-*-times-medium-r-*-*-160-*-*-*-*iso8859-1",
565             XmFontList, 0, &arg0); if ((arg0) ac++);
566     XtSetArg((arg[1]), XmNlabelString,
567             "XtConvertForm, \"Browse\"", ac++);
568     XtSetArg((arg[1]), XmNlabelString, "Browse");
569     XtSetArg((arg[1]), XmNwidth, 40); ac++;
570     XtSetArg((arg[1]), XmNlabelString, "10");
571     XtSetArg((arg[1]), XmNwidth, 72); ac++;
572     XtSetArg((arg[1]), XmNheight, 35); ac++;
573     browseButton = XmCreatePushButton(form,
574                                         "browseButton",
575                                         args,
576                                         ac);
577     XtManageChild(browseButton,
578                 xmactivateCallback, browseButtonCallback, (XtPointer) 0);
579     XtAddCallback(browseButton, xmactivateCallback, browseButtonCallback, (XtPointer) 0);
580     ac = 0;
581     XtSetArg((arg[1]), XmNlabelString, "OutputField");
582     XtSetArg((arg[1]), XmNfontList,
583             "XtConvertForm, \"-*-times-medium-r-*-*-160-*-*-*-*iso8859-1",
584             XmFontList, 0, &arg0); if ((arg0) ac++);
585     XtSetArg((arg[1]), XmNlabelString, "Output File");
586     XtSetArg((arg[1]), XmNlabel, "Output File");
587     XtSetArg((arg[1]), XmNwidth, 20); ac++;
588     XtSetArg((arg[1]), XmNlabelString, "Output File");
589     XtSetArg((arg[1]), XmNwidth, 20); ac++;
590     XtSetArg((arg[1]), XmNlabelString, "Output File");
591     XtSetArg((arg[1]), XmNwidth, 10); ac++;
592     XtSetArg((arg[1]), XmNlabelString, "Output File");
593     XtSetArg((arg[1]), XmNwidth, 110); ac++;
594     XtSetArg((arg[1]), XmNlabelString, "Output File");
595     XtSetArg((arg[1]), XmNwidth, 19); ac++;
596     XtSetArg((arg[1]), XmNlabelString, "Output File");
597     XtSetArg((arg[1]), XmNwidth, 20); ac++;
598     XtSetArg((arg[1]), XmNlabelString, "Output File");
599     XtSetArg((arg[1]), XmNwidth, 10); ac++;
600     XtSetArg((arg[1]), XmNlabelString, "Output File");
601     XtSetArg((arg[1]), XmNwidth, 110); ac++;
602     XtSetArg((arg[1]), XmNlabelString, "Output File");
603     XtSetArg((arg[1]), XmNwidth, 19); ac++;
604     XtSetArg((arg[1]), XmNlabelString, "Output File");
605     XtSetArg((arg[1]), XmNwidth, 10); ac++;
606     XtSetArg((arg[1]), XmNlabelString, "Output File");
607     XtSetArg((arg[1]), XmNwidth, 110); ac++;
608     XtSetArg((arg[1]), XmNlabelString, "Output File");
609     XtSetArg((arg[1]), XmNwidth, 157); ac++;
610     XtSetArg((arg[1]), XmNlabelString, "395"); ac++;
611     selectFileDialog = XmCreatePopUpShell("selectFileDialog",
612                                         xmfileSelectFileDialogClass,
613                                         mainWindow,
614                                         args,
615                                         ac);
616     ac = 0;
617     XtSetArg((arg[1]), XmNlabelString, "186"); ac++;
618     XtSetArg((arg[1]), XmNlabel, "186"); ac++;
619     XtSetArg((arg[1]), XmNwidth, 77); ac++;
620     XtSetArg((arg[1]), XmNlabelString, "437"); ac++;
621     XtSetArg((arg[1]), XmNwidth, 395); ac++;
622     XmFileSelectionBox = XmCreateFileSelectionBox("selectFileDialog",
623                                         xmfileSelectFileDialogClass,
624                                         mainWindow,
625                                         args,
626                                         ac);
627     XtAddCallback(inputFileSelectionBox, xmokCallback, okFileDialogCallback, (XtPointer)
628                                         Pointer(0));
629     ac = 0;
630     XtAddCallback(inputFileSelectionBox, xmcancelCallback, cancelFileDialogCallback, (XtPointer)
631                                         ac);
632     XtSetArg((arg[1]), XmNlabelString, "284"); ac++;
633     XmFileSelectionBox = XmCreatePopUpShell("aboutHelpDialogShell",
634                                         xmfileHelpDialogClass,
635                                         mainWindow,
636                                         args,
637                                         ac);
638     ac = 0;
639     XtSetArg((arg[1]), XmNlabelString, "200raisePolicy", XmRESIZE_GROW); ac++;
640     XtSetArg((arg[1]), XmNlabel, "200raisePolicy", XmRESIZE_GROW); ac++;
641     aboutHelpDialogShell = XmCreatePopUpShell("aboutHelpDialogShell",
642                                         xmfileHelpDialogClass,
643                                         mainWindow,
644                                         args,
645                                         ac);

```

10/16/02
15:35:35

creation-C.C

11

10/16/02
15:35:35

creation-C.C

12

```
644 XSetArg(args [ac], XmNheight, 312); ac++;
645 aboutHelpBulletinBoard = XmCreateScrolledWindow(aboutHelpPanel,
646 "aboutHelpBulletinBoard",
647 args,
648 ac);
649 ac = 0;
650 XSetArg(args [ac], XmNscrollingPolicy, XmAUTOMATIC); ac++;
651 XSetArg(args [ac], XmNsensitive, 20); ac++;
652 XSetArg(args [ac], XmNlabelString, "aboutScrolledWindow",
653 XmStringCreate("About Help", 20)); ac++;
654 XSetArg(args [ac], XmNlabel, XmStringCreate("About Help", 20)); ac++;
655 XSetArg(args [ac], XmNheight, 210); ac++;
656 aboutScrolledWindow = XmCreateScrolledWindow(aboutHelpBulletinBoard,
657 "aboutScrolledWindow",
658 args,
659 ac);
660 XtManageChild(aboutScrolledWindow);
661
662 ac = 0;
663 XSetArg(args [ac], XmNalignment, XmLEFTCENTER); ac++;
664 XSetArg(args [ac], XmFontList,
665 BX_CONVERT(aboutScrolledWindow, "Helvetica-Bold", "130"));
666 XSetArg(args [ac], XmNfontList, 0, XmFontList);
667 XSetArg(args [ac], XmNlabelString, "aboutScrolledWindow");
668 BX_CONVERT(aboutScrolledWindow, ":"~":" Minichsets"\rt"\":r"\Record socket in
669 sessions"\rt"\into a file"\rt"\":r"\National Research Council\rt"\:r\ Da
in
670 Johnston\rt"\:r\ All rights reserved\rt"\:");
671 XSetArg(args [ac], XmNlabelString, "aboutScrolledWindow");
672 XSetArg(args [ac], XmNheight, 250); ac++;
673 aboutHelpLabel = XmCreateLabel(aboutScrolledWindow,
674 "aboutHelpLabel",
675 args,
676 ac);
677 XtManageChild(aboutHelpLabel);
678 ac = 0;
679 XSetArg(args [ac], XmNlabelString,
680 BX_CONVERT(aboutHelpPanel,
681 "Dismise", "aboutHelpString", 0, XmFont));
682 if ((argok)) ac++;
683 XSetArg(args [ac], XmNsensitive, 20); ac++;
684 XSetArg(args [ac], XmNlabel, XmStringCreate("Dismise", 50)); ac++;
685 XSetArg(args [ac], XmNheight, 127); ac++;
686 aboutDismisePushbutton = XmCreatePushButton(aboutHelpPanel,
687 "aboutDismisePushbutton",
688 args,
689 ac);
690 XtManageChild(aboutDismisePushbutton);
691 XtAddCallback(aboutDismisePushbutton, XmactivateCallback, (XtP
692 aboutDismise);
693 ac = 0;
694 XSetArg(args [ac], XmNameHelpWidget, helpCascadeButton); ac++;
695 XtSetValues(imenuar, args, ac);
696 ac = 0;
697 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
698 XSetArg(args [ac], XmNleftOffset, 160); ac++;
699 XSetArg(args [ac], XmNsensitive, 200); ac++;
700 XSetArg(args [ac], XmNleftOffset, 140); ac++;
701 XtSetValues(stopPushButton, args, ac);
702 ac = 0;
703 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
704 XSetArg(args [ac], XmNleftOffset, 410); ac++;
705 XSetArg(args [ac], XmNleftPosition, 4); ac++;

706 XSetArg(args [ac], XmNtopOffset, 70); ac++;
707 XtSetValues(imenuar, args, ac);
708 ac = 0;
709 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
710 XSetArg(args [ac], XmNleftOffset, 200); ac++;
711 XSetArg(args [ac], XmNtopOffset, 200); ac++;
712 XtSetValues(label, args, ac);
713 ac = 0;
714 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
715 XSetArg(args [ac], XmNleftOffset, 380); ac++;
716 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
717 XSetArg(args [ac], XmNleftOffset, 140); ac++;
718 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
719 XSetArg(args [ac], XmNleftOffset, 20); ac++;
720 XSetArg(args [ac], XmNtopOffset, 200); ac++;
721 XtSetValues(quitePushButton, args, ac);
722
723 ac = 0;
724 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
725 XSetArg(args [ac], XmNleftOffset, 280); ac++;
726 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
727 XSetValues(pausePushButton, args, ac);
728 ac = 0;
729 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
730 XSetArg(args [ac], XmNleftOffset, 20); ac++;
731 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
732 XSetValues(recordPushButton, args, ac);
733 ac = 0;
734 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
735 XSetArg(args [ac], XmNleftOffset, 20); ac++;
736 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
737 XSetArg(args [ac], XmNleftOffset, 140); ac++;
738 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
739 XSetValues(recordPushButton, args, ac);
740 ac = 0;
741 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
742 XSetArg(args [ac], XmNleftOffset, 120); ac++;
743 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
744 XSetArg(args [ac], XmNleftOffset, 80); ac++;
745 XSetValues(portTextfield, args, ac);
746 ac = 0;
747 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
748 XSetArg(args [ac], XmNleftOffset, 40); ac++;
749 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
750 XSetArg(args [ac], XmNleftOffset, 80); ac++;
751 XSetValues(portLabel, args, ac);
752 ac = 0;
753 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
754 XSetArg(args [ac], XmNleftOffset, 480); ac++;
755 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
756 XSetArg(args [ac], XmNleftOffset, 10); ac++;
757 XSetValues(braceButton, args, ac);
758 ac = 0;
759 XSetArg(args [ac], XmNleftAttachment, XmATTACH_FORM); ac++;
760 XSetArg(args [ac], XmNleftOffset, 110); ac++;
761 XSetArg(args [ac], XmNleftOffset, 10); ac++;
762 XSetValues(outputFileTextfield, args, ac);
763 ac = 0;
764 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
765 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
766 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
767 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
768 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
769 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
770 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
771 XSetArg(args [ac], XmNleftAttachment, XmATTACH_POSITION); ac++;
```

10/16/02
15:35:35

creation-c.c

13

```
772 XxBetAny(args[ac], xmrightPosition, 21); ac++;
773 XxBetAny(args[ac], xmleftOffset, 20); ac++;
774 XxBetValues(outputsLabel, args, ac);
775
776
777 /* Begin user code block <and CreateMainWindow> */
778 /* End user code block <and CreateMainWindow> */
779 return( mainWindow );
780 }
781 }
```


10/16/02
15:35:37

creation-c.h

1

```
1  /* WARNING: This file is overwritten at code generation time.
2   * Any changes to this file will be lost.
3   */
4  /*
5   */
6  /*
7   * Created by Builder Xcessory Version 5.0
8   * Generated by Code Generator Xcessory 5.0 (05/22/98)
9   */
10 /*
11 #ifndef creation_c_H
12 #define creation_c_H
13 */
14 /*
15  * Global widget declarations.
16  * _EXTERNAL is set to extern if the
17  * .defs file is not included from the
18  * main file.
19  */
20 #ifdef DECLARE_BY_GLOBALS
21 #define EXTERNAL
22 #else
23 #define EXTERNAL extern
24 #endif
25 /*
26 /*
27  * Start Global Widget Declarations .
28  */
29 EXTERNAL Widget
30 EXTERNAL Widget
31 EXTERNAL Widget
32 EXTERNAL Widget
33 EXTERNAL Widget
34 EXTERNAL Widget
35 EXTERNAL Widget
36 EXTERNAL Widget
37 EXTERNAL Widget
38 EXTERNAL Widget
39 EXTERNAL Widget
40 /*
41  * End Global Widget Declarations .
42  */
43 #endif
```


10/16/02
15:35:47

1 main-c.C
1 10/16/02
1 15:35:47

```
1 /*  
2 * README: Portions of this file are merged at file generation.  
3 * time. Edits can be made "only" in between specified code blocks, look  
4 * for keywords <begin user code> and <end user code>.  
5 */  
6 /* Generated by the ICS Builder Xcessory (BX).  
7 *  
8 * Builder Xcessory Version 5.0  
9 * Code Generator Xcessory 5.0 (05/22/98)  
10 */  
11 /* File record/main-c.c  
12 * Usergroup mod_record.  
13 */  
14 /* Begin user code block <file_comments> */  
15 /*.*/  
16 /* End user code block <file_comments> */  
17 /* File record/main-c.c  
18 * Usergroup mod_record.  
19 */  
20 /* This file is mostly created by the user interface builder tool  
21 * Builder Xcessory version 5.0. There are only a few places where  
22 * we can add our input.  
23 */  
24 /* The purpose of this file is to create the user interface, set some  
25 * of the initial variables and user interface widget states, and  
26 * then loop waiting for (and responding to) events.  
27 */  
28 /* It is the response to the events, mainly in the callbacks source  
29 * file that will define the logic of the program.  
30 */  
31 /* Todo We should add logic to the main program to support '-h' or  
32 * '-help' as a valid command line argument. It would simply cause  
33 * the "usage" message to be written.  
34 */  
35 /* Code  
36 * All content of this file is copyrighted to the  
37 * National Research Council of Canada.  
38 */  
39 /* All rights reserved.  
40 */  
41 /* Author Daniel F. Johnston  
42 */  
43 /* Version August 2002  
44 */  
45 /* $Id: main-c,v 1.6 2002/08/19 18:46:03 Johnston Exp $  
46 */  
47 /* End user code block <file_comments> */  
48 */  
49 /* Motif required Headers  
50 */  
51 /*include <X11/StringDefs.h>  
52 */  
53 /*include <Xm/Xm.h>  
54 */  
55 /*if (XmVersion > 1.002)  
56 */  
57 /*include <Xm/MemUtil.h>  
58 */  
59 /* Globally included information.  
60 */  
61 */  
62 */  
63 */  
64 /* Headers for classes used in this program  
65 */  
66 */
```

main-c.C
2

