CHAMELEON 32
C DEVELOPMENT SYSTEM
TASK COMMUNICATION LIBRARY

Version 1.0

TEKELEC
26580 Agoura Road
Calabasas, California
91302

Part Number 910-3439

July 24, 1990

Copyright© 1990, Tekelec.

All Rights reserved.

This document in whole or in part, may not be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine-readable form without prior written consent from *TEKELEC*.

Tekelec® is a registered trademark of *TEKELEC*.
Chameleon® is a registered trademark of *TEKELEC*.

TABLE OF CONTENTS

CHAPTER	DESCRIPTION	PAGE
ONE	**CHAMELEON 32 APPLICATION PROGRAMMING INTERFACE**	
	Introduction	1-1
	Installation Instructions	1-3
	User Interface Components	1-4
	Application Programming Interface Library	1-5
	API Functions	1-5
	API Requests	1-7
TWO	**APPLICATION PROGRAMMING INTERFACE DEVELOPMENT ENVIRONMENT**	
	Introduction	2-1
	Application Programming Interface Files	2-1
	Building an Interface File	2-3
	Global Initializations	2-3
THREE	**APPLICATION PROGRAMMING INTERFACE LIBRARY FUNCTIONS**	
	Introduction	3-1
	addNewLine	3-3
	cSToggle	3-4
	eraseEOS	3-6
	fillBoxArea	3-7
	getBoxArea	3-8
	getFileChoice	3-9
	initUI	3-12
	unMark	3-13
	userInterface	3-14
FOUR	**APPLICATION PROGRAMMING INTERFACE LIBRARY REQUESTS**	
	Introduction	4-1
	Notes	4-2
	Requests	
	BOX_INPUT	4-3
	BOX_REQ	4-6
	DSP_REQ	4-13
	ERASE_FIELD	4-15
	ERASEB_REQ	4-17
	ERASEW_REQ	4-18
	INPUT_REQ	4-19
	REL_REQ	4-26
	WINDOW_REQ	4-27

TABLE OF CONTENTS (continued)

CHAPTER	DESCRIPTION	PAGE
FIVE	**APPLICATION PROGRAMMING INTERFACE EXAMPLES**	
	Introduction	5-1
	Example One: Pull Down Menu Logic	5.1-1
	Example Two: Parameter Input	5.2-1
	Example Three: Listing Files from a Directory	5.3-1
Appendix A	Inlcude File UI.H	
Appendix B	Include File MAINSYM.H	

Chapter 1:
CHAMELEON 32
APPLICATION PROGRAMMING INTERFACE

Introduction

The Application Programming Interface (API) is designed to provide a uniform user interface for applications created on the Chameleon 32. These applications and the user interface are created within the Chameleon 32 C Development System.

The look and feel provided by this package is similar to pull down menu packages common in many other environments.

The library provides tools to develop standardized and modular application code facilitating enhancements, readability and transferability of code.

This document assumes some familiarity with the Chameleon 32 C Package. For more detailed information than what is provided here, refer to the *Chameleon 32 C Manual, Volume IV.*

Applications developed using the API can be run on both a Chameleon 32 and a Chameleon 20 containing the C run-time module.

API provides a quick and effective way to develop a user interface. Using this type of interface, you can:

- transfer information to and from the application
- provide easy access to the current configuration parameters
- modify the parameters during runtime
- verify that any changes made are within a valid range
- make selections from a list of options
- select a file from a specified path
- chain lists and parameter input fields in any order

The functions within the API library provide you with complete control over all of the attributes regarding both the look and function of each part of the display.

- The boxes and windows can be overlayed.
- For each box or window, the following attributes are easily controlled:
  - location of the window or box on the screen
  - appearance of the window or box including:
    - borders
    - surrounding arrows
    - highlighting of current selection
    - position of the text within the box or window
    - color of the text, highlight and outline
    - parameter input as either hex, integer or string
    - automatic range checking
    - prompting the user when input is required

Installation Instructions

The API software is installed through the Chameleon 32 C shell using the batch file ***INSTALL*** included on the diskette.

To install the software, at the C prompt %, enter:

BATCH B:\INSTALL

This batch file creates the necessary directories and copies both the Library and Example files. The files are copied to the directories shown in Figure 1.1.

FILENAME	DESCRIPTION	DIRECTORY
libui.a	API Library	A:\LIB
mainsym.h	General symbols	A:\INCLUDE
ui.h	API specific symbols	A:\INCLUDE
SELECT.C	Ex. 1, Pull Down Menus	A:\USR\API\EX1
MAKEFILE	Creates example 1	A:\USR\API\EX1
UITAB.C	API table for example 1	A:\USR\API\EX1
UITAB.H	API externals for example 1	A:\USR\API\EX1
EX2.C	Ex. 2, Parameter Input	A:\USR\API\EX2
MAKEFILE	Creates example 2	A:\USR\API\EX2
UITAB.C	API table for example 2	A:\USR\API\EX2
UITAB.H	API externals for example 2	A:\USR\API\EX2
EX3.C	Ex. 3, Listing Files	A:\USR\API\EX3
MAKEFILE	Creates example 3	A:\USR\API\EX3
UITAB.C	API table for example 3	A:\USR\API\EX3
UITAB.H	API externals for example 3	A:\USR\API\EX3

Figure 1.1: Installation and File Directories

Note that the UITAB.C and UITAB.H files for each example are not the same. They contain the unique definitions that create the display for each example.

User Interface Components

A user interface created using the Application Programming Interface consists of boxes and windows. A box, or list selector, contains several strings of information, with each string made up of one field. These are typically used to display a group of options, for example commands or messages, and then to accept a selection from the user.

There are two types of windows. The simplest type displays strings of information that can scroll either up or down within the window. The second type displays strings of information at a fixed location.

Within the second type of window, the information displayed can take two forms, either an unformated string or a field made up of two parts, a title or description and a value. A field can be used to accept user input, this is called an input field. A window can contain a sequence of input fields.

Figure 1.2 is an example of an interface created using the application programming interface. This menu is used in the NT/TE Simulator for the Chameleon 32 or 20. It consists of three windows and one box.

- The first window displays the menu strip along the top edge of the page. On this window, the borders and the selection highlight are turned off. This is an unformatted string.
- The second window displays the name of the application along the bottom edge.

