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

Johnston, Daniel F.

Publisher's version / Version de l'éditeur:
https://doi.org/10.4224/21272477
IMTI ITFI; no. IMTI-TR-011, 2002-10-16

INTEGRATED MANUFACTURING TECHNOLOGIES INSTITUTE

INTSTITUT DES TECHNOLOGIES DE FABRICATION INTÉGRÉE

Pages: 162

REPORT
RAPPORT

Date: 2002 October 16

Fig.
Diag. 19

SAP Project #
de Projet de SAP 55-W99

For
Pour IMTI

Unclassified
(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

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

List of Figures

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 extent 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 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

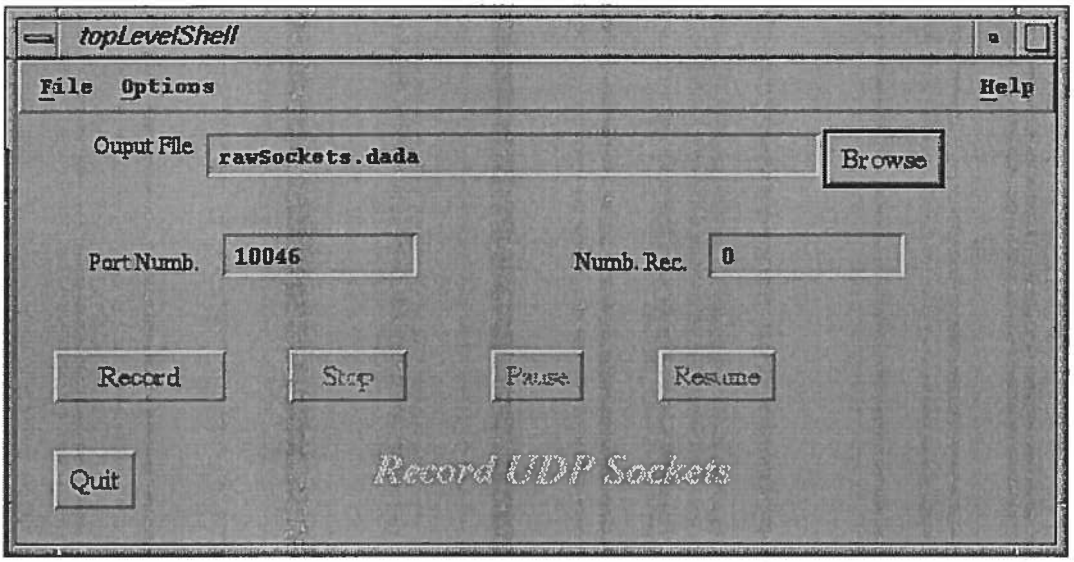

Figure 2.2: Socket recording - main window

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

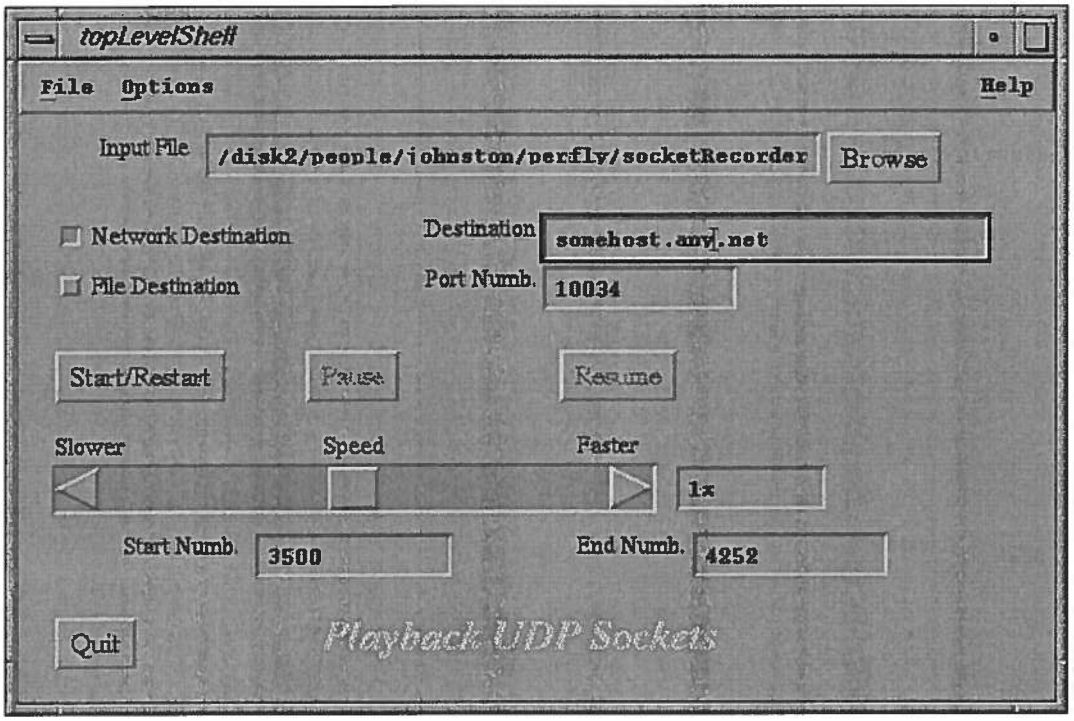

Figure 2.3: Socket playback - main window

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 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/Restart" 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/Restart" 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/Restart" 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/Restart" 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/Restart" 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.

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".

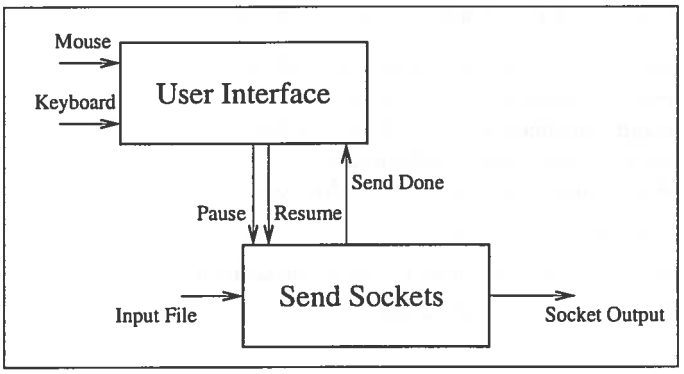

Figure 3.1: Two processes with signals

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.

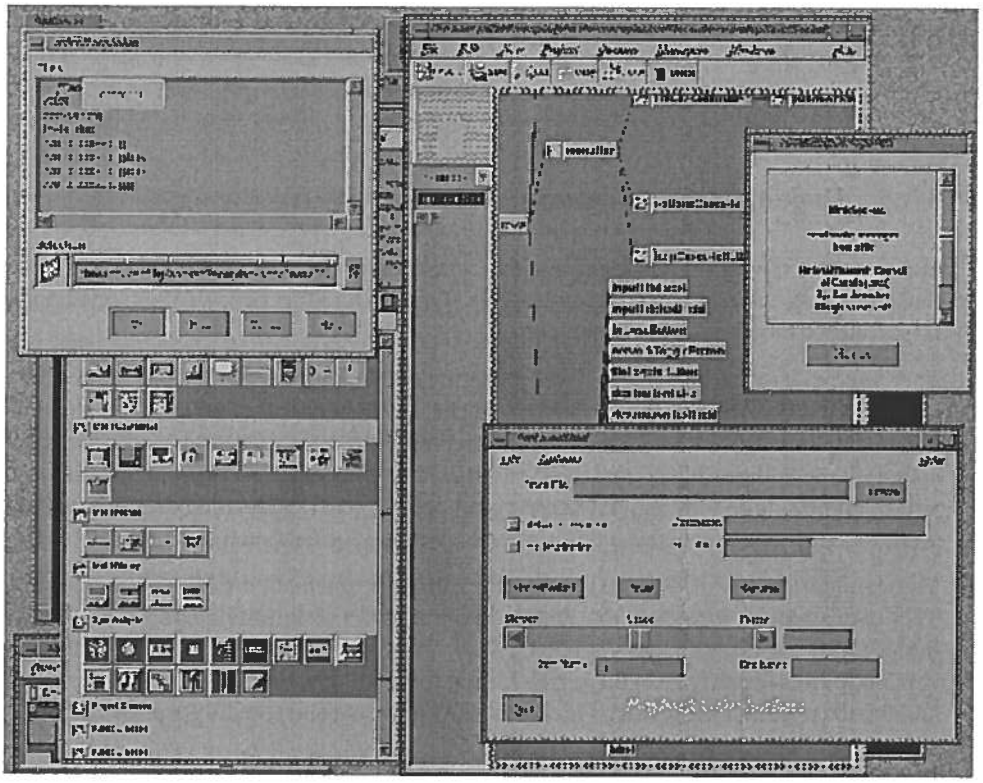

Figure 3.2: Builder Xcessory version 5.0

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 knowledgable 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.

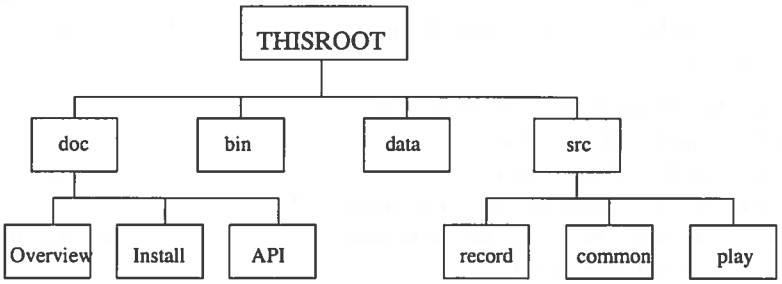

Figure 3.3: Source code directory structure

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 in found under the 'src' directory. There are three subdirectories;

- 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 subdirectories 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:

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:

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:

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:

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 declaired '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 okFilePlayDialogCallback (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 trys 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 existance 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 existance 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 it's 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 trys 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 existance 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 clild 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 existance 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 existance 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 contect 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 dor 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 XtAppContext _globalMemory::mainAppContext

the application contect 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>
```

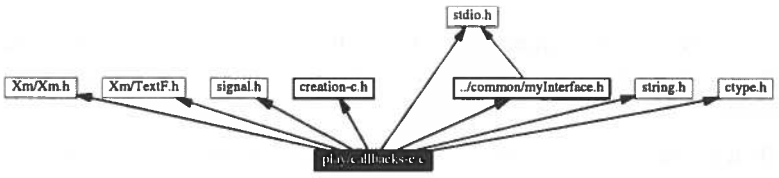

Figure 11.1: Include dependency graph for play_callbacks-c.c

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 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)

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.

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) ()

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>
```

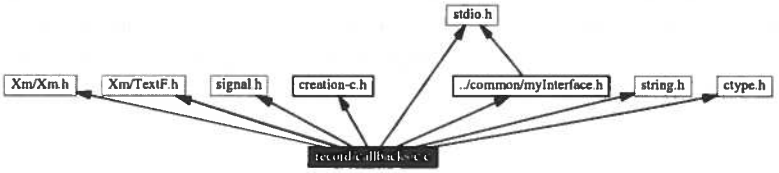

Figure 11.2: Include dependency graph for record_callbacks-c.c

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.

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"
```

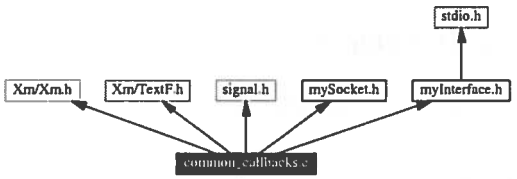

Figure 11.3: Include dependency graph for common_callbacks.c

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.

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"
```

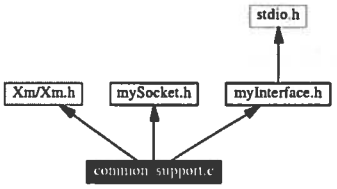

Figure 11.4: Include dependency graph for common_support.c

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"
```

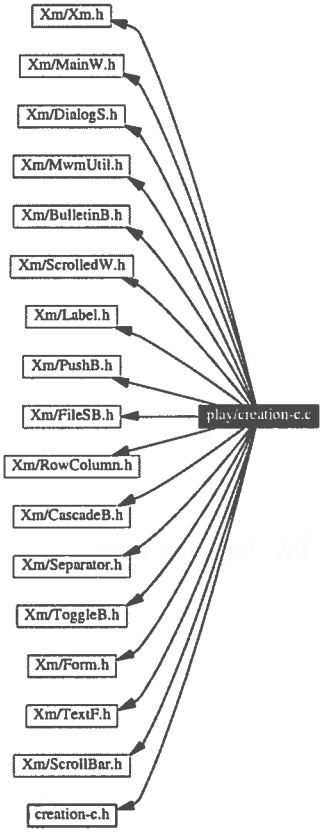

Figure 11.5: Include dependency graph for play_creation-c.c

Functions

- void **RegisterBxConverters** (XtAppContext)
- 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.

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"
```

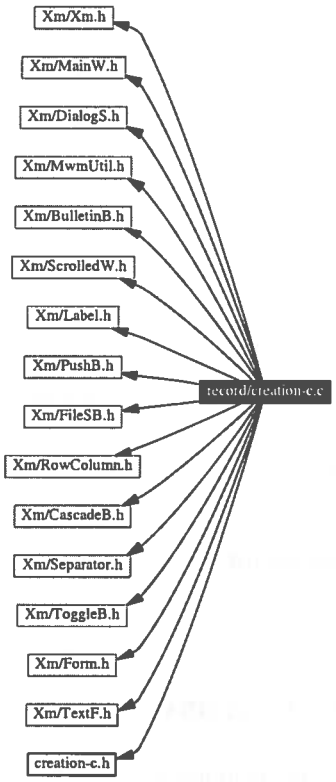

Figure 11.6: Include dependency graph for record_creation-c.c

Functions

- void **RegisterBxConverters** (XtAppContext)
- 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

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"
```

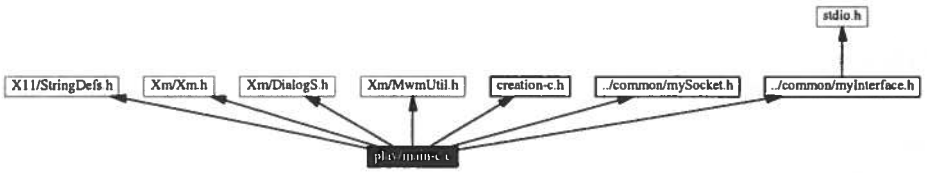

Figure 11.7: Include dependency graph for play_main-c.c

Defines

- #define **BX_APP_CLASS** "BuilderProduct"

Functions

- void **RegisterBxConverters** (XtAppContext)
- 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 responce to the events, mainly in the callbacks source file that will define the logic of the program.

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"
```

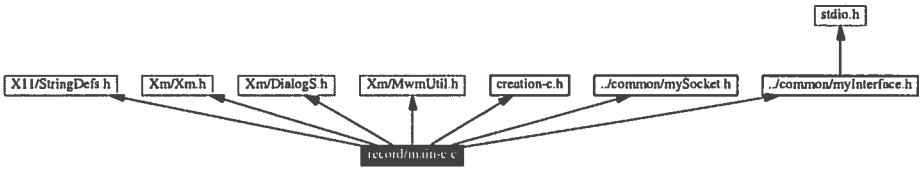

Figure 11.8: Include dependency graph for record_main-c.c

Defines

- #define **BX_APP_CLASS** "BuilderProduct"

Functions

- void **RegisterBxConverters** (XtAppContext)
- 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 responce 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.

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:

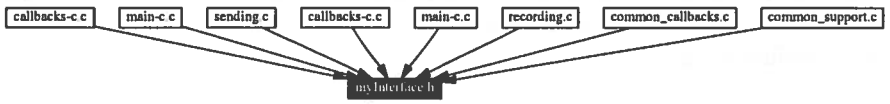

Figure 11.10: Include dependency graph for myInterface.h

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 then 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 consistant with the set of globals created by the user interface builder tool for some elements of the user interface widgets.

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 dor 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:

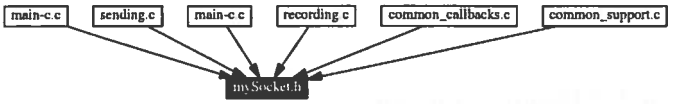

Figure 11.11: Include dependency graph for mySocket.h

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

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.

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"
```

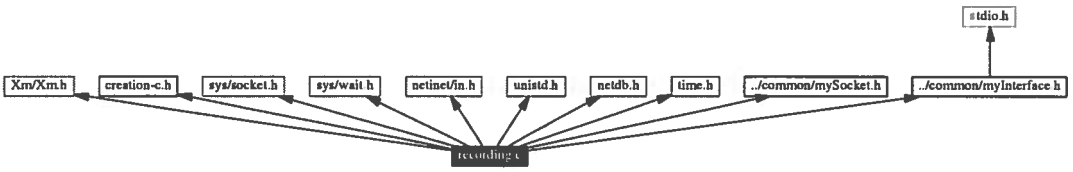

Figure 11.12: Include dependency graph for recording.c

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.

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"
```

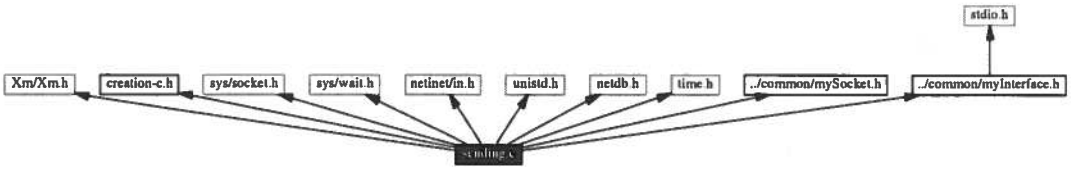

Figure 11.13: Include dependency graph for sending.c

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.

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.