```
67 /* Common constant and pixmap declarations.  
68 */  
69 #include "creation-c.h"  
70 */  
71 /* Convenience functions from utilities file.  
72 */  
73 /* Convenience functions from utilities file.  
74 */  
75 /* Convenience functions from utilities file.  
76 extern void RegisterBnCConverters(XtAppContext);  
77 extern XtPointers BX_CONVERT(Widge, char *, int, Boolean *);  
78 extern XtPointers BX_CONVERT(Widget, char *, int, Boolean *);  
79 extern XtPointers BX_SPECIZE(Widget);  
80 extern void BX_SETUP_PGM(Widge, XtPointers, XtEvent *, Boolean *);  
81 extern Pixmap_XPM_PIXMAP(Widge, ArgList, Cardinal *, Pixel);  
82 extern void BX_SET_BACKGROUND_COLOUR(Widget, ArgList, Cardinal *, Pixel);  
83 */  
84 /* Begin user code block <globals> */  
85 #include "../common/mylocket.h"  
86 #include "../common/myInterface.h"  
87 // the block of user defined Globals shared by this interface  
88 GlobalMemory shared;  
89 /* End user code block <globals> */  
90 */  
91 /* Change this line via the Output Application Names Dialog.  
92 */  
93 /* Define BX_APP_CLASS "BuilderProduct"  
94 */  
95 int main( int argc, char **argv )  
96 {  
97     Widget parent;  
98     XtAppContext app;  
99     Arg args[256];  
100    Cardinal ac;  
101    Boolean argok=false;  
102    Widget topLevelShell;  
103    Window mainWindow;  
104    Widget  
105    /* Begin user code block <declarations> */  
106    /* End user code block <declarations> */  
107    /* The applicationShell is created as an unrealised  
108     * parent for multiple topLevelShells. The topLevelShells  
109     * are created as popout children of the applicationsShell.  
110     * This is a recommendation of Paul Asente & Ralph Swick in  
111     * X Window System Toolkit, p. 677.  
112 */  
113    /* The applicationShell is created as an unrealised  
114     * parent for multiple topLevelShells. The topLevelShells  
115     * are created as popout children of the applicationsShell.  
116     * This is a recommendation of Paul Asente & Ralph Swick in  
117     * X Window System Toolkit, p. 677.  
118     *endif  
119     *if(XlibSpecificationRelease  
120     *else  
121     *if (XlibSpecificationRelease>=5)  
122     *else  
123     *else  
124     *(Cardinal *) argv,  
125     *endif  
126     *endif  
127     argv, NULL,  
128     NULL );  
129 */  
130 RegisterBnCConverters(app);  
131 #if (Xversion >= 1002)  
132 XmSetPARENTSofRMModelConverter();
```

10/16/02
15:35:47

main-c.c

10/16/02
15:35:47

3

main-c.C

4

```
133 #endif
134 /* Begin user code block <create_shells> */
135 ac = (int) argc; /* use the variable to surprise compiler warnings */
136 /* End user code block <create_shells> */
137
138 /*
139 * Create classes and widgets used in this program.
140 */
141
142 /* Begin user code block <create_topLevelShell> */
143 /*
144 *ingroup mod_record
145
146 This main program is mostly computer generated. We only modify it
147 (between the "user code block commands") to create our global memory
148 structure and initialize it based on the command line arguments.
149
150 /* copy our current application context to make it global */
151 shared.mainAppContext = app;
152
153 /* start by assuming there is no supplied args */
154 shared.outputFileName[0] = shared.portNumber[0] = '\0';
155
156 shared.outputFileName[0] = shared.portNumber[0];
157
158 switch ( argc )
159 {
160     /* we should have both args, port and filename */
161     /* process the filename arg */
162     if( !shared.filename = fopen( argv[1], "w" ) )
163         printf("Unmain! Cannot open file '%s'\n", argv[1] );
164     usageRecord();
165     exit(1);
166
167     strcpy( shared.outputFileName, argv[2] );
168     /* no 'break' here, fall through to process arg 2 */
169
170     case 2:
171         /* we have 1 arg, the port number */
172         /* process the port number here */
173         if( sscanf( argv[1], "%u", shared.portNum ) != 1 )
174             printf("Unbot a valid port number '%s'\n", argv[1] );
175         usageRecord();
176         exit(1);
177
178         strcpy( shared.portNumber, argv[1] );
179         break;
180     case 1:
181         /* we have no user supplied command args */
182         break;
183
184     default:
185         usageRecord();
186         exit(1);
187         break;
188
189     /* End user code block <create_topLevelShell> */
190
191     ac = 0;
192     XCreateRecord( argv[ac], XSMR, 562); ac++;
193     XCreateRecord( argv[ac], XSMR, 656); ac++;
194     XCreateRecord( argv[ac], XSMR, 619); ac++;
195     XCreateRecord( argv[ac], XSMR, 250); ac++;
196     topLevelShell = XCreatePopUpShell("topLevelShell",
197     topLevelShellWidgetSetClass,
198
199     Parent,
200     ac,
201     ac);
202     mainWindow = (Widget)CreateMainWindow( topLevelShell );
203     XtManageChild(mainWindow);
204     XtPopUp(ktParent(mainWindow), xtDrawables);
205
206     /* Begin user code block <app_procedures> */
207     /* set the initial state of all the GUI interface widgets */
208     /* the default input file name */
209     if( strlen( shared.outputFileName ) == 0 )
210     {
211         XmTextFieldGetString( outputFileNameField, "<none>" );
212     }
213     XmTextFieldGetString( outputFileNameField, shared.outputFileName );
214
215     /* the default value for the network port number if not user supplied */
216     if( strlen( shared.portNumber[0] ) == 0 )
217     {
218         shared.portNumber = DEFAULT_PORT_NUMBER;
219         sprintf( shared.portNumber, "%d", shared.portNumber );
220         XmTextFieldGetString( portTextfield, shared.portNumber );
221     }
222     XmTextFieldGetString( portTextfield, shared.portNumber );
223     sscanf( shared.portNumber, "%d", shared.portNumber );
224
225     /* the initial 'enabled' stat for pushbuttons */
226     /* record enabled, stop, pause and resume disabled */
227     /* record sensitive, recordpushbutton, true */ /* it already is, by default */
228     XSetSensitive( stopPushButton, False );
229     XSetSensitive( recordPushButton, False );
230     XSetSensitive( pausePushButton, False );
231     XSetSensitive( resumePushButton, False );
232
233     /* the dialog box for file selection is not visible initially */
234     XtManageChild(inputFileSelectionBox);
235     /* the dialog box for help (aboutHelpPanel) is not visible initially */
236     XtManageChild(aboutHelpPanel);
237     shared.messageNumber = 0;
238     XmTextFieldGetString( numRecrTextfield, "0" );
239
240     /* we are not sending yet */
241     /* shared.childProcess = 0;
242
243     /* End user code block <app_procedures> */
244
245     /* Begin user code block <main_loop> */
246     /* End user code block <main_loop> */
247     /* End user code block <main_loop> */
248
249     XAppMainLoop(app);
250
251     /* A return value regardless of whether or not the main loop ends.
252
253     */
254
255 }
```

10/16/02
15:35:56

record_support.c

1

```
1  /*\file record_support.c
2   \ingroup mod_record
3
4   This file contains the support function needed for the
5   recording application.  We have a usage printing function.
6
7   \code
8
9   Copyright (c) 2002
10  All content of this file is copyrighted to the
11  National Research Council of Canada.
12  All rights reserved.
13  \endcode
14  \author Daniel V. Johnston
15  \date May 2003
16
17  $Id: record_support.c,v 1.4 2002/10/16 18:22:18 Johnston Exp $
18  */
19
20  /*\ingroup mod_record
21
22  This function will be called when there is an error with the user
23  supplied command line arguments for the socket recording application.
24  It will simply print out the list of standard and optional command
25  arguments, and then finish.
26  */
27  void
28  usagerecord( void )
29  {
30     printf("\nthe required command is of the form...\n");
31     printf("  recordSocket [portnum] [filename]\n");
32     printf("    - where 'portnum' is the port number to listen to\n");
33     printf("    - and 'filename' is the file to save socket data\n");
34
35 }
```

36

10/16/02
15:36:04

recording.c

1

10/16/02
15:36:04

```
1  /** \file recording.c
2   * \ingroup mod_record
3   *
4   * This file contains the support functions needed for creating
5   * the socket and reading socket messages. We also provide
6   * functions for controlling the sending process, i.e.
7   * pause/resume, sending done, etc.
8   *
9   */
10  \code
11  Copyright (c) 2002
12  All content of this file is copyrighted to the
13  National Research Council of Canada.
14  All rights reserved.
15  Vendorcode Daniel F. Johnston
16  Author Daniel F. Johnston
17  \since Aug 2002
18  $Id: recording.c,v 1.7 2002/10/16 18:13:43 johnston Exp $
19  */
20  */
21  #include <xm/mn.h>
22  #include "creation-c.h"
23  #include "socket.h"
24  #include "sys/socket.h"
25  #include "sys/wait.h"
26  #include <netinet/in.h>
27  #include <netdb.h>
28  #include <time.h>
29  #include "common.h"
30  #include ".../common/mySocket.h"
31  #include ".../common/myInterface.h"
32  // the byte array that will hold a UDP socket message before storing
33  // char buf[MAX_MESSAGE_SIZE];
34  /*
35  */
36  /**
37  * \ingroup mod_record
38  * This function will create a socket. It uses the supplied
39  * port number to create a socket that can read messages on the local host.
40  * Param port is the integer port number to use for the new socket
41  * \return an error indication (if < 0) or the integer value of the socket
42  */
43  int createReadSocket( int port )
44  {
45  /*
46  * int sock;
47  * struct sockaddr_in server;
48  * sock = socket(AF_INET, SOCK_DGRAM, 0);
49  * if ( sock < 0 )
50  * {
51  *   perror("Opening recording stream socket");
52  *   return -1;
53  * }
54  * /* Just in case, we clear the structure */
55  * bzero((char *)server, sizeof (server));
56  * /* Assign client port number */
57  * server.sin_addr.s_addr = INADDR_ANY;
58  * /* we will read from any valid address */
59  * server.sin_port = port;
60  * /* name the socket */
61  */
62  */
63  */
64  */
65  */
66  */

    /**
     * \file recording.c
     * \ingroup mod_record
     * \brief This file contains the support functions needed for creating
     * the socket and reading socket messages. We also provide
     * functions for controlling the sending process, i.e.
     * pause/resume, sending done, etc.
     */

```

recording.c

2

```
67  if( bind( sock, (struct sockaddr *)server, sizeof server) < 0 )
68  {
69    perror("Binding recording stream socket");
70    return -1;
71  }
72  return sock;
73 }
74 */
75 /**
76 * the timer callback to reset the state of the
77 * record/stop/resume buttons
78 */
79 /**
80 * \ingroup mod_record
81 * This function will be called only if the recording child process
82 * is terminated. We will reset the record, stop, pause and
83 * resume buttons to their
84 * default state. This state is the 'start' button enabled, and the
85 * others disabled. It will also close the output file and
86 * write the number of messages stored into the user interface.
87 */
88 void
89 resetButtons()
90 {
91  extern GlobalMemory shared;
92  char tmphuf [MAX_BUTTON_TEXT_SIZE];
93 /**
94 * disable the pause, resume and stop buttons, reenable the record */
95 XSetSensitive( recordPushButton, True );
96 XSetSensitive( stopPushButton, False );
97 XSetSensitive( pausePushButton, False );
98 XSetSensitive( resumePushButton, False );
99 /**
100 * the child process is no more */
101 shared.childProcess = 0;
102 /**
103 * re-open the output file to read size */
104 shared.filerecipep = fopen( shared.outputFileName, "r" );
105 /**
106 * read the number of messages in the file */
107 shared.messageNumber = getNmbrMessages( shared.filerecipe );
108 fclose( shared.filerecipe );
109 sprintf( tmpbuf, "%d", shared.messageNumber );
110 numPartFieldString( numPartField, tmpbuf );
111 /**
112 * close down the existing socket */
113 closed.shared.socketInbnd = 0;
114 /**
115 * the "child should pause" signal handler */
116 /**
117 * \ingroup mod_record
118 * This function will be called by the child (reading) process in response
119 * to a 'pause' signal. The function will make sure that the process ID
120 * is correct, and if it is, it will cause the process to pause. The reading
121 * process will remain in this state until another signal changes it.
122 */
123 void
124 readProcessPause()
125 {
126  int pid;
127  int status;
128  extern GlobalMemory shared;
129  if( ( pid = wait(status) ) == -1 )
130  {
131    /**
132     * an error of some kind (probably a fork error) - ignore it */
133  }

    /**
     * \file recording.c
     * \ingroup mod_record
     * \brief This function will be called by the child (reading) process in response
     * to a 'pause' signal. The function will make sure that the process ID
     * is correct, and if it is, it will cause the process to pause. The reading
     * process will remain in this state until another signal changes it.
     */

```

10/16/02
15:36:04

recording.c

10/16/02
15:36:04

3

```
133     if( shared.globalDebugMode == 1 )
134         printf("Child process pause: fork error!\n");
135     return;
136 }
137 if( shared.globalDebugMode == 1 )
138     printf("Child process pause: pid %d received!\n", pid);
139     return; /* we resume only if 'resumed' by the main process */
140 if( pid == shared.childProcess )
141 {
142     /* we pause the current child process */
143     pause();
144     return; /* we resume only if 'resumed' by the main process */
145 }
146 /* else */
147 /* some other process id */
148 }
149 */
150 /* the "child should stop" signal handler */
151 /*+
152 \ingroup mod_record
153 This function will be called by the child (reading) process in response
154 to a 'stop' signal. The function will make sure that the process ID
155 is correct and, if it is, it will cause the process to exit. This
156 should trigger the 'child terminated' function in the main process.
157 */
158 void readprocessStop()
159 {
160     extern GlobalMemory shared;
161     fclose( shared.filabecrip );
162     /* we end the current child process */
163     exit(255);
164 }
165 */
166 /* the "child dies" signal handler */
167 /*+
168 \ingroup mod_record
169 This function will be called by the child (reading) process when
170 this child process is
171 done, i.e. when terminated by an error or the user. This
172 signal handler is used to queue an event for
173 the user interface to switch the
174 button states as required for the user interface.
175 */
176 */
177 void readprocessDone()
178 {
179     readprocessOne()
180 {
181     int pid;
182     int status;
183     extern GlobalMemory shared;
184     if( (pid = wait(status)) == -1 )
185     {
186         /* an error of some kind (probably a fork error) - ignore it */
187         if( shared.globalDebugMode == 1 )
188             printf("Child process done: fork error!\n");
189     }
190     return;
191 }
192 if( shared.globalDebugMode == 1 )
193     printf("Child process done: pid %d received!\n", pid);
194     return;
195 if( pid == shared.childProcess )
196     /* our expected child process died, reset the button state */
197     /* we cannot do it here because the child process cannot
198 */

199     /*
200     * access the same user interface as the main program
201     */
202     if( XAppAddTimeOut( shared.mainAppContext, 0, resetReadButtons, NULL ) );
203     return;
204 }
205 /* some other process died */
206 */
207 */
208 /* read the messages into the file */
209 /*+
210 \ingroup mod_record
211 This function will be called by the child (reading) process, and this
212 process will spend most of its time here in this loop. The child
213 process will loop until killed (stopped) by the main program. When the
214 process is done, the child process will trigger a signal to the main
215 process to let it know the child process is finished.
216 */
217 /*+
218 \param sock is the open socket we use for reading messages
219 \param file is the file descriptor for the (already opened) file
220 into which we will store the socket messages and delays.
221 \return an error indication (< 0) if there are sending problems
222 */
223 int readmessages( FILE *file )
224 {
225     extern GlobalMemory shared;
226     timespec t;
227     int length;
228     char buffer[MAX_MESSAGE_SIZE];
229     struct sockaddr_in client;
230     int clientfd;
231     /* get the current time of day as a reference */
232     clock_gettime( CLOCK_REALTIME, &t );
233     for ( ; ; /* loop waiting for messages */
234     {
235         /* initialise the client name len := 0, otherwise no client address */
236         /*+
237         \idnameclient = siseof( client );
238         /* in case there was stuff in structure from previous messages */
239         /*+
240         memset( client, '\0', sizeof(client) );
241         */
242         length = recvfrom( shared.socketfd, buffer, MAX_MESSAGE_SIZE, 0,
243                           &client, &clientfd );
244         /*+
245         if( length < 0 && shared.GlobalDebugMode == 1 )
246             perror("recvfrom error!\n");
247         */
248         if( clientfd != client.ssin_family )
249         {
250             /*+
251             if( clientfd > 0 && shared.GlobalDebugMode == 1 )
252                 printf("Client address is '%s' \n", (char *)inet_ntoa(client.sin_addr));
253             */
254             /* we need to null terminate the received message */
255             buffer[length] = '\0';
256             /*+
257             if( get the current time of day to calculate the delta */
258             /*+
259             clock_gettime( CLOCK_REALTIME, &currenttime );
260             delaytime.tv_sec = currenttime.tv_sec - previoustime.tv_sec;
261             delaytime.tv_nsec = currenttime.tv_nsec - previoustime.tv_nsec;
262             previoustime.tv_sec = currenttime.tv_sec;
263             previoustime.tv_nsec = currenttime.tv_nsec;
264         */
265     }
```

recording.c

4

10/16/02
15:36:04

recording.c

5