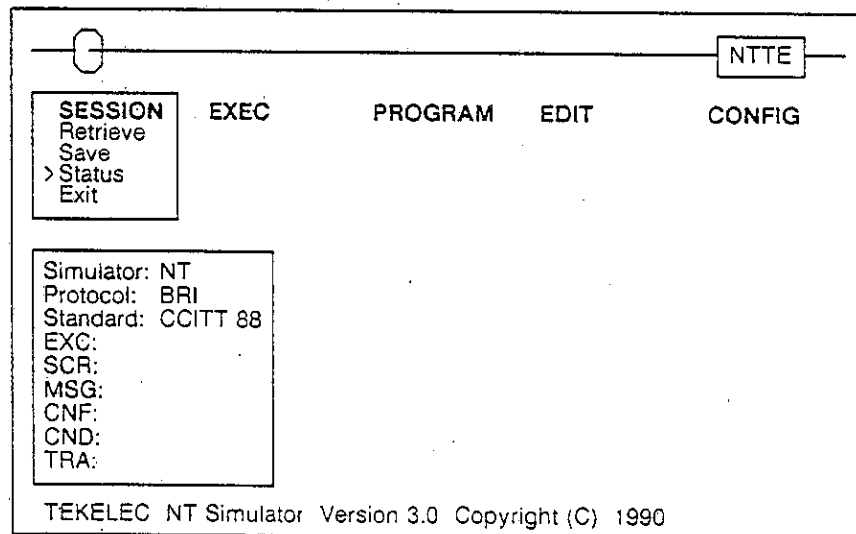


Figure 1.2: The User Interface

- The status shown on the page is shown within the third window. Each field has a description with an associated value. For example, the description of the first variable is *Simulator* and the value is *NT*. This is an example of a window containing fields.

- The box is the session list selector. It is used to select from a group of session commands.

A standard set of keys are used to move around or exit from the boxes and windows. The arrow keys, ←↑↓→, are used to move within a frame, where a frame is either a box or a window. The keys ***GO***, ***ESC***, ***CAN*** and ***RTN*** are used to exit a frame. The action taken is determined through the software.

Application Programming Interface Library

The Application Programming Interface Library provides nine functions and nine commands or requests. The functions, described in Chapter 3, use the data structures established through the requests, to display the appropriate information. This section provides a brief description of each function and request.

FUNCTIONS

Two of the functions provide the majority of the API functionality.

- **initUI**

  This function is used to initialize the user interface. It is used only once, when the interface is first started. It performs the following functions:

  - it specifies the window and box administration areas
  - it defines the number of windows and boxes that make up the display
  - it initiates an error window

- **userInterface**

  The user interface is a request oriented library. To initiate any of the nine requests, the appropriate data structures are set up and a call is made to the function **userInterface.**

  This is the primary function within the API. All requests are initiated through this function.

The seven additional functions, shown in Figure 1.3, are used in conjunction with the request BOX_REQ. They are used to control the information displayed within a box.

FUNCTION	OPERATION DESCRIPTION
addNewLine	Adds a new line to a list selector.
cSToggle	Toggles between a tag, for example the character *, and a blank space at a designated location within a list selector.
eraseEOS	Erases the screen from line 3 downward.
fillBoxArea	Initializes a box for use.
getBoxArea	Allocates space to the scrolling area of a list selector.
getFileChoice	Initializes a list box to display file names from a specified path.
unMark	Removes all *Marks* set within a list selector.

Figure 1.3: Additional API Functions

Each of these functions are described in detail in Chapter 3. Examples showing their use are provided in Chapter 5.

REQUESTS

The nine requests are summarized in Figure 1.4. The details can be found in Chapter 4.

REQUEST	PAGE	OPERATION DESCRIPTION
BOX_INPUT	4.3	Create a list selector at run-time. This allows a dynamic creation of choices.
BOX_REQ	4.5	Display a box or list selector.
DSP_REQ	4.11	Display text within the window.
ERASE_FIELD	4.13	Erase an entire field, both the value and the description, from the screen.
ERASEB_REQ	4.15	Erase a box or list selector.
ERASEW_REQ	4.16	Erase a window from the screen.
INPUT_REQ	4.17	Display a sequence of fields to be edited.
REL_REQ	4.22	Releases the memory allocated for a specific window.
WINDOW_REQ	4.23	Initialize the window description.

Figure 1.4: Application Programming Interface Requests

Chapter 2:
APPLICATION PROGRAMMING INTERFACE DEVELOPMENT ENVIRONMENT

Introduction

The Application Programming Interface (API) is accessed in the format of a library similar to the other libraries available within the Chameleon 32 C Development system. Refer to the *Chameleon 32 C Manual, Volume IV* for general information on include files, library files and the method of building an application.

Application Programming Interface Files

There are five files used during the development of an application program interface. The first three are provided with the Application Programming Interface package. They provide the following functions:

- **libui.a**

  The file libui.a is the library file which contains all of the logic behind the library functions. It is located in the directory **A:\LIB.**

  This file is used at link time. For example -LUI in the command, **cc -0 MENU uitab.c -LUI**, links the ui.a library to your source code.

- **ui.h**

  The file ui.h is the header file which includes all of the definitions and structures used as arguments for the library functions. It is located in the directory **A:\include.**

  Ui.h is shown in Appendix A of this document. It must be included in the source code as shown in the examples found in Chapter 5.

  ```
  #include <ui.h>
  ```

- **mainsym.h**

  The file mainsym.h is the header file which includes the definitions for internal type declarations and Chameleon specific attributes such as key codes or colors. It is located in the directory **A:\include.**

  Mainsym.h is shown in Appendix B of this document. It must be included in the source code as shown in the examples provided in Chapter 5.

  ```
  #include <mainsym.h>
  ```

  NOTE: mainsym.h must be included before ui.h.

The remaining two files are application unique. They are part of the application files. These files are not required but are recommended to provide a uniformity between all applications using this package.

- **uitab.c**

  The file uitab.c is the user interface program file which contains the initialization of structure parameters for each menu request. It is located in the development directory.

- **uitab.h**

  The file uitab.h is the header file which includes the external declaration of the declarations made in uitab.c. It is located in the development directory.