Chapter 13

Software Index

Index

Appendix A

Common Source Code

```
/**
    \file common_callbacks.c
    \ingroup mod_common

    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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since August 2002

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

*/

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

/// the shared, global variable structure
extern globalMemory shared;
/// the global reference to the file browser window
extern Widget   inputFileSelectionBox;
/// the global reference to the help display window
extern Widget   aboutHelpBulletinBoard;


/**
    \ingroup mod_common
    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.
    \param 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)
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
debugToggleCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    if( XmToggleButtonGetState( w ) == True )
        shared.globalDebugMode = 1;
    else
        shared.globalDebugMode = 0;
}

/**
    \ingroup mod_common
    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.
```



```
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
aboutHelpCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    /* show the small message dialog */
    XtManageChild(aboutHelpBulletinBoard);
}

/**
    \ingroup mod_common
    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.
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
aboutDismissCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    /* remove the file selection dialog */
    XtUnmanageChild(aboutHelpBulletinBoard);
}

/**
    \ingroup mod_common
    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.
    \todo We should provide a mechanism to change the name
    of the html document used for help.
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to client data
    defined for the callback, in this case the name of the file to open
    \param call_data the standard event information provided by X11-Motif
*/
void
usageHelpCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    char commandLine[MAX_NAME_TEXT_SIZE];
    strcpy(commandLine, "netscape ");
    strcat(commandLine, (char *)client_data);
    strcat(commandLine, " &");
    system(commandLine);
}

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

```
133 */
134 void
135 browseButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
136 {
137     /* show the file selection dialog */
138     XtManageChild(inputFileSelectionBox);
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
154 cancelFileDialogCallback( Widget w, XtPointer client_data, XtPointer call_data)
155 {
156     /* remove the file selection dialog */
157     XtUnmanageChild(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
172 quitButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
173 {
174     if( shared.childProcess != 0 )
175     /* kill the current child process if any */
176         kill( shared.childProcess, SIGTERM );
177
178     /* we close the input file, close the socket, and exit the program */
179     if( shared.fileDescrip != NULL )
180         fclose( shared.fileDescrip );
181
182     /* close down the socket */
183     close( shared.socketNumb );
184
185     exit(1);
186 }
187
```

```
/**
    \file common_support.c
    \ingroup mod_common

   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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since May 2002

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

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

/**
    \ingroup mod_common
    This function is declaired '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.
    \param 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.
    \param length is the length of the existing character array, and on
    return it will be equal to the total new length of the string.
    \param text is the array of characters to be appended to the existing(?)
    string
*/
static void
AppendString(char **return_text, int *length, char *text)
{
    *length += strlen(text);
    if ( *return_text == (char *)NULL )
    {
        *return_text = (char *)XtCalloc(*length + 1, sizeof(char));
    }
    else
    {
        *return_text = (char *)XtRealloc(*return_text, *length + 1);
    }
    strcat(*return_text, text);
}

#ifndef XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
#define XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG XmSTRING_COMPONENT_CHARSET
#endif

/**
    \ingroup mod_common
    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.
    \param xmstr is the Motif XmString variable to be converted.
    \param return_text is the character array that will hold the converted
    text
    \return true or false, i.e. whether a conversion to character array
    was possible or not
*/
Boolean
XmStringToString(XmString xmstr, char **return_text)
{
    Boolean                     done = False;
    XmStringContext             context;
    char                        *text;
    XmStringCharSet             tag;
    XmStringDirection           direction;
    int                         length;
    XmStringComponentType       type;
    XmStringComponentType       unknownTag;
    unsigned short              unknownLen;
    unsigned char               *unknownVal;

    length = 0;
    *return_text = NULL;

    if ( !XmStringInitContext(&context, xmstr) ) return(False);

    while( !done )
    {
        type = XmStringGetNextComponent(context, &text, &tag, &direction,
                                        &unknownTag, &unknownLen, &unknownVal);
        switch( type )
        {
        case XmSTRING_COMPONENT_TEXT:
        case XmSTRING_COMPONENT_LOCALE_TEXT:
            AppendString(return_text, &length, text);
            AppendString(return_text, &length, "\0");
            XtFree(text);
            break;
        case XmSTRING_COMPONENT_SEPARATOR:
            AppendString(return_text, &length, "\n");
            break;

        case XmSTRING_COMPONENT_USER_BEGIN:
        case XmSTRING_COMPONENT_USER_END:
            /* IGNORED */
            break;

        case XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG:
            XtFree((char *)tag);
            break;

        case XmSTRING_COMPONENT_DIRECTION:
            break;

        case XmSTRING_COMPONENT_UNKNOWN:
            XtFree((char *)unknownVal);
            break;

        case XmSTRING_COMPONENT_END:
        default:
            done = True;
        }
    }

    XmStringFreeContext(context);
    return(True);
```

```
}

/* read the number of messages in the file */
/**
    \ingroup mod_common
    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.
    \param fileD is the file descriptor for the (already opened) file
     which contains the socket messages and delays.
    \return the number of messages in the file, or 0 if there was an
     error reading the file or an invalid file.
*/
int
getNumbMessages( FILE *fileD )
        {
        long dummy;
        int length;
        char buffer[MAX_MESSAGE_SIZE];
        int itemsRead;
        int count;

        count = 0;
        for(;;count++)
                {
                /* read the first part of the delay from the file */
                itemsRead = fread( &dummy, sizeof(long), 1, fileD );
                if( itemsRead != 1 )
                        return count;
                /* read the second part of the delay from the file */
                itemsRead = fread( &dummy, sizeof(long), 1, fileD );
                if( itemsRead != 1 )
                        return count;
                /* read the size of the message from the file */
                itemsRead = fread( &length, sizeof(int), 1, fileD );
                if( itemsRead != 1 )
                        return count;
                if( length < 0 || length > MAX_MESSAGE_SIZE )
                        return 0;
                itemsRead = fread( buffer, length, 1, fileD  );
                if( itemsRead != 1 )
                        return count;
                }
        }

/* read the number of messages in a user supplied file name */
/**
    \ingroup mod_common
    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.
    \param fileName is the name of a file which (we hope) contains socket
    messages and delay values.
    \return the number of messages in the file, or 0 if there was an
     error reading the file or an invalid file.
*/
int
readNumberMessages( char *fileName )
        {
        extern globalMemory shared;
        int count;

```



```
        /* close the current file (if one open), open and size the new */
        if( strlen( shared.inputFileName ) != 0 )
                fclose( shared.fileDescrip );

        if( !(shared.fileDescrip = fopen( fileName, "r")) )
                {
                printf("\n(support) Cannot open file '%s'\n", fileName );
                count = 0;
                }
        else
                {
                /* if there is a valid input file, get the number of messages in it */
                count = getNumbMessages( shared.fileDescrip );
                }
        /* reset the file read pointer to the start of the file */
        fclose( shared.fileDescrip );
        shared.fileDescrip = fopen( fileName, "r");
        if( shared.globalDebugMode == 1 )
                printf("File message count =  %d\n", count);
        return count;
        }

```

```
/**
    \file myInterface.h
    \ingroup mod_common

    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 consistant with the set of globals created by the user interface
    builder tool for some elements of the user interface widgets.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since April 2002

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

#include <stdio.h>

/**
    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_NAME_TEXT_SIZE 124

/**
    Motif GUI interface defined constants
    This constant will specify the biggest number of text
    digits we will deal with from small text fields.
*/
#define MAX_NUMBER_TEXT_SIZE 20

/**

  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 dor 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.
    \remark This requirement for global scope would disappear if
    the code was re-written into the C++ computer language.
*/
typedef struct _globalMemory
{
        /// the character array used to hold the name of the input file
```



```
        char inputFileName[MAX_NAME_TEXT_SIZE];
        /// the character array used to hold the name of the output file
        char outputFileName[MAX_NAME_TEXT_SIZE];
        /// the network address to send messages to, or output file name
        char address[MAX_NAME_TEXT_SIZE];
        /// port number (as a text string) to send/listen to
        char portNumbText[MAX_NUMBER_TEXT_SIZE];
        /// port number (as an integer) to send/listen messages to
        int  portNumb;
        /// the input or output socket file, as a file descriptor
        FILE *fileDescrip;
        /// the number of messages currently stored in the file
        int messageNumber;
        /// the message number in file to start sending
        int  startMesNumb;
        /// the message number in file to stop sending
        int  endMesNumb;
        /// the socket we create when sending/reading via the network
        int socketNumb;
        /// tha rate we send, ie faster/slower than recorded
        float speedFactor;
        /// when this flag is set, print extra debug statements
        int globalDebugMode;
        /// the process id of the sending/reading sub-process
        int childProcess;
        /// the application contect of main process
        XtAppContext mainAppContext;
}
globalMemory;


#ifndef DOXYGEN_SHOULD_IGNORE_THIS
/*-----------------------------------------------------------------------------
 *   These are function prototypes for all functions.
 *   They are already collected by doxygen in appropriate 'modules' and
 *    documented there, so these (required) prototypes would generate a
 *    duplicate list without the 'ifndef' statement above.
 */

/* global functions in support source file */
Boolean XmStringToString(XmString , char**);
int getNumbMessages( FILE *);
int readNumberMessages( char* );

/* global functions in sending source file */
void usageSend();
int createSendSocket( char *, int );
int sendSocketMessage( int, char*, int);
void resetSendButtons();
void sendProcessPause();
void sendProcessStop();
void sendProcessDone();
int sendMessages( FILE *, FILE * );
void fileToggleCallback( Widget, XtPointer, XtPointer );

/* global functions in reading source file */
void usageRecord();
int createReadSocket( int );
int readSocketMessage( int, char*, int);
void resetReadButtons();
void readProcessPause();
void readProcessStop();
void readProcessDone();
int readMessages( FILE *);

#endif /* DOXYGEN_SHOULD_IGNORE_THIS */
```

```
/**
    \file mySocket.h
    \ingroup mod_common

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

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since April 2002

    $Id: mySocket.h,v 1.3 2002/08/19 18:46:19 johnston Exp $
*/
/// the default value for the address of the local computer
#define LOCAL_ADDRESS "localhost"

/// the default value we will use for a network port number
#define DEFAULT_PORT_NUMBER 10025

/// the biggest socket message we will deal with
#define MAX_MESSAGE_SIZE 1024

```

Appendix B

Record Program Source Code

```
/*
 * README: This file is appended to at file generation time.
 * Edits can be made throughout the file
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */
#include <Xm/Xm.h>
#include <Xm/TextF.h>
#include <signal.h>
#include "creation-c.h"
#include "../common/myInterface.h"

/*
 * Standard includes for builtins.
 */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/*
 * Macros to make code look nicer between ANSI and K&R.
 */
#ifndef ARGLIST
#if (NeedFunctionPrototypes == 0)
#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;
#else
#define PROTOTYPE(p)    p
#define ARGLIST(p)      (
#define ARG(a, b)       a b,
#define GRA(a, b)       a b)
#ifdef __cplusplus
#define UARG(a, b)      a,
#define GRAU(a, b)      a)
#else
#define UARG(a, b)      a b,
#define GRAU(a, b)      a b)
#endif
#endif
#endif

Widget          BxFindTopShell PROTOTYPE((Widget));
WidgetList      BxWidgetIdsFromNames PROTOTYPE((Widget, char*, char*));

/**
    \file record/callbacks-c.c
    \ingroup mod_record

    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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since April 2002

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

/// the shared, global variable structure
extern globalMemory shared;

/**
    \ingroup mod_record
    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.
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
recordButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    char tmpBuf[MAX_NUMBER_TEXT_SIZE];

    /* read all the fields on the user interface to make sure we have
     *  the most up-to-date values for each option
     */

    /* get the output file name */
    strcpy(shared.outputFileName, XmTextFieldGetString( outputFileTextField ));

    /* the socket port number we are listening to */
    strcpy(tmpBuf, XmTextFieldGetString( portTextField ));
    if( sscanf( tmpBuf, "%d", &shared.portNumb) != 1 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }

    /* create socket */
    shared.socketNumb = createReadSocket( shared.portNumb );
    if( shared.socketNumb < 0 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }

    /* open file at beginning and reset message counter */
    shared.fileDescrip = fopen( shared.outputFileName, "w" );
    /* (re)set the frame counter */
    shared.messageNumber = 0;
    /* write the current value of the counter into the text field display */
    XmTextFieldSetString( numbRecTextField, "0" );

```

```
133     /* if there is a suspended child process already, terminate it first */
134     if( shared.childProcess != 0 )
135         /* kill the current (suspended) child process */
136         kill( shared.childProcess, SIGTERM );
137
138     /* we need to know when the (new) recording process is finished */
139     signal( SIGCHLD, readProcessDone );
140
141     /* fork a subprocess to run the long recording proceedure */
142     switch( 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( SIGABRT, readProcessStop );
149         readMessages( shared.fileDescrip );
150         /* when the record process is terminated, exit with signal set */
151         _exit(255);
152     case -1:
153         XBell( XtDisplay(w), 100 );
154         printf("Fork() of record messages failed!\n");
155         return;
156         }
157
158     /* the main (motif) program continues... */
159     if( shared.globalDebugMode == 1 )
160         printf("Read process started: pid %d !\n", shared.childProcess);
161      /* set pushbutton state - disable the start button, enable the stop/pause */
162     XtSetSensitive( recordPushButton, False );
163     XtSetSensitive( stopPushButton, True );
164     XtSetSensitive( pausePushButton, True );
165     XtSetSensitive( 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     push buttons. Since this button can only be active when the program
173     is actually reading, we know that there is a sub-process in existance
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     \param 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     \param client_data is a pointer to any (optional) client data
182     defined for the callback (none in this case)
183     \param 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 sigval sval;
189     /* abort the current running child process */
190     sval.sival_int = 1;
191     if( sigqueue( shared.childProcess, SIGABRT, sval ) < 0 )
192         if( shared.globalDebugMode == 1 )
193             printf("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     interface decides to respond to the user activating one of the 'Pause'
200     push buttons. Since this button can only be active when the program
201     is actually reading, we know that there is a sub-process in existance
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 message
205     counter into the text field on the display.
206     \param 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     \param client_data is a pointer to any (optional) client data
210     defined for the callback (none in this case)
211     \param call_data the standard event information provided by X11-Motif
212 */
213 void
214 pauseRecButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
215 {
216     union sigval sval;
217     /* disable the pause button, enable the record, stop and resume */
218     XtSetSensitive( recordPushButton, True );
219     XtSetSensitive( stopPushButton, True );
220     XtSetSensitive( pausePushButton, False );
221     XtSetSensitive( resumePushButton, True );
222     sval.sival_int = 1;
223     if( sigqueue( shared.childProcess, SIGSTOP, sval ) < 0 )
224         if( shared.globalDebugMode == 1 )
225             printf("Pause: cannot queue signal to child\n");
226 }
227
228 /**
229     \ingroup mod_record
230     This callback is the function that will be called when the Motif user
231     interface decides to respond to the user activating the 'Resume'
232     push buttons. Since this button can only be active when the program
233     is reading but in a paused state, we know that there is a sub-process
234     in existance and that it is not running.  We want to tell the
235     sub-process to continue, and change the state of active buttons in
236     the interface to show this change in the sub-process.
237     \param w the widget that caused the event and this callback.
238     \param client_data is a pointer to any (optional) client data
239     defined for the callback (none in this case)
240     \param call_data the standard event information provided by X11-Motif
241 */
242 void
243 resumeButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
244 {
245     union sigval sval;
246     /* disable the resume button, reenable the pause */
247     XtSetSensitive( recordPushButton, False );
248     XtSetSensitive( stopPushButton, True );
249     XtSetSensitive( pausePushButton, True );
250     XtSetSensitive( resumePushButton, False );
251     /* start recording messages into current position in file */
252     sval.sival_int = 1;
253     if( sigqueue( shared.childProcess, SIGCONT, sval ) < 0 )
254         if( shared.globalDebugMode == 1 )
255             printf("Resume: cannot queue signal to child\n");
256 }
257
258
259 /**
260     \ingroup mod_record
261     This callback is the function that will be called when the Motif user
262     interface puts up a file selection dialog and the user selects a file
263     and presses the 'OK' button.  We try to check the validity of the
264     new file name by trying to read how many socket messages it
```

```
265     contains.  If the file is invalid, the user interface will just
266     'beep' as the user trys to 'ok' the selection of an improper file.
267     \param w the widget that caused the event and this callback.  The push
268     button (ok) on the file selection dialog in this case.
269     \param client_data is a pointer to any (optional) client data
270     defined for the callback (none in this case)
271     \param call_data the standard event information provided by X11-Motif
272 */
273 void
274 okFileDialogCallback( Widget w, XtPointer client_data, XtPointer call_data)
275 {
276     char        *file;
277     char        currentDirectory[MAX_NAME_TEXT_SIZE];
278     int         result;
279     XmFileSelectionBoxCallbackStruct    *fsb =
280         (XmFileSelectionBoxCallbackStruct *)call_data;
281
282     if ( XmStringToString(fsb->value, &file) )
283     {
284         /* copy the filename selected, but don't copy the directory part */
285         getwd( currentDirectory );
286         /* if the first part of the selected file is the same, cut it off */
287         result = strncmp( file, currentDirectory, strlen(currentDirectory));
288         if( result == 0 )
289             {
290             shared.outputFileName[0] = '.'; /* substitute for current dir */
291             strcpy( shared.outputFileName+1, file+strlen(currentDirectory) );
292             }
293         else
294             /* better copy the whole directory */
295             strcpy( shared.outputFileName, file );
296         XmTextFieldSetString( outputFileTextField, shared.outputFileName );
297         XtFree((char *)file);
298     }
299     else
300         printf("Invalid Selection\n");
301
302     /* remove the file selection dialog */
303     XtUnmanageChild(inputFileSelectionBox);
304 }
305
```

```
/*
 * README: Portions of this file are merged at file generation
 * time. Edits can be made *only* in between specified code blocks, look
 * for keywords <Begin user code> and <End user code>.
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */


/* Begin user code block <file_comments> */
/**
    \file record/creation-c.c
    \ingroup mod_record

    In this source file is the source code generated automatically
    by the user interface builder software.  It will use this source
    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since July 2002

    $Id: creation-c.c,v 1.6 2002/08/19 19:55:02 johnston Exp $
*/
/* End user code block <file_comments> */

#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>

/*
 * Global declarations are now stored in the header file.
 *
 * If DECLARE_BX_GLOBALS is defined then this header file
 * declares the globals, otherwise it just externs them.
 */
#define DECLARE_BX_GLOBALS

/*
 * Globally included information.
 */


/*
 * Common constant and pixmap declarations.
 */
```



```
#include "creation-c.h"

/*
 * Convenience functions from utilities file.
 */
extern void RegisterBxConverters(XtAppContext);
extern XtPointer BX_CONVERT(Widget, char *, char *, int, Boolean *);
extern XtPointer BX_DOUBLE(double);
extern XtPointer BX_SINGLE(float);
extern void BX_MENU_POST(Widget, XtPointer, XEvent *, Boolean *);
extern Pixmap XPM_PIXMAP(Widget, char **);
extern void BX_SET_BACKGROUND_COLOR(Widget, ArgList, Cardinal *, Pixel);