```
265      /* save the current time stamp, the message size
266      * and the message into the file
267      */
268      if( fwrite( &deltaTime.tv_sec, sizeof(long), 1, fileD ) != 1 )
269      {
270          perror("write (delta) to file error\n");
271          exit(1);
272      }
273      if( fwrite( &deltaTime.tv_nsec, sizeof(long), 1, fileD ) != 1 )
274      {
275          perror("write (nsec) to file error\n");
276          perror("Write (delta) to file error\n");
277          exit(1);
278      }
279      if( fwrite( &length, sizeof(int), 1, fileD ) != 1 )
280      {
281          perror("write (length) to file error\n");
282          exit(1);
283      }
284      /*
285      /* now write the actual message */
286      if( fwrite( buffer, length, 1, fileD ) != 1 )
287      {
288          perror("Write (buffer) to file error\n");
289          exit(1);
290      }
291      /*
292      /* end of forever loop */
293  }
```

Appendix C

Playback Program Source Code

10/16/02
15:32:21

callbacks-C.C

10/16/02
15:32:21

1

```
1 /*  
2 * WARNING: This file is appended to at file generation time.  
3 * Edits can be made throughout the file.  
4 */  
5 /* Generated by the ICS Builder accessory (ix). */  
6 /* Builder Accessory Version 5.0 */  
7 /* Code Generator Accessory 5.0 (05/22/98)  
8 */  
9 /*  
10 */  
11 #include <cm/m.h>  
12 #include <cm/mestf.h>  
13 #include <cm/mestf.h>  
14 #include <signal.h>  
15 #include "creation-c.h"  
16 #include ".../common/ayninterface.h"  
17 /* Standard includes for builtins.  
18 */  
19 /* Macros to make code look nicer between ANSI and K&R.  
20 */  
21 #include <stdio.h>  
22 #include <string.h>  
23 #include <cctype.h>  
24 /* If (needFunctionPrototypes == 0)  
25 */  
26 /*#ifndef ANSI_C  
27 #define ANSI_C  
28 #endif  
29 #if (needFunctionPrototypes == 0)  
30 #define PROTOTYPE(p) ()  
31 #define ARGLIST(p)()  
32 #define ARG(a, b) a, b  
33 #define GRP(a, b) a, b  
34 #define GRD(a, b) a, b  
35 #define GRAD(a, b) a, b  
36 #else  
37 #define PROTOTYPE(p) p  
38 #define ARGLIST(p)  
39 #define ARG(a, b) a, b  
40 #define GRN(a, b) a, b  
41 #endif  
42 #define _oplumberus  
43 #define _usage(a, b) a,  
44 #define _GRN(a, b) a,  
45 #define _GRD(a, b) a, b,  
46 #define _GRAD(a, b) a, b,  
47 #endif  
48 #endif  
49 #endif  
50 #include "BridgedropShell_prototype.h"  
51 Widget BridgedropShell_prototype(Widget, char*, char*);  
52 WidgetList BridgedropfdPrototype(Widget);  
53 /*  
54 */  
55 /* File play/callbacks-c.c  
56 */  
57 /* In this source file are the definitions for the callback routines  
58 * that will be called by Motif and X11 when the user clicks on  
59 * push buttons, toggle buttons and pull down menu options for  
60 * the 'playback nodes' application.  
61 */  
62 Some callback functions are simple enough to fully define here.  
63 But, many require a call to some of the support and sending  
64 functions defined in other source files. A few will actually  
65 start other process or send signals, and the support functions  
66
```

callbacks-C.C

10/16/02
15:32:21

2

```
67 will define what happens when these signals are processed.  
68 */  
69 \code  
70 Copyright (c) 2002  
71 All content of this file is copyrighted to the  
72 National Research Council of Canada.  
73 All rights reserved.  
74 \endcode  
75 Author: Daniel F. Johnston  
76 \since April 2003  
77 /*  
78 #include "callbacks-c.c" 1.10 2002/08/20 15:37:01 Johnston Exp $  
79 */  
80 // the shared, global variable structure  
81 extern GlobalMemory shared;  
82 /*  
83 */  
84 /*  
85 */  
86 /*ingroup mod_play  
87 This callback is the function that will be called when the Motif user  
88 interface selects one of the 'enable file output' toggle buttons. It  
89 will make sure that all the file buttons are enabled, and that all  
90 the network buttons are disabled.  
91 */  
92 /*param w the widget that caused the event and this callback. It  
93 could be one of two, but we don't care which in this case.  
94 */  
95 /*param client data is a pointer to any (optional) client data  
96 */  
97 /*param call_data the standard event information provided by X11-Motif  
98 */  
99 /*ingroup mod_play  
100 This callback is the function that will be called when the Motif user  
101 interface has selected a file destination, turn on file */  
102 /*param client data is a pointer to any (optional) client data  
103 */  
104 /*param call_data the standard event information provided by X11-Motif  
105 */  
106 /*  
107 */  
108 /*  
109 */  
110 /*param client data is a pointer to any (optional) client data  
111 */  
112 /*param w the widget that caused the event and this callback. It  
113 could be one of two, but we don't care which in this case.  
114 */  
115 /*param client data is a pointer to any (optional) client data  
116 */  
117 /*param call_data the standard event information provided by X11-Motif  
118 */  
119 /*  
120 */  
121 /* param has selected a network destination, turn off file if set */  
122 /*param client data is a pointer to any (optional) client data  
123 */  
124 /*param client data is a pointer to any (optional) client data  
125 */  
126 /*param client data is a pointer to any (optional) client data  
127 */  
128 */  
129 /*  
130 */  
131 /*  
132 */  
133 /*  
134 */  
135 /*param call_data is the function that will be called when the Motif user
```

10/16/02
15:33:21

callbacks-c.C

3 15:32:21

133 Interface changed the 'speed' slider on the user interface. It
134 will read the new value from the slider, and compute a speed
135 multiplier to show in the associated text area. This multiplier will
136 also be saved in the shared global structure.
137 Todo We should modify the slider to provide a wider range of
138 speed multiplier values, including one which could implement
139 a single step (single message) mode.
140

141 \param w the widget that caused the event and this callback. We ask
142 the slider widget for it's current value.
143 \param client_data is a pointer to any (optional) client data
144 defined for the callback (none in this case)
145 \param call_data the standard event information provided by X11-Motif

146 void speedSliderCallback(Widget w, XtPointer client_data, XtPointer call_data)

147 {
148 int speedVal;
149 int speedVal;
150 XtyGetValues(w, XmValue, &speedVal, NULL);
151 if(shared.globalDebugMode == 1)
152 printf("New speed slider = %d\n", speedVal);
153 if(speedVal < 15)
154 {
155 shared.speedFactor = 0.25f; /* 1/4 speed */
156 XmTextfieldSetString(speedTextfield, "1/4x");
157 }
158 else if(speedVal > 15 && speedVal < 30)
159 {
160 shared.speedFactor = 0.333333f; /* 1/3 speed */
161 XmTextfieldSetString(speedTextfield, "1/3x");
162 }
163 else if(speedVal > 30 && speedVal < 45)
164 {
165 shared.speedFactor = 0.5f; /* 1/2 speed */
166 XmTextfieldSetString(speedTextfield, "1/2x");
167 }
168 else if(speedVal > 45 && speedVal < 60)
169 {
170 shared.speedFactor = 1.0f; /* normal ix speed */
171 XmTextfieldSetString(speedTextfield, "ix");
172 }
173 else if(speedVal > 60 && speedVal < 75)
174 {
175 shared.speedFactor = 2.0f; /* 2x speed */
176 XmTextfieldSetString(speedTextfield, "2x");
177 }
178 else if(speedVal > 75 && speedVal < 90)
179 {
180 shared.speedFactor = 3.0f; /* 3x speed */
181 XmTextfieldSetString(speedTextfield, "3x");
182 }
183 else
184 {
185 shared.speedFactor = 4.0f; /* 4x speed */
186 XmTextfieldSetString(speedTextfield, "4x");
187 }
188 }

189 }
190 /*
191 \ingroup mod_play
192 This callback is the function that will be called when the Motif user
193 interface puts up a file selection dialog and the user selects a file
194 and presses the 'OK' button. We try to check the validity of the
195 new file name by trying to read how many socket messages it
196 contains. If the file is invalid, the user interface will just
197 beep, as the user tries to 'ok' the selection of an improper file.
198
199 \param w the widget that caused the event and this callback. The push
200 button ('ok') on the file selection dialog in this case.
201 \param client_data is a pointer to any (optional) client data
202 defined for the callback (none in this case)
203 \param call_data the standard event information provided by X11-Motif
204 */
205 void okfileplayialogCallback(Widget w, XtPointer client_data, XtPointer call_data)
206 {
207 char *file;
208 char currentDirectory[XM_NAME_TEXT_SIZE];
209 int result;
210 XmFileSelectionBoxCallbackStruct *fab =
211 (XmFileSelectionBoxCallbackStruct *)call_data;
212 (XmFileSelectionBoxCallbackStruct *)call_data;
213 if(XmStringToString(fab->value, file))
214 {
215 /* copy the filenames selected, but don't copy the directory part */
216 getd(currentDirectory);
217 /* if the first part of the selected file is the same, cut it off */
218 result = strncmp(file, currentDirectory, strlen(currentDirectory));
219 if(result == 0)
220 {
221 /* shared.importFileName[0] = '.'; /* substitute for current dir */
222 shared.importFileName, file[strlen(currentDirectory)];
223 }
224 }
225 /* better copy the whole directory */
226 strcpy(shared.importFileName, file);
227 XmTextfieldSetString(inputTextfield, shared.importFileName);
228 shared.endOfFileNum = readNumberOfMessages(inputTextfield);
229 /* set the end message number to be the max messages in file */
230 if(shared.endOfFileNum == readNumberOfMessages(inputTextfield, shared.endOfFileNum))
231 {
232 XmTextfieldSetString(inputTextfield, shared.endOfFileNum);
233 XmTextfieldSetString(endOfFileField, currentDirectory);
234 if(shared.endOfFileNum == 0)
235 {
236 XmTextfieldSetString(endOfFileField, "100");
237 return;
238 }
239 }
240 else
241 {
242 printf("Invalid Selection\n");
243 /* remove the file selection dialog */
244 XmManagerGetChild(dialogFileSelectionBox);
245 }
246 }/*
247 \ingroup mod_play
248 This callback is the function that will be called when the Motif user
249 interface decides to respond to the user activating one of the 'Pause'
250 push buttons. Since this button can only be active when the program
251 is actually sending, we know that there is a sub-process in existence
252 and that it is running. We want to stop the sub-process, and change
253 the state of active buttons in the interface to allow the user to
254 start it up again.
255 \param w the widget that caused the event and this callback. It
256 could be one of several widgets in this case, and we don't care
257 which one.
258 \param client_data is a pointer to any (optional) client data
259 defined for the callback (none in this case)
260 \param call_data the standard event information provided by X11-Motif
261 */
262 void pauseplaybuttoncallback(Widget w, XtPointer client_data, XtPointer call_data)
263 {
264 }

10/16/02
15:32:21

callbacks-C.C

10/16/02
15:32:21

5

callbacks-C.C

6

```
265 {
266     union signal_sval;
267     /* disable the pause button, enable the restart and resume */
268     XSetSensitive( startPushButton, True );
269     XSetSensitive( pausePushButton, False );
270     XSetSensitive( resumePushButton, True );
271     eval.sival_int = 1;
272     if( siqqueue( shared.childProcess, SIGSTOP, eval ) < 0 )
273     {
274         printf("Pause: cannot queue signal to child\n");
275     }
276 }
277 /**
278 *ingroup mod_play
279 This callback is the function that will be called when the Motif user
280 interface decides to respond to the user activating the 'Resume'
281 push buttons. Since this button can only be active when the program
282 is sending but in a paused state, we know that there is a sub-process
283 in existence and that it is not running. We want to tell the
284 sub-process to continue, and change the state of active buttons in
285 the interface to show this change in the sub-process.
286
287 Vararg w the widget that caused the event and this callback.
288 Vararg client_data is pointer to any (optional) client data
289 defined for this callback (none in this case)
290
291 void
292 resumePlayButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
293 {
294     union signal_sval;
295     /* disable the resume button, reusable the pause */
296     XSetSensitive( startPushButton, False );
297     XSetSensitive( pausePushButton, True );
298     XSetSensitive( resumePushButton, True );
299     /* start sending messages from current position in file */
300     eval.sival_int = 1;
301     if( siqqueue( shared.childProcess, SIGCONT, eval ) < 0 )
302     {
303         printf("Resume: cannot queue signal to child\n");
304     }
305 }
306 /**
307 *ingroup mod_play
308 This callback is the function that will be called when the Motif user
309 interface decides to respond to the user activating the 'Start'
310 push button. This is the most complex of the user-modifiable fields,
311 read the current values for most of the user-modifiable fields,
312 create the socket or output file, re-open the input file, read the
313 start and end message values, and fork a subprocess to send
314 the socket messages. If all this works, then we change the
315 state of the user interface buttons and we are done.
316 Vararg w the widget that caused the event and this callback.
317 Vararg client_data is a pointer to any (optional) client data
318 defined for the callback (none in this case)
319 Vararg call_data the standard event information provided by X11-Motif
320
321 void
322 startButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
323 {
324     FILE *savfileDescrip; /* the output file */
325     char tmpbuf[MAX_STRING_TEXT_SIZE];
326     int count;
327
328     /* read all the fields on the user interface to make sure we have
329       the most up-to-date value for each option
330   */