Building an Interface File

The following example illustrates how these files are used to build the executable interface file.

```
cc -o MENU menu.c uitab.c -LUI
```

Each part of this command is defined as follows:

- cc The command used to compile and link.
- -o This option for the cc command is used to name the resulting executable file other than the standard A.out. In this example, the file is output to MENU.
- MENU This is the name associated with the -o command. After a successful link and compile, it will contain the executable file.
- menu.c The main program of the sample application.
- uitab.c Initialization code for the request parameters.
- -LUI The user interface library (libui.a) to link with.

Global Initializations

When using the API library, the following structures and definitions must be declared:

NUM_OF_WINDOWS

NUM_OF_BOXES

DISPLAY dsp [NUM_OF_WINDOWS]
BOX box [NM_OF_BOXES]

error_window
err_str
errDsp
noTitle

The example shown on the following page illustrates these initializations. Further examples can be found in Chapter 5.

```
/*  Mandatory definitions and external declarations,
    for example in the file uitab.h */

#define   ERROR_WIN          0
                                  /* These parameters        */
#define   NUM_OF_WINDOWS     30   /* define the number of    */
# define  NUM_OF_BOXES       30   /* windows and boxes       */
                                  /* active at any time      */
                                  /* within an application.  */
extern DISPLAY               dsp[];
extern BOX                   box[];

extern WINDOWREQ  error_win;

extern FIELD errStr = {22, 1, NIL};

/*  Mandatory declarations, for example in the file uitab.c        */

DISPLAY  dsp[NUM_OF_WINDOWS];  /*  System configuration.  */
BOX      box[NUM_OF_BOXES];    /*  System configuration.  */

FIELD errStr = {22, 1, NIL};

DSPREQ   errDsp =
    {
    DSP_REQ,
    0,
    ERR_WIN,
    (byte *) &errStr
    };

FIELD  noTitle = {1, 1, ""};        /*  empty string               */

WINDOWREQ  err_win =
    {WINDOW-REQ, 0, ERROR-WIN, STATIC, 40, 40, '2', 1, 20,
     FALSE, 20, 20, NOF, FALSE, '7', &noTitle, NIL};
```

Chapter 3:
APPLICATION PROGRAMMING INTERFACE LIBRARY FUNCTIONS

Introduction

A user interface created using API directs any input from the keyboard or output to the screen through a set of requests. All types of requests are initiated through the function **userInterface()** with the first parameter defining the request type. This function, along with **initUI** for interface initialization provide the main functionality of the user interface. These are summarized in Figure 3.1.

REQUEST	PAGE	OPERATION DESCRIPTION
initUI	3.12	Initializes the user interface.
userInterface	3.14	Provides access to the user interface.

Figure 3.1: API (Main) Functions

The seven additional functions can be thought of as Help functions. They are used with the request types BOX_REQ and BOX_INPUT. These functions are summarized in Figure 3.2.

REQUEST	PAGE	OPERATION DESCRIPTION
addNewLine	3.3	Adds a new line to a list selector.
cSToggle	3.4	Toggles between a tag, for eachample the character *, and a blank space at a designated location .
eraseEOS	3.6	Erases the screen from line 3 downward.
fillBoxArea	3.7	Initializes a box for use.
getBoxArea	3.8	Allocates space to the scrolling area of a list selector.
getFileChoice	3.9	Initializes a list box to display file names from a specified path.
unMark	3.13	Removes all *Marks* set within a list selector. (See cSToggle to remove just one *Mark*.)

Figure 3.2: Additional API (Help) Functions

A complete description of both types of functions can be found in an alphabetical listing in the following pages.

The structures corresponding to the different request types, shown in capital letters, are defined in the include file, ui.h , and in Chapter 4 of this document.

addNewLine()

Declaration

```
addNewLine (s, str)
SCRAREA *s;
byte       *str;
```

Description

The function addNewLine inserts one line at a time to a list selector. This function should be used in conjunction with getBoxArea().

AddNewLine() should be used when fillBoxArea() is not convenient, for example, when the contents of a list selector is determined during runtime of the application or the entries are to be retrieved from a file.

The last line inserted must be equal to the empty string. An error will occur if more lines are added than defined by the BOXREQ parameter lines. You can however, have less entries.

The parameters used with this function are defined as follows:

- *s — A pointer to the scrolling area of a BOXREQ.
- *str — A pointer to the string to be inserted. All inserted strings must be the same length.

Return Value

None

Related Functions

getBoxArea()

Example

The following example uses addNewLine to define the contents of a list selector.

```
BOXREQ  myBox = {.......};          /* initialized parameters*/

getBoxArea (&myBox);
        :
        :
addNewLine (myBox.p, "Title String");
addNewLine (myBox.p, "My Choice");
addNewLine (myBox.p, "Your Choice");
addNewLine (myBox.p, "");           /*mandatory last string*/
        :
        :
userInterface (&myBox, &conf, dsp, box);
```

cSToggle ()

Declaration

```
cSToggle (s, n, mode, ch, ch1)
SCRAREA *s;
int        n, mode;
char       ch, ch1;
```

Description

The cSToggle function is used to mark a specified position within a box or list selector with a character, for example an asterisk. This feature can be used in two ways.

- When multiple selections are made within the list selector, this can be used to mark each selection.
- For a single selection box, it can be used to clarify the selection.

The parameters used with this function are defined as follows:

s	A pointer to the scrolling area within a BOXREQ
n	The position within the box
mode	0 Toggle
	1 Set
ch	First marker character
ch1	Second marker character

If mode is set to Toggle, the function checks for the current state of the marker. If it currently contains *ch1*, it is set to ch. For anything else, it is set to *ch1*.

Note: In order to use this feature, each selectable entry in a box should begin with a blank.

The structure BOXREQ is defined in Chapter 4 with the command BOX_REQ.

Return Value

The output is a value of 0 or 1.

- 0 Set to first marker, ch.
- 1 Set to second marker, ch1.

Related Functions

The function **unMark()** can be used to clear all marks within a box.