/*
 * Declarations for callbacks and handlers.
 */
#ifndef DOXYGEN_SHOULD_IGNORE_THIS
extern void aboutDismissCallback(Widget, XtPointer, XtPointer);
extern void okFileDialogCallback(Widget, XtPointer, XtPointer);
extern void cancelFileDialogCallback(Widget, XtPointer, XtPointer);
extern void browseButtonCallback(Widget, XtPointer, XtPointer);
extern void recordButtonCallback(Widget, XtPointer, XtPointer);
extern void stopButtonCallback(Widget, XtPointer, XtPointer);
extern void pauseRecButtonCallback(Widget, XtPointer, XtPointer);
extern void resumeButtonCallback(Widget, XtPointer, XtPointer);
extern void quitButtonCallback(Widget, XtPointer, XtPointer);
extern void debugToggleCallback(Widget, XtPointer, XtPointer);
extern void usageHelpCallback(Widget, XtPointer, XtPointer);
extern void aboutHelpCallback(Widget, XtPointer, XtPointer);
#endif
/*
 * Create the mainWindow hierarchy of widgets.
 */
Widget
CreatemainWindow(Widget parent)
{
    Cardinal ac = 0;
    Arg      args[256];
    Cardinal cdc = 0;
    Boolean  argok = False;
    Widget   mainWindow;
    Widget   aboutHelpDialogShell;
    Widget   aboutHelpLabel;
    Widget   aboutDismissPushButton;
    Widget   selectFileDialog;
    Widget   menuBar;
    Widget   fileCascadeButton;
    Widget   pulldownMenu;
    Widget   openFilePushButton;
    Widget   separator1;
    Widget   recordFilePushButton;
    Widget   stopFilePushButton;
    Widget   pauseFilePushButton;
    Widget   resumeFilePushButton;
    Widget   separator2;
    Widget   quitFilePushButton;
    Widget   optionsCascadeButton;
    Widget   pulldownMenu1;
    Widget   helpCascadeButton;
    Widget   pulldownMenu2;
    Widget   usageHelpPushButton;
    Widget   aboutHelpPushButton;
    Widget   form;
    Widget   numberRecordedLabel;
    Widget   label;
    Widget   quitPushButton;
```

```
133     Widget   portLabel;
134     Widget   browseButton;
135     Widget   outputFileLabel;
136
137     /*
138      * Register the converters for the widgets.
139      */
140     RegisterBxConverters(XtWidgetToApplicationContext(parent));
141     XtInitializeWidgetClass((WidgetClass)xmMainWindowWidgetClass);
142     XtInitializeWidgetClass((WidgetClass)xmDialogShellWidgetClass);
143     XtInitializeWidgetClass((WidgetClass)xmBulletinBoardWidgetClass);
144     XtInitializeWidgetClass((WidgetClass)xmScrolledWindowWidgetClass);
145     XtInitializeWidgetClass((WidgetClass)xmLabelWidgetClass);
146     XtInitializeWidgetClass((WidgetClass)xmPushButtonWidgetClass);
147     XtInitializeWidgetClass((WidgetClass)xmFileSelectionBoxWidgetClass);
148     XtInitializeWidgetClass((WidgetClass)xmRowColumnWidgetClass);
149     XtInitializeWidgetClass((WidgetClass)xmCascadeButtonWidgetClass);
150     XtInitializeWidgetClass((WidgetClass)xmRowColumnWidgetClass);
151     XtInitializeWidgetClass((WidgetClass)xmSeparatorWidgetClass);
152     XtInitializeWidgetClass((WidgetClass)xmToggleButtonWidgetClass);
153     XtInitializeWidgetClass((WidgetClass)xmFormWidgetClass);
154     XtInitializeWidgetClass((WidgetClass)xmTextFieldWidgetClass);
155
156     ac = 0;
157     XtSetArg(args[ac], XmNx, 562); ac++;
158     XtSetArg(args[ac], XmNy, 656); ac++;
159     XtSetArg(args[ac], XmNwidth, 619); ac++;
160     XtSetArg(args[ac], XmNheight, 290); ac++;
161     mainWindow = XmCreateMainWindow(parent,
162         "mainWindow",
163         args,
164         ac);
165
166     ac = 0;
167     XtSetArg(args[ac], XmNwidth, 619); ac++;
168     XtSetArg(args[ac], XmNheight, 28); ac++;
169     menuBar = XmCreateMenuBar(mainWindow,
170         "menuBar",
171         args,
172         ac);
173     XtManageChild(menuBar);
174
175     ac = 0;
176     XtSetArg(args[ac], XmNmnemonic, 'F'); ac++;
177     XtSetArg(args[ac], XmNlabelString,
178         BX_CONVERT(menuBar, "File",
179         XmRXmString, 0, &argok)); if (argok) ac++;
180     XtSetArg(args[ac], XmNx, 5); ac++;
181     XtSetArg(args[ac], XmNy, 2); ac++;
182     XtSetArg(args[ac], XmNwidth, 43); ac++;
183     XtSetArg(args[ac], XmNheight, 24); ac++;
184     fileCascadeButton = XmCreateCascadeButton(menuBar,
185         "fileCascadeButton",
186         args,
187         ac);
188     XtManageChild(fileCascadeButton);
189
190     ac = 0;
191     XtSetArg(args[ac], XmNx, 0); ac++;
192     XtSetArg(args[ac], XmNy, 0); ac++;
193     XtSetArg(args[ac], XmNwidth, 74); ac++;
194     XtSetArg(args[ac], XmNheight, 154); ac++;
195     pulldownMenu = XmCreatePulldownMenu(XtParent(fileCascadeButton),
196         "pulldownMenu",
197         args,
198         ac);
```



```
199
200     ac = 0;
201     XtSetArg(args[ac], XmNmnemonic, 'O'); ac++;
202     XtSetArg(args[ac], XmNlabelString,
203         BX_CONVERT(pulldownMenu, "Open...",
204         XmRXmString, 0, &argok)); if (argok) ac++;
205     openFilePushButton = XmCreatePushButton(pulldownMenu,
206         "openFilePushButton",
207         args,
208         ac);
209     XtManageChild(openFilePushButton);
210     XtAddCallback(openFilePushButton, XmNactivateCallback, browseButtonCallback, (XtPoint
er)0);
211
212     ac = 0;
213     separator1 = XmCreateSeparator(pulldownMenu,
214         "separator1",
215         args,
216         ac);
217     XtManageChild(separator1);
218
219     ac = 0;
220     XtSetArg(args[ac], XmNmnemonic, 'E'); ac++;
221     XtSetArg(args[ac], XmNlabelString,
222         BX_CONVERT(pulldownMenu, "Record",
223         XmRXmString, 0, &argok)); if (argok) ac++;
224     recordFilePushButton = XmCreatePushButton(pulldownMenu,
225         "recordFilePushButton",
226         args,
227         ac);
228     XtManageChild(recordFilePushButton);
229     XtAddCallback(recordFilePushButton, XmNactivateCallback, recordButtonCallback, (XtPoi
nter)0);
230
231     ac = 0;
232     XtSetArg(args[ac], XmNmnemonic, 'S'); ac++;
233     XtSetArg(args[ac], XmNlabelString,
234         BX_CONVERT(pulldownMenu, "Stop",
235         XmRXmString, 0, &argok)); if (argok) ac++;
236     stopFilePushButton = XmCreatePushButton(pulldownMenu,
237         "stopFilePushButton",
238         args,
239         ac);
240     XtManageChild(stopFilePushButton);
241     XtAddCallback(stopFilePushButton, XmNactivateCallback, stopButtonCallback, (XtPointer
)0);
242
243     ac = 0;
244     XtSetArg(args[ac], XmNmnemonic, 'P'); ac++;
245     XtSetArg(args[ac], XmNlabelString,
246         BX_CONVERT(pulldownMenu, "Pause",
247         XmRXmString, 0, &argok)); if (argok) ac++;
248     pauseFilePushButton = XmCreatePushButton(pulldownMenu,
249         "pauseFilePushButton",
250         args,
251         ac);
252     XtManageChild(pauseFilePushButton);
253     XtAddCallback(pauseFilePushButton, XmNactivateCallback, pauseRecButtonCallback, (XtPo
inter)0);
254
255     ac = 0;
256     XtSetArg(args[ac], XmNmnemonic, 'R'); ac++;
257     XtSetArg(args[ac], XmNlabelString,
258         BX_CONVERT(pulldownMenu, "Resume",
259         XmRXmString, 0, &argok)); if (argok) ac++;
260     resumeFilePushButton = XmCreatePushButton(pulldownMenu,
```

```
261         "resumeFilePushButton",
262         args,
263         ac);
264     XtManageChild(resumeFilePushButton);
265     XtAddCallback(resumeFilePushButton, XmNactivateCallback, resumeButtonCallback, (XtPoi
nter)0);
266
267     ac = 0;
268     separator2 = XmCreateSeparator(pulldownMenu,
269         "separator2",
270         args,
271         ac);
272     XtManageChild(separator2);
273
274     ac = 0;
275     XtSetArg(args[ac], XmNmnemonic, 'Q'); ac++;
276     XtSetArg(args[ac], XmNlabelString,
277         BX_CONVERT(pulldownMenu, "Quit",
278         XmRXmString, 0, &argok)); if (argok) ac++;
279     quitFilePushButton = XmCreatePushButton(pulldownMenu,
280         "quitFilePushButton",
281         args,
282         ac);
283     XtManageChild(quitFilePushButton);
284     XtAddCallback(quitFilePushButton, XmNactivateCallback, quitButtonCallback, (XtPointer
)0);
285
286     ac = 0;
287     XtSetArg(args[ac], XmNsubMenuId, pulldownMenu); ac++;
288     XtSetValues(fileCascadeButton, args, ac);
289
290     ac = 0;
291     XtSetArg(args[ac], XmNmnemonic, 'O'); ac++;
292     XtSetArg(args[ac], XmNlabelString,
293         BX_CONVERT(menuBar, "Options",
294         XmRXmString, 0, &argok)); if (argok) ac++;
295     XtSetArg(args[ac], XmNx, 55); ac++;
296     XtSetArg(args[ac], XmNy, 2); ac++;
297     XtSetArg(args[ac], XmNwidth, 74); ac++;
298     XtSetArg(args[ac], XmNheight, 24); ac++;
299     optionsCascadeButton = XmCreateCascadeButton(menuBar,
300         "optionsCascadeButton",
301         args,
302         ac);
303     XtManageChild(optionsCascadeButton);
304
305     ac = 0;
306     XtSetArg(args[ac], XmNx, 0); ac++;
307     XtSetArg(args[ac], XmNy, 0); ac++;
308     XtSetArg(args[ac], XmNwidth, 118); ac++;
309     XtSetArg(args[ac], XmNheight, 30); ac++;
310     pulldownMenu1 = XmCreatePulldownMenu(XtParent(optionsCascadeButton),
311         "pulldownMenu1",
312         args,
313         ac);
314
315     ac = 0;
316     XtSetArg(args[ac], XmNlabelString,
317         BX_CONVERT(pulldownMenu1, "debug mode",
318         XmRXmString, 0, &argok)); if (argok) ac++;
319     debugToggleButton = XmCreateToggleButton(pulldownMenu1,
320         "debugToggleButton",
321         args,
322         ac);
323     XtManageChild(debugToggleButton);
324     XtAddCallback(debugToggleButton, XmNvalueChangedCallback, debugToggleCallback, (XtPoi
```



```
nter)0);
325
326     ac = 0;
327     XtSetArg(args[ac], XmNsubMenuId, pulldownMenu1); ac++;
328     XtSetValues(optionsCascadeButton, args, ac);
329
330     ac = 0;
331     XtSetArg(args[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
332     XtSetArg(args[ac], XmNmnemonic, 'H'); ac++;
333     XtSetArg(args[ac], XmNlabelString,
334         BX_CONVERT(menuBar, "Help",
335         XmRXmString, 0, &argok)); if (argok) ac++;
336     XtSetArg(args[ac], XmNx, 564); ac++;
337     XtSetArg(args[ac], XmNy, 2); ac++;
338     XtSetArg(args[ac], XmNwidth, 50); ac++;
339     XtSetArg(args[ac], XmNheight, 24); ac++;
340     helpCascadeButton = XmCreateCascadeButton(menuBar,
341         "helpCascadeButton",
342         args,
343         ac);
344     XtManageChild(helpCascadeButton);
345
346     ac = 0;
347     XtSetArg(args[ac], XmNx, 0); ac++;
348     XtSetArg(args[ac], XmNy, 0); ac++;
349     XtSetArg(args[ac], XmNwidth, 63); ac++;
350     XtSetArg(args[ac], XmNheight, 54); ac++;
351     pulldownMenu2 = XmCreatePulldownMenu(XtParent(helpCascadeButton),
352         "pulldownMenu2",
353         args,
354         ac);
355
356     ac = 0;
357     XtSetArg(args[ac], XmNmnemonic, 'U'); ac++;
358     XtSetArg(args[ac], XmNlabelString,
359         BX_CONVERT(pulldownMenu2, "Usage",
360         XmRXmString, 0, &argok)); if (argok) ac++;
361     usageHelpPushButton = XmCreatePushButton(pulldownMenu2,
362         "usageHelpPushButton",
363         args,
364         ac);
365     XtManageChild(usageHelpPushButton);
366     XtAddCallback(usageHelpPushButton, XmNactivateCallback, usageHelpCallback, (XtPointer
)"recordUsageHelp.html");
367
368     ac = 0;
369     XtSetArg(args[ac], XmNmnemonic, 'A'); ac++;
370     XtSetArg(args[ac], XmNlabelString,
371         BX_CONVERT(pulldownMenu2, "About",
372         XmRXmString, 0, &argok)); if (argok) ac++;
373     aboutHelpPushButton = XmCreatePushButton(pulldownMenu2,
374         "aboutHelpPushButton",
375         args,
376         ac);
377     XtManageChild(aboutHelpPushButton);
378     XtAddCallback(aboutHelpPushButton, XmNactivateCallback, aboutHelpCallback, (XtPointer
)0);
379
380     ac = 0;
381     XtSetArg(args[ac], XmNsubMenuId, pulldownMenu2); ac++;
382     XtSetValues(helpCascadeButton, args, ac);
383
384     ac = 0;
385     XtSetArg(args[ac], XmNresizePolicy, XmRESIZE_GROW); ac++;
386     XtSetArg(args[ac], XmNwidth, 619); ac++;
387     XtSetArg(args[ac], XmNheight, 262); ac++;
```

```
388     form = XmCreateForm(mainWindow,
389         "form",
390         args,
391         ac);
392     XtManageChild(form);
393
394     ac = 0;
395     XtSetArg(args[ac], XmNfontList,
396         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
397         XmRFontList, 0, &argok)); if (argok) ac++;
398     XtSetArg(args[ac], XmNlabelString,
399         BX_CONVERT(form, "Stop",
400         XmRXmString, 0, &argok)); if (argok) ac++;
401     XtSetArg(args[ac], XmNx, 160); ac++;
402     XtSetArg(args[ac], XmNy, 140); ac++;
403     XtSetArg(args[ac], XmNwidth, 74); ac++;
404     XtSetArg(args[ac], XmNheight, 35); ac++;
405     stopPushButton = XmCreatePushButton(form,
406         "stopPushButton",
407         args,
408         ac);
409     XtManageChild(stopPushButton);
410     XtAddCallback(stopPushButton, XmNactivateCallback, stopButtonCallback, (XtPointer)0);

411
412     ac = 0;
413     XtSetArg(args[ac], XmNeditable, False); ac++;
414     XtSetArg(args[ac], XmNx, 410); ac++;
415     XtSetArg(args[ac], XmNy, 70); ac++;
416     XtSetArg(args[ac], XmNwidth, 120); ac++;
417     XtSetArg(args[ac], XmNheight, 30); ac++;
418     numbRecTextField = XmCreateTextField(form,
419         "numbRecTextField",
420         args,
421         ac);
422     XtManageChild(numbRecTextField);
423
424     ac = 0;
425     XtSetArg(args[ac], XmNfontList,
426         BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
427         XmRFontList, 0, &argok)); if (argok) ac++;
428     XtSetArg(args[ac], XmNlabelString,
429         BX_CONVERT(form, "Numb. Rec.",
430         XmRXmString, 0, &argok)); if (argok) ac++;
431     XtSetArg(args[ac], XmNx, 330); ac++;
432     XtSetArg(args[ac], XmNy, 80); ac++;
433     XtSetArg(args[ac], XmNwidth, 70); ac++;
434     XtSetArg(args[ac], XmNheight, 19); ac++;
435     numberRecordedLabel = XmCreateLabel(form,
436         "numberRecordedLabel",
437         args,
438         ac);
439     XtManageChild(numberRecordedLabel);
440
441     ac = 0;
442     XtSetArg(args[ac], XmNalignment, XmALIGNMENT_CENTER); ac++;
443     XtSetArg(args[ac], XmNsensitive, False); ac++;
444     XtSetArg(args[ac], XmNfontList,
445         BX_CONVERT(form, "-*-times-bold-i-*-*-*-240-*-*-*-*-iso8859-1",
446         XmRFontList, 0, &argok)); if (argok) ac++;
447     XtSetArg(args[ac], XmNforeground,
448         BX_CONVERT(form, "#fffd7",
449         XmRPixel, 0, &argok)); if (argok) ac++;
450     XtSetArg(args[ac], XmNlabelString,
451         BX_CONVERT(form, "Record UDP Sockets",
452         XmRXmString, 0, &argok)); if (argok) ac++;
```



```
453     XtSetArg(args[ac], XmNx, 200); ac++;
454     XtSetArg(args[ac], XmNy, 200); ac++;
455     XtSetArg(args[ac], XmNwidth, 239); ac++;
456     XtSetArg(args[ac], XmNheight, 29); ac++;
457     label = XmCreateLabel(form,
458         "label",
459         args,
460         ac);
461     XtManageChild(label);
462
463     ac = 0;
464     XtSetArg(args[ac], XmNfontList,
465         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
466         XmRFontList, 0, &argok)); if (argok) ac++;
467     XtSetArg(args[ac], XmNlabelString,
468         BX_CONVERT(form, "Quit",
469         XmRXmString, 0, &argok)); if (argok) ac++;
470     XtSetArg(args[ac], XmNx, 20); ac++;
471     XtSetArg(args[ac], XmNy, 200); ac++;
472     XtSetArg(args[ac], XmNwidth, 52); ac++;
473     XtSetArg(args[ac], XmNheight, 39); ac++;
474     quitPushButton = XmCreatePushButton(form,
475         "quitPushButton",
476         args,
477         ac);
478     XtManageChild(quitPushButton);
479     XtAddCallback(quitPushButton, XmNactivateCallback, quitButtonCallback, (XtPointer)0);

480
481     ac = 0;
482     XtSetArg(args[ac], XmNfontList,
483         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
484         XmRFontList, 0, &argok)); if (argok) ac++;
485     XtSetArg(args[ac], XmNlabelString,
486         BX_CONVERT(form, "Resume",
487         XmRXmString, 0, &argok)); if (argok) ac++;
488     XtSetArg(args[ac], XmNx, 380); ac++;
489     XtSetArg(args[ac], XmNy, 140); ac++;
490     XtSetArg(args[ac], XmNwidth, 74); ac++;
491     XtSetArg(args[ac], XmNheight, 35); ac++;
492     resumePushButton = XmCreatePushButton(form,
493         "resumePushButton",
494         args,
495         ac);
496     XtManageChild(resumePushButton);
497     XtAddCallback(resumePushButton, XmNactivateCallback, resumeButtonCallback, (XtPointer
)0);
498
499     ac = 0;
500     XtSetArg(args[ac], XmNfontList,
501         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
502         XmRFontList, 0, &argok)); if (argok) ac++;
503     XtSetArg(args[ac], XmNlabelString,
504         BX_CONVERT(form, "Pause",
505         XmRXmString, 0, &argok)); if (argok) ac++;
506     XtSetArg(args[ac], XmNx, 280); ac++;
507     XtSetArg(args[ac], XmNy, 140); ac++;
508     XtSetArg(args[ac], XmNwidth, 59); ac++;
509     XtSetArg(args[ac], XmNheight, 35); ac++;
510     pausePushButton = XmCreatePushButton(form,
511         "pausePushButton",
512         args,
513         ac);
514     XtManageChild(pausePushButton);
515     XtAddCallback(pausePushButton, XmNactivateCallback, pauseRecButtonCallback, (XtPointe
r)0);
```