331     /* read the current end number before it is overwritten */
332     strcpy(tmpbuf, XtextFieldGetString( endMessageTextField ));
333     if( sget( tmpbuf, "id", &shared.endMessageId ) != 1 )
334     {
335         XtDisplay(w), 100 );
336         return;
337     }
338
339     /* get the input file name */
340     strcpy(shared.inputFileName, XtextFieldGetString( inputFileTextField );
341     count = readnumbMessages( shared.inputFileName );
342     if( count == 0 )
343     {
344         XBell( XtDisplay(w), 100 );
345         return;
346     }
347     /* reset back to the the user selected end number for the (new) file */
348     strcpy( tmpbuf, "id", shared.endMessageId );
349     XtextFieldSetString( endMessageTextField, tmpbuf );
350
351     strcpy(tmpbuf, XtextFieldGetString( startMessageTextField ),;
352     if( sget( tmpbuf, "id", &shared.startMessageId ) != 1 )
353     {
354         XBell( XtDisplay(w), 100 );
355         return;
356     }
357     if( shared.startMessageId < 1 )
358     {
359         XBell( XtDisplay(w), 100 );
360         return;
361     }
362     if( shared.endMessageId <= shared.startMessageId )
363     {
364         XBell( XtDisplay(w), 100 );
365         return;
366     }
367     /* the destination address for the socket (or file name ) */
368     strcpy(shared.address, XtextFieldGetString( destinationTextField ) );
369
370     /* the destination port for the socket */
371     strcpy(tmpbuf, XtextFieldGetString( portTextField ) );
372     if( sget( tmpbuf, "id", &shared.portnum ) != 1 )
373     {
374         XBell( XtDisplay(w), 100 );
375         return;
376     }
377
378     savfileDescrip = NULL;
379     /* if we have the normal, address destination, then create socket */
380     if( XmoggleButtonGetState( networkToggleButton ) == TRUE )
381     {
382         shared.sockfd = createSendSocket( shared.address, shared.portnum );
383         if( shared.sockfd < 0 )
384         {
385             XBell( XtDisplay(w), 100 );
386             return;
387         }
388     }
389
390 }
391
392     /* we will save the output in the destination file, try to open it */
393     {
394         savfileDescrip = fopen( shared.address, "w" );
395         if( savfileDescrip == NULL )
396         {

397     }
```

10/16/02
15:32:21

callbacks-c.c

7

```
397     XBell( XtDisplay(w), 100 );
398
399     return;
400   /* we can write to the file, send the information */
401
402   /* start sending messages from start of file */
403   /* (re)set the frame counter */
404   if( shared.childNumber == 1 )
405     shared.messageNumber = 1;
406   if( shared.childProcess != 0 )
407     /* kill the current (suspended) child process */
408     kill( shared.childProcess, SIGTERM );
409
410   /* we need to know when the sending process is finished */
411   signal( SIGCHLD, endProcessDone );
412   /* fork a subprocess to run the long send procedure */
413   switch( fork( shared.childProcess == FORK ) )
414   {
415     case 0: /* child process */
416       /* we want the child process to respond to a pause/resume control */
417       signal( SIGPOLL, sendProcessPause );
418       sendMessage( shared.sendFileDescrip, savefileDescrip );
419       /* when the send process is done, exit with signal set */
420       exit(255);
421
422     XBell( XtDisplay(w), 100 );
423     print("fork() of send messages failed\n");
424   }
425
426   /* the main (notif) program continues. */
427   /* set prohibition state - disable the start button, enable the pause */
428   XSetItem( startPubButton, False );
429   XSetItem( pausePubButton, True );
430   XSetItem( resumePubButton, False );
431
432   /* if we have a file destination, then close file */
433   if( networkPubButtonGetFile( networkPubButton ) == TRUE )
434     ifclose( savefileDescrip );
435   else
436     /* close the socket */
437     close( shared.socketHandle );
438
439 }
440
```

10/16/02 15:32:29

1

creation-c.c

```
1  /*  
2   * REAMER: Portions of this file are merged at file generation  
3   * time. Warts can be made "only" in between specified code blocks, look  
4   * for keywords <Begin user code> and <End user code>  
5   */  
6  /*  
7   * Generated by the ICS Builder accessory (ix).  
8   *  
9   * Builder Accessory Version 5.0  
10  * Code Generator Accessory 5.0 (05/22/98)  
11  */  
12  /*  
13  */  
14  /* Begin user code block <file_comments> */  
15  /*  
16  */  
17  \file play/creation-c.c  
18  \ingroup mod-play  
19  */  
20  In this source file is the code generated automatically  
21  by the user interface builder software. It will use this source  
22  again, preserving user supplied code between the special brackets,  
23  when the software is used to modify the interface.  
24  (code)  
25  Copyright (c) 2002  
26  All content of this file is copyrighted to the  
27  National Research Council of Canada.  
28  All rights reserved.  
29  \endcode  
30  \author Daniel F. Johnston  
31  \since July 2002  
32  $Id: creation-c.c,v 1.9 2002/10/16 18:20:27 johnston Exp $  
33  */  
34  /* End user code block <file_comments> */  
35  */  
36  */  
37  \include <cm/cm.h>  
38  \include <cm/mainW.h>  
39  \include <cm/dialog.h>  
40  \include <cm/menut.h>  
41  \include <cm/multisec.h>  
42  \include <cm/scrollbar.h>  
43  \include <cm/label.h>  
44  \include <cm/pubh.h>  
45  \include <cm/filesh.h>  
46  \include <cm/bottomum.h>  
47  \include <cm/cascade.h>  
48  \include <cm/separat.h>  
49  \include <cm/toggle.h>  
50  \include <cm/form.h>  
51  \include <cm/testf.h>  
52  \include <cm/scrollbar.h>  
53  */  
54  /*  
55  * Global declarations are now stored in the header file.  
56  * If DECLARE_EX_GLOBALS is defined then this header file  
57  * declares the globals, otherwise it just exports them.  
58  */  
59  */  
60  #define DECLARE_EX_GLOBALS  
61  */  
62  /* Globally included information.  
63  */  
64  */  
65  */  
66  */
```

10/16/02
15:32:29

2

creation-c.c

```
67  /* Common constant and pixmap declarations.  
68  */  
69  #include "creation-c.h"  
70  /*  
71  * Convenience functions from utilities file.  
72  */  
73  extern void RegisterXConverters(XtAppContext);  
74  /*  
75  */  
76  extern XtPointer X_CONVERTER(XtWidget, char *, int, Boolean);  
77  extern XtPointer X_DOUBLE(double);  
78  extern XtPointer X_STRING( float );  
79  extern void X_MENU_POPS(XtWidget, XtPointer, XEvent *, Boolean);  
80  extern Pixmap XWM_PIXMAP(XtWidget, char **);  
81  extern void X_SET_BACKGROUND_COLOR(XtWidget, Arglist, Cardinal *, Pixel);  
82  /*  
83  */  
84  /* Declarations for callbacks and handlers.  
85  */  
86  #ifndef DDXCORE SHOULD IGNORE THIS  
87  extern void aboutAboutMiscCallback(XtWidget, XtPointer, XtPointer);  
88  extern void offFilePlayCallback(XtWidget, XtPointer, XtPointer, XtPointer);  
89  extern void cancelFileSaveCallback(XtWidget, XtPointer, XtPointer);  
90  extern void aboutAboutMisc(XtWidget, XtPointer, XtPointer);  
91  extern void startAutomatic(XtWidget, XtPointer, XtPointer);  
92  extern void pauseAutomatic(XtWidget, XtPointer, XtPointer);  
93  extern void resumeAutomatic(XtWidget, XtPointer, XtPointer);  
94  extern void quitButtonOnCallback(XtWidget, XtPointer, XtPointer);  
95  extern void networkIconCallback(XtWidget, XtPointer, XtPointer);  
96  extern void debTogglePeopleCallback(XtWidget, XtPointer, XtPointer);  
97  extern void usageHelpCallback(XtWidget, XtPointer, XtPointer);  
98  extern void aboutHelpCallback(XtWidget, XtPointer, XtPointer);  
99  extern void aboutHelpAbout(XtWidget, XtPointer, XtPointer);  
100  extern void speedSliderCallback(XtWidget, XtPointer, XtPointer);  
101  */  
102  /*  
103  */  
104  /* Create the mainWindow hierarchy of widgets.  
105  */  
106  Widget CreateMainWindow(Widget parent)  
107  {  
108    Cardinal ac = 0;  
109    Arg args[256];  
110    Cardinal cdc = 0;  
111    Boolean arg0 = False;  
112    mainWindow;  
113    Widget aboutHelpDialogShell;  
114    Widget aboutHelpLabel;  
115    Widget aboutAboutLabel;  
116    Widget selectFileDialog;  
117    Widget membBar;  
118    Widget fileMenuBar;  
119    Widget pullDownMenu;  
120    Widget openFilePushButton;  
121    Widget separator;  
122    Widget startFilePushButton;  
123    Widget pauseFilePushButton;  
124    Widget resumeFilePushButton;  
125    Widget separator;  
126    Widget quitFilePushButton;  
127    Widget optionCascadeButton;  
128    Widget pullDownMenu1;  
129    Widget helpCascadeButton;  
130    Widget pullDownMenu2;  
131    Widget usageHelpButton;  
132    Widget
```

10/16/02
15:32:29

creation-c.c

3
15:32:29

```
133     Widget
134     widget
135     Widget
136     Widget
137     Widget
138     Widget
139     Widget
140     Widget
141     Widget
142     Widget
143     Widget
144     Widget
145     Widget
146     Widget
147
148     /* Register the converters for the widgets.
149 */
150     RegisterAndConverters(widgetClass, mainWindow);
151     XInitializeWidgetClass(widgetClass);
152     XInitializeWidgetClass(widgetClass);
153     XInitializeWidgetClass(widgetClass);
154     XInitializeWidgetClass(widgetClass);
155     XInitializeWidgetClass(widgetClass);
156     XInitializeWidgetClass(widgetClass);
157     XInitializeWidgetClass(widgetClass);
158     XInitializeWidgetClass(widgetClass);
159     XInitializeWidgetClass(widgetClass);
160     XInitializeWidgetClass(widgetClass);
161     XInitializeWidgetClass(widgetClass);
162     XInitializeWidgetClass(widgetClass);
163     XInitializeWidgetClass(widgetClass);
164     XInitializeWidgetClass(widgetClass);
165     XInitializeWidgetClass(widgetClass);
166     XInitializeWidgetClass(widgetClass);
167     ac = 0;
168     XSetCard(args[ac], XmWidth, 612); ac++;
169     XSetCard(args[ac], XmHeight, 414); ac++;
170     XSetCard(args[ac], XmWidth, 619); ac++;
171     XSetCard(args[ac], XmHeight, 385); ac++;
172     XSetCard(args[ac], XmWidth, 385); ac++;
173     mainWindow = XmCreateMainWindow(parent,
174                                     "mainWindow",
175                                     args,
176                                     ac);
177
178     ac = 0;
179     XSetCard(args[ac], XmWidth, 619); ac++;
180     XSetCard(args[ac], XmHeight, 28); ac++;
181     menuBar = XmCreateMenuBar(mainWindow,
182                               "menuBar",
183                               args,
184                               ac);
185     XtManageChild(menuBar);
186
187     ac = 0;
188     XSetCard(args[ac], XmWidth, 21); ac++;
189     XSetCard(args[ac], XmLabelString,
190              "CONFIRM", 0, args); if (args) ac++;
191     XmFormString(args[ac], XmWidth, 43); ac++;
192     XSetCard(args[ac], XmWidth, 5); ac++;
193     XSetCard(args[ac], XmWidth, 21); ac++;
194     XSetCard(args[ac], XmWidth, 24); ac++;
195     fileCascadeButton = XmCreateCascadeButton(menuBar,
196                                              "fileCascadeButton",
197                                              "fileCascadeButton");
198
199     XmManageChild(fileCascadeButton);
200     ac = 0;
201     XSetCard(args[ac], XmWidth, 0); ac++;
202     XSetCard(args[ac], XmWidth, 0); ac++;
203     XSetCard(args[ac], XmWidth, 74); ac++;
204     XSetCard(args[ac], XmWidth, 130); ac++;
205     XmFormString(args[ac], XmWidth, 0); ac++;
206     pullDownMenu = XmCreatePullDownMenu(XtParent(fileCascadeButton),
207                                         "pullDownMenu",
208                                         args,
209                                         ac);
210
211     ac = 0;
212     XSetCard(args[ac], XmWidth, 0); ac++;
213     XSetCard(args[ac], XmLabelString,
214             "BROWSE", 0, args); if (args) ac++;
215     XmFormString(args[ac], XmWidth, "Open..."); ac++;
216     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
217     openFilePushButton = XmCreatePushButton(pullDownMenu,
218                                             "openFilePushButton",
219                                             args,
220                                             ac);
221     XmManageChild(openFilePushButton);
222     XtAddCallback(openFilePushButton, XmactivateCallback, browseButtonCallback, (XtPointer)
223 er);
223     ac = 0;
224     XmCreateSeparator(pullDownMenu,
225                       "separator1",
226                       args,
227                       ac);
228     XmManageChild(separator1);
229
230     separator1 = XmCreateSeparator(pullDownMenu,
231                                   "separator1",
232                                   args,
233                                   ac);
234     XSetCard(args[ac], XmWidth, 0); ac++;
235     XmFormString(args[ac], XmWidth, "Start",
236                  0, args); if (args) ac++;
237     startFilePushButton = XmCreatePushButton(pullDownMenu,
238                                             "startFilePushButton",
239                                             args,
240                                             ac);
241     XmManageChild(startFilePushButton);
242     XtAddCallback(startFilePushButton, XmactivateCallback, startButtonCallback, startButtonCallback,
243                 (XtPointer)
244 er);
245     ac = 0;
246     XSetCard(args[ac], XmWidth, 0); ac++;
247     XmFormString(args[ac], XmWidth, "PAUSE"); ac++;
248     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
249     pauseFilePushButton = XmCreatePushButton(pullDownMenu,
250                                         "pauseFilePushButton",
251                                         args,
252                                         ac);
253     XmManageChild(pauseFilePushButton);
254     XtAddCallback(pauseFilePushButton, XmactivateCallback, pauseButtonCallback, pauseButtonCallback,
255                 (XtPointer)
256 er);
257     XSetCard(args[ac], XmWidth, 0); ac++;
258     XmFormString(args[ac], XmWidth, "RESUME"); ac++;
259     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
260     resumeFilePushButton = XmCreatePushButton(pullDownMenu,
261                                             "resumeFilePushButton",
262                                             args,
```

creation-c.C

4

10/16/02
15:32:29

```
199     XmManageChild(fileCascadeButton);
200
201     ac = 0;
202     XSetCard(args[ac], XmWidth, 0); ac++;
203     XSetCard(args[ac], XmWidth, 0); ac++;
204     XSetCard(args[ac], XmWidth, 74); ac++;
205     XSetCard(args[ac], XmWidth, 130); ac++;
206     XmFormString(args[ac], XmWidth, 0); ac++;
207     pullDownMenu = XmCreatePullDownMenu(XtParent(fileCascadeButton),
208                                         "pullDownMenu",
209                                         args,
210                                         ac);
211
212     ac = 0;
213     XSetCard(args[ac], XmWidth, 0); ac++;
214     XSetCard(args[ac], XmLabelString,
215             "BROWSE", 0, args); if (args) ac++;
216     XmFormString(args[ac], XmWidth, "Open..."); ac++;
217     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
218     openFilePushButton = XmCreatePushButton(pullDownMenu,
219                                             "openFilePushButton",
220                                             args,
221                                             ac);
222     XmManageChild(openFilePushButton);
223     XtAddCallback(openFilePushButton, XmactivateCallback, browseButtonCallback, browseButtonCallback,
224                 (XtPointer)
225 er);
226     ac = 0;
227     XmCreateSeparator(pullDownMenu,
228                       "separator1",
229                       args,
230                       ac);
231     XmManageChild(separator1);
232
233     separator1 = XmCreateSeparator(pullDownMenu,
234                                   "separator1",
235                                   args,
236                                   ac);
237     XSetCard(args[ac], XmWidth, 0); ac++;
238     XmFormString(args[ac], XmWidth, "Start",
239                  0, args); if (args) ac++;
240     startFilePushButton = XmCreatePushButton(pullDownMenu,
241                                             "startFilePushButton",
242                                             args,
243                                             ac);
244     XmManageChild(startFilePushButton);
245     XtAddCallback(startFilePushButton, XmactivateCallback, startButtonCallback, startButtonCallback,
246                 (XtPointer)
247 er);
248     ac = 0;
249     XSetCard(args[ac], XmWidth, 0); ac++;
250     XmFormString(args[ac], XmWidth, "PAUSE"); ac++;
251     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
252     pauseFilePushButton = XmCreatePushButton(pullDownMenu,
253                                         "pauseFilePushButton",
254                                         args,
255                                         ac);
256     XSetCard(args[ac], XmWidth, 0); ac++;
257     XmFormString(args[ac], XmWidth, "RESUME"); ac++;
258     XmCreatePushButton(0, "xrclick", 0, args); if (args) ac++;
259     resumeFilePushButton = XmCreatePushButton(pullDownMenu,
260                                             "resumeFilePushButton",
261                                             args,
```

10/16/02
15:32:29

creation-c.c

5

10/16/02
15:32:29

creation-c.c

6