Examples

The first example illustrates the function when several test cases are chosen from the list for a particular scenario. Each test case selected is marked with an asterisk. Note that a test case can be deselected by chosing it again and calling the function **csToggle()** which is set to toggle.

```
Test Cases
  tst1.tst
* tst2.tst
* tst3.tst
  tst4.tst
* tst5.tst
* tst6.tst
  tst7.tst
  tst8.tst
```

```
for (;;)
    {
    userInterface(&test_box,&conf.dps,box);
    if (conf.exe==GO or conf.ext==ESC) break;
    csToggle (text_box.p, conf.choice, 0, '*', ' ');
    test_box.setRow = conf.row;/* see BOX_REQ for */
                              /*description of setRow   */
    test_box.offset = conf.choice;
    }
    test_box.setRow = test_box.row;
    test_box.offset = 0;
```

In the second example, a list of .c files is presented in a list selector. The file tst3.c has been selected. The function csToggle() is used to mark the selection, then redisplay the box. At this point, no additional choices are given to the user.

```
Files
  tst1.c
  tst2.c
* tst3.c
  tst4.c
  tst5.c
  tst6.c
```

```
userInterface(&file_box,&conf,dps,box);
csToggle (file_box.p, conf.choice, 0, '*', '*');
file_box.choice = FALSE;
userInterface(&file_box,&conf,dps,box);
file_box.choice = TRUE;
  .
  .
  .
```

eraseEOS()

Declaration	eraseEOS()
Description	This function erases the screen from line 3 downward. It is useful in conjunction with pull down menu logic. There are no parameters required with this function.
Return Value	None
Related Functions	None

fillBoxArea ()

Declaration

```
fillBoxArea (req, strlist)
BOXREQ   *req;
byte     *strlist[];
```

Description

This function initializes the scrolling linked list located within the structure BOXREQ. This must be done once, typically in the beginning of the program, before a box or list selector can be accessed through a call to **userInterface()**.

The parameters are defined as follows:

req	A pointer to BOXREQ
strlist	address of the array containing the strings to be entered in the list box

The structure BOXREQ is defined in Chapter 4 with the command BOX_REQ.

Return Value

None

Related Functions

None

Example

This example illustrates the use of the function **fillBoxArea()**.

```
BOXREQ   message_list = {.......}    ;  /* initialized with parameters */
byte     *messList[] = {"     SETUP",          /* observe that each */
                        "     ALERT",          /*  string must have */
                        "   CALL PR",          /*   the same length */
                                   .
                                   .
                                   .
                       };
fillBoxArea (&message_list, messList);
```

See the beginning of example 1 in Chapter 5 for an efficient way of initializing several boxes in a row.

getBoxArea()

Declaration

```
getBoxArea (breq)
BOXREQ   *breq;
```

Description

This function allocates space to the scrolling area of a list selector. The linked list is also initialized. If the area needs to be re-initialized at any point, this function can be called again.

No strings are put into the list selector. This is done with the function addNewLine().

The pointer *breq, points to the structure BOXREQ. This strucutre is define in Chapter 4 with the command BOX_REQ.

Return Value

None

Related Functions

addNewLine()
fillBoxArea()

getFileChoice ()

Declaration

```
getFileChoice        (boxName, fPath, ext, bTitle, errMsg,
                      insFlag, inserts, fnum, conf)

BOXREQ   *boxName;
byte     *fPath;
byte     *ext;
byte     *bTitle;
byte     *errMsg;
int      insFlag
byte     ** inserts;
int      fnum;
BOXCONF  *conf;
```

Description

This function is used to display a list of files. The function reads the directory specified by the path for each occurance of a file with the specified extension. For each occurance, the filename is loaded into the list selector.

A call to the function **fillBoxArea()** is not necessary to initialize the scroll area within the box as that is done within the function **getFileChoice().**

The parameters used with this function are defined as follows:

*boxName	A pointer to BOXREQ
*fpath	The directory path
*ext	The file extension, three character maximum without the period
*bTitle	The title string to be displayed within the list selector
*errMsg	The error string that will be displayed at the bottom of the screen if no files exist. It is recommended to end this string with **Press CAN to continue** since this is required to continue.
insFlag †	When set to TRUE, this inserts the number of lines specified in fnum into the list box. Otherwise, set to FALSE.
**inserts †	A pointer to an array of strings to be inserted when insFlag = TRUE. Otherwise set to NIL.

fnum †	When insFlag is set to true, this is the number of lines to be inserted.
*conf	A pointer to the BOXCONF structure.

The structures BOXREQ and BOXCONF are defined in Chapter 4 with the command BOX_REQ.

† This function contains an optional insert feature that permits you to insert strings into the list selector prior to the filenames. The insert strings are defined by an array of pointers. This utilizes the parameters **insFlag, **inserts** and **fnum.**

The second example illustrates this feature.

Note: Each time the function getFileChoice is called, the BOXREQ scroll area is reinitialized and all entries are reloaded into the list selector.

Refer to Example 3 in Chapter 5 at the function handle_load(), for an illustration on using this function in conjunction with csToggle.

Return Value

The BOXCONF structure contains exit information.

This parameter conf.str contains the filename which can be used to open the specified file.

Related Functions

None

Example

The first example uses the function **getFileChoice()** to display a box containing the file names in the directory **A:\TEKELEC\SYSTEM** with the extension .co. It does not include the optional inserts.

```
BOXREQ          my_box = {.......};          /* initialized parameters */

getFileChoice   (&my_box, "\\Tekelec\\system\\", "co", "CO FILES",
                "Sorry no .co files.  Press CAN to continue", FALSE,
                 NIL, 0, &conf);

                /* conf.str contains file name */
```

The second example illustrates that same list selector with the additional inserts. Note that the strings within the insert must be the same length as the title string. The length of the title string must be calculated to fit within the length specified by the parameter len in the BOXREQ structure.

```
BOXREQ          my_box = {.......};          /* initialized parameters */

byte *myInserts[] =    {"   NEW    ",
                        " DELETE   ",
                        " DEFAULT  "};

getFileChoice   (&my_box, "\\Tekelec\\system\\", "co", "CO FILES",
                "Sorry no .co files.  Press CAN to continue", TRUE,
                 myInserts, 3, &conf);

                /* conf.str contains file name */
```

initUI ()

Declaration

```
initUI (dsp, box, req, nw, nb)

DISPLAY    *dsp;
BOX        *box;
WINDOWREQ  *req;
int        nw;
int        nb;
```

Description

This function initializes the user interface. initUI() must be called before any other call is made to the interface.