```
516
517     ac = 0;
518     XtSetArg(args[ac], XmNfontList,
519         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
520         XmRFontList, 0, &argok)); if (argok) ac++;
521     XtSetArg(args[ac], XmNlabelString,
522         BX_CONVERT(form, "Record",
523         XmRXmString, 0, &argok)); if (argok) ac++;
524     XtSetArg(args[ac], XmNx, 20); ac++;
525     XtSetArg(args[ac], XmNy, 140); ac++;
526     XtSetArg(args[ac], XmNwidth, 106); ac++;
527     XtSetArg(args[ac], XmNheight, 35); ac++;
528     recordPushButton = XmCreatePushButton(form,
529         "recordPushButton",
530         args,
531         ac);
532     XtManageChild(recordPushButton);
533     XtAddCallback(recordPushButton, XmNactivateCallback, recordButtonCallback, (XtPointer
)0);
534
535     ac = 0;
536     XtSetArg(args[ac], XmNx, 120); ac++;
537     XtSetArg(args[ac], XmNy, 70); ac++;
538     XtSetArg(args[ac], XmNwidth, 120); ac++;
539     XtSetArg(args[ac], XmNheight, 30); ac++;
540     portTextField = XmCreateTextField(form,
541         "portTextField",
542         args,
543         ac);
544     XtManageChild(portTextField);
545
546     ac = 0;
547     XtSetArg(args[ac], XmNfontList,
548         BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
549         XmRFontList, 0, &argok)); if (argok) ac++;
550     XtSetArg(args[ac], XmNlabelString,
551         BX_CONVERT(form, "Port Numb.",
552         XmRXmString, 0, &argok)); if (argok) ac++;
553     XtSetArg(args[ac], XmNx, 40); ac++;
554     XtSetArg(args[ac], XmNy, 80); ac++;
555     XtSetArg(args[ac], XmNwidth, 70); ac++;
556     XtSetArg(args[ac], XmNheight, 19); ac++;
557     portLabel = XmCreateLabel(form,
558         "portLabel",
559         args,
560         ac);
561     XtManageChild(portLabel);
562
563     ac = 0;
564     XtSetArg(args[ac], XmNfontList,
565         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
566         XmRFontList, 0, &argok)); if (argok) ac++;
567     XtSetArg(args[ac], XmNlabelString,
568         BX_CONVERT(form, "Browse",
569         XmRXmString, 0, &argok)); if (argok) ac++;
570     XtSetArg(args[ac], XmNx, 480); ac++;
571     XtSetArg(args[ac], XmNy, 10); ac++;
572     XtSetArg(args[ac], XmNwidth, 72); ac++;
573     XtSetArg(args[ac], XmNheight, 35); ac++;
574     browseButton = XmCreatePushButton(form,
575         "browseButton",
576         args,
577         ac);
578     XtManageChild(browseButton);
579     XtAddCallback(browseButton, XmNactivateCallback, browseButtonCallback, (XtPointer)0);
```



```
580
581     ac = 0;
582     XtSetArg(args[ac], XmNx, 110); ac++;
583     XtSetArg(args[ac], XmNy, 10); ac++;
584     XtSetArg(args[ac], XmNwidth, 370); ac++;
585     XtSetArg(args[ac], XmNheight, 30); ac++;
586     outputFileTextField = XmCreateTextField(form,
587         "outputFileTextField",
588         args,
589         ac);
590     XtManageChild(outputFileTextField);
591
592     ac = 0;
593     XtSetArg(args[ac], XmNfontList,
594         BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
595         XmRFontList, 0, &argok)); if (argok) ac++;
596     XtSetArg(args[ac], XmNlabelString,
597         BX_CONVERT(form, "Ouput File",
598         XmRXmString, 0, &argok)); if (argok) ac++;
599     XtSetArg(args[ac], XmNx, 20); ac++;
600     XtSetArg(args[ac], XmNy, 10); ac++;
601     XtSetArg(args[ac], XmNwidth, 110); ac++;
602     XtSetArg(args[ac], XmNheight, 19); ac++;
603     outputFileLabel = XmCreateLabel(form,
604         "outputFileLabel",
605         args,
606         ac);
607     XtManageChild(outputFileLabel);
608
609     ac = 0;
610     XtSetArg(args[ac], XmNwidth, 457); ac++;
611     XtSetArg(args[ac], XmNheight, 396); ac++;
612     selectFileDialog = XtCreatePopupShell("selectFileDialog",
613         xmDialogShellWidgetClass,
614         mainWindow,
615         args,
616         ac);
617
618     ac = 0;
619     XtSetArg(args[ac], XmNx, 186); ac++;
620     XtSetArg(args[ac], XmNy, 77); ac++;
621     XtSetArg(args[ac], XmNwidth, 457); ac++;
622     XtSetArg(args[ac], XmNheight, 396); ac++;
623     inputFileSelectionBox = XmCreateFileSelectionBox(selectFileDialog,
624         "outputFileSelectionBox",
625         args,
626         ac);
627     XtAddCallback(inputFileSelectionBox, XmNokCallback, okFileDialogCallback, (XtPointer)
0);
628     XtAddCallback(inputFileSelectionBox, XmNcancelCallback, cancelFileDialogCallback, (Xt
Pointer)0);
629
630     ac = 0;
631     XtSetArg(args[ac], XmNwidth, 284); ac++;
632     XtSetArg(args[ac], XmNheight, 312); ac++;
633     aboutHelpDialogShell = XtCreatePopupShell("aboutHelpDialogShell",
634         xmDialogShellWidgetClass,
635         mainWindow,
636         args,
637         ac);
638
639     ac = 0;
640     XtSetArg(args[ac], XmNresizePolicy, XmRESIZE_GROW); ac++;
641     XtSetArg(args[ac], XmNx, 8); ac++;
642     XtSetArg(args[ac], XmNy, 32); ac++;
643     XtSetArg(args[ac], XmNwidth, 284); ac++;
```

```
644     XtSetArg(args[ac], XmNheight, 312); ac++;
645     aboutHelpBulletinBoard = XmCreateBulletinBoard(aboutHelpDialogShell,
646         "aboutHelpBulletinBoard",
647         args,
648         ac);
649
650     ac = 0;
651     XtSetArg(args[ac], XmNscrollingPolicy, XmAUTOMATIC); ac++;
652     XtSetArg(args[ac], XmNx, 20); ac++;
653     XtSetArg(args[ac], XmNy, 20); ac++;
654     XtSetArg(args[ac], XmNwidth, 250); ac++;
655     XtSetArg(args[ac], XmNheight, 210); ac++;
656     aboutScrolledWindow = XmCreateScrolledWindow(aboutHelpBulletinBoard,
657         "aboutScrolledWindow",
658         args,
659         ac);
660     XtManageChild(aboutScrolledWindow);
661
662     ac = 0;
663     XtSetArg(args[ac], XmNalignment, XmALIGNMENT_CENTER); ac++;
664     XtSetArg(args[ac], XmNfontList,
665         BX_CONVERT(aboutScrolledWindow, "-*-helvetica-bold-r-*-*-*-130-*-*-*-*-iso8859-1"
,
666         XmRFontList, 0, &argok)); if (argok) ac++;
667     XtSetArg(args[ac], XmNlabelString,
668         BX_CONVERT(aboutScrolledWindow, ":::t\"   MimicSockets\":t\"\":t\"Record socket m
essages\":t\"into a file\":t\"\":t\"National Research Council\":t\"  of Canada (2002)\":t\"By: Da
n Johnston\":t\"All rights reserved!\"",
669         XmRXmString, 0, &argok)); if (argok) ac++;
670     XtSetArg(args[ac], XmNrecomputeSize, False); ac++;
671     XtSetArg(args[ac], XmNwidth, 220); ac++;
672     XtSetArg(args[ac], XmNheight, 250); ac++;
673     aboutHelpLabel = XmCreateLabel(aboutScrolledWindow,
674         "aboutHelpLabel",
675         args,
676         ac);
677     XtManageChild(aboutHelpLabel);
678
679     ac = 0;
680     XtSetArg(args[ac], XmNlabelString,
681         BX_CONVERT(aboutHelpBulletinBoard, "Dismiss",
682         XmRXmString, 0, &argok)); if (argok) ac++;
683     XtSetArg(args[ac], XmNx, 70); ac++;
684     XtSetArg(args[ac], XmNy, 250); ac++;
685     XtSetArg(args[ac], XmNwidth, 127); ac++;
686     XtSetArg(args[ac], XmNheight, 35); ac++;
687     aboutDismissPushButton = XmCreatePushButton(aboutHelpBulletinBoard,
688         "aboutDismissPushButton",
689         args,
690         ac);
691     XtManageChild(aboutDismissPushButton);
692     XtAddCallback(aboutDismissPushButton, XmNactivateCallback, aboutDismissCallback, (XtP
ointer)0);
693     ac = 0;
694     XtSetArg(args[ac], XmNmenuHelpWidget, helpCascadeButton); ac++;
695     XtSetValues(menuBar, args, ac);
696
697     ac = 0;
698     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
699     XtSetArg(args[ac], XmNleftOffset, 160); ac++;
700     XtSetArg(args[ac], XmNtopOffset, 140); ac++;
701     XtSetValues(stopPushButton, args, ac);
702
703     ac = 0;
704     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
705     XtSetArg(args[ac], XmNleftOffset, 410); ac++;
```



```
706     XtSetArg(args[ac], XmNtopOffset, 70); ac++;
707     XtSetValues(numRecTextField, args, ac);
708
709     ac = 0;
710     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
711     XtSetArg(args[ac], XmNleftOffset, 200); ac++;
712     XtSetArg(args[ac], XmNtopOffset, 200); ac++;
713     XtSetValues(label, args, ac);
714
715     ac = 0;
716     XtSetArg(args[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
717     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
718     XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_NONE); ac++;
719     XtSetArg(args[ac], XmNleftOffset, 20); ac++;
720     XtSetArg(args[ac], XmNtopOffset, 200); ac++;
721     XtSetValues(quitPushButton, args, ac);
722
723     ac = 0;
724     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
725     XtSetArg(args[ac], XmNleftOffset, 380); ac++;
726     XtSetArg(args[ac], XmNtopOffset, 140); ac++;
727     XtSetValues(resumePushButton, args, ac);
728
729     ac = 0;
730     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
731     XtSetArg(args[ac], XmNleftOffset, 280); ac++;
732     XtSetArg(args[ac], XmNtopOffset, 140); ac++;
733     XtSetValues(pausePushButton, args, ac);
734
735     ac = 0;
736     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
737     XtSetArg(args[ac], XmNleftOffset, 20); ac++;
738     XtSetArg(args[ac], XmNtopOffset, 140); ac++;
739     XtSetValues(recordPushButton, args, ac);
740
741     ac = 0;
742     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
743     XtSetArg(args[ac], XmNleftOffset, 120); ac++;
744     XtSetArg(args[ac], XmNtopOffset, 70); ac++;
745     XtSetValues(portTextField, args, ac);
746
747     ac = 0;
748     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
749     XtSetArg(args[ac], XmNleftOffset, 40); ac++;
750     XtSetArg(args[ac], XmNtopOffset, 80); ac++;
751     XtSetValues(portLabel, args, ac);
752
753     ac = 0;
754     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
755     XtSetArg(args[ac], XmNleftOffset, 480); ac++;
756     XtSetArg(args[ac], XmNtopOffset, 10); ac++;
757     XtSetValues(browseButton, args, ac);
758
759     ac = 0;
760     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
761     XtSetArg(args[ac], XmNleftOffset, 110); ac++;
762     XtSetArg(args[ac], XmNtopOffset, 10); ac++;
763     XtSetValues(outputFileTextField, args, ac);
764
765     ac = 0;
766     XtSetArg(args[ac], XmNtopAttachment, XmATTACH_POSITION); ac++;
767     XtSetArg(args[ac], XmNrightAttachment, XmATTACH_POSITION); ac++;
768     XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
769     XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_NONE); ac++;
770     XtSetArg(args[ac], XmNbottomOffset, 0); ac++;
771     XtSetArg(args[ac], XmNtopPosition, 4); ac++;
```

```
772     XtSetArg(args[ac], XmNrightPosition, 21); ac++;
773     XtSetArg(args[ac], XmNleftOffset, 20); ac++;
774     XtSetValues(outputFileLabel, args, ac);
775
776
777
778     /* Begin user code block <end_CreatemainWindow> */
779     /* End user code block <end_CreatemainWindow> */
780     return( mainWindow );
781 }
```

```
 1  /*
 2   * WARNING: This file is overwritten at code generation time.
 3   * Any changes to this file will be lost.
 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_BX_GLOBALS
21  #define EXTERNAL
22  #else
23  #define EXTERNAL extern
24  #endif
25
26  /*
27   * Start Global Widget Declarations.
28   */
29  EXTERNAL Widget   aboutHelpBulletinBoard;
30  EXTERNAL Widget   aboutScrolledWindow;
31  EXTERNAL Widget   inputFileSelectionBox;
32  EXTERNAL Widget   debugToggleButton;
33  EXTERNAL Widget   stopPushButton;
34  EXTERNAL Widget   numbRecTextField;
35  EXTERNAL Widget   resumePushButton;
36  EXTERNAL Widget   pausePushButton;
37  EXTERNAL Widget   recordPushButton;
38  EXTERNAL Widget   portTextField;
39  EXTERNAL Widget   outputFileTextField;
40  /*
41   * End Global Widget Declarations.
42   */
43  #endif
```

```
/*
 * README: Portions of this file are merged at file generation
 * time. Edits can be made *only* in between specified code blocks, look
 * for keywords <Begin user code> and <End user code>.
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */


/* Begin user code block <file_comments> */
/**
    \file record/main-c.c
    \ingroup mod_record

    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 responce 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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since August 2002

    $Id: main-c.c,v 1.6 2002/08/19 18:46:03 johnston Exp $
*/
/* End user code block <file_comments> */

/*
 * Motif required Headers
 */
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#if (XmVersion >= 1002)
#include <Xm/RepType.h>
#endif
#include <Xm/MwmUtil.h>

/*
 * Globally included information.
 */


/*
 * Headers for classes used in this program
 */
```



```

/*
 * Common constant and pixmap declarations.
 */
#include "creation-c.h"

/*
 * Convenience functions from utilities file.
 */
extern void RegisterBxConverters(XtAppContext);
extern XtPointer BX_CONVERT(Widget, char *, char *, int, Boolean *);
extern XtPointer BX_DOUBLE(double);
extern XtPointer BX_SINGLE(float);
extern void BX_MENU_POST(Widget, XtPointer, XEvent *, Boolean *);
extern Pixmap XPM_PIXMAP(Widget, char **);
extern void BX_SET_BACKGROUND_COLOR(Widget, ArgList, Cardinal *, Pixel);

/* Begin user code block <globals> */
#include "../common/mySocket.h"
#include "../common/myInterface.h"
/// the block of user defined globals shared by this interface
globalMemory shared;
/* End user code block <globals> */

/*
 * Change this line via the Output Application Names Dialog.
 */
#define BX_APP_CLASS "BuilderProduct"

int main( int argc, char **argv)
{
    Widget       parent;
    XtAppContext app;
    Arg          args[256];
    Cardinal     ac;
    Boolean      argok=False;
    Widget   topLevelShell;
    Widget   mainWindow;

    /* Begin user code block <declarations> */
    /* End user code block <declarations> */

    /*
     * The applicationShell is created as an unrealized
     * parent for multiple topLevelShells.  The topLevelShells
     * are created as popup children of the applicationShell.
     * This is a recommendation of Paul Asente & Ralph Swick in
     * _X_Window_System_Toolkit_ p. 677.
     */

    parent = XtVaAppInitialize ( &app, BX_APP_CLASS, NULL, 0,
#ifndef XlibSpecificationRelease
    (Cardinal *) &argc,
#else
#if (XlibSpecificationRelease>=5)
    &argc,
#else
    (Cardinal *) &argc,
#endif
#endif
    argv, NULL,
    NULL );

    RegisterBxConverters(app);
#if (XmVersion >= 1002)
    XmRepTypeInstallTearOffModelConverter();
```

```
133 #endif
134
135     /* Begin user code block <create_shells> */
136     ac = (int) argok;  /* use the variable to surpress compiler warnings */
137     /* End user code block <create_shells> */
138
139     /*
140      * Create classes and widgets used in this program.
141      */
142
143     /* Begin user code block <create_topLevelShell> */
144     /**
145     \ingroup mod_record
146
147     This main program is mostly computer generated.  We only modify it
148     (between the "user code block" comments) to create our global memory
149     structure and initalize it based on the command line arguments.
150
151     */
152     /* copy our current application context to make it global */
153     shared.mainAppContext = app;
154
155     /* start by assuming there is no supplied args */
156     shared.outputFileName[0] = shared.portNumbText[0] = '\0';
157
158     switch ( argc )
159     {
160     case 3:
161         /* we should have both args, port and filename */
162         /* process the filename arg */
163         if( !(shared.fileDescrip = fopen( argv[2], "w")) )
164               {
165               printf("\n(main) Cannot open file '%s'\n", argv[2] );
166               usageRecord();
167               exit(1);
168               }
169         strcpy( shared.outputFileName, argv[2] );
170         /* no 'break' here, fall through to process arg 2 */
171     case 2:
172         /* we have 1 arg, the port number */
173         /* process the port number here */
174         if( sscanf( argv[1], "%u", &shared.portNumb ) != 1 )
175               {
176               printf("\nNot a valid port number '%s'\n", argv[1] );
177               usageRecord();
178               exit(1);
179               }
180         strcpy( shared.portNumbText, argv[1] );
181         break;
182     case 1:
183         /* we have no user supplied command args */
184         break;
185     default:
186         usageRecord();
187         exit(1);
188         break;
189     }
190     /* End user code block <create_topLevelShell> */
191
192     ac = 0;
193     XtSetArg(args[ac], XmNx, 562); ac++;
194     XtSetArg(args[ac], XmNy, 656); ac++;
195     XtSetArg(args[ac], XmNwidth, 619); ac++;
196     XtSetArg(args[ac], XmNheight, 290); ac++;
197     topLevelShell = XtCreatePopupShell("topLevelShell",
198         topLevelShellWidgetClass,
```