```
args,
    ac);
XtManageChild(resumeFilePushButton);
XtAddCallback(resumeFilePushButton, XmactivateCallback, resumePlayButtonCallback, (XtPointer)0),
    args,
    ac);
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325

args,
    ac);
XtManageChild(resumeFilePushButton);
XtAddCallback(resumeFilePushButton, XmactivateCallback, resumePlayButtonCallback, (XtPointer)0),
    args,
    ac);
XtManageChild(separatord);
    "separator2",
    args,
    ac);
XtManageChild(separatord2);
    args,
    ac);
ac = 0;
separatord2 = XmcreateSeparator(pulldownMenu,
    "separator2",
    args,
    ac);
XtSetArg(args[ac], XmNemonic, 'Q'); ac++;
    BX_CONVERT(pulldownMenu, XmLabelString,
    XmString, 0, XmGetString, 0, XmGetRString); if (argok) ac++;
    XmCreateToggleButton, 0, XmGetRString); if (argok) ac++;
    fileOpToggleButton = XmcreateToggleButton(pulldownMenu,
    "fileOpToggleButton",
    args,
    ac);
XtManageChild(fileOpToggleButton);
    args,
    ac);
XtAddCallback(fileOpToggleButton, XmvalueChangedCallback, fileToggleCallback, (XtPointer)
    int);
    ac = 0;
XtSetArg(args[ac], XmLabelString,
    BX_CONVERT(pulldownMenu, "debug mode",
    XmString, 0, XmGetString, 0, XmGetRString); if (argok) ac++;
    XmCreateToggleButton, 0, XmGetRString); if (argok) ac++;
    debugOpButton = XmcreateToggleButton(pulldownMenu,
    "debugOpButton",
    args,
    ac);
XtManageChild(debugOpButton);
    args,
    ac);
XtAddCallback(debugOpButton, XmvalueChangedCallback, debugToggleCallback, (XtPointer)
    int);
    ac = 0;
XtSetArg(args[ac], XmLabelString,
    BX_CONVERT(pulldownMenu, pulldownMenu), ac++);
XtSetValues(optionsCascadeButton, args, ac);
    ac = 0;
XtSetArg(args[ac], XmLabelString,
    XmNsubMenuItem, pulldownMenu); ac++;
    ac++;
```

```
316
317
318
319
320
321
322
323
324
325

args,
    ac);
XtManageChild(optionFilePushButton);
    args,
    ac);
XtAddCallback(optionFilePushButton, XmactivateCallback, quitButtonCallback, (XtPointer)
    int);
    ac = 0;
XtSetArg(args[ac], XmLabelString,
    XmNsubMenuItem, pulldownMenu); ac++;
    ac++;
```

10/16/02
15:32:29

creation-c.c

10/16/02
15:32:29

creation-c.c

8

```

389 XtManageChild(usageHelpPushButton); XmactivateCallback, usageHelpCallback, (XtPointer
390 XtAddCallback(usageHelpPushButton, XmactivateCallback, usageHelpCallback, (XtPointer
391 "playbackUsageHelp.html"));
392 ac = 0;
393 XtSetArg(args[ac], XmNameString, 'A'); ac++;
394 XtSetArg(args[ac], XmLabelString, XmLabelString);
395 Xm_CONVERT((pulldownMenu1, "About", XmLabelString, 0, (argok)); if ((argok) ac++);
396 Xm_CONVERT((pulldownMenu1, "AboutHelpPushButton", 0, (argok)); ac++);
397 aboutHelpPushButton = XmCreatePushButton(pulldownMenu1,
398 "AboutHelpPushButton",
399 args,
400 ac);
401 XtManageChild(aboutHelpPushButton), XmactivateCallback, aboutHelpCallback,
402 XtAddCallback(aboutHelpPushButton, XmactivateCallback, aboutHelpCallback, (XtPointer
403 ac));
404 ac = 0;
405 XtSetArg(args[ac], XmSubMenuItem, pulldownMenu2); ac++;
406 Xm_SETValues(helpCascadeButton, args, ac);
407 ac = 0;
408 ac = 0;
409 XtSetArg(args[ac], XmPolicyPolicy, XmPERIME_GROW); ac++;
410 XtSetArg(args[ac], XmLabelString, XmLabelString);
411 XtSetArg(args[ac], XmFontList, XmFontList);
412 XtSetArg(args[ac], XmHeight, 357); ac++;
413 form = XmCreateForm(mainWindow,
414 "Form",
415 args,
416 ac);
417 XtManageChild(form);
418 ac = 0;
419 XtSetArg(args[ac], XmLabelString, XmLabelString_CENTER); ac++;
420 XtSetArg(args[ac], XmFontList, XmFontList);
421 XtSetArg(args[ac], XmFontList, XmFontList);
422 Xm_CONVERT((form, "--bold-i--e--20--e--e--iso0859-1",
423 XmFontList, 0, (argok)); if ((argok) ac++);
424 XtSetArg(args[ac], XmForeground, XmForeground);
425 XmLabel, 0, (argok); ac++;
426 XtSetArg(args[ac], XmLabelString, XmLabelString);
427 Xm_CONVERT((form, "PlayBack UDP Sockets",
428 XmFontString, 0, (argok)); if ((argok) ac++);
429 XtSetArg(args[ac], XmLabel, XmLabel);
430 XtSetArg(args[ac], XmFontList, XmFontList);
431 XtSetArg(args[ac], XmHeight, 100); ac++;
432 XtSetArg(args[ac], XmWidth, 300); ac++;
433 XtSetArg(args[ac], XmHeight, 239); ac++;
434 label = XmCreateLabel(form,
435 "Label",
436 args,
437 ac);
438 XtManageChild(label);
439 ac = 0;
440 XtSetArg(args[ac], XmEditTable, True); ac++;
441 XtSetArg(args[ac], XmLabel, XmLabel);
442 XtSetArg(args[ac], XmFontList, XmFontList);
443 XtSetArg(args[ac], XmFontList, XmFontList);
444 XtSetArg(args[ac], XmWidth, 120); ac++;
445 XtSetArg(args[ac], XmHeight, 30); ac++;
446 endNumField = XmCreateTextField(form,
447 "endNumField",
448 args,
449 ac);
450 XmManageChild(endNumField);
451 ac = 0;
452
)

```

```

453 XtSetArg(args[ac], XmFontList,
454 Xm_CONVERT((form, "--times-medium-r--e--iso0859-1",
455 XmFontList, 0, (argok)); if ((argok) ac++);
456 XmSetArg(args[ac], XmLabelString,
457 Xm_CONVERT((form, "End Num.", XmLabelString),
458 XmFontString, 0, (argok)); ac++);
459 XtSetArg(args[ac], XmLabel, XmLabel);
460 Xm_CONVERT((form, "End Num.", 2000, 330); ac++);
461 XtSetArg(args[ac], XmFontList, XmFontList);
462 XtSetArg(args[ac], XmHeight, 19); ac++;
463 endNumLabel = XmCreateLabel(form,
464 "endNumLabel",
465 args,
466 ac);
467 XtManageChild(endNumLabel);
468 ac = 0;
469 XmSetArg(args[ac], XmValue, "1"); ac++;
470 XmSetArg(args[ac], XmEditable, True); ac++;
471 XmSetArg(args[ac], XmLabel, XmLabel);
472 XmSetArg(args[ac], XmFontList, XmFontList);
473 XmSetArg(args[ac], XmLabel, XmLabel);
474 XmSetArg(args[ac], XmFontList, XmFontList);
475 XmSetArg(args[ac], XmHeight, 30); ac++;
476 startNumField = XmCreateTextField(form,
477 "startNumField",
478 args,
479 XmManageChild(startNumField);
480 ac = 0;
481 XmSetArg(args[ac], XmFontList,
482 Xm_CONVERT((form, "--times-medium-r--e--iso0859-1",
483 XmFontList, 0, (argok)); if ((argok) ac++);
484 XmSetArg(args[ac], XmLabelString,
485 Xm_CONVERT((form, "Start Num.", XmLabelString),
486 XmFontString, 0, (argok)); if ((argok) ac++);
487 XmSetArg(args[ac], XmLabel, XmLabel);
488 XmSetArg(args[ac], XmFontList, XmFontList);
489 XmSetArg(args[ac], XmHeight, 74); ac++;
490 XmSetArg(args[ac], XmWidth, 200); ac++;
491 XmSetArg(args[ac], XmLabel, XmLabel);
492 XmSetArg(args[ac], XmFontList, XmFontList);
493 XmSetArg(args[ac], XmHeight, 93); ac++;
494 startNumLabel = XmCreateLabel(form,
495 "startNumLabel",
496 args,
497 XmManageChild(startNumLabel);
498 ac = 0;
499 XmSetArg(args[ac], XmFontList,
500 Xm_CONVERT((form, "--times-medium-r--e--iso0859-1",
501 XmFontList, 0, (argok)); if ((argok) ac++);
502 XmSetArg(args[ac], XmLabelString,
503 XmSetArg(args[ac], XmLabel, XmLabel);
504 XmSetArg(args[ac], XmFontList, XmFontList);
505 speedMField = XmCreateTextField(form,
506 "speedMField",
507 args,
508 ac);
509 XmManageChild(speedMField);
510 ac = 0;
511 XmSetArg(args[ac], XmFontList,
512 Xm_CONVERT((form, "--times-medium-r--e--iso0859-1",
513 XmFontList, 0, (argok)); if ((argok) ac++);
514 XmSetArg(args[ac], XmLabelString,
515 Xm_CONVERT((form, "Paser",
516 XmFontString, 0, (argok)); if ((argok) ac++);
517 XmManageLabel(args[ac], XmLabel);
518
)

```

10/16/02
15:32:29

creation-c.c

10/16/02
15:32:29

9

creation-c.c

```
519 XtSetArg(args[ac], XmNlabelString, "Speed"); if (argok) ac++;
520 XtSetArg(args[ac], XmNheight, 40); ac++;
521 XtSetArg(args[ac], XmNheight, 19); ac++;
522 fastervlabel = XmCreateLabel(form,
523 "fastervlabel",
524 args,
525 ac);
526 XmManageChild(fastervlabel);
527 ac = 0;
528 args[0];
529 XmFontList, XmFontList,
530 BX_CONVERT(form, "+-times-medium-r---+--160---+---iso8859-1",
531 XmFontList, 0, sargok); if (argok) ac++;
532 XtSetArg(args[ac], XmLabelString,
533 BX_CONVERT(form, "Slow",
534 XmFontList, 0, sargok); if (argok) ac++;
535 XtSetArg(args[ac], XmNlabelString, "Slow");
536 XtSetArg(args[ac], XmNheight, 20); ac++;
537 XtSetArg(args[ac], XmNwidth, 45); ac++;
538 XtSetArg(args[ac], XmNheight, 20); ac++;
539 alowerlabel = XmCreateLabel(form,
540 "alowerlabel",
541 args,
542 ac);
543 XmManageChild(alowerlabel);
544 ac = 0;
545 XtSetArg(args[ac], XmFontList,
546 XmFontList, 0, sargok); if (argok) ac++;
547 BX_CONVERT(form, "+-times-medium-r---+--160---+---iso8859-1",
548 XmFontList, 0, sargok); if (argok) ac++;
549 XtSetArg(args[ac], XmLabelString,
550 BX_CONVERT(form, "Speed",
551 XmFontString, 0, sargok); if (argok) ac++;
552 XtSetArg(args[ac], XmNheight, 180); ac++;
553 XtSetArg(args[ac], XmNlabelString, "Up");
554 XtSetArg(args[ac], XmNwidth, 40); ac++;
555 XtSetArg(args[ac], XmNheight, 19); ac++;
556 speedlabel = XmCreateLabel(form,
557 "speedlabel",
558 ac);
559 XmManageChild(speedlabel);
560 ac = 0;
561 XtSetArg(args[ac], XmNlabelString, "Indirect");
562 XtSetArg(args[ac], XmNheight, 45); ac++;
563 XtSetArg(args[ac], XmNlabelString, "Horizontal");
564 XtSetArg(args[ac], XmNheight, 100); ac++;
565 XtSetArg(args[ac], XmNlabelString, "Vertical");
566 XtSetArg(args[ac], XmNlabelString, "Horizontal");
567 XtSetArg(args[ac], XmNheight, 20); ac++;
568 XtSetArg(args[ac], XmNlabelString, "Vertical");
569 XtSetArg(args[ac], XmNwidth, 360); ac++;
570 XtSetArg(args[ac], XmNheight, 30); ac++;
571 speedscrollbar = XmCreateScrollbar(form,
572 "speedscrollbar",
573 args,
574 ac);
575 XmManageChild(speedscrollbar);
576 XmAddCallback(speedscrollbar, XmValueChangedCallback, speedSliderCallback, (XtPointer)
      x);
577 ac = 0;
578 XtSetArg(args[ac], XmFontList,
579 BX_CONVERT(form, "+-times-medium-r---+--160---+---iso8859-1",
580 XmFontList, 0, sargok); if (argok) ac++;
581 XtSetArg(args[ac], XmLabelString,
582 XmCreatePushButton, "startPushbutton",
583 BX_CONVERT(form, "Quit",
584 XmFontString, 0, sargok); if (argok) ac++;
585 XtSetArg(args[ac], XmNlabelString, "End");
586 XtSetArg(args[ac], XmNheight, 300); ac++;
587 XtSetArg(args[ac], XmNwidth, 50); ac++;
588 XtSetArg(args[ac], XmNlabelString, "End");
589 XmCreatePushButton, "quitPushbutton",
590 args,
591 ac);
592 ac = 0;
593 XtManageChild(quitPushbutton);
594 XtAddCallback(quitPushbutton, XmActivateCallback, quitButtonCallback, (XtPointer)0);
595 ac = 0;
596 XtSetArg(args[ac], XmFontList,
597 BX_CONVERT(form, "+-times-medium-r---+--160---+---iso8859-1",
598 XmFontList, 0, sargok); if (argok) ac++;
599 XmLabelString, 0, sargok); if (argok) ac++;
600 XtConvert(form, "Pause",
601 XmLabelString, 0, sargok); if (argok) ac++;
602 XtSetArg(args[ac], XmNlabelString, "Pause");
603 XtSetArg(args[ac], XmNheight, 320); ac++;
604 XtSetArg(args[ac], XmNlabelString, "Resume");
605 XtSetArg(args[ac], XmNwidth, 140); ac++;
606 XtSetArg(args[ac], XmNlabelString, "Resume");
607 XmCreatePushButton, "resumePushbutton",
608 args,
609 ac);
610 XtManageChild(resumePushbutton);
611 XtAddCallback(resumePushbutton, XmActivateCallback, resumePlayButtonCallback, (XtPointer)
      timer);
612 ac = 0;
613 XtSetArg(args[ac], XmFontList,
614 XmFontList, 0, sargok); if (argok) ac++;
615 XmLabelString, 0, sargok); if (argok) ac++;
616 XmLabelString, 0, sargok); if (argok) ac++;
617 XmLabelString, 0, sargok); if (argok) ac++;
618 XtConvert(form, "Pause",
619 XmLabelString, 0, sargok); if (argok) ac++;
620 XtSetArg(args[ac], XmNlabelString, "Pause");
621 XtSetArg(args[ac], XmNheight, 170); ac++;
622 XtSetArg(args[ac], XmNwidth, 140); ac++;
623 XtSetArg(args[ac], XmNlabelString, "Resume");
624 XtSetArg(args[ac], XmNheight, 35); ac++;
625 XmCreatePushButton, "PausePushbutton",
626 "PausePushbutton",
627 args;
628 ac = 0;
629 XtManageChild(pausePushbutton);
630 XtAddCallback(pausePushbutton, XmActivateCallback, pausePlayButtonCallback, (XtPointer)
      timer);
631 ac = 0;
632 XtSetArg(args[ac], XmFontList,
633 XmFontList, 0, sargok); if (argok) ac++;
634 XmLabelString, 0, sargok); if (argok) ac++;
635 XmLabelString, 0, sargok); if (argok) ac++;
636 XtConvert(form, "Start/Restart",
637 XmLabelString, 0, sargok); if (argok) ac++;
638 XtSetArg(args[ac], XmNlabelString, "Start");
639 XtSetArg(args[ac], XmNheight, 20); ac++;
640 XtSetArg(args[ac], XmNwidth, 140); ac++;
641 XtSetArg(args[ac], XmNlabelString, "Start");
642 XtSetArg(args[ac], XmNheight, 106); ac++;
643 XmCreatePushButton, "startPushbutton",
644 "startPushbutton",
645 args,
646 ac);
```

10

creation-c.c

10/16/02
15:32:29

10/16/02
15:32:29

11

creation-C.C

Creation-C.C

12