The parameters used with this function are defined as follows:

*dsp	A pointer to the window administration area
*box	A pointer to the list box administration area
*req	window initiation of ERROR_WINDOW (This is required)
nw	NUM_OF_WINDOWS
nb	NUM_OF_BOXES

The structure types DISPLAY and BOX are internal to the userInterface function. In order to use the API, both of these administration areas must be initialized within the user application. This is typically done within uitab.c.

Refer to Chapter 2 for the declaration of dsp, box, NUM_OF_WINDOWS, NUM_OF_BOXES and ERROR_WINDOW

The structure type WINDOWREQ is defined in Chapter 4 with the command WINDOW_REQ.

Return Value

None.

unMark ()

Declaration

```
unMark (s)
SCRAREA     *s;
```

Description

This function removes all marks used to identify selections within a list box. The parameter used with this function is defined as follows:

*s	A scroll area within the box to be cleared

Return Value

None

Related Functions

See **cSToggle()** for information on marking selections.

Example

The following example clears all selections within the box specified by *myBox*.

```
BOXREQ mybox = {.......}; /* initialized parameters */
unMark (myBox.p);
```

userInterface ()

Declaration

```
userInterface (req, conf, dsp, box)
byte        *req;
byte        *conf;
DISPLAY     *dsp;
BOX         *box:
```

Description

This function gives the user access to the user interface. Each of the requests or commands described in Chapter 4 is initiated through a call to this function.

The parameters used with this function are defined as follows:

req	A pointer to the structure containing the request type or event
conf	A pointer to the return value
dsp	A pointer to the window administration area
box	A pointer to the list box administration area

Note: The administration areas *dsp* and *box* must have been previously initialized by calling the function **initUI()**. This is typically done at the beginning of the program.

Return Value

The output is put in a structure of the type CONFIRM, where applicable. This is defined uniquely for each of the requests in Chapter 4.

Chapter 4:
APPLICATION PROGRAMMING INTERFACE LIBRARY REQUESTS

Introduction

The Application Programming Interface library provides nine requests or commands. A brief description of each of these is shown in Figure 4.1.

REQUEST	PAGE	OPERATION DESCRIPTION
BOX_INPUT	4.3	Create a user edited list selector. This allows runtime configured lists of choices.
BOX_REQ	4.4	Display a box or list selector.
DSP_REQ	4.7	Display text within the window.
ERASE_FIELD	4.9	Erase an entire field, both the value and the description, within a window.
ERASEB_REQ	4.11	Erase a box or list selector.
ERASEW_REQ	4.12	Erase a window from the screen.
INPUT_REQ	4.13	Display a sequence of fields to be edited.
REL_REQ	4.17	Releases the memory allocated for a specific window.
WINDOW_REQ	4.18	Initialize the window description.

Figure 4.1: Application Programming Interface Requests

Each of these requests is made up of structures defining the parameters required to complete the request. Once the parameters within each structure are defined, a call is made to the function **userInterface**. It utilizes the information in the structures to complete the request.

The **userInterface** function, along with **initUI** and the additional API (Help) functions are described in Chapter 3 of this document.

Notes

- All definitions of constants (uppercase parameters) can be found in the files **a:\include\ui.h** and **mainsym.h.**
- The KEYS shown for each command are those keys that will return execution to the calling procedure. You must define the response to each of these keys.
- When colors are used within the request, they are identified using the following numbering scheme shown in Figure 4.2.

COLOR	IDENTIFICATION
BLACK	'0'
RED	'1'
GREEN	'2'
YELLOW	'3'
BLUE	'4'
MAGENTA	'5'
CYAN	'6'
WHITE	'7'

Figure 4.2: Display Color Encoding

Within this table, the right column is used as an entry to a request. Using the actual name of the color will cause an error.

BOX_INPUT

Description

The request BOX_INPUT is used to create a list box of selections at run-time. This allows for dynamic creation of choices within a box.

The command BOX_INPUT is similar to the command BOX_REQ. You must be familiar with that command before you use BOX_INPUT.

The following keys are used to edit the fields:

- **CTRL A** Append
- **CTRL I** Insert
- **CTRL D** Delete line

Keys

The following keys can be used to exit from the list selector:

- ESC
- CANCEL
- GO

Parameters

The structure type BOXREQ is used to define the box. This is the same structure type used for the BOX_REQ command. The only difference between the two structures will be the event parameter. In this case it is set to BOX_INPUT.

All of the other parameters are described in detail within the BOX_REQ command. The structure is as follows:

```
typedef struct
    {
    int        event;
    int        taskId;
    int        box;
    int        len;
    byte       color;
    int        col;
    int        row;
    int        clear;
    int        choice;
    int        maxRow;
    int        frame;
    byte       bcolor;
    byte       rev;
    int        lines;
    SCRAREA    *p;
    int        offset;
    int        setRow;
    }BOXREQ;
```

An array of pointers, containing at least two entries, must be initialized before you can initialize a BOX_INPUT. This can be done by calling **fillBoxArea()**.

The example shown on the following will illustrate.

Return Structure

The return parameter **conf** is of the type BOXCONF. This structure contains the information regarding the selection within the list selector, the keystroke used to exit the list, the current selection position on the screen and a pointer to the selected string.

The structure is defined as follows:

```
typedef struct
    {
    int event;
    int exit;
    int choice;
    byte *str;
    int row;
    }BOXCONF;
```

event	BOX_CONF
exit	The key used to exit the list selector (GO, RTN, ESC, CAN, LEFT or RIGHT.)
choice	The number of the selected parameter where 1 is the first non-title string
*str	A pointer to the actual string chosen.
row	The row number on the screen. This defines where the selection is actually located. This value is useful in combination with **setRow**.

Example

The example shown below illustrates the logic used to edit the contents of a list selector. Note that an entry can always be removed by the user by pressing ***CTRL-D***.

This example shows the editing of completely free format data (gets()). This logic is normally replaced by some format specific to the application.