```
199         parent,
200         args,
201         ac);
202     mainWindow = (Widget)CreatemainWindow(topLevelShell);
203     XtManageChild(mainWindow);
204     XtPopup(XtParent(mainWindow), XtGrabNone);
205
206     /* Begin user code block <app_procedures> */
207     /* set the initial state of all the GUI interface widgets */
208
209     /* the default input file name */
210     if( strlen( shared.outputFileName ) == 0 )
211         XmTextFieldSetString( outputFileTextField, "<none>");
212     else
213         XmTextFieldSetString( outputFileTextField, shared.outputFileName );
214
215     /* the default value for the network port number if not user supplied */
216     if( strlen( shared.portNumbText ) == 0 )
217     {
218         shared.portNumb = DEFAULT_PORT_NUMBER;
219         sprintf( shared.portNumbText, "%d", shared.portNumb );
220         XmTextFieldSetString( portTextField, shared.portNumbText );
221     }
222     else
223         XmTextFieldSetString( portTextField, shared.portNumbText );
224     sscanf( shared.portNumbText, "%d", &shared.portNumb );
225
226     /* the initial 'enabled' stat for pushbuttons */
227     /* record enabled, stop, pause and resume disabled */
228     XtSetSensitive( recordPushButton, True ); /* it already is, by default */
229     XtSetSensitive( stopPushButton, False );
230     XtSetSensitive( pausePushButton, False );
231     XtSetSensitive( resumePushButton, False );
232
233     /* the dialog box for file selection is not visible initially */
234     XtUnmanageChild(inputFileSelectionBox);
235     /* the dialog box for help (about) is not visible initially */
236     XtUnmanageChild(aboutHelpBulletinBoard);
237
238     shared.messageNumber = 0;
239     XmTextFieldSetString( numbRecTextField, "0" );
240
241     /* we are not sending yet */
242     shared.childProcess = 0;
243
244     /* End user code block <app_procedures> */
245
246     /* Begin user code block <main_loop> */
247     /* End user code block <main_loop> */
248
249     XtAppMainLoop(app);
250
251     /*
252      * A return value regardless of whether or not the main loop ends.
253      */
254      return(0);
255 }
```

```
 1  /**
 2      \file record_support.c
 3      \ingroup mod_record
 4
 5     This file contains the support function needed for the
 6      recording application.  We have a usage printing function.
 7
 8      \code
 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 F. Johnston
15      \since May 2002
16
17      $Id: record_support.c,v 1.4 2002/10/16 18:22:18 johnston Exp $
18  */
19
20  /**
21      \ingroup mod_record
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
```

```
/**
    \file recording.c
    \ingroup mod_record

   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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since Aug 2002

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

#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"

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

/**
    \ingroup mod_record
    This function will create a socket.  It uses the supplied
    port number to create a socket that can read messages on the local host.
    \param port is the integer port number to use for the new socket
    \return an error indication (if < 0) or the integer value of the socket
*/
int
createReadSocket( int port )
    {
    int sock;
    struct sockaddr_in server;

    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if ( sock < 0)
        {
        perror("opening recording stream socket");
        return -1;
        }

    /* just in case, we clear the structure */
    bzero((char *)&server, sizeof (server));

    /* we will read from any valid address */
    server.sin_addr.s_addr = INADDR_ANY;

    /* Assign client port number */
    server.sin_family = AF_INET;
    server.sin_port = port;

    /* name the socket */
```



```
    if( bind( sock, (struct sockaddr *)&server, sizeof server) < 0 )
        {
        perror("binding recording stream socket");
        return -1;
        }

    return sock;
    }

/* the timer callback to reset the state of the
 *  record/stop/pause/resume buttons
 */
/**
    \ingroup mod_record
    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.
*/
void
resetReadButtons()
    {
    extern globalMemory shared;
    char tmpBuf[MAX_NUMBER_TEXT_SIZE];

    /* disable the pause, resume and stop buttons, reenable the record */
    XtSetSensitive( recordPushButton, True );
    XtSetSensitive( stopPushButton, False );
    XtSetSensitive( pausePushButton, False );
    XtSetSensitive( resumePushButton, False );

    /* the child process is no more */
    shared.childProcess = 0;

    /* re-open the output file  to read size */
    shared.fileDescrip = fopen( shared.outputFileName, "r" );
    /* read the number of messages in the file */
    shared.messageNumber = getNumbMessages( shared.fileDescrip );
    fclose( shared.fileDescrip );
    sprintf( tmpBuf, "%d", shared.messageNumber );
    XmTextFieldSetString( numbRecTextField, tmpBuf );

    /* close down the existing socket */
    close( shared.socketNumb );
    }

/* the "child should pause" signal handler */
/**
    \ingroup mod_record
    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.
*/
void
readProcessPause()
    {
    int pid;
    int status;
    extern globalMemory shared;

    if(( pid = wait(&status)) == -1 )
        /* an error of some kind (probably a fork error) - ignore it */
        {
```

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



```
199              *   access the same user interface as the main program
200              */
201             {
202             XtAppAddTimeOut( shared.mainAppContext, 0, resetReadButtons, NULL );
203             return;
204             }
205         /* else */
206         /* some other process died */
207         }
208
209     /* read the messages into the file */
210     /**
211         \ingroup mod_record
212         This function will be called by the child (reading) process, and this
213         process will spend most of its time here in this loop.  The child
214         process will loop until killed (stopped) by the main program.  When the
215         file is done, the child process will trigger a signal to the main
216         process to let it know the child process is finished.
217         \param sock is the open socket we use for reading messages
218         \param fileD is the file descriptor for the (already opened) file
219          into which we will store the socket messages and delays.
220         \return an error indication (< 0) if there are sending problems
221     */
222     int
223     readMessages( FILE *fileD )
224         {
225         extern globalMemory shared;
226         timespec_t previousTime, currentTime, deltaTime;
227         int length;
228         char buffer[MAX_MESSAGE_SIZE];
229         struct sockaddr_in client;
230         int clientNameLen;
231
232         /* get the current time of day as a reference */
233         clock_gettime( CLOCK_REALTIME, &previousTime);
234
235         for (;;) /* loop waiting for messages */
236             {
237             /* initialize the client name len != 0, otherwise no client address */
238             clientNameLen = sizeof ( client );
239             /* in case there was stuff in structure from previous messages */
240     /*      bzero(client, sizeof(client) );
241     */
242
243             length = recvfrom( shared.socketNumb, buffer, MAX_MESSAGE_SIZE, 0,
244                                &client, &clientNameLen );
245             if( length < 0 && shared.globalDebugMode == 1 )
246                 {
247                 perror("recvfrom error\n");
248                 }
249
250             if( clientNameLen > 0 && shared.globalDebugMode == 1 )
251                 {
252                 printf("Client address is '%s' \n",(char *)inet_ntoa(client.sin_addr));
253                 }
254
255             /* we need to null terminate the received message */
256             buffer[length] = '\0';
257
258             /* get the current time of day to calculate the delta */
259             clock_gettime( CLOCK_REALTIME, &currentTime);
260             deltaTime.tv_sec = currentTime.tv_sec - previousTime.tv_sec;
261             deltaTime.tv_nsec = currentTime.tv_nsec - previousTime.tv_nsec;
262             previousTime.tv_sec = currentTime.tv_sec;
263             previousTime.tv_nsec = currentTime.tv_nsec;
264
```

```
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 (delta1) to file error\n");
271             exit(1);
272             }
273
274         if( fwrite( &deltaTime.tv_nsec, sizeof(long), 1, fileD ) != 1 )
275             {
276             perror("Write (delta2) to file error\n");
277             exit(1);
278             }
279
280         if( fwrite( &length, sizeof(int), 1, fileD ) != 1 )
281             {
282             perror("Write (length) to file error\n");
283             exit(1);
284             }
285
286         /* now write the actual message */
287         if( fwrite( buffer, length, 1, fileD ) != 1 )
288               {
289               perror("Write (buffer) to file error\n");
290               exit(1);
291               }
292         }  /* end of forever loop */
293     }
294
```

Appendix C

Playback Program Source Code

```
/*
 * README: This file is appended to at file generation time.
 * Edits can be made throughout the file
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */
#include <Xm/Xm.h>
#include <Xm/TextF.h>
#include <signal.h>
#include "creation-c.h"
#include "../common/myInterface.h"

/*
 * Standard includes for builtins.
 */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/*
 * Macros to make code look nicer between ANSI and K&R.
 */
#ifndef ARGLIST
#if (NeedFunctionPrototypes == 0)
#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;
#else
#define PROTOTYPE(p)    p
#define ARGLIST(p)      (
#define ARG(a, b)       a b,
#define GRA(a, b)       a b)
#ifdef __cplusplus
#define UARG(a, b)      a,
#define GRAU(a, b)      a)
#else
#define UARG(a, b)      a b,
#define GRAU(a, b)      a b)
#endif
#endif
#endif

Widget          BxFindTopShell PROTOTYPE((Widget));
WidgetList      BxWidgetIdsFromNames PROTOTYPE((Widget, char*, char*));

/**
    \file play/callbacks-c.c
    \ingroup mod_play

    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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since April 2002

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

/// the shared, global variable structure
extern globalMemory shared;

/**
    \ingroup mod_play
    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.
    \param 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.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
fileToggleCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    /* user has selected a file destination, turn on file */
    XmToggleButtonSetState( fileToggleButton, True, False );
    XmToggleButtonSetState( fileOptToggleButton, True, False );
    /* set both network toggles as false */
    XmToggleButtonSetState( networkToggleButton, False, False );
    XmToggleButtonSetState( networkOptToggleButton, False, False );
}

/**
    \ingroup mod_play
    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.
    \param 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.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
networkToggleCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    /* user has selected a network destination, turn off file if set */
    XmToggleButtonSetState( fileToggleButton, False, False );
    XmToggleButtonSetState( fileOptToggleButton, False, False );
    /* set both network toggles as true */
    XmToggleButtonSetState( networkToggleButton, True, False );
    XmToggleButtonSetState( networkOptToggleButton, True, False );
}

/**
    \ingroup mod_play
    This callback is the function that will be called when the Motif user
```

```
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     \param w the widget that caused the event and this callback. We ask
141     the slider widget for it's current value.
142     \param client_data is a pointer to any (optional) client data
143     defined for the callback (none in this case)
144     \param call_data the standard event information provided by X11-Motif
145 */
146 void
147 speedSliderCallback( Widget w, XtPointer client_data, XtPointer call_data)
148 {
149     int speedVal;
150
151     XtVaGetValues(w, XmNvalue, &speedVal, NULL);
152     if( shared.globalDebugMode == 1 )
153         printf("New speed slider = %d\n", speedVal );
154     if( speedVal < 15 )
155         {
156         shared.speedFactor = 0.25f;  /* 1/4 speed */
157         XmTextFieldSetString( speedTextField, "1/4x" );
158         }
159     else if( speedVal >= 15  && speedVal < 30 )
160         {
161         shared.speedFactor = 0.333333f;  /* 1/3 speed */
162         XmTextFieldSetString( speedTextField, "1/3x" );
163         }
164     else if( speedVal >= 30  && speedVal < 45 )
165         {
166         shared.speedFactor = 0.5f;  /* 1/2 speed */
167         XmTextFieldSetString( speedTextField, "1/2x" );
168         }
169     else if( speedVal >= 45  && speedVal < 60 )
170         {
171         shared.speedFactor = 1.0f;  /* normal 1x speed */
172         XmTextFieldSetString( speedTextField, "1x" );
173         }
174     else if( speedVal >= 60  && speedVal < 75 )
175         {
176         shared.speedFactor = 2.0f;  /* 2x speed */
177         XmTextFieldSetString( speedTextField, "2x" );
178         }
179     else if( speedVal >= 75  && speedVal < 90 )
180         {
181         shared.speedFactor = 3.0f;  /* 3x speed */
182         XmTextFieldSetString( speedTextField, "3x" );
183         }
184     else
185         {
186         shared.speedFactor = 4.0f;  /* 4x speed */
187         XmTextFieldSetString( speedTextField, "4x" );
188         }
189 }
190
191 /**
192     \ingroup mod_play
193     This callback is the function that will be called when the Motif user
194     interface puts up a file selection dialog and the user selects a file
195     and presses the 'OK' button.  We try to check the validity of the
196     new file name by trying to read how many socket messages it
197     contains.  If the file is invalid, the user interface will just
198     'beep' as the user trys to 'ok' the selection of an improper file.
```



```
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
206 okFilePlayDialogCallback( Widget w, XtPointer client_data, XtPointer call_data)
207 {
208     char        *file;
209     char        currentDirectory[MAX_NAME_TEXT_SIZE];
210     int         result;
211     XmFileSelectionBoxCallbackStruct    *fsb =
212         (XmFileSelectionBoxCallbackStruct *)call_data;
213
214     if ( XmStringToString(fsb->value, &file) )
215     {
216         /* copy the filename selected, but don't copy the directory part */
217         getwd( currentDirectory );
218         /* if the first part of the selected file is the same, cut it off */
219         result = strncmp( file, currentDirectory, strlen(currentDirectory));
220         if( result == 0 )
221             {
222             shared.inputFileName[0] = '.'; /* substitute for currect dir */
223             strcpy( shared.inputFileName+1, file+strlen(currentDirectory) );
224             }
225         else
226             /* better copy the whole directory */
227             strcpy( shared.inputFileName, file );
228         XmTextFieldSetString( inputFileTextField, shared.inputFileName );
229         shared.endMesNumb = readNumberMessages( shared.inputFileName );
230         /* set the end message number to be the max messages in file */
231         sprintf( currentDirectory, "%d", shared.endMesNumb );
232         XmTextFieldSetString( endNumbTextField, currentDirectory );
233         XtFree((char *)file);
234         if( shared.endMesNumb == 0 )
235             {
236             XBell( XtDisplay(w), 100 );
237             return;
238             }
239     }
240     else
241         printf("Invalid Selection\n");
242
243     /* remove the file selection dialog */
244     XtUnmanageChild(inputFileSelectionBox);
245 }
246
247 /**
248     \ingroup mod_play
249     This callback is the function that will be called when the Motif user
250     interface decides to respond to the user activating one of the 'Pause'
251     push buttons. Since this button can only be active when the program
252     is actually sending, we know that there is a sub-process in existance
253     and that it is running.  We want to stop the sub-process, and change
254     the state of active buttons in the interface to allow the user to
255     start it up again.
256     \param w the widget that caused the event and this callback.  It
257     could be one of several widgets in this case, and we don't care
258     which one.
259     \param client_data is a pointer to any (optional) client data
260     defined for the callback (none in this case)
261     \param call_data the standard event information provided by X11-Motif
262 */
263 void
264 pausePlayButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
```

```
{
    union sigval sval;
    /* disable the pause button, enable the restart and resume */
    XtSetSensitive( startPushButton, True );
    XtSetSensitive( pausePushButton, False );
    XtSetSensitive( resumePushButton, True );
    sval.sival_int = 1;
    if( sigqueue( shared.childProcess, SIGSTOP, sval ) < 0 )
        if( shared.globalDebugMode == 1 )
            printf("Pause: cannot queue signal to child\n");
}

/**
    \ingroup mod_play
    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 existance 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.
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
resumePlayButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    union sigval sval;
    /* disable the resume button, reenable the pause */
    XtSetSensitive( startPushButton, False );
    XtSetSensitive( pausePushButton, True );
    XtSetSensitive( resumePushButton, False );
    /* start sending messages from current position in file */
    sval.sival_int = 1;
    if( sigqueue( shared.childProcess, SIGCONT, sval ) < 0 )
        if( shared.globalDebugMode == 1 )
            printf("Resume: cannot queue signal to child\n");
}

/**
    \ingroup mod_play
    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.
    \param w the widget that caused the event and this callback.
    \param client_data is a pointer to any (optional) client data
    defined for the callback (none in this case)
    \param call_data the standard event information provided by X11-Motif
*/
void
startButtonCallback( Widget w, XtPointer client_data, XtPointer call_data)
{
    FILE *saveFileDescrip;      /* the output file */
    char tmpBuf[MAX_NUMBER_TEXT_SIZE];
    int count;

    /* read all the fields on the user interface to make sure we have
     *  the most up-to-date value for each option
     */
```



```

    /* read the current end number before it is overwritten */
    strcpy(tmpBuf, XmTextFieldGetString( endNumbTextField ));
    if( sscanf( tmpBuf, "%d", &shared.endMesNumb ) != 1 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }

    /* get the input file name */
    strcpy(shared.inputFileName, XmTextFieldGetString( inputFileTextField ));
    count = readNumberMessages( shared.inputFileName );
    if( count == 0 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }
    /* reset back to the the user selected end number for the (new?) file */
    sprintf( tmpBuf, "%d", shared.endMesNumb );
    XmTextFieldSetString( endNumbTextField, tmpBuf );

    strcpy(tmpBuf, XmTextFieldGetString( startNumbTextField ));
    if( sscanf( tmpBuf, "%d", &shared.startMesNumb ) != 1 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }
    if( shared.startMesNumb < 1 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }
    if( shared.endMesNumb <= shared.startMesNumb )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }

    /* the destination address for the socket (or file name ) */
    strcpy(shared.address, XmTextFieldGetString( destinationTextField ));

    /* the destination port for the socket */
    strcpy(tmpBuf, XmTextFieldGetString( portTextField ));
    if( sscanf( tmpBuf, "%d", &shared.portNumb) != 1 )
        {
        XBell( XtDisplay(w), 100 );
        return;
        }

    saveFileDescrip = NULL;
    /* if we have the normal, address destination, then create socket */
    if( XmToggleButtonGetState( networkToggleButton ) == True )
        {
        shared.socketNumb = createSendSocket( shared.address, shared.portNumb );
        if( shared.socketNumb < 0 )
            {
            XBell( XtDisplay(w), 100 );
            return;
            }
        }
    else
        /* we will save the output in the destination file, try to open it */
        {
        saveFileDescrip = fopen( shared.address, "w");
        if( saveFileDescrip == NULL )
            {
```

```
397             XBell( XtDisplay(w), 100 );
398             return;
399             }
400         /* we can write to the file, send the information */
401         }
402
403     /* start sending messages from start of file */
404     /* (re)set the frame counter */
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, sendProcessDone );
412     /* fork a subprocess to run the long send proceedure */
413     switch( shared.childProcess = fork() )
414         {
415     case 0: /* child process */
416         /* we want the child process to respond to a pause/resume control */
417         signal( SIGSTOP, sendProcessPause );
418         sendMessages( shared.fileDescrip, saveFileDescrip );
419         /* when the send process is done, exit with signal set */
420         _exit(255);
421     case -1:
422         XBell( XtDisplay(w), 100 );
423         printf("Fork() of send messages failed!\n");
424         return;
425         }
426
427     /* the main (motif) program continues... */
428     /* set pushbutton state - disable the start button, enable the pause */
429     XtSetSensitive( startPushButton, False );
430     XtSetSensitive( pausePushButton, True );
431     XtSetSensitive( resumePushButton, False );
432
433     /* if we have a file destination, then close file */
434     if( XmToggleButtonGetState( networkToggleButton ) == True )
435         fclose( saveFileDescrip );
436     else
437         /* close the socket */
438         close( shared.socketNumb );
439 }
440
```