```
647 xtManageChild(startPushButton), xmCreatePushButton, startButtonCallback, (xtPointer)0
648 xtAddCallback(startPushButton, xmActivateCallback, startButtonCallback, (xtPointer)0
),
649 ac = 0;
650 xSetLabel(args[ac], Xnum, 310); ac++;
651 xSetLabel(args[ac], Xnum, 90); ac++;
652 xSetLabel(args[ac], Xnum, 120); ac++;
653 xSetLabel(args[ac], XnumWidth, 118); ac++;
654 xSetLabel(args[ac], XnumHeight, 27); ac++;
655 portLabel = xmCreateLabel(form,
656 "portLabel",
657 args,
658 ac);
659 xManageChild(portTextField);
660 ac = 0;
661 xSetLabel(args[ac], XmFontList,
662 XmConvex((form, "-*-times-medium-r-*-*-140-*-*-*-iso8859-1",
663 XmFontList, 0, sargok)); if (argok) ac++;
664 xSetLabel(args[ac], XmLabelString,
665 XmConvex((form, "Port Num",
666 XmFontString, 0, sargok)); if (argok) ac++;
667 xSetLabel(args[ac], Xnum, 200); ac++;
668 xSetLabel(args[ac], Xnum, 50); ac++;
669 xSetLabel(args[ac], Xnum, 60); ac++;
670 xSetLabel(args[ac], XnumWidth, 149); ac++;
671 xSetLabel(args[ac], XnumHeight, 19); ac++;
672 portLabel = xmCreateLabel(form,
673 "portLabel",
674 args,
675 ac);
676 xManageChild(portLabel);
677
678 ac = 0;
679 xSetLabel(args[ac], Xnum, 310); ac++;
680 xSetLabel(args[ac], Xnum, 60); ac++;
681 xSetLabel(args[ac], XnumWidth, 270); ac++;
682 xSetLabel(args[ac], XmLabelString,
683 destinationTextField = xmCreateTextField(form,
684 "destinationTextField",
685 args,
686 ac);
687 xManageChild(destinationTextField);
688 ac = 0;
689 xSetLabel(args[ac], XmFontList,
690 XmConvex((form, "-*-times-medium-r-*-*-140-*-*-*-iso8859-1",
691 XmFontList, 0, sargok)); if (argok) ac++;
692 xSetLabel(args[ac], XmLabelString,
693 XmConvex((form, "Destination",
694 XmFontString, 0, sargok)); if (argok) ac++;
695 xSetLabel(args[ac], Xnum, 200); ac++;
696 xSetLabel(args[ac], Xnum, 60); ac++;
697 xSetLabel(args[ac], XnumWidth, 59); ac++;
698 xSetLabel(args[ac], Xnum, 19); ac++;
699 xSetLabel(args[ac], XmLabelString,
700 destinationLabel = xmCreateLabel(form,
701 "destinationLabel",
702 ac);
703
704 xManageChild(destinationLabel);
705
706 ac = 0;
707 xSetLabel(args[ac], XmFontList,
708 xSetLabel(args[ac], XmFontList,
709 XmConvex((form, "-*-times-medium-r-*-*-140-*-*-*-iso8859-1",
710 XmFontList, 0, sargok)); if (argok) ac++;
711 xSetLabel(args[ac], XmLabelString,
```

```
712 XmConvex((form, "File Destination",
713 XmFontList, 0, sargok)); if (argok) ac++;
714 xSetLabel(args[ac], Xnum, 20); ac++;
715 xSetLabel(args[ac], Xnum, 90); ac++;
716 xSetLabel(args[ac], XnumWidth, 118); ac++;
717 xSetLabel(args[ac], XmHeight, 27); ac++;
718 filePopUpButton = xmCreatePopUpButton(form,
719 "filePopUpButton",
720 args,
721 ac);
722 xManageChild(filePopUpButton, XmValueChangeCallback, filePopUpButton,
723 xtAddCallback(filePopUpButton, XmValueChangeCallback, filePopUpButton, (xtPoint
724 ex=0));
725 ac = 0;
726 xSetLabel(args[ac], XmFontList,
727 XmConvex((args[ac], XmFontList,
728 XmFontList, 0, sargok)); if (argok) ac++);
729 xSetLabel(args[ac], XmLabelString,
730 XmConvex((form, "Network Destination",
731 XmFontList, 0, sargok)); if (argok) ac++;
732 xSetLabel(args[ac], XmFontList, 0, sargok)); if (argok) ac++;
733 xSetLabel(args[ac], Xnum, 20); ac++;
734 xSetLabel(args[ac], Xnum, 60); ac++;
735 xSetLabel(args[ac], XnumWidth, 149); ac++;
736 xSetLabel(args[ac], XmHeight, 27); ac++;
737 networkPopUpButton = xmCreatePopUpButton(form,
738 "networkPopUpButton",
739 args,
740 ac);
741 xManageChild(networkPopUpButton, XmValueChangeCallback, networkPopUpButton,
742 xtAddCallback(networkPopUpButton, XmValueChangeCallback, networkPopUpButton,
743 args));
744 ac = 0;
745 xSetLabel(args[ac], XmFontList,
746 XmConvex((form, "-*-times-medium-r-*-*-160-*-*-*-iso8859-1",
747 XmFontList, 0, sargok)); if (argok) ac++);
748 xSetLabel(args[ac], XmLabelString,
749 XmConvex((form, "Browse",
750 XmFontString, 0, sargok)); if (argok) ac++;
751 xSetLabel(args[ac], Xnum, 480); ac++;
752 xSetLabel(args[ac], XmHeight, 10); ac++;
753 xSetLabel(args[ac], XnumWidth, 72); ac++;
754 xSetLabel(args[ac], XmFontList,
755 browseButton = xmCreatePushButton(form,
756 "browseButton",
757 args,
758 ac);
759 xManageChild(browseButton, xmActivateCallback, browseButton, browseButtonCallba
760 xtAddCallback(browseButton, xmActivateCallback, browseButton, (xtPointer)0);
761 ac = 0;
762 xSetLabel(args[ac], XmFontList,
763 xSetLabel(args[ac], Xnum, 110); ac++;
764 xSetLabel(args[ac], Xnum, 10); ac++;
765 xSetLabel(args[ac], XnumWidth, 370); ac++;
766 xSetLabel(args[ac], XmHeight, 30); ac++;
767 inputFileNameField = xmCreateTextField(form,
768 "inputFileNameField",
769 args,
770 ac);
771 xManageChild(inputFileNameField);
772
773 ac = 0;
774 xSetLabel(args[ac], XmFontList,
```

10/16/02
15:32:29

creation-c.c

10/16/02
15:32:29

creation-c.C

14

```

    MM_CONVERT(form, "-o-times-medium-r--*-140-*-*-iso8859-1",
    XmPortString, 0, XmPortString);
    if ((argok) ac++)
    XSetLabel(args[ac], XmLabelString,
    MM_CONVERT(form, "Input File",
    XmPortString, 0, XmPort));
    if ((argok) ac++)
    XmPString(args[ac], XmString, 20); ac++;
    XBelArc(args[ac], XmArc, XmString, 11); ac++;
    XSetArc(args[ac], XmWidth, 110); ac++;
    XSetArc(args[ac], XmHeight, 19); ac++;
    XSetArc(args[ac], XmWidth, 19); ac++;
    XmCreateLabel(inputFileLabel, "inputFileLabel",
    args,
    ac);
    XmManageChild(inputFileLabel);
    784    ac = 0;
    XBelArc(args[ac], XmWidth, 45); ac++;
    XmSetArc(args[ac], XmHeight, 386); ac++;
    selectFileDialog = XmCreatePopUpShell("selectFileDialog",
    xmFileDialogGetClass,
    mainWindow,
    args,
    ac);
    797    ac = 0;
    798    ac = 0;
    799    XBelArc(args[ac], XmWidth, 3); ac++;
    800    XBelArc(args[ac], XmY, 86); ac++;
    801    XBelArc(args[ac], XmWidth, 45); ac++;
    802    XBelArc(args[ac], XmHeight, 386); ac++;
    803    XBelArc(args[ac], XmWidth, 386); ac++;
    804    XmCreateFileSelectionBox(selectFileDialog,
    "inputFileSelectionBox",
    args,
    ac);
    805    XAddCallback(inputFileSelectionBox, XmOkCallback,
    okFilePlayDialogCallBack, (XtPointer)
    ac);
    806    XAddCallback(inputFileSelectionBox, XmCancelCallback,
    cancelFileDialogCallBack, (XtPointer)
    ac);
    807    XAddCallback(inputFileSelectionBox, XmDismissCallback,
    aboutDismissCallBack,
    (XtPointer)0);
    808    XAddCallback(inputFileSelectionBox, XmHelpCallback,
    cancelFileDialogCallBack,
    (XtPointer)
    ac);
    809    XmAddChild(aboutHelpDialogShell, XmHelpCallBack,
    cancelFileDialogCallBack,
    (XtPointer)
    ac);
    810    ac = 0;
    811    XBelArc(args[ac], XmWidth, 284); ac++;
    812    XBelArc(args[ac], XmHeight, 312); ac++;
    813    aboutHelpDialogShell = XmCreatePopUpShell("aboutHelpDialogShell",
    xmFileDialogGetClass,
    mainWindow,
    args,
    ac);
    814    ac = 0;
    815    XBelArc(args[ac], XmResizablePolicy, XmRESIZE_GROW);
    816    mainWindow,
    817    args,
    818    ac);
    819    ac = 0;
    820    XBelArc(args[ac], XmLabelString,
    XmCREATEFORM, 0);
    821    ac = 0;
    822    XBelArc(args[ac], XmWidth, 968); ac++;
    823    XBelArc(args[ac], XmY, 122); ac++;
    824    XBelArc(args[ac], XmWidth, 284); ac++;
    825    XBelArc(args[ac], XmHeight, 312); ac++;
    826    aboutHelpBulletinBoard = XmCreateBulletinBoard(aboutHelpDialogShell,
    "aboutHelpBulletinBoard",
    args,
    ac);
    827    ac = 0;
    828    XBelArc(args[ac], XmScrollingPolicy, XmAUTOMATIC);
    829    ac);
    830    ac = 0;
    831    XBelArc(args[ac], XmLabelString,
    XmCREATEFORM, 0);
    832    ac = 0;
    833    XBelArc(args[ac], XmWidth, 20); ac++;
    834    XBelArc(args[ac], XmY, 20); ac++;
    835    XBelArc(args[ac], XmWidth, 250); ac++;
    836    XBelArc(args[ac], XmHeight, 210); ac++;
    837    aboutScrolledWindow = XmCreateScrolledWindow(aboutHelpBulletinBoard,
    "aboutScrolledWindow",
    args,
    ac);
    838    ac = 0;
    839    ac);
    840    ac);
    841    XmManageChild(aboutScrolledWindow);
    842    ac = 0;
    843    XBelArc(args[ac], XmLabelString, "About");
    844    XBelArc(args[ac], XmFontList,
    XM_CENTRE);
    845    XBelArc(args[ac], XmFontList,
    XM_CENTRE);
    846    MM_CONVERT(aboutScrolledWindow, "-*-helvetica-bold-r--*-130-*-*-iso8859-1";
    847    XmFontList, 0, XmFont);
    if ((argok) ac++);
    848    XBelArc(args[ac], XmLabelString);
    849    MM_CONVERT(aboutScrolledWindow, ":::t:""\\"";Send socket message
    "from a File"::t:\\"";National Research Council\\t\\"; of Canada (2002)\\t\\";Dan
    Johnson\\t\\";All rights reserved\\t\";
    850    XmFontList, 0, XmFont);
    if ((argok) ac++);
    851    XBelArc(args[ac], XmLabelString, "About");
    852    XBelArc(args[ac], XmWidth, 230); ac++;
    853    XBelArc(args[ac], XmHeight, 250); ac++;
    854    aboutHelpLabel = XmCreateLabel(aboutScrolledWindow,
    "aboutHelpLabel",
    args,
    ac);
    855    ac);
    856    ac);
    857    ac);
    858    ac);
    859    ac = 0;
    860    XBelArc(args[ac], XmLabelString,
    MM_CONVERT(aboutHelpBulletinBoard, "Minimal"));
    861    XBelArc(args[ac], XmLabelString, "About");
    862    MM_CONVERT(aboutHelpBulletinBoard, "Minimal");
    863    XmFontList, 0, XmFont);
    if ((argok) ac++);
    864    XBelArc(args[ac], XmLabelString, "About");
    865    XBelArc(args[ac], XmWidth, 250); ac++;
    866    XBelArc(args[ac], XmHeight, 250); ac++;
    867    XBelArc(args[ac], XmLabelString, "About");
    868    aboutHelpSubButton = XmCreatePushButton(aboutHelpBulletinBoard,
    "aboutHelpSubButton",
    args,
    ac);
    869    ac);
    870    ac);
    871    XBelArc(aboutBisim, ssPushButton,
    aboutDismissCallBack,
    aboutActivateCallBack,
    aboutDissmissCallBack,
    aboutHelpCallBack, (XtPointer)
    dint);
    872    XAddCallback(aboutBisim, ssPushButton,
    aboutDismissCallBack,
    aboutActivateCallBack,
    aboutDissmissCallBack,
    aboutHelpCallBack, (XtPointer)
    dint);
    873    ac = 0;
    874    XBelArc(args[ac], XmNoneHelpWidget, helpCascadeButton);
    875    XBelArcValues(menuBar, args, ac);
    876    XBelArcValues(endnumBar, args, ac);
    877    ac = 0;
    878    XBelArc(args[ac], XmLeftAttachment, XmATTACH_FORM);
    879    XBelArc(args[ac], XmLeftAttachment, 400); ac++;
    880    XBelArc(args[ac], XmTopOffset, 250); ac++;
    881    XBelArcValues(endnumBar, args, ac);
    882    XBelArcValues(endnumBar, args, ac);
    883    ac = 0;
    884    XBelArc(args[ac], XmLeftAttachment, XmATTACH_FORM);
    885    XBelArc(args[ac], XmLeftAttachment, 350); ac++;
    886    XBelArc(args[ac], XmTopOffset, 250); ac++;
    887    XBelArcValues(endnumBar, args, ac);
    888    XBelArcValues(endnumBar, args, ac);
    889    ac = 0;
    890    XBelArc(args[ac], XmLeftAttachment, XmATTACH_FORM);
    891    XBelArc(args[ac], XmLeftAttachment, 250); ac++;
    892    XBelArcValues(startNumBar, args, ac);
    893    XBelArcValues(startNumBar, args, ac);
    894    ac = 0;
    895    XBelArc(args[ac], XmLeftAttachment, XmATTACH_FORM);
    896    XBelArc(args[ac], XmTopOffset, 140); ac++;
    897    XBelArc(args[ac], XmLeftAttachment, 250); ac++;
    898    XBelArc(args[ac], XmTopOffset, 60); ac++;
    899    XBelArc(args[ac], XmLeftAttachment, 250); ac++;
    900    XBelArcValues(startNumBar, args, ac);
  
```

10/16/02
15:32:29

creation-c.c

10/16/02
15:32:29

creation-c.c

16