For example, a string displayed within another list selector can be retrieved and copied into the edited list. Note that each string must be adjusted to be the same length.

```
BOXREQ   my_box = {BOX_INPUT, ......},      /*initialized parameters*/

byte     *myBox[] = {"This is the title",
                     "     End Str     "};

fillBoxArea (&my_box, myBox);     /*in the beginning of the program*/
          .
          .
          .
exit = FALSE;
do
  {
   userInterface (&my_box, &conf, dsp, box);
   switch (conf.exit)
    {
     CNTRL_A:
     CNTRL_I: positionCursor (conf.row, my_box.col);
              gets (conf.str);                  /*get an unformatted*/
              fillToLength (conf.str, my_box.len)       /* string*/
              my_box.setRow = conf.row;           /*make sure it has*/
              my_box.offset = conf.choice;        /*the same length */
              break;                              /*as other strings*/
    ESC:
    CAN:      fillBoxArea (&my_box, myBox);        /*reset contents*/
              exit = TRUE;                         /* to original  */
              break;
    GO:       exit = TRUE;                        /*Exit by keeping */
              break;                              /*editions made in*/
    }                                             /*  this session  */
  }
while (!exit);
```

BOX_REQ

Description

A BOX_REQ is used to display a list of selections from which a choice can be made. The UP and DOWN arrow keys are used to move from one choice to another within the box. When the list of choices is longer than the frame size, the list will scroll.

This request returns a structure, type BOXCONF, as described below. This structure contains information about the selection made.

Chapter 5 provides an example showing how this request is used to implement a pull down menu application.

Keys

The following keys can be used to exit from a list selector:

- GO
- ESC
- CAN
- RIGHT
- LEFT
- RETURN

Parameters

The structure type BOXREQ is used to define the box. This includes information on the length of the text strings, the color the border, text and reverse video, the column and row of locations for the text strings, size and amount of interaction possible. The structure is defined as follows:

```
typedef struct
   {
   int        event;
   int        taskId;
   int        box;
   int        len;
   byte       color;
   int        col;
   int        row;
   int        clear;
   int        choice;
   int        maxRow;
   int        frame;
   byte       bcolor;
   byte       rev;
   int        lines;
   SCRAREA    *p;
   int        offset;
   int        setRow;
   }BOXREQ;
```

Each of these parameters are defined on the following page.

event	BOX_REQ.
taskId	Reserved. Set to 0
box	The box identification number, n, where $0 \leq n \leq$ NUM_OF_BOXES. Observe that each BOXREQ definition is associated with a unique box number. The recommended procedure is to increment the value for each box definition.
len	The length of the string to be displayed +1. This determines how the outline is drawn around the text. All strings to be displayed should be of the length = len (including the \∅).
color	The color of the text, this is specified according to the Figure 4.2.
col	The starting column number for each text string.
row	The row number to position the first text string. The top line is in row 1 so begin with row 2 when using a frame.
clear	Determines if the area under the box will be erased before displaying. This is important with overlapping boxes. • TRUE erase area before displaying • FALSE don't erase area before displaying (this can be a time saving selection.)
choice	Determines if the interface will wait for user input before returning to the application. • TRUE wait for user input • FALSE do not wait for user input. Return to the application immediately after displaying the list. See the function **cSToggle()** for an example of this.
maxRow	Defines the number of rows that will be shown within the box and locates the the bottom border. Refer to the notes following these descriptions for information on how this works with *lines*.
frame	A box can be bounded by a frame, and arrows on any of the sides. This parameter is used to select which portions of the border and which arrows will be displayed.

The following border and arrow selections are available:

FRAME COMMANDS	DESCRIPTION	ARROW COMMANDS	DESCRIPTION
FRM	A complete frame	ARS	All four arrows
TOPF	Top of frame	TAL	Top arrow only
BOTF	Bottom of frame	BAR	Bottom arrow only
RIGHTF	Right side of frame	RAR	Right arrow only
LEFTF	Left side of frame	LAR	Left arrow only
NOF	No frame		

Figure 4.3: Frame and Arrow Commands.

Multiple commands can be combined using +. For example, to display a complete border with arrows on the left and right, enter **FRM + LAR + RAR**.

bcolor — The color of the border, specified according to Figure 4.2.

rev — The color of the selection highlight if it is enabled (see *choice*). It is specified according to Figure 4.2.

lines — The total number of strings contained in the box. (Add two for the title and terminating line.)

Refer to the notes following these descriptions for information on how this works with *maxRow*.

*p — This should always be set to NIL. It is initialized by the call to **fillBoxArea()**, **getFileChoice()** or **getBoxArea()**.

offset — The line number of the selection currently highlighted. This refers to the line number within the box. It is initialized to 0.

Refer to the notes following these descriptions for information on how this works with *setRow*.

setRow — The position (row number on the screen) of the selection currently highlighted. This should be initialized to the same value as row.

Refer to the notes following these descriptions for information on how this works with *offset*.

Notes: Some of the parameters seem to identify the same attributes. This section is meant to clarify those parameters and show how they work together to provide flexibility within the user interface.

Initialization of a List Selector

A list selector differs from other API requests in that it consists of two building blocks. First, the BOXREQ structure itself, and secondly an array of pointers to strings that correspond to the possible selections within the list selector. The scrolling area within the BOXREQ (SCRAREA *p;) is initialized with these strings by using the function **fillBoxArea()**. See the previous chapter for details.

The array of pointers has the following format:

```
byte  *example [] = {" TITLE STRING",
                     " FIRST CHOICE",
                     "SECOND CHOICE",
                           :
                     " END STRING "};
```

When using fillBoxArea(), the parameter lines within the BOXREQ must be initialized to the exact number of strings as declared in the array of pointers.

All strings except the TITLE STRING and END STRING are selectable. If a title and end string are not required in the list selector being defined, both strings can be initialized to an empty string "". This displays only the selectable strings.

NOTE: All of the strings initialized within the array of pointers must be of the same length.

If the selections are to be marked using the function **cSToggle()**, the first character in each selectable string should be a blank.

Note: The functions **getBoxArea()** and **addNewLine()** can be used to build a list selector by adding one line at a time to the end of the list. These functions are described in detail in Chapter 3.

maxRow vs. lines

These parameters together define the size of the box and the number of fields to be displayed within the box. This is most easily seen by an example.

```
   Session
 > Choice 1
   Choice 2
   Choice 3
   Choice 4
   Choice 5
   Choice 6
   Exit 7
```

The Session list selector shown here is made up of seven selectable strings and a title. The end string is an empty string (""). To accomodate this list, *lines* should be set to nine (7 lines plus 2). This determines the total number of selectable strings including the title and terminating line. It does not determine *maxRow*.