```
/*
 * README: Portions of this file are merged at file generation
 * time. Edits can be made *only* in between specified code blocks, look
 * for keywords <Begin user code> and <End user code>.
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */


/* Begin user code block <file_comments> */
/**
    \file play/creation-c.c
    \ingroup mod_play

    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.
    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since July 2002

    $Id: creation-c.c,v 1.9 2002/10/16 18:20:27 johnston Exp $
*/
/* End user code block <file_comments> */

#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>

/*
 * Global declarations are now stored in the header file.
 *
 * If DECLARE_BX_GLOBALS is defined then this header file
 * declares the globals, otherwise it just externs them.
 */
#define DECLARE_BX_GLOBALS

/*
 * Globally included information.
 */


```



```
/*
 * Common constant and pixmap declarations.
 */
#include "creation-c.h"

/*
 * Convenience functions from utilities file.
 */
extern void RegisterBxConverters(XtAppContext);
extern XtPointer BX_CONVERT(Widget, char *, char *, int, Boolean *);
extern XtPointer BX_DOUBLE(double);
extern XtPointer BX_SINGLE(float);
extern void BX_MENU_POST(Widget, XtPointer, XEvent *, Boolean *);
extern Pixmap XPM_PIXMAP(Widget, char **);
extern void BX_SET_BACKGROUND_COLOR(Widget, ArgList, Cardinal *, Pixel);

/*
 * Declarations for callbacks and handlers.
 */
#ifndef DOXYGEN_SHOULD_IGNORE_THIS
extern void aboutDismissCallback(Widget, XtPointer, XtPointer);
extern void okFilePlayDialogCallback(Widget, XtPointer, XtPointer);
extern void cancelFileDialogCallback(Widget, XtPointer, XtPointer);
extern void browseButtonCallback(Widget, XtPointer, XtPointer);
extern void startButtonCallback(Widget, XtPointer, XtPointer);
extern void pausePlayButtonCallback(Widget, XtPointer, XtPointer);
extern void resumePlayButtonCallback(Widget, XtPointer, XtPointer);
extern void quitButtonCallback(Widget, XtPointer, XtPointer);
extern void networkToggleCallback(Widget, XtPointer, XtPointer);
extern void fileToggleCallback(Widget, XtPointer, XtPointer);
extern void debugToggleCallback(Widget, XtPointer, XtPointer);
extern void usageHelpCallback(Widget, XtPointer, XtPointer);
extern void aboutHelpCallback(Widget, XtPointer, XtPointer);
extern void speedSliderCallback(Widget, XtPointer, XtPointer);
#endif

/*
 * Create the mainWindow hierarchy of widgets.
 */
Widget
CreatemainWindow(Widget parent)
{
    Cardinal ac = 0;
    Arg      args[256];
    Cardinal cdc = 0;
    Boolean  argok = False;
    Widget   mainWindow;
    Widget   aboutHelpDialogShell;
    Widget   aboutHelpLabel;
    Widget   aboutDismissPushButton;
    Widget   selectFileDialog;
    Widget   menuBar;
    Widget   fileCascadeButton;
    Widget   pulldownMenu;
    Widget   openFilePushButton;
    Widget   separator1;
    Widget   startFilePushButton;
    Widget   pauseFilePushButton;
    Widget   resumeFilePushButton;
    Widget   separator2;
    Widget   quitFilePushButton;
    Widget   optionsCascadeButton;
    Widget   pulldownMenu1;
    Widget   helpCascadeButton;
    Widget   pulldownMenu2;
    Widget   usageHelpPushButton;
```

```
133     Widget   aboutHelpPushButton;
134     Widget   form;
135     Widget   label;
136     Widget   endNumbLabel;
137     Widget   startNumbLabel;
138     Widget   fasterLabel;
139     Widget   slowerLabel;
140     Widget   speedLabel;
141     Widget   speedScrollBar;
142     Widget   quitPushButton;
143     Widget   portLabel;
144     Widget   destinationLabel;
145     Widget   browseButton;
146     Widget   inputFileLabel;
147
148     /*
149      * Register the converters for the widgets.
150      */
151     RegisterBxConverters(XtWidgetToApplicationContext(parent));
152     XtInitializeWidgetClass((WidgetClass)xmMainWindowWidgetClass);
153     XtInitializeWidgetClass((WidgetClass)xmDialogShellWidgetClass);
154     XtInitializeWidgetClass((WidgetClass)xmBulletinBoardWidgetClass);
155     XtInitializeWidgetClass((WidgetClass)xmScrolledWindowWidgetClass);
156     XtInitializeWidgetClass((WidgetClass)xmLabelWidgetClass);
157     XtInitializeWidgetClass((WidgetClass)xmPushButtonWidgetClass);
158     XtInitializeWidgetClass((WidgetClass)xmFileSelectionBoxWidgetClass);
159     XtInitializeWidgetClass((WidgetClass)xmRowColumnWidgetClass);
160     XtInitializeWidgetClass((WidgetClass)xmCascadeButtonWidgetClass);
161     XtInitializeWidgetClass((WidgetClass)xmRowColumnWidgetClass);
162     XtInitializeWidgetClass((WidgetClass)xmSeparatorWidgetClass);
163     XtInitializeWidgetClass((WidgetClass)xmToggleButtonWidgetClass);
164     XtInitializeWidgetClass((WidgetClass)xmFormWidgetClass);
165     XtInitializeWidgetClass((WidgetClass)xmTextFieldWidgetClass);
166     XtInitializeWidgetClass((WidgetClass)xmScrollBarWidgetClass);
167
168     ac = 0;
169     XtSetArg(args[ac], XmNx, 612); ac++;
170     XtSetArg(args[ac], XmNy, 414); ac++;
171     XtSetArg(args[ac], XmNwidth, 619); ac++;
172     XtSetArg(args[ac], XmNheight, 385); ac++;
173     mainWindow = XmCreateMainWindow(parent,
174         "mainWindow",
175         args,
176         ac);
177
178     ac = 0;
179     XtSetArg(args[ac], XmNwidth, 619); ac++;
180     XtSetArg(args[ac], XmNheight, 28); ac++;
181     menuBar = XmCreateMenuBar(mainWindow,
182         "menuBar",
183         args,
184         ac);
185     XtManageChild(menuBar);
186
187     ac = 0;
188     XtSetArg(args[ac], XmNmnemonic, 'F'); ac++;
189     XtSetArg(args[ac], XmNlabelString,
190         BX_CONVERT(menuBar, "File",
191         XmRXmString, 0, &argok)); if (argok) ac++;
192     XtSetArg(args[ac], XmNx, 5); ac++;
193     XtSetArg(args[ac], XmNy, 2); ac++;
194     XtSetArg(args[ac], XmNwidth, 43); ac++;
195     XtSetArg(args[ac], XmNheight, 24); ac++;
196     fileCascadeButton = XmCreateCascadeButton(menuBar,
197         "fileCascadeButton",
198         args,
```



```
199         ac);
200     XtManageChild(fileCascadeButton);
201
202     ac = 0;
203     XtSetArg(args[ac], XmNx, 0); ac++;
204     XtSetArg(args[ac], XmNy, 0); ac++;
205     XtSetArg(args[ac], XmNwidth, 74); ac++;
206     XtSetArg(args[ac], XmNheight, 130); ac++;
207     pulldownMenu = XmCreatePulldownMenu(XtParent(fileCascadeButton),
208         "pulldownMenu",
209         args,
210         ac);
211
212     ac = 0;
213     XtSetArg(args[ac], XmNmnemonic, 'O'); ac++;
214     XtSetArg(args[ac], XmNlabelString,
215         BX_CONVERT(pulldownMenu, "Open...",
216         XmRXmString, 0, &argok)); if (argok) ac++;
217     openFilePushButton = XmCreatePushButton(pulldownMenu,
218         "openFilePushButton",
219         args,
220         ac);
221     XtManageChild(openFilePushButton);
222     XtAddCallback(openFilePushButton, XmNactivateCallback, browseButtonCallback, (XtPoint
er)0);
223
224     ac = 0;
225     separator1 = XmCreateSeparator(pulldownMenu,
226         "separator1",
227         args,
228         ac);
229     XtManageChild(separator1);
230
231     ac = 0;
232     XtSetArg(args[ac], XmNmnemonic, 'S'); ac++;
233     XtSetArg(args[ac], XmNlabelString,
234         BX_CONVERT(pulldownMenu, "Start",
235         XmRXmString, 0, &argok)); if (argok) ac++;
236     startFilePushButton = XmCreatePushButton(pulldownMenu,
237         "startFilePushButton",
238         args,
239         ac);
240     XtManageChild(startFilePushButton);
241     XtAddCallback(startFilePushButton, XmNactivateCallback, startButtonCallback, (XtPoint
er)0);
242
243     ac = 0;
244     XtSetArg(args[ac], XmNmnemonic, 'P'); ac++;
245     XtSetArg(args[ac], XmNlabelString,
246         BX_CONVERT(pulldownMenu, "Pause",
247         XmRXmString, 0, &argok)); if (argok) ac++;
248     pauseFilePushButton = XmCreatePushButton(pulldownMenu,
249         "pauseFilePushButton",
250         args,
251         ac);
252     XtManageChild(pauseFilePushButton);
253     XtAddCallback(pauseFilePushButton, XmNactivateCallback, pausePlayButtonCallback, (XtP
ointer)0);
254
255     ac = 0;
256     XtSetArg(args[ac], XmNmnemonic, 'R'); ac++;
257     XtSetArg(args[ac], XmNlabelString,
258         BX_CONVERT(pulldownMenu, "Resume",
259         XmRXmString, 0, &argok)); if (argok) ac++;
260     resumeFilePushButton = XmCreatePushButton(pulldownMenu,
261         "resumeFilePushButton",
```

```
    262         args,
    263         ac);
    264     XtManageChild(resumeFilePushButton);
    265     XtAddCallback(resumeFilePushButton, XmNactivateCallback, resumePlayButtonCallback, (X
tPointer)0);
    266
    267     ac = 0;
    268     separator2 = XmCreateSeparator(pulldownMenu,
    269         "separator2",
    270         args,
    271         ac);
    272     XtManageChild(separator2);
    273
    274     ac = 0;
    275     XtSetArg(args[ac], XmNmnemonic, 'Q'); ac++;
    276     XtSetArg(args[ac], XmNlabelString,
    277         BX_CONVERT(pulldownMenu, "Quit",
    278         XmRXmString, 0, &argok)); if (argok) ac++;
    279     quitFilePushButton = XmCreatePushButton(pulldownMenu,
    280         "quitFilePushButton",
    281         args,
    282         ac);
    283     XtManageChild(quitFilePushButton);
    284     XtAddCallback(quitFilePushButton, XmNactivateCallback, quitButtonCallback, (XtPointer
)0);
    285
    286     ac = 0;
    287     XtSetArg(args[ac], XmNsubMenuId, pulldownMenu); ac++;
    288     XtSetValues(fileCascadeButton, args, ac);
    289
    290     ac = 0;
    291     XtSetArg(args[ac], XmNmnemonic, 'O'); ac++;
    292     XtSetArg(args[ac], XmNlabelString,
    293         BX_CONVERT(menuBar, "Options",
    294         XmRXmString, 0, &argok)); if (argok) ac++;
    295     XtSetArg(args[ac], XmNx, 55); ac++;
    296     XtSetArg(args[ac], XmNy, 2); ac++;
    297     XtSetArg(args[ac], XmNwidth, 74); ac++;
    298     XtSetArg(args[ac], XmNheight, 24); ac++;
    299     optionsCascadeButton = XmCreateCascadeButton(menuBar,
    300         "optionsCascadeButton",
    301         args,
    302         ac);
    303     XtManageChild(optionsCascadeButton);
    304
    305     ac = 0;
    306     XtSetArg(args[ac], XmNx, 0); ac++;
    307     XtSetArg(args[ac], XmNy, 0); ac++;
    308     XtSetArg(args[ac], XmNwidth, 173); ac++;
    309     XtSetArg(args[ac], XmNheight, 78); ac++;
    310     pulldownMenu1 = XmCreatePulldownMenu(XtParent(optionsCascadeButton),
    311         "pulldownMenu1",
    312         args,
    313         ac);
    314
    315     ac = 0;
    316     XtSetArg(args[ac], XmNmnemonic, 'N'); ac++;
    317     XtSetArg(args[ac], XmNlabelString,
    318         BX_CONVERT(pulldownMenu1, "Network Destination",
    319         XmRXmString, 0, &argok)); if (argok) ac++;
    320     networkOptToggleButton = XmCreateToggleButton(pulldownMenu1,
    321         "networkOptToggleButton",
    322         args,
    323         ac);
    324     XtManageChild(networkOptToggleButton);
    325     XtAddCallback(networkOptToggleButton, XmNvalueChangedCallback, networkToggleCallback,
```



```
 (XtPointer)0);
    326
    327     ac = 0;
    328     XtSetArg(args[ac], XmNmnemonic, 'F'); ac++;
    329     XtSetArg(args[ac], XmNlabelString,
    330         BX_CONVERT(pulldownMenu1, "File Destination",
    331         XmRXmString, 0, &argok)); if (argok) ac++;
    332     fileOptToggleButton = XmCreateToggleButton(pulldownMenu1,
    333         "fileOptToggleButton",
    334         args,
    335         ac);
    336     XtManageChild(fileOptToggleButton);
    337     XtAddCallback(fileOptToggleButton, XmNvalueChangedCallback, fileToggleCallback, (XtPo
inter)0);
    338
    339     ac = 0;
    340     XtSetArg(args[ac], XmNlabelString,
    341         BX_CONVERT(pulldownMenu1, "debug mode",
    342         XmRXmString, 0, &argok)); if (argok) ac++;
    343     debugToggleButton = XmCreateToggleButton(pulldownMenu1,
    344         "debugToggleButton",
    345         args,
    346         ac);
    347     XtManageChild(debugToggleButton);
    348     XtAddCallback(debugToggleButton, XmNvalueChangedCallback, debugToggleCallback, (XtPoi
nter)0);
    349
    350     ac = 0;
    351     XtSetArg(args[ac], XmNsubMenuId, pulldownMenu1); ac++;
    352     XtSetValues(optionsCascadeButton, args, ac);
    353
    354     ac = 0;
    355     XtSetArg(args[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
    356     XtSetArg(args[ac], XmNmnemonic, 'H'); ac++;
    357     XtSetArg(args[ac], XmNlabelString,
    358         BX_CONVERT(menuBar, "Help",
    359         XmRXmString, 0, &argok)); if (argok) ac++;
    360     XtSetArg(args[ac], XmNx, 564); ac++;
    361     XtSetArg(args[ac], XmNy, 2); ac++;
    362     XtSetArg(args[ac], XmNwidth, 50); ac++;
    363     XtSetArg(args[ac], XmNheight, 24); ac++;
    364     helpCascadeButton = XmCreateCascadeButton(menuBar,
    365         "helpCascadeButton",
    366         args,
    367         ac);
    368     XtManageChild(helpCascadeButton);
    369
    370     ac = 0;
    371     XtSetArg(args[ac], XmNx, 0); ac++;
    372     XtSetArg(args[ac], XmNy, 0); ac++;
    373     XtSetArg(args[ac], XmNwidth, 63); ac++;
    374     XtSetArg(args[ac], XmNheight, 56); ac++;
    375     pulldownMenu2 = XmCreatePulldownMenu(XtParent(helpCascadeButton),
    376         "pulldownMenu2",
    377         args,
    378         ac);
    379
    380     ac = 0;
    381     XtSetArg(args[ac], XmNmnemonic, 'U'); ac++;
    382     XtSetArg(args[ac], XmNlabelString,
    383         BX_CONVERT(pulldownMenu2, "Usage",
    384         XmRXmString, 0, &argok)); if (argok) ac++;
    385     usageHelpPushButton = XmCreatePushButton(pulldownMenu2,
    386         "usageHelpPushButton",
    387         args,
    388         ac);
```

```
    XtManageChild(usageHelpPushButton);
    XtAddCallback(usageHelpPushButton, XmNactivateCallback, usageHelpCallback, (XtPointer
)"playbackUsageHelp.html");

    ac = 0;
    XtSetArg(args[ac], XmNmnemonic, 'A'); ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(pulldownMenu2, "About",
        XmRXmString, 0, &argok)); if (argok) ac++;
    aboutHelpPushButton = XmCreatePushButton(pulldownMenu2,
        "aboutHelpPushButton",
        args,
        ac);
    XtManageChild(aboutHelpPushButton);
    XtAddCallback(aboutHelpPushButton, XmNactivateCallback, aboutHelpCallback, (XtPointer
)0);

    ac = 0;
    XtSetArg(args[ac], XmNsubMenuId, pulldownMenu2); ac++;
    XtSetValues(helpCascadeButton, args, ac);

    ac = 0;
    XtSetArg(args[ac], XmNresizePolicy, XmRESIZE_GROW); ac++;
    XtSetArg(args[ac], XmNwidth, 619); ac++;
    XtSetArg(args[ac], XmNheight, 357); ac++;
    form = XmCreateForm(mainWindow,
        "form",
        args,
        ac);
    XtManageChild(form);

    ac = 0;
    XtSetArg(args[ac], XmNalignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(args[ac], XmNsensitive, False); ac++;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-bold-i-*-*-*-240-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNforeground,
        BX_CONVERT(form, "#ffffd7",
        XmRPixel, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Playback UDP Sockets",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 180); ac++;
    XtSetArg(args[ac], XmNy, 300); ac++;
    XtSetArg(args[ac], XmNwidth, 239); ac++;
    XtSetArg(args[ac], XmNheight, 29); ac++;
    label = XmCreateLabel(form,
        "label",
        args,
        ac);
    XtManageChild(label);

    ac = 0;
    XtSetArg(args[ac], XmNeditable, True); ac++;
    XtSetArg(args[ac], XmNx, 400); ac++;
    XtSetArg(args[ac], XmNy, 250); ac++;
    XtSetArg(args[ac], XmNwidth, 120); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    endNumbTextField = XmCreateTextField(form,
        "endTextField",
        args,
        ac);
    XtManageChild(endNumbTextField);

    ac = 0;
```



```
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "End Numb.",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 330); ac++;
    XtSetArg(args[ac], XmNy, 250); ac++;
    XtSetArg(args[ac], XmNwidth, 69); ac++;
    XtSetArg(args[ac], XmNheight, 19); ac++;
    endNumbLabel = XmCreateLabel(form,
        "endNumbLabel",
        args,
        ac);
    XtManageChild(endNumbLabel);

    ac = 0;
    XtSetArg(args[ac], XmNvalue, "1"); ac++;
    XtSetArg(args[ac], XmNeditable, True); ac++;
    XtSetArg(args[ac], XmNx, 140); ac++;
    XtSetArg(args[ac], XmNy, 250); ac++;
    XtSetArg(args[ac], XmNwidth, 120); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    startNumbTextField = XmCreateTextField(form,
        "startNumbTextField",
        args,
        ac);
    XtManageChild(startNumbTextField);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Start Numb.",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 60); ac++;
    XtSetArg(args[ac], XmNy, 250); ac++;
    XtSetArg(args[ac], XmNwidth, 74); ac++;
    XtSetArg(args[ac], XmNheight, 19); ac++;
    startNumbLabel = XmCreateLabel(form,
        "startNumbLabel",
        args,
        ac);
    XtManageChild(startNumbLabel);

    ac = 0;
    XtSetArg(args[ac], XmNeditable, False); ac++;
    XtSetArg(args[ac], XmNx, 390); ac++;
    XtSetArg(args[ac], XmNy, 210); ac++;
    XtSetArg(args[ac], XmNwidth, 93); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    speedTextField = XmCreateTextField(form,
        "speedTextField",
        args,
        ac);
    XtManageChild(speedTextField);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Faster",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 330); ac++;
```