```
901      ac = 0;
902      XtSetArg(args[ac], XmRLeftAttachment, XmRLeftAttachment, XmRMatchPosition) ; ac++;
903      XtSetArg(args[ac], XmRRightAttachment, XmRRightAttachment, XmRMatchPosition) ; ac++;
904      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
905      XtSetArg(args[ac], XmRightAttachment, XmRightAttachment, 30) ; ac++;
906      XtSetArg(args[ac], XmLeftOffset, 300) ; ac++;
907      XtSetArg(args[ac], XmTopOffset, 210) ; ac++;
908      XtSetValues(destinationTextField, args, ac);
909
910      ac = 0;
911      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
912      XtSetArg(args[ac], XmLeftOffset, 330) ; ac++;
913      XtSetArg(args[ac], XmTopOffset, 190) ; ac++;
914      XtSetValues(fastLabel, args, ac);
915
916      ac = 0;
917      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
918      XtSetArg(args[ac], XmLeftOffset, 20) ; ac++;
919      XtSetArg(args[ac], XmTopOffset, 190) ; ac++;
920      XtSetValues(slowLabel, args, ac);
921
922      ac = 0;
923      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
924      XtSetArg(args[ac], XmLeftOffset, 180) ; ac++;
925      XtSetArg(args[ac], XmTopOffset, 190) ; ac++;
926      XtSetValues(speedLabel, args, ac);
927
928      ac = 0;
929      XtSetArg(args[ac], XmTopAttachment, XmTopAttachment, XmRMatchForm) ; ac++;
930      XtSetArg(args[ac], XmBottomAttachment, XmBottomAttachment, XmRMatchForm) ; ac++;
931      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
932      XtSetArg(args[ac], XmLeftOffset, 20) ; ac++;
933      XtSetValues(scrollBar, args, ac);
934
935      ac = 0;
936      XtSetArg(args[ac], XmTopAttachment, XmTopAttachment, XmRMatchForm) ; ac++;
937      XtSetArg(args[ac], XmBottomAttachment, XmBottomAttachment, XmRMatchForm) ; ac++;
938      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchPosition) ; ac++;
939      XtSetArg(args[ac], XmBottomPosition, 95) ; ac++;
940      XtSetArg(args[ac], XmLeftOffset, 20) ; ac++;
941      XtSetArg(args[ac], XmTopOffset, 300) ; ac++;
942      XtSetValues(scrollButton, args, ac);
943
944      ac = 0;
945      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
946      XtSetArg(args[ac], XmLeftOffset, 240) ; ac++;
947      XtSetArg(args[ac], XmLeftOffset, 90) ; ac++;
948      XtSetValues(startPushButton, args, ac);
949
950      ac = 0;
951      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
952      XtSetArg(args[ac], XmLeftOffset, 310) ; ac++;
953      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
954      XtSetArg(args[ac], XmLeftOffset, 90) ; ac++;
955      XtSetValues(portLabel, args, ac);
956
957      ac = 0;
958      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
959      XtSetArg(args[ac], XmLeftOffset, 90) ; ac++;
960      XtSetValues(portLabel, args, ac);
961
962      ac = 0;
963      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
964      XtSetArg(args[ac], XmLeftOffset, 310) ; ac++;
965      XtSetArg(args[ac], XmLeftOffset, 60) ; ac++;
966
```

```
967      XtSetValues(destinationTextField, args, ac);
968      ac = 0;
969      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
970      XtSetArg(args[ac], XmLeftOffset, 240) ; ac++;
971      XtSetArg(args[ac], XmTopOffset, 60) ; ac++;
972      XtSetValues(destinationLabel, args, ac);
973
974      ac = 0;
975      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
976      XtSetArg(args[ac], XmLeftOffset, 20) ; ac++;
977      XtSetArg(args[ac], XmTopOffset, 90) ; ac++;
978      XtSetValues(filterGGLEButton, args, ac);
979
980      ac = 0;
981      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
982      XtSetArg(args[ac], XmLeftOffset, 20) ; ac++;
983      XtSetArg(args[ac], XmTopOffset, 60) ; ac++;
984      XtSetValues(neutralToggleButton, args, ac);
985
986      ac = 0;
987      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
988      XtSetArg(args[ac], XmLeftOffset, 480) ; ac++;
989      XtSetArg(args[ac], XmTopOffset, 10) ; ac++;
990      XtSetValues(brownLabel, args, ac);
991
992      ac = 0;
993      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
994      XtSetArg(args[ac], XmLeftOffset, 110) ; ac++;
995      XtSetArg(args[ac], XmTopOffset, 10) ; ac++;
996      XtSetValues(inputLabel, args, ac);
997
998      ac = 0;
999      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchPosition) ; ac++;
1000      XtSetArg(args[ac], XmRightAttachment, XmRightAttachment, XmRMatchPosition) ; ac++;
1001      XtSetArg(args[ac], XmLeftOffset, 3) ; ac++;
1002      XtSetArg(args[ac], XmRightOffset, 21) ; ac++;
1003      XtSetArg(args[ac], XmBottomLeftOffset, 0) ; ac++;
1004      XtSetArg(args[ac], XmBottomRightPosition, 3) ; ac++;
1005      XtSetArg(args[ac], XmRightPosition, 20) ; ac++;
1006      XtSetValues(inputLabel, args, ac);
1007
1008      ac = 0;
1009      XtSetArg(args[ac], XmLeftAttachment, XmLeftAttachment, XmRMatchForm) ; ac++;
1010
1011      /* Begin user code block <and_CreatMainWindow> */
1012      /* End user code block <and_CreatMainWindow> */
1013      return( mainWindow );
1014
1015 }
```

10/16/02
15:32:37

creation-c.h

1

```
1  /*  
2   * WARNING: This file is overwritten at code generation time.  
3   * Any changes to this file will be lost.  
4   */  
5  /*  
6  * Created by Builder Xcessory Version 5.0  
7  * Generated by Code Generator Xcessory 5.0 (05/22/98)  
8  */  
9  /*  
10  */  
11 #ifndef creation_c_H  
12 #define creation_c_H  
13 /*  
14  * Global widget declarations.  
15  * - EXTERNAL is set to extern if the  
16  *   code file is not included from the  
17  *   main file.  
18  */  
19 /*  
20 #ifdef DECLARE_EX_GLOBALS  
21 #define EXTERNAL  
22 #else  
23 #define EXTERNAL extern  
24 #endif  
25 /*  
26  * Start Global Widget Declarations.  
27  */  
28 /*  
29 EXTERNAL Widget  
30 EXTERNAL Widget  
31 EXTERNAL Widget  
32 EXTERNAL Widget  
33 EXTERNAL Widget  
34 EXTERNAL Widget  
35 EXTERNAL Widget  
36 EXTERNAL Widget  
37 EXTERNAL Widget  
38 EXTERNAL Widget  
39 EXTERNAL Widget  
40 EXTERNAL Widget  
41 EXTERNAL Widget  
42 EXTERNAL Widget  
43 EXTERNAL Widget  
44 EXTERNAL Widget  
45 EXTERNAL Widget  
46 /*  
47  * End Global Widget Declarations.  
48 */  
49 #endif
```


10/16/02
15:32:45

main-c.C

1

```
1 /* NAME: Portions of this file are merged at file generation
2 * time. Edits can be made "only" in between specified code blocks, look
3 * for keywords <begin user code> and <end user code>.
4 */
5 /*
6 * Generated by the ICS Builder Accessory (IK).
7 * Builder Accessory Version 5.0
8 * Code Generator Xaccessory 5.0 (05/22/98)
9 */
10 /*
11 * Generated Xaccessory 5.0 (05/22/98)
12 */
13 /*
14 /* Begin user code block <file_comments> */
15 /*
16 /* File play/main-c.c
17 \ingroup mod_Play
18 */
19 /*
20 This file is mostly created by the user interface builder tool
21 Builder Accessory version 5.0. There are only a few places where
22 we can add our input.
23
24 The purpose of this file is to create the user interface, set some
25 of the initial variables and user interface widget states, and
26 then loop waiting for (and responding to) events.
27
28 It is the response to the events, mainly in the callbacks source
29 file that will define the logic of the program.
30 */
31 /*
32 Copyright (c) 2002
33 All content of this file is copyrighted to the
34 National Research Council of Canada.
35 All rights reserved.
36
37 Author Daniel F. Johnston
38 Licence April 2002
39 */
40 $Id: main-c.C,v 1.6 2002/08/19 18:46:55 johnston Exp $
41 */
42 /* End user code block <file_comments> */
43 */
44 /*
45 * Motif required Headers
46 */
47 #include <x11/StringDefs.h>
48 #include <xm/Xm/Dialog.h>
49 #if (XmVersion >= 1002)
50 #include <xm/RapType.h>
51 #endif
52 #include <xm/Menutab.h>
53 /*
54 * Headers for classes used in this program
55 */
56 /*
57 * Globally included information.
58 */
59 /*
60 */
61 /*
62 */
63 /*
64 */
65 /*
66 */

/* Headers for classes used in this program
* Globally included information.
* Common constant and pixmap declarations.
*/
```

10/16/02
15:32:45

main-c.C

2

```
67 #include "creation-c.h"
68 /*
69 * Convenience functions from utilities file.
70 */
71 /*
72 extern void RegisterXmConverters(XmApplicationContext);
73 extern XmPointer Xm_CONVERT_WIDGET, char *, int, Boolean *);
74 extern XmPointer Xm_DOUBLE(double);
75 extern XmPointer Xm_STRING(float);
76 extern void Xm_MENU_POST(Widget, XmEvent *, Boolean *);
77 extern Plenum Xm_SET_WIDGET, char **;
78 extern void Xm_SET_BACKGROUND_COLOR(Widget, Arglist, Cardinal *, Pixel);
79
80 /* Begin user code block <global> */
81 #include "../common/mySocket.h"
82 #include "../common/myInterface.h"
83 // the block of user defined Globals shared by this interface
84 globalMemory shared;
85 /* End user code block <global> */
86
87 /*
88 * change this line via the Output Application Names Dialog.
89 */
90 #define IX_APP_CLASS "BuilderProduct"
91 int main( argc, char **argv )
92 {
93     Widget parent;
94     XmAppContext app;
95     Arg args[256];
96     Cardinal aci;
97     Boolean autoRelease;
98     Widget topLevelWindow;
99     mainWindow;
100    Widget
101
102    /* Begin user code block <declarations> */
103    char tempBuf[MAX_NUMERIC_TEXT_SIZE];
104    /* End user code block <declarations> */
105
106    /*
107     * The applicationShell is created as an unrealized
108     * parent for multiple topLevelWindows. The topLevelWindows
109     * are created as popup children of the applicationShell.
110     * This is a recommendation of Paul Asente & Ralph Swick in
111     * X-Window Toolkit, P. 677.
112
113    parent = XmAppInitialize( app, IX_APP_CLASS, NULL, 0,
114    #ifndef XmBSpecificationRelease
115        #if (XmVersion >= 1002)
116            (Cardinal *) argv,
117        #else
118            #if (XmSpecificationRelease>=5)
119                argv,
120            #else
121                (Cardinal *) argv,
122            #endif
123            #endif
124            argv, NULL,
125            NULL );
126
127    RegisterXmConverters(app);
128    #if (XmVersion >= 1002)
129        XmSetTypeInstallForModelConverter();
130    #endif
131
132    /* Begin user code block <create_Shells> */
133
```

10/16/02
15:32:45

main-c.c

10/16/02
15:32:45

3

```
133 ac = (int) argc; /* use the variable to suppress compiler warnings */
134 /* End user code block <create_shells> */
135
136 /* Create classes and widgets used in this program.
137 */
138 /*/
139 /* Begin user code block <create_toplevelShell> */
140
141 /*/
142 Vingroup mod_play
143
144 This main program is mostly computer generated. We only modify it
145 (between the "user code block" command) to create our global memory
146 structure and initialise it based on the command line arguments.
147
148 /*
149 /* copy our current application context to make it global */
150 shared.mainAppContext = app;
151
152 /* start by assuming there's no supplied args */
153 shared.addresses[0] = shared.addresses[0]
154 - shared.portNumbers[0] = '\0';
155
156 switch ( argc )
157 {
158 case 4:
159 /* we should have all 3 args, destination, port and filename */
160 /* process the filename arg */
161 if( !(shared.filerefDescr = fopen( argv[3], "r" )) )
162 {
163 printf("\n(main) Cannot open file '%s'\n", argv[3] );
164 usageSend();
165 exit(1);
166 }
167 strcpy( shared.inputFileName, argv[3] );
168 /* no 'break' here, fall through to process argv[2] */
169
170 /* we have 2 args, the destination and the port number */
171 /* process the port number here */
172 if( sscanf ( argv[2], "%u", &shared.portNumber ) != 1 )
173 {
174 printf("\nNot a valid port number '%s'\n", argv[2] );
175 usageSend();
176 exit(1);
177 }
178 strcpy( shared.portNumbers, argv[2] );
179 /* no 'break' here, fall through to process argv[1] */
180
181 /* we have 1 arg, the destination address */
182 strcpy( shared.address, argv[1] );
183 break;
184
185 /* we have no user supplied command args */
186
187 /* no 'break' here, fall through to process argv[1] */
188 usageSend();
189 exit(1);
190
191 */
192 /* End user code block <create_toplevelShell> */

193 ac = 0;
194 XSetArg(argv[ac], XmNx, 612); ac++;
195 XSetArg(argv[ac], XmNy, 414); ac++;
196 XSetArg(argv[ac], XmWidth, 384); ac++;
197 XSetArg(argv[ac], XmHeight, 385); ac++;

198 XSetSensitive( argv[ac], XmSensitive, True ); /* it already is, by default */

199 XSetPushbutton( fileOpPrgleButton, False ); /* the normal 'network' state */

200 XSetPushbutton( networkToggleButton, True, False );
201 XSetPushbutton( networkOpPrgleButton, True, False );
202 XSetPushbutton( networkOpPrgleState( networkOpPrgleButton, False, False ) );
203 XSetPushbutton( fileOpPrgleButton, True, False );
204 XSetPushbutton( fileOpPrgleState( fileOpPrgleButton, False, False ) );
205 XSetPushbutton( fileOpPrgleState( fileOpPrgleButton, False, False ) );
206
207 /* the initial 'enabled' stat for pushbuttons */