Displaying all of the choices may take up more space on the screen than allocated for this box. By setting *maxRow* to row +4, five rows of information are displayed at any time.

The additional information is displayed by scrolling through the choices using the arrow keys.

```
   Session   <- row
 > Choice 1
   Choice 2
   Choice 3
   Choice 4  <- maxRow
```

```
   Choice 3
   Choice 4
   Choice 5
   Choice 6
 > Exit 7
```

By pressing the down arrow three times, the title and first selections are scrolled out of the box and replaced with the additional selections.

If the first string was originally placed in row 2, Choice 3 is now in row 2 and Exit in row 6.

offset vs. setRow

These parameters are used to redisplay a box with the highlight on the most recent selection. This is done by combining a location within the list (*offset*) and the actual location on the screen (*setRow*).

Another example will illustrate.

```
  Session
> Choice 1
  Choice 2
  Choice 3
  Choice 4
```

The box is initially displayed with the first selection highlighted. This is set up as:

my_box.offset = 0
my_box.setRow = row

You can then move down to Choice 3 and select it.

```
  Session
  Choice 1
  Choice 2
> Choice 3
  Choice 4
```

To automatically highlight Choice 3 when the box is redisplayed, set:

my_box.offset = *conf.choice*
my_box.setRow = *conf.row*

These parameters are located within the structure BOXCONF. This is the information returned by the function as defined below.

Refer to the function **csToggle()** for an example of how this is used.

Returned Structure

The return parameter **conf** for a BOX_REQ is of the type BOXCONF. This structure contains the information regarding the selection within the list selector, the keystroke used to exit the list, the current selection position on the screen and a pointer to the selected string.

The structure is defined as follows:

```
typedef struct
    {
    int event;
    int exit;
    int choice;
    byte *str;
    int row;
    }BOXCONF;
```

event	BOX_CONF
exit	The key used to exit the list selector (GO, RTN, ESC, CAN, LEFT or RIGHT.)
choice	The number of the selected parameter where 1 is the first non-title string
*str	A pointer to the actual string chosen.
row	The row number on the screen. This defines where the selection is actually located. This value is useful in combination with **setRow.**

Example

The following example sets up a list of choices (Box1[]) with LIST 1 as the titles and Choice 1 initially highlighted. The outline is magenta and the text is yellow.

Since maxRow - row + 2 < lines (5-2+2<7), the list will scroll.

```
#define  BOX_1  1

byte *Box1[] =
    {"LIST1      ",
    "Choice1   ",
    "Choice2   ",
    "Choice3   ",
    "Choice4   ",
    "Choice5   ",
    ""};                        /*The end string is the empty string*/

BOXREQ  box_1  =
    {BOX_REQ, 0, BOX_1, 11, '3', 26, 2, TRUE,
       TRUE, 5, FRM, '6', '5', 7, NIL, 0, 2};
```

The call to the function is made as follows:

```
fillBoxArea (&box_1, Box1);    /* Initialize box to contain strings*/
    .
    .
    .
userInterface (&box_1, &conf, dsp, box);
```

DSP_REQ

Description

This command is used to display text within a window. The type of window, defined by WINDOW_REQ, determines how the command is used. The window must be initialized using the WINDOW_REQ command before using DSP_REQ. Refer to WINDOW_REQ for a description of the different types of windows.

Note that this command does not redisplay the border around the text.

Keys

None

Parameters

The structure type DSPREQ is used to specify the information to be displayed. It is defined as follows:

DSPREQ This identifies the window to contain the display and a pointer to the text to be displayed.

```
typedef struct
    {
    int event;
    int taskId;
    int window;
    byte *text;
    }DSPREQ;
```

event DSP_REQ

taskId This bit is reserved. It is always set to 0.

window The window identification number, where
0 ≤ n ≤ NUM_OF_WINDOWS

*text If the window is defined as **SCROLLING**, then this is the pointer to the string to be displayed.

If the window is defined as **STATIC**, with the output field in the WINDOW_REQ defined as NIL, then the pointer is to a structure of the type FIELD.

Otherwise, with the window defined as **STATIC** and the output field defined to point at a FIELD_SEQ, this parameter is set to NIL.

Example

The three examples that follow illustrate using the command DSP_REQ.

The first example illustrates a window type defined as **SCROLLING.**

```
#define TEST_WIN 1
WINDOWREQ mywindow = {WINDOW_REQ, TEST_WIN, ..., SCROLLING, ...},
byte  testStr[] = "Test DSP_REQ";
DSPREQ  myDSP = {DSP_REQ, 0, TEST_WIN, NIL};
userInterface (&myWindow, &ch, dsp, box);                    /*byte ch*/
                :
myDsp.txt = testStr;
userInterface (&myDsp, &ch, dsp, box);
```

The second example illustrates a **STATIC** window with the output field defined as NIL.

```
#define TEST_WIN 1
byte  testStr[] = "Test DSP_REQ";
FIELD       errStr   = {10, 1, NIL};
DSPREQ      myDSP    = {DSP_REQ, 0, TEST_WIN, (byte*)&errStr};
WINDOWREQ  myWindow = {WINDOW_REQ, TEST_WIN,...,STATIC,...,NIL};
userInterface (&myWindow, &ch, dsp, box);                    /*byte ch*/
                :
errStr.str = testStr;
userInterface (&myDsp, &ch, dps, box);
```

The third example illustrates a **STATIC** window with the output field set as a pointer to FIELD_SEQ. The DSP_REQ redisplays only the FIELD_DEF entries with the change flag set to true. When the DSP_REQ is completed, all change flags are set to false.

```
#define TEST_WIN 1
FIELD_DEF  e1 = {.....};
FIELD_DEF  e2 = {.....};
FIELD_SEQ  ee = {&e1, &e2, NIL};
DSP_REQ  myDsp = {DSP_REQ, 0, TEST_WIN, NIL};
WINDOWREQ  myWindow = {WINDOW_REQ, ..., STATIC, ..., &ee};
userInterface (&myWindow, &ch, dsp, box);                    /*byte ch*/
                :
i=0;
while (ee.f[i]) ee.f [i++]→changed = TRUE      /* set all changed
                                     flags to TRUE, in other situations
                   you may only want to set the flags selectively */
userInterface (&myDsp, &ch, dsp, box);
```

ERASE_FIELD

Description

This is a request to erase a specific field from a window. This will erase both the description and the associated value.