```
519     XtSetArg(args[ac], XmNy, 190); ac++;
520     XtSetArg(args[ac], XmNwidth, 41); ac++;
521     XtSetArg(args[ac], XmNheight, 19); ac++;
522     fasterLabel = XmCreateLabel(form,
523         "fasterLabel",
524         args,
525         ac);
526     XtManageChild(fasterLabel);
527
528     ac = 0;
529     XtSetArg(args[ac], XmNfontList,
530         BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
531         XmRFontList, 0, &argok)); if (argok) ac++;
532     XtSetArg(args[ac], XmNlabelString,
533         BX_CONVERT(form, "Slower",
534         XmRXmString, 0, &argok)); if (argok) ac++;
535     XtSetArg(args[ac], XmNx, 20); ac++;
536     XtSetArg(args[ac], XmNy, 190); ac++;
537     XtSetArg(args[ac], XmNwidth, 45); ac++;
538     XtSetArg(args[ac], XmNheight, 20); ac++;
539     slowerLabel = XmCreateLabel(form,
540         "slowerLabel",
541         args,
542         ac);
543     XtManageChild(slowerLabel);
544
545     ac = 0;
546     XtSetArg(args[ac], XmNfontList,
547         BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
548         XmRFontList, 0, &argok)); if (argok) ac++;
549     XtSetArg(args[ac], XmNlabelString,
550         BX_CONVERT(form, "Speed",
551         XmRXmString, 0, &argok)); if (argok) ac++;
552     XtSetArg(args[ac], XmNx, 180); ac++;
553     XtSetArg(args[ac], XmNy, 190); ac++;
554     XtSetArg(args[ac], XmNwidth, 40); ac++;
555     XtSetArg(args[ac], XmNheight, 19); ac++;
556     speedLabel = XmCreateLabel(form,
557         "speedLabel",
558         args,
559         ac);
560     XtManageChild(speedLabel);
561
562     ac = 0;
563     XtSetArg(args[ac], XmNvalue, 45); ac++;
564     XtSetArg(args[ac], XmNmaximum, 100); ac++;
565     XtSetArg(args[ac], XmNhighlightThickness, 0); ac++;
566     XtSetArg(args[ac], XmNorientation, XmHORIZONTAL); ac++;
567     XtSetArg(args[ac], XmNx, 20); ac++;
568     XtSetArg(args[ac], XmNy, 210); ac++;
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     XtManageChild(speedScrollBar);
576     XtAddCallback(speedScrollBar, XmNvalueChangedCallback, speedSliderCallback, (XtPointe
r)0);
577
578     ac = 0;
579     XtSetArg(args[ac], XmNfontList,
580         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
581         XmRFontList, 0, &argok)); if (argok) ac++;
582     XtSetArg(args[ac], XmNlabelString,
583         BX_CONVERT(form, "Quit",
```



```
584         XmRXmString, 0, &argok)); if (argok) ac++;
585     XtSetArg(args[ac], XmNx, 20); ac++;
586     XtSetArg(args[ac], XmNy, 300); ac++;
587     XtSetArg(args[ac], XmNwidth, 52); ac++;
588     XtSetArg(args[ac], XmNheight, 39); ac++;
589     quitPushButton = XmCreatePushButton(form,
590         "quitPushButton",
591         args,
592         ac);
593     XtManageChild(quitPushButton);
594     XtAddCallback(quitPushButton, XmNactivateCallback, quitButtonCallback, (XtPointer)0);
595
596     ac = 0;
597     XtSetArg(args[ac], XmNfontList,
598         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
599         XmRFontList, 0, &argok)); if (argok) ac++;
600     XtSetArg(args[ac], XmNlabelString,
601         BX_CONVERT(form, "Resume",
602         XmRXmString, 0, &argok)); if (argok) ac++;
603     XtSetArg(args[ac], XmNx, 320); ac++;
604     XtSetArg(args[ac], XmNy, 140); ac++;
605     XtSetArg(args[ac], XmNwidth, 74); ac++;
606     XtSetArg(args[ac], XmNheight, 35); ac++;
607     resumePushButton = XmCreatePushButton(form,
608         "resumePushButton",
609         args,
610         ac);
611     XtManageChild(resumePushButton);
612     XtAddCallback(resumePushButton, XmNactivateCallback, resumePlayButtonCallback, (XtPoi
nter)0);
613
614     ac = 0;
615     XtSetArg(args[ac], XmNfontList,
616         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
617         XmRFontList, 0, &argok)); if (argok) ac++;
618     XtSetArg(args[ac], XmNlabelString,
619         BX_CONVERT(form, "Pause",
620         XmRXmString, 0, &argok)); if (argok) ac++;
621     XtSetArg(args[ac], XmNx, 170); ac++;
622     XtSetArg(args[ac], XmNy, 140); ac++;
623     XtSetArg(args[ac], XmNwidth, 59); ac++;
624     XtSetArg(args[ac], XmNheight, 35); ac++;
625     pausePushButton = XmCreatePushButton(form,
626         "pausePushButton",
627         args,
628         ac);
629     XtManageChild(pausePushButton);
630     XtAddCallback(pausePushButton, XmNactivateCallback, pausePlayButtonCallback, (XtPoint
er)0);
631
632     ac = 0;
633     XtSetArg(args[ac], XmNfontList,
634         BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
635         XmRFontList, 0, &argok)); if (argok) ac++;
636     XtSetArg(args[ac], XmNlabelString,
637         BX_CONVERT(form, "Start/Restart",
638         XmRXmString, 0, &argok)); if (argok) ac++;
639     XtSetArg(args[ac], XmNx, 20); ac++;
640     XtSetArg(args[ac], XmNy, 140); ac++;
641     XtSetArg(args[ac], XmNwidth, 106); ac++;
642     XtSetArg(args[ac], XmNheight, 35); ac++;
643     startPushButton = XmCreatePushButton(form,
644         "startPushButton",
645         args,
646         ac);
```

```
    XtManageChild(startPushButton);
    XtAddCallback(startPushButton, XmNactivateCallback, startButtonCallback, (XtPointer)0
);

    ac = 0;
    XtSetArg(args[ac], XmNx, 310); ac++;
    XtSetArg(args[ac], XmNy, 90); ac++;
    XtSetArg(args[ac], XmNwidth, 120); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    portTextField = XmCreateTextField(form,
        "portTextField",
        args,
        ac);
    XtManageChild(portTextField);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Port Numb.",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 240); ac++;
    XtSetArg(args[ac], XmNy, 90); ac++;
    XtSetArg(args[ac], XmNwidth, 70); ac++;
    XtSetArg(args[ac], XmNheight, 19); ac++;
    portLabel = XmCreateLabel(form,
        "portLabel",
        args,
        ac);
    XtManageChild(portLabel);

    ac = 0;
    XtSetArg(args[ac], XmNx, 310); ac++;
    XtSetArg(args[ac], XmNy, 60); ac++;
    XtSetArg(args[ac], XmNwidth, 270); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    destinationTextField = XmCreateTextField(form,
        "destinationTextField",
        args,
        ac);
    XtManageChild(destinationTextField);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Destination",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 240); ac++;
    XtSetArg(args[ac], XmNy, 60); ac++;
    XtSetArg(args[ac], XmNwidth, 69); ac++;
    XtSetArg(args[ac], XmNheight, 19); ac++;
    destinationLabel = XmCreateLabel(form,
        "destinationLabel",
        args,
        ac);
    XtManageChild(destinationLabel);

    ac = 0;
    XtSetArg(args[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
```



```
        BX_CONVERT(form, "File Destination",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 20); ac++;
    XtSetArg(args[ac], XmNy, 90); ac++;
    XtSetArg(args[ac], XmNwidth, 118); ac++;
    XtSetArg(args[ac], XmNheight, 27); ac++;
    fileToggleButton = XmCreateToggleButton(form,
        "fileToggleButton",
        args,
        ac);
    XtManageChild(fileToggleButton);
    XtAddCallback(fileToggleButton, XmNvalueChangedCallback, fileToggleCallback, (XtPoint
er)0);

    ac = 0;
    XtSetArg(args[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Network Destination",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 20); ac++;
    XtSetArg(args[ac], XmNy, 60); ac++;
    XtSetArg(args[ac], XmNwidth, 149); ac++;
    XtSetArg(args[ac], XmNheight, 27); ac++;
    networkToggleButton = XmCreateToggleButton(form,
        "networkToggleButton",
        args,
        ac);
    XtManageChild(networkToggleButton);
    XtAddCallback(networkToggleButton, XmNvalueChangedCallback, networkToggleCallback, (X
tPointer)0);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
        BX_CONVERT(form, "-*-times-medium-r-*-*-*-160-*-*-*-*-iso8859-1",
        XmRFontList, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNlabelString,
        BX_CONVERT(form, "Browse",
        XmRXmString, 0, &argok)); if (argok) ac++;
    XtSetArg(args[ac], XmNx, 480); ac++;
    XtSetArg(args[ac], XmNy, 10); ac++;
    XtSetArg(args[ac], XmNwidth, 72); ac++;
    XtSetArg(args[ac], XmNheight, 35); ac++;
    browseButton = XmCreatePushButton(form,
        "browseButton",
        args,
        ac);
    XtManageChild(browseButton);
    XtAddCallback(browseButton, XmNactivateCallback, browseButtonCallback, (XtPointer)0);

    ac = 0;
    XtSetArg(args[ac], XmNx, 110); ac++;
    XtSetArg(args[ac], XmNy, 10); ac++;
    XtSetArg(args[ac], XmNwidth, 370); ac++;
    XtSetArg(args[ac], XmNheight, 30); ac++;
    inputFileTextField = XmCreateTextField(form,
        "inputFileTextField",
        args,
        ac);
    XtManageChild(inputFileTextField);

    ac = 0;
    XtSetArg(args[ac], XmNfontList,
```

```
775             BX_CONVERT(form, "-*-times-medium-r-*-*-*-140-*-*-*-*-iso8859-1",
776             XmRFontList, 0, &argok)); if (argok) ac++;
777         XtSetArg(args[ac], XmNlabelString,
778             BX_CONVERT(form, "Input File",
779             XmRXmString, 0, &argok)); if (argok) ac++;
780         XtSetArg(args[ac], XmNx, 20); ac++;
781         XtSetArg(args[ac], XmNy, 11); ac++;
782         XtSetArg(args[ac], XmNwidth, 110); ac++;
783         XtSetArg(args[ac], XmNheight, 19); ac++;
784         inputFileLabel = XmCreateLabel(form,
785             "inputFileLabel",
786             args,
787             ac);
788         XtManageChild(inputFileLabel);
789
790         ac = 0;
791         XtSetArg(args[ac], XmNwidth, 457); ac++;
792         XtSetArg(args[ac], XmNheight, 396); ac++;
793         selectFileDialog = XtCreatePopupShell("selectFileDialog",
794             xmDialogShellWidgetClass,
795             mainWindow,
796             args,
797             ac);
798
799         ac = 0;
800         XtSetArg(args[ac], XmNx, 3); ac++;
801         XtSetArg(args[ac], XmNy, 86); ac++;
802         XtSetArg(args[ac], XmNwidth, 457); ac++;
803         XtSetArg(args[ac], XmNheight, 396); ac++;
804         inputFileSelectionBox = XmCreateFileSelectionBox(selectFileDialog,
805             "inputFileSelectionBox",
806             args,
807             ac);
808         XtAddCallback(inputFileSelectionBox, XmNokCallback, okFilePlayDialogCallback, (XtPoin
ter)0);
809         XtAddCallback(inputFileSelectionBox, XmNcancelCallback, cancelFileDialogCallback, (Xt
Pointer)0);
810
811         ac = 0;
812         XtSetArg(args[ac], XmNwidth, 284); ac++;
813         XtSetArg(args[ac], XmNheight, 312); ac++;
814         aboutHelpDialogShell = XtCreatePopupShell("aboutHelpDialogShell",
815             xmDialogShellWidgetClass,
816             mainWindow,
817             args,
818             ac);
819
820         ac = 0;
821         XtSetArg(args[ac], XmNresizePolicy, XmRESIZE_GROW); ac++;
822         XtSetArg(args[ac], XmNx, 968); ac++;
823         XtSetArg(args[ac], XmNy, 122); ac++;
824         XtSetArg(args[ac], XmNwidth, 284); ac++;
825         XtSetArg(args[ac], XmNheight, 312); ac++;
826         aboutHelpBulletinBoard = XmCreateBulletinBoard(aboutHelpDialogShell,
827             "aboutHelpBulletinBoard",
828             args,
829             ac);
830
831         ac = 0;
832         XtSetArg(args[ac], XmNscrollingPolicy, XmAUTOMATIC); ac++;
833         XtSetArg(args[ac], XmNx, 20); ac++;
834         XtSetArg(args[ac], XmNy, 20); ac++;
835         XtSetArg(args[ac], XmNwidth, 250); ac++;
836         XtSetArg(args[ac], XmNheight, 210); ac++;
837         aboutScrolledWindow = XmCreateScrolledWindow(aboutHelpBulletinBoard,
838             "aboutScrolledWindow",
```



```
839             args,
840             ac);
841         XtManageChild(aboutScrolledWindow);
842
843         ac = 0;
844         XtSetArg(args[ac], XmNalignment, XmALIGNMENT_CENTER); ac++;
845         XtSetArg(args[ac], XmNfontList,
846             BX_CONVERT(aboutScrolledWindow, "-*-helvetica-bold-r-*-*-*-130-*-*-*-*-iso8859-1"
,
847             XmRFontList, 0, &argok)); if (argok) ac++;
848         XtSetArg(args[ac], XmNlabelString,
849             BX_CONVERT(aboutScrolledWindow, ":::t\"   MimicSockets\":t\"\":t\"Send socket mes
sages\":t\"from a file\":t\"\":t\"National Research Council\":t\"  of Canada (2002)\":t\"By: Dan
Johnston\":t\"All rights reserved!\"",
850             XmRXmString, 0, &argok)); if (argok) ac++;
851         XtSetArg(args[ac], XmNrecomputeSize, False); ac++;
852         XtSetArg(args[ac], XmNwidth, 220); ac++;
853         XtSetArg(args[ac], XmNheight, 250); ac++;
854         aboutHelpLabel = XmCreateLabel(aboutScrolledWindow,
855             "aboutHelpLabel",
856             args,
857             ac);
858         XtManageChild(aboutHelpLabel);
859
860         ac = 0;
861         XtSetArg(args[ac], XmNlabelString,
862             BX_CONVERT(aboutHelpBulletinBoard, "Dismiss",
863             XmRXmString, 0, &argok)); if (argok) ac++;
864         XtSetArg(args[ac], XmNx, 70); ac++;
865         XtSetArg(args[ac], XmNy, 250); ac++;
866         XtSetArg(args[ac], XmNwidth, 127); ac++;
867         XtSetArg(args[ac], XmNheight, 35); ac++;
868         aboutDismissPushButton = XmCreatePushButton(aboutHelpBulletinBoard,
869             "aboutDismissPushButton",
870             args,
871             ac);
872         XtManageChild(aboutDismissPushButton);
873         XtAddCallback(aboutDismissPushButton, XmNactivateCallback, aboutDismissCallback, (XtP
ointer)0);
874         ac = 0;
875         XtSetArg(args[ac], XmNmenuHelpWidget, helpCascadeButton); ac++;
876         XtSetValues(menuBar, args, ac);
877
878         ac = 0;
879         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
880         XtSetArg(args[ac], XmNleftOffset, 400); ac++;
881         XtSetArg(args[ac], XmNtopOffset, 250); ac++;
882         XtSetValues(endNumbTextField, args, ac);
883
884         ac = 0;
885         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
886         XtSetArg(args[ac], XmNleftOffset, 330); ac++;
887         XtSetArg(args[ac], XmNtopOffset, 250); ac++;
888         XtSetValues(endNumbLabel, args, ac);
889
890         ac = 0;
891         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
892         XtSetArg(args[ac], XmNleftOffset, 140); ac++;
893         XtSetArg(args[ac], XmNtopOffset, 250); ac++;
894         XtSetValues(startNumbTextField, args, ac);
895
896         ac = 0;
897         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
898         XtSetArg(args[ac], XmNleftOffset, 60); ac++;
899         XtSetArg(args[ac], XmNtopOffset, 250); ac++;
900         XtSetValues(startNumbLabel, args, ac);
```

```
901
902         ac = 0;
903         XtSetArg(args[ac], XmNrightAttachment, XmATTACH_POSITION); ac++;
904         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
905         XtSetArg(args[ac], XmNrightPosition, 78); ac++;
906         XtSetArg(args[ac], XmNleftOffset, 390); ac++;
907         XtSetArg(args[ac], XmNtopOffset, 210); ac++;
908         XtSetValues(speedTextField, args, ac);
909
910         ac = 0;
911         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
912         XtSetArg(args[ac], XmNleftOffset, 330); ac++;
913         XtSetArg(args[ac], XmNtopOffset, 190); ac++;
914         XtSetValues(fasterLabel, args, ac);
915
916         ac = 0;
917         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
918         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
919         XtSetArg(args[ac], XmNtopOffset, 190); ac++;
920         XtSetValues(slowerLabel, args, ac);
921
922         ac = 0;
923         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
924         XtSetArg(args[ac], XmNleftOffset, 180); ac++;
925         XtSetArg(args[ac], XmNtopOffset, 190); ac++;
926         XtSetValues(speedLabel, args, ac);
927
928         ac = 0;
929         XtSetArg(args[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
930         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
931         XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_NONE); ac++;
932         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
933         XtSetArg(args[ac], XmNtopOffset, 210); ac++;
934         XtSetValues(speedScrollBar, args, ac);
935
936         ac = 0;
937         XtSetArg(args[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
938         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
939         XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_POSITION); ac++;
940         XtSetArg(args[ac], XmNbottomPosition, 95); ac++;
941         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
942         XtSetArg(args[ac], XmNtopOffset, 300); ac++;
943         XtSetValues(quitPushButton, args, ac);
944
945         ac = 0;
946         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
947         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
948         XtSetArg(args[ac], XmNtopOffset, 140); ac++;
949         XtSetValues(startPushButton, args, ac);
950
951         ac = 0;
952         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
953         XtSetArg(args[ac], XmNleftOffset, 310); ac++;
954         XtSetArg(args[ac], XmNtopOffset, 90); ac++;
955         XtSetValues(portTextField, args, ac);
956
957         ac = 0;
958         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
959         XtSetArg(args[ac], XmNleftOffset, 240); ac++;
960         XtSetArg(args[ac], XmNtopOffset, 90); ac++;
961         XtSetValues(portLabel, args, ac);
962
963         ac = 0;
964         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
965         XtSetArg(args[ac], XmNleftOffset, 310); ac++;
966         XtSetArg(args[ac], XmNtopOffset, 60); ac++;
```