208 /* start enabled, pass and release disabled */
209 XSetSensitive( startPushbutton, True ); /* it already is, by default */

210 XSetPushbutton( pausePushbutton, False );
```

main-c.C

4

```
199 topLevelShell = XtCreatePopupShell("topLevelShell",
200 topLevelShellWidgetClass,
201 parent,
202 args,
203 ac);
204 mainWindow = (Widget)CreateMainWindow( topLevelShell );
205 XtPopUp(XParent(mainWindow),
206 XtGrabNone);
207
208 /* Begin user code block <app_procedures> */
209 /* set the initial state of all the GUI interface widgets */
210 /* the default address for network destination */
211 if( shared.address == 0 )
212 {
213 XmTextfieldSetString( destinationTextField, LOCAL_ADDRESS );
214 strcpy( shared.address, LOCAL_ADDRESS );
215 }
216 else
217 XmTextfieldSetString( destinationTextField, shared.address );
218
219 /* the default input file name */
220 if( shared.inputFileName == 0 )
221 XmTextfieldSetString( inputFileNameField, "<none>" );
222 else
223 XmTextfieldSetString( inputFileNameField, shared.inputFileName );
224
225 /* the default value for the network port number if not user supplied */
226 if( shared.portNumber == 0 )
227 {
228 shared.portNumber = DEFAULT_PORT_NUMBER;
229 sprintf( shared.portNumber, "%d", shared.portNumber );
230 }
231 else
232 XmTextfieldSetString( portNumberField, shared.portNumber );
233
234 XmTextfieldSetString( portNumberField, shared.portNumber );
235 scanC( shared.portNumber );
236
237 /* the initial value for the speed multiplier */
238 shared.SpeedFactor = 1.0f;
239
240 /* the initial values for the toggle buttons */
241 XmToggleButtonSetState( debugOpPrgleButton, False, False );
242 shared.globalDebugMode = 0;
243
244 if( shared.portNumber == 0 )
245 {
246 /* the output to file mode */
247 XmToggleButtonSetState( networkToggleButton, False, False );
248 XmToggleButtonSetState( networkOpPrgleButton, False, False );
249 XmToggleButtonSetState( fileToggleButton, True, False );
250 XmToggleButtonSetState( fileOpPrgleButton, True, False );
251 }
252
253 {
254 /* the normal 'network' state */
255 XmToggleButtonSetState( networkOpPrgleButton, True, False );
256 XmToggleButtonSetState( networkOpPrgleState( networkOpPrgleButton, True, False ) );
257 XmToggleButtonSetState( fileOpPrgleButton, False, False );
258 XmToggleButtonSetState( fileOpPrgleState( fileOpPrgleButton, False, False ) );
259 }
260
261 /* the initial 'enabled' stat for pushbuttons */
262 /* start enabled, pass and release disabled */
263 XSetSensitive( startPushbutton, True ); /* it already is, by default */
264 XSetPushbutton( pausePushbutton, False );
```

10/16/02
15:32:45

main.c.c

5

```
265 xSetSensitive( resumePushButton, False );
266 /* the dialog box for file selection is not visible initially */
268 xThmManagerChildInputFileSelectionBox;
269 /* the dialog box for help (about) is not visible initially */
270 xThmManagerChild(aboutHelpInBoard);
271
272 shared.startMsgnum = 1;
273 /* read the number of messages in the input file and set count */
274 if( strlen(shared.inputFileName) != 0 )
275 {
276     shared.endMsgnum = readNumberOfMessages( shared.inputFileName );
277
278     if( shared.endMsgnum == 0 )
279     {
280         xBell( xDisplay(mainWindow), 100 );
281     }
282
283     shared.endMsgnum = 0;
284     sprintf( tmpBuf, "%d", shared.endMsgnum );
285     XSetField( fieldString( endMsgnumField, tmpBuf ) );
286
287 /* we are not sending yet */
288 shared.childProcess = 0;
289 /* End user code block <app_procedures> */
290
291 /* Begin user code block <main_loop> */
292 /* End user code block <main_loop> */
293
294 xAppMainLoop(app);
295
296 /* A return value regardless of whether or not the main loop ends.
297 */
298 /* */
299 /* */
300
301 }
```


10/16/02
15:32:58

play_support.c

1

```
1  /* File play_support.c
2   \ingroup mod_play
3
4   This file contains the support function needed for the
5   application. We have usage printing function and motif
6   string manipulation
7
8   \code
9
10  Copyright (c) 2002
11  All content of this file is copyrighted to the
12  National Research Council of Canada.
13  All rights reserved.
14  \endcode
15  Author Daniel F. Johnston
16  \since May 2002
17
18  $Id: play_support.c,v 1.3 2002/08/19 18:46:55 johnston Exp $
19  */
20
21 // a counter of where we are in reading file
22 unsigned int fnameNumber;
23
24 /**
25  \ingroup mod_play
26  This function will be called when there is an error with the user-
27  supplied command line arguments for the socket playback program.
28  It will simply print out the list of standard and optional
29  command arguments, and then finish.
30 */
31 void usageSend( void )
32 {
33     printf("Usage required command is of the form...\n");
34     printf("  sendSocket [destination [portnum [filename]]]\n");
35     printf("    - where 'destination' is the network or file destination\n");
36     printf("    - where 'portnum' is the port number (0 indicates file)\n");
37     printf("    - and 'filename' is the file with socket data\n");
38 }
39
40
41
```


10/16/02
15:33:09

sending.c

```
1  /*  
2   File sending.c  
3   \ingroup mod_play  
4   This file contains the support functions needed for creating  
5   the socket and sending socket messages. We also provide  
6   functions for controlling the sending process, i.e. speed,  
7   pause/resume, sending done, etc.  
8   */  
9  
10  \code  
11  Copyright (c) 2002  
12  All content of this file is copyrighted to the  
13  National Research Council of Canada.  
14  All rights reserved.  
15  \endcode  
16  Author Daniel F. Johnston  
17  Since May 2002  
18  $Id: sending.c,v 1.10 2002/10/16 18:10:30 johnston Exp $  
19  
20  */  
21  #include <xm/2m.h>  
22  #include "creation-c.h"  
23  #include "sys/socket.h"  
24  #include "sys/types.h"  
25  #include "sys/wait.h"  
26  #include "netstat/in.h"  
27  #include "cmisid.h"  
28  #include "netdb.h"  
29  #include "time.h"  
30  #include ".../common/mySocket.h"  
31  #include ".../common/myInterface.h"  
32  #include ".../xm.h"  
33  // the byte array that will hold a UDP socket message for sending  
34  char sendbuf[MAX_MESSAGE_SIZE];  
35  // the socket address structure for the destination (if defined)  
36  struct sockaddr_in server;  
37  /*  
38  \ingroup mod_play  
39  This function will create a socket. It uses the supplied  
40  network address and port number to create a socket that  
41  can write data to that address.  
42  \param address is the TCP/IP network address of the network node we  
43  want to send data to. It can be in a name format, or an address  
44  in the IP /dot/ address format.  
45  \param port is the integer port number to use for this new socket  
46  \param hostent is the hostent structure provided by the system  
47  \return an error indication (if < 0) or the integer value of the socket  
48  */  
49  int createSendSocket( char *address, int port )  
50  {  
51      int sock;  
52      struct hostent *hp, *gethostbyname();  
53  
54      sock = socket(AF_INET, SOCK_DGRAM, 0);  
55      if ( sock < 0 )  
56      {  
57          perror("opening stream socket");  
58          return -1;  
59      }  
60  }  
61  
62  /* Connect socket using hostname provided */  
63  hp = gethostbyname( address );  
64  if ( hp == 0 )  
65  {  
66      fprintf(stderr, "%s: unknown host, "\
```

10/16/02
15:33:09

sending.c

```
67      enter valid host name\n", address );  
68  }  
69  /*  
70  \ingroup mod_play  
71  bssrc((char *)server, sizeof (server));  
72  copy((char *)hp->h_addr, (char *)server.sin_addr, hp->h_length);  
73  /* Assign client port number */  
74  server.sin_family = AF_INET;  
75  server.sin_port = port;  
76  return sock;  
77  }  
78  /* send the user supplied message */  
79  /*  
80  \ingroup mod_play  
81  This function will send a message via an existing socket.  
82  \param sock is the integer socket to use for sending  
83  \param buffer is an array of bytes to be sent via the socket.  
84  \param contents do not have to be only ASCII characters.  
85  \param sizebuffer is the integer length of the user supplied message  
86  \return an error indication (if < 0) or the number of characters sent  
87  */  
88  int  
89  sendSocketMessage( int sock, char *buffer, int sizebuffer )  
90  {  
91      int i;  
92      if ( i = sendto(sock, buffer, sizebuffer, 0,  
93                      (struct sockaddr *)server, sizeof server ) < 0 )  
94      {  
95          perror("writing to socket");  
96          return -1;  
97      }  
98      return i;  
99  }  
100 /*  
101  \ingroup mod_play  
102  This function will reset the start, pause and resume buttons to their  
103  default state. This state is the start button enabled, and the  
104  others disabled.  
105  */  
106 void  
107 resetSendButtons()  
108 {  
109     extern GlobalMemory shared;  
110     /* disable the pause and resume button, reenable the start */  
111     xSetSensitive( startPushButton, True );  
112     xSetSensitive( pausePushButton, False );  
113     xSetSensitive( resumePushButton, False );  
114     /* the child process is no more */  
115     shared.childprocess = 0;  
116  
117     /* the "child should pause" signal handler */  
118  }  
119 /*  
120  \ingroup mod_play  
121  This function will be called by the child (sending) process in response  
122  to a 'pause' signal. The function will make sure that the process ID  
123  is correct and, if it is, it will cause the process to 'pause'. The sending  
124  process will remain in this state until another signal changes it.  
125  
126  */  
127 void  
128 sendProcessPause()  
129 {  
130     int pid;  
131     int status;
```

2

10/16/02
15:33:09

sending.c

3
15:33:09

```
133     extern GlobalMemory shared;
134
135     if( (pid = wait(status)) == -1 )
136         /* an error of some kind (probably a fork error) - ignore it */
137
138     if( shared.GlobalDebugode == 1 )
139         printf("child process pause: fork error!\n");
140
141     return;
142 }
143
144     if( pid == shared.childProcess )
145         /* we pause the current child process */
146         pause();
147
148     /* else */
149     /* some other process id */
150     if( shared.GlobalDebugode == 1 )
151         printf("Child process pause: pid %d received!\n", pid);
152
153 }
154
155 /* the "child die" signal handler */
156
157     /*ingroup mod_play
158     * This function will be called by the child (sending) process whenever it is
159     * done, i.e. when the complete file of socket messages has been sent. This
160     * signal is used to queue an event for the user interface to switch the
161     * button states as required by this change in program state.
162 */
163 void
164 sendProcessDone()
165 {
166     int pid;
167     int status;
168     extern GlobalMemory shared;
169
170     if( (pid = wait(status)) == -1 )
171         /* an error of some kind (probably a fork error) - ignore it */
172
173     if( shared.GlobalDebugode == 1 )
174         printf("child process done: fork error!\n");
175
176     return;
177 }
178
179 if( pid == shared.childProcess )
180     /* our expected child process did, reset the button state */
181     /* we cannot do it here because the child process cannot
182     * access the same user interface as the main program
183
184     XAppAddPrimeOut( shared.mainAppContext, 0, resetSendButtons, NULL );
185
186     /* send the messages from the file, at selected speed */
187
188     /* else */
189     /* some other process died */
190     if( shared.GlobalDebugode == 1 )
191         printf("Child process done: pid %d received!\n", pid);
192
193     /* send the messages from the file, at selected speed */
194
195     /*ingroup mod_play
196     * This function will be called by the child (sending) process and the
197     * process will spend most of its time here in this loop. The child
198     * process will loop until all the file of messages has been sent, or
199     */
```

```
199     it can be paused by a 'signal' from the main program. When the
200     file is done, the child process will trigger a signal to the main
201     process to let it know the file is finished.
202     \param filfd is the file descriptor for the (already opened) file
203     which contains the socket messages and delays.
204     \param outfd is the file descriptor of the (already opened) output
205     file used to save the (subset) of the messages if the destination
206     selected is not an network address
207     \return an error indication (< 0) if there are sending problems
208 */
209     int
210     sendMessages( FILE *file, FILE *outfile )
211     {
212         extern GlobalMemory shared;
213
214         timeSpec t_delay;
215         int itemsRead;
216         int itemsWritten;
217
218         for(;shared.messageNumber<=shared.endMessageNumber++;
219             /* read the first part of the delay from the file */
220             itemsRead = fread( &delay.tv_sec, sizeof(long), 1, filfd );
221             if( itemsRead != 1 )
222                 return 0;
223             if( itemsRead == 1 )
224                 /* read the second part of the delay from the file */
225                 itemsRead = fread( &delay.tv_nsec, sizeof(long), 1, filfd );
226             if( itemsRead == 1 )
227                 if( itemsRead != 1 )
228                     return 0;
229             /* read the size of the message from the file */
230             itemsRead = fread( &length, sizeof(int), 1, filfd );
231             if( itemsRead != 1 )
232                 return 0;
233             if( length < 0 || length > MAX_MESSAGE_SIZE )
234                 return -1;
235             itemsRead = fread( buffer, length, 1, filfd );
236             if( itemsRead != 1 )
237                 return 0;
238
239             /* only send/save after we read past the start number */
240             if( shared.messageNumber >= shared.startMsgNmbr )
241                 {
242                     if( !XtToggleButtonGetState( networkToggleButton )
243                         == False )
244                         /* if we have a file destination, save all */
245                         itemsWrote = fwrite( &delay.tv_sec, sizeof(long), 1, outfile );
246                         if( itemsWrote != 1 )
247                             return 0;
248                         itemsWrote = fwrite( &delay.tv_nsec, sizeof(long), 1, outfile );
249                         if( itemsWrote != 1 )
250                             return 0;
251                         itemsWrote = fwrite( &length, sizeof(int), 1, outfile );
252                         if( itemsWrote != 1 )
253                             return 0;
254                         itemsWrote = fwrite( buffer, length, 1, outfile );
255                         if( itemsWrote != 1 )
256                             return 0;
257
258             }
259
260         /* delay.tv_sec /= shared.speedFactor;
261         delay.tv_nsec /= shared.speedFactor;
262         nanosleep(&delay, &NULL);
263         /* write the character string to the socket
264         */
```

4
10/16/02
15:33:09

Sending.C

10/16/02
15:33:09

sending.c

5

```
* and indicate an error if unsuccessful
265
266    /*
267     * If (sendto(shared.socketNumb, buffer, length, 0,
268     *             (struct sockaddr *)server, sizeof server ) < 0)
269     {
270         perror("writing socket");
271         exit(1);
272     }
273
274     if( shared.GlobalDebugMode == 1 )
275         printf("%s %d\n", shared.messageNumber);
276     */
277
278     /* we completed the send! */
279
280
281 }
```

Appendix D

Other Source Code

10/16/02
15:37:18

listing.sh

1

```
#!/bin/sh
#
# simple script to generate listing files of source
# Convert them to postscript, and then print them
# two pages per page.
#
# If [-s $1]
# then
# Get a copy of the file with line numbers
nl -ba $1 > $1".lnt"
# convert to postscript
/usr/lib/print/laptops -O -E $1 -N -U -Tl.0in -F0 $1".lnt" > $1".ps"
# convert postscript to 2-pages-per-page Postscript
/disk2/people/oliver/bin/gnup -2 -m.5in -d1 < $1".ps" > $1".2ps"
else
echo ""
fi
echo "File not found"
exit 1
fi
```

10/16/02
15:37:18