```
967         XtSetValues(destinationTextField, args, ac);
968
969         ac = 0;
970         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
971         XtSetArg(args[ac], XmNleftOffset, 240); ac++;
972         XtSetArg(args[ac], XmNtopOffset, 60); ac++;
973         XtSetValues(destinationLabel, args, ac);
974
975         ac = 0;
976         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
977         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
978         XtSetArg(args[ac], XmNtopOffset, 90); ac++;
979         XtSetValues(fileToggleButton, args, ac);
980
981         ac = 0;
982         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
983         XtSetArg(args[ac], XmNleftOffset, 20); ac++;
984         XtSetArg(args[ac], XmNtopOffset, 60); ac++;
985         XtSetValues(networkToggleButton, args, ac);
986
987         ac = 0;
988         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
989         XtSetArg(args[ac], XmNleftOffset, 480); ac++;
990         XtSetArg(args[ac], XmNtopOffset, 10); ac++;
991         XtSetValues(browseButton, args, ac);
992
993         ac = 0;
994         XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
995         XtSetArg(args[ac], XmNleftOffset, 110); ac++;
996         XtSetArg(args[ac], XmNtopOffset, 10); ac++;
997         XtSetValues(inputFileTextField, args, ac);
998
999         ac = 0;
1000        XtSetArg(args[ac], XmNtopAttachment, XmATTACH_POSITION); ac++;
1001        XtSetArg(args[ac], XmNrightAttachment, XmATTACH_POSITION); ac++;
1002        XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
1003        XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_NONE); ac++;
1004        XtSetArg(args[ac], XmNbottomOffset, 0); ac++;
1005        XtSetArg(args[ac], XmNtopPosition, 3); ac++;
1006        XtSetArg(args[ac], XmNrightPosition, 21); ac++;
1007        XtSetArg(args[ac], XmNleftOffset, 20); ac++;
1008        XtSetValues(inputFileLabel, args, ac);
1009
1010
1011
1012        /* Begin user code block <end_CreatemainWindow> */
1013        /* End user code block <end_CreatemainWindow> */
1014        return( mainWindow );
1015    }
```

```
 1  /*
 2   * WARNING: This file is overwritten at code generation time.
 3   * Any changes to this file will be lost.
 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_BX_GLOBALS
21  #define EXTERNAL
22  #else
23  #define EXTERNAL extern
24  #endif
25
26  /*
27   * Start Global Widget Declerations.
28   */
29  EXTERNAL Widget   aboutHelpBulletinBoard;
30  EXTERNAL Widget   aboutScrolledWindow;
31  EXTERNAL Widget   inputFileSelectionBox;
32  EXTERNAL Widget   networkOptToggleButton;
33  EXTERNAL Widget   fileOptToggleButton;
34  EXTERNAL Widget   debugToggleButton;
35  EXTERNAL Widget   endNumbTextField;
36  EXTERNAL Widget   startNumbTextField;
37  EXTERNAL Widget   speedTextField;
38  EXTERNAL Widget   resumePushButton;
39  EXTERNAL Widget   pausePushButton;
40  EXTERNAL Widget   startPushButton;
41  EXTERNAL Widget   portTextField;
42  EXTERNAL Widget   destinationTextField;
43  EXTERNAL Widget   fileToggleButton;
44  EXTERNAL Widget   networkToggleButton;
45  EXTERNAL Widget   inputFileTextField;
46  /*
47   * End Global Widget Declarations.
48   */
49  #endif
```

```
/*
 * README: Portions of this file are merged at file generation
 * time. Edits can be made *only* in between specified code blocks, look
 * for keywords <Begin user code> and <End user code>.
 */
/*
 * Generated by the ICS Builder Xcessory (BX).
 *
 * Builder Xcessory Version 5.0
 * Code Generator Xcessory 5.0 (05/22/98)
 *
 */


/* Begin user code block <file_comments> */
/**
    \file play/main-c.c
    \ingroup mod_play

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

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since April 2002

    $Id: main-c.c,v 1.6 2002/08/19 18:46:55 johnston Exp $
*/
/* End user code block <file_comments> */

/*
 * Motif required Headers
 */
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#if (XmVersion >= 1002)
#include <Xm/RepType.h>
#endif
#include <Xm/MwmUtil.h>

/*
 * Globally included information.
 */


/*
 * Headers for classes used in this program
 */

/*
 * Common constant and pixmap declarations.
 */
```



```
#include "creation-c.h"

/*
 * Convenience functions from utilities file.
 */
extern void RegisterBxConverters(XtAppContext);
extern XtPointer BX_CONVERT(Widget, char *, char *, int, Boolean *);
extern XtPointer BX_DOUBLE(double);
extern XtPointer BX_SINGLE(float);
extern void BX_MENU_POST(Widget, XtPointer, XEvent *, Boolean *);
extern Pixmap XPM_PIXMAP(Widget, char **);
extern void BX_SET_BACKGROUND_COLOR(Widget, ArgList, Cardinal *, Pixel);

/* Begin user code block <globals> */
#include "../common/mySocket.h"
#include "../common/myInterface.h"
/// the block of user defined globals shared by this interface
globalMemory shared;
/* End user code block <globals> */

/*
 * Change this line via the Output Application Names Dialog.
 */
#define BX_APP_CLASS "BuilderProduct"

int main( int argc, char **argv)
{
    Widget       parent;
    XtAppContext app;
    Arg          args[256];
    Cardinal     ac;
    Boolean      argok=False;
    Widget   topLevelShell;
    Widget   mainWindow;

    /* Begin user code block <declarations> */
    char tmpBuf[MAX_NUMBER_TEXT_SIZE];
    /* End user code block <declarations> */

    /*
     * The applicationShell is created as an unrealized
     * parent for multiple topLevelShells.  The topLevelShells
     * are created as popup children of the applicationShell.
     * This is a recommendation of Paul Asente & Ralph Swick in
     * _X_Window_System_Toolkit_ p. 677.
     */

    parent = XtVaAppInitialize ( &app, BX_APP_CLASS, NULL, 0,
#ifndef XlibSpecificationRelease
    (Cardinal *) &argc,
#else
#if (XlibSpecificationRelease>=5)
    &argc,
#else
    (Cardinal *) &argc,
#endif
#endif
    argv, NULL,
    NULL );

    RegisterBxConverters(app);
#if (XmVersion >= 1002)
    XmRepTypeInstallTearOffModelConverter();
#endif

    /* Begin user code block <create_shells> */
```

```
133     ac = (int) argok;  /* use the variable to surpress compiler warnings */
134     /* End user code block <create_shells> */
135
136     /*
137      * Create classes and widgets used in this program.
138      */
139
140     /* Begin user code block <create_topLevelShell> */
141     /**
142     \ingroup mod_play
143
144     This main program is mostly computer generated.  We only modify it
145     (between the "user code block" comments) to create our global memory
146     structure and initalize 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 is no supplied args */
153     shared.inputFileName[0] = shared.address[0]
154                             = shared.portNumbText[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.fileDescrip = 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 arg 2 */
169     case 3:
170         /* we have 2 args, the destination and the port number */
171         /* process the port number here */
172         if( sscanf( argv[2], "%u", &shared.portNumb ) != 1 )
173               {
174               printf("\nNot a valid port number '%s'\n", argv[2] );
175               usageSend();
176               exit(1);
177               }
178         strcpy( shared.portNumbText, argv[2] );
179         /* no 'break' here, fall through to process arg 1 */
180     case 2:
181         /* we have 1 arg, the destination address */
182         strcpy( shared.address, argv[1] );
183         break;
184     case 1:
185         /* we have no user supplied command args */
186         break;
187     default:
188         usageSend();
189         exit(1);
190         break;
191     }
192     /* End user code block <create_topLevelShell> */
193
194     ac = 0;
195     XtSetArg(args[ac], XmNx, 612); ac++;
196     XtSetArg(args[ac], XmNy, 414); ac++;
197     XtSetArg(args[ac], XmNwidth, 619); ac++;
198     XtSetArg(args[ac], XmNheight, 385); ac++;
```



```
199     topLevelShell = XtCreatePopupShell("topLevelShell",
200         topLevelShellWidgetClass,
201         parent,
202         args,
203         ac);
204     mainWindow = (Widget)CreatemainWindow(topLevelShell);
205     XtManageChild(mainWindow);
206     XtPopup(XtParent(mainWindow), 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( strlen( 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( strlen( shared.inputFileName ) == 0 )
221         XmTextFieldSetString( inputFileTextField, "<none>");
222     else
223         XmTextFieldSetString( inputFileTextField, shared.inputFileName );
224
225     /* the default value for the network port number if not user supplied */
226     if( strlen( shared.portNumbText ) == 0 )
227     {
228         shared.portNumb = DEFAULT_PORT_NUMBER;
229         sprintf( shared.portNumbText, "%d", shared.portNumb );
230         XmTextFieldSetString( portTextField, shared.portNumbText );
231     }
232     else
233         XmTextFieldSetString( portTextField, shared.portNumbText );
234     sscanf( shared.portNumbText, "%d", &shared.portNumb );
235
236     /* the initial value for the speed multiplyer */
237     XmTextFieldSetString( speedTextField, "1x" );
238     shared.speedFactor = 1.0f;
239
240     /* the initial values for the toggle buttons */
241     XmToggleButtonSetState( debugToggleButton, False, False );
242     shared.globalDebugMode = 0;
243     /* the toggle initial values depends upon port number */
244     if( shared.portNumb == 0 )
245         {
246         /* the output to file mode */
247         XmToggleButtonSetState( networkToggleButton, False, False );
248         XmToggleButtonSetState( networkOptToggleButton, False, False );
249         XmToggleButtonSetState( fileToggleButton, True, False );
250         XmToggleButtonSetState( fileOptToggleButton, True, False );
251         }
252     else
253         {
254         /* the normal 'network' state */
255         XmToggleButtonSetState( networkToggleButton, True, False );
256         XmToggleButtonSetState( networkOptToggleButton, True, False );
257         XmToggleButtonSetState( fileToggleButton, False, False );
258         XmToggleButtonSetState( fileOptToggleButton, False, False );
259         }
260
261     /* the initial 'enabled' stat for pushbuttons */
262     /* start enabled, pause and resume disabled */
263     XtSetSensitive( startPushButton, True ); /* it already is, by default */
264     XtSetSensitive( pausePushButton, False );
```

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

```
/**
    \file play_support.c
    \ingroup mod_play

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

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since May 2002

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

/// a counter of where we are in reading file
unsigned int frameNumber;

/**
    \ingroup mod_play
    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.
*/
void
usageSend( void )
        {
        printf("\nThe required command is of the form...\n");
        printf("   sendSocket {destination [portNum [filename]]]\n");
        printf("       - where 'destination' is the network  or file destination\n");
        printf("       - where 'portNum' is the port number (0 indicates file)\n");
        printf("       - and  'filename' is the file with socket data\n");
        }


```

```
/**
    \file sending.c
    \ingroup mod_play

   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.

    \code
    Copyright (c) 2002
    All content of this file is copyrighted to the
    National Research Council of Canada.
    All rights reserved.
    \endcode
    \author Daniel F. Johnston
    \since May 2002

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

#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"

/// the byte array that will hold a UDP socket message for sending
char sendBuf[MAX_MESSAGE_SIZE];
/// the socket address structure for the destination (if defined)
struct sockaddr_in server;

/**
    \ingroup mod_play
    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.
    \param 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.
    \param port is the integer port number to use for the new socket
    \return an error indication (if < 0) or the integer value of the socket
*/
int
createSendSocket( char *address, int port )
    {
    int sock;
    struct hostent *hp, *gethostbyname();

    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if ( sock < 0)
        {
        perror("opening stream socket");
        return -1;
        }

    /* Connect socket using hostname provided */
    hp = gethostbyname( address );
    if (hp == 0)
        {
        fprintf(stderr, "%s: unknown host, \
```



```
                enter valid host name\n", address);
        return -1;
        }
    bzero((char *)&server, sizeof (server));
    bcopy(hp->h_addr, (char *)&server.sin_addr, hp->h_length);
    /*   Assign client port number */
    server.sin_family = AF_INET;
    server.sin_port = port;
    return sock;
    }

/* send the user supplied message */
/**
    \ingroup mod_play
    This function will send a message via an existing socket.
    \param sock is the integer socket to use for sending
    \param buffer is an array of bytes to be sent via the socket.  Buffer
      contents do not have to be only ASCII characters.
    \param sizeBuffer is the integer length of the user supplied message
    \return an error indication (if < 0) or the number of characters sent
*/
int
sendSocketMessage( int sock, char *buffer, int sizeBuffer )
    {
    int i;
    if (i = sendto(sock, buffer, sizeBuffer, 0,
                (struct sockaddr *)&server, sizeof server  ) < 0)
        {
        perror("writing to socket");
        return -1;
        }
    return i;
    }

/* the timer callback to reset the state of the start/pause/resume buttons */
/**
    \ingroup mod_play
    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.
*/
void
resetSendButtons()
    {
    extern globalMemory shared;
    /* disable the pause and resume button, reenable the start */
    XtSetSensitive( startPushButton, True );
    XtSetSensitive( pausePushButton, False );
    XtSetSensitive( resumePushButton, False );
    /* the child process is no more */
    shared.childProcess = 0;
    }

/* the "child should pause" signal handler */
/**
    \ingroup mod_play
    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.
*/
void
sendProcessPause()
    {
    int pid;
    int status;
```

```
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.globalDebugMode == 1 )
139             printf("Child process pause: fork error!\n");
140         return;
141         }
142
143     if( pid == shared.childProcess )
144         {
145         /* we pause the current child process */
146         pause();
147         return; /* we return only if 'resumed' by the main process */
148         }
149     /* else */
150     /* some other process id */
151     if( shared.globalDebugMode == 1 )
152         printf("Child process pause: pid %d received!\n", pid);
153     }
154
155 /* the "child dies" signal handler */
156 /**
157     \ingroup mod_play
158     This function will be called by the child (sending) process when 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.globalDebugMode == 1 )
174             printf("Child process done: fork error!\n");
175         return;
176         }
177
178     if( pid == shared.childProcess )
179         /* our expected child process died, reset the button state */
180         /* we cannot do it here because the child process cannot
181          *   access the same user interface as the main program
182          */
183         {
184             XtAppAddTimeOut( shared.mainAppContext, 0, resetSendButtons, NULL );
185             return;
186         }
187     /* else */
188     /* some other process died */
189     if( shared.globalDebugMode == 1 )
190         printf("Child process done: pid %d received!\n", pid);
191     }
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     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 fileD is the file descriptor for the (already opened) file
203      which contains the socket messages and delays.
204     \param outFileD 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 *fileD, FILE *outFileD )
211     {
212     extern globalMemory shared;
213     timespec_t delay;
214     int length;
215     char buffer[MAX_MESSAGE_SIZE];
216     int itemsRead;
217     int itemsWrote;
218
219     for(;shared.messageNumber<=shared.endMesNumb;shared.messageNumber++)
220         {
221         /* read the first part of the delay from the file */
222         itemsRead = fread( &delay.tv_sec, sizeof(long), 1, fileD );
223         if( itemsRead != 1 )
224             return 0;
225         /* read the second part of the delay from the file */
226         itemsRead = fread( &delay.tv_nsec, sizeof(long), 1, fileD );
227         if( itemsRead != 1 )
228             return 0;
229         /* read the size of the message from the file */
230         itemsRead = fread( &length, sizeof(int), 1, fileD );
231         if( itemsRead != 1 )
232             return 0;
233         if( length < 0 || length > MAX_MESSAGE_SIZE )
234             return -1;
235         itemsRead = fread( buffer, length, 1, fileD  );
236         if( itemsRead != 1 )
237             return 0;
238
239         /* only send/save after we read past the start number */
240         if( shared.messageNumber >= shared.startMesNumb )
241             {
242             if( XmToggleButtonGetState( networkToggleButton )
243                     == False)
244                 {
245                 /* if we have a file destination, save all */
246                 itemsWrote = fwrite( &delay.tv_sec, sizeof(long),1,outFileD);
247                 if( itemsWrote != 1 )
248                     return 0;
249                 itemsWrote = fwrite( &delay.tv_nsec, sizeof(long),1,outFileD);
250                 if( itemsWrote != 1 )
251                     return 0;
252                 itemsWrote = fwrite( &length, sizeof(int),1,outFileD);
253                 if( itemsWrote != 1 )
254                     return 0;
255                 itemsWrote = fwrite( buffer, length, 1, outFileD);
256                 if( itemsWrote != 1 )
257                     return 0;
258                 }
259             else
260                 {
261                 delay.tv_sec /= shared.speedFactor;
262                 delay.tv_nsec /= shared.speedFactor;
263                 nanosleep( &delay, NULL);
264                 /*  write the character string to the socket
```

```
265                  *   and indicate an error if unsuccessful
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("Fs %d\n", shared.messageNumber);
276             }
277         }
278     /* we completed the send! */
279     return 0;
280     }
281
```

Appendix D

Other Source Code

```
 1  #!/bin/sh
 2  #
 3  # simple script to generate listing files of source
 4  #  convert them to postscript, and then print them
 5  #  two pages per page.
 6
 7  if [ -s $1 ]
 8  then
 9  # Get a copy of the file with line numbers
10    nl -ba $1 > $1".lst"
11  # convert to postscript
12    /usr/lib/print/lptops -G -E $1 -N1 -W -U -T1.0in -B1.0in -FCB $1".lst" > $1".ps"
13  # convert postscript to 2-pages-per-page postscript
14    /disk2/people/oliver/bin/psnup -2 -m0.5in -d1 < $1".ps" > $1".2ps"
15  else
16    echo ""
17    echo "file not found"
18    echo ""
19    exit 1
20  fi
```