This command is valid only for a STATIC window request with the output field set to FIELD_SEQ. Refer to the command WINDOW_REQ for details on this type of window.

The structure type FIELD_DEF is defined with the command WINDOW_REQ. The *changed* field within that structure must be set to ERASE_FIELD for this command. Once the command is executed, the *changed* flag is set to false.

Keys

None

Parameters

The structure type ERASEFIELD is used to specify the window to be erased. It is defined as follows:

ERASEFIELD This specifies the type of request and window identification.

```
typedef struct
    {
    int event;
    int taskId;
    int window;
    }ERASEFIELD;
```

event	ERASE_FIELD
taskID	This bit is reserved. It is always set to 0.
window	The window identification number, where $0 \leq n \leq$ NUM_OF_WINDOWS

Example

This example initializes a window with three sets of fields, each a description and a value. The window is identified as window #5. Any of the fields (e0, e1 or ez) which have FIELD_DEF.changed set to ERASE_FIELD, will be erased when an ERASE_FIELD request is initiated.

```
FIELD_DEF e0 =
  {TRUE, 5, 7,"Desc. field1", 5, 25,init. value1};
FIELD_DEF e1 =
  {TRUE, 6, 7,"Desc. field2", 6, 25,init. value2};
FIELD_DEF ez =
  {TRUE, 7, 7,"Desc. field3", 7, 25,init. value3};
FIELD_SEQ ee = { &e0, &e1, &ez, NIL};
WINDOWREQ  dsp_win  =
  {WINDOW_REQ, 0, 5, STATIC, 40,60,'3', 5, 5,
  FALSE, 10, 20, FRM, FALSE, '5', &noTitle, &ee };
```

The FIELD_DEF structure for field *ez* is then changed to indicate that it will be erased and the ERASE_FIELD structure set up.

```
ez.changed = ERASE_FIELD
ERASEFIELD era = {ERASE_FIELD, 0, 5 };
userInterface (&dsp_win, &ch, dsp, box);
    .
    .
    .
userInterface (&era, &ch, dsp, box);
```

ERASEB_REQ

Description

This requests that an entire list box be erased from the screen.

Keys

None

Parameters

The structure required for this request is of the type ERASEREQ. It is defined as follows:

```
typedef struct
    {
    int event;
    int taskId;
    int box;
    }ERASEREQ;
```

event	ERASEB_REQ
taskId	This bit is reserved. It is always set to 0.
box	The box identification number, where $0 \leq n \leq$ NUM_OF_BOXES

Returned Values

None

Example

This example displays and erases a box.

```
#define  MY_BOX 1
BOXREQ    myBox    = {BOX_REQ, 0, MY_BOX, ...};
ERASEREQ  eraMyBox = {ERASEB_REQ, 0, MY_BOX};

userInterface (&myBox, &conf, dsp, box;
        .
        .
        .
userInterface (&eraMyBox, &conf, dsp, box};
```

ERASEW_REQ

Description

This requests that a window be erased from the screen.

Keys

None

Parameters

The structure required for this request is of the type ERASERQ. It is defined as follows:

```
typedef struct
    {
    int event;
    int taskId;
    int box;
    }ERASEREQ;
```

event	ERASEW_REQ
taskId	This bit is reserved. It is always set to 0.
box	The identification number for the window, where $0 \le n \le$ NUM_OF_WINDOWS

Returned Values

None

Example

This example displays and erases a window.

```
#define  MY_WINDOW 1
WINDOWREQ  myWindow   = {WINDOW_REQ, 0, MY_WINDOW, ...};
ERASEREQ  eraMyWindow = {ERASEW_REQ, 0, MY_WINDOW};

userInterface (&myWindow, &conf, dsp, box;
      :
      :
userInterface (&eraMyWindow, &conf, dsp, box};
```

INPUT_REQ

Description

This command is used to display a sequence of fields to be edited, for example, configuration parameters.

The fields can be different types, integers, strings, hex or binary. The min & max parameters of the INPUT_FIELD_TYPE are used to limit the range of values that can be entered for a particular field.

Use the arrow keys, ↑↓→←, to move around between the fields.

The fields making up the sequence are numbered. The arrow keys are set to these numbers in order to control the sequence the fields are selected in.

The following keys can be used during runtime operation to modify the field values.

- CTRL-N Go to the next field
- CTRL-P Go to the previous field
- CTRL-I Insert mode (Default mode is overwrite)
- CTRL-D Delete to end of line
- CTRL-A Go to the beginning of the line
- CTRL-E Go to the end of the line
- RETURN Go to the next field
- Space Bar Toggle between preset values

A unique prompt can be associated with each input field. This text is displayed, in a position specified within the request, each time the cursor is positioned at that field. The prompt text is typically used to tell the user the permitted range for a value.

Messages indicating the current mode, either insert or blank for overwrite, or an error when a value outside of the specified range are displayed. These are located as specified in the INPUT_REQ structure.

This field does not display a frame around the input parameters. To display a frame, initiate a static WINDOW_REQ without fields before calling INPUT_REQ.

The recommended strategy to use this command is to define a structure type corresponding to the input field sequence defined for the screen. Declare two structures of this type. The first will be used as the configuration description for the application and the second to be used as a work area for the INPUT_REQ. If the input fields for the INPUT_REQ definition are set to point at entries in the work structure, the following logic is both efficient and convenient.

```
SCREEN_DESR_TYPE param,paramWork;   /* Structure describing screen */
INPUTREQ  myInput = {.....}               /*initialized parameters*/
paramWork = param;                  /* assign the work structure the values
                                       of the current configuration.  */
userInterface (&myInput, &resultChar, dsp, box);
if (resultChar == GO)               /* Use the parameters only if the user*/
    param = paramWork;              /* decided to save the configuration */
```

Keys

The following keys can be used to exit from or cancel the command:

- CAN
- ESC
- GO

Parameters

There are three types of structures required to initiate an INPUT_REQ. They are defined on the following pages.

This command uses different color definitions than those previously defined (Figure 4.2). These are shown in Figure 4.4. Refer to the examples at the end of the command to see how these are used.

BLACK
RED
GREEN
YELLOW
BLUE
MAGENTA
CYAN
WHITE

Figure 4.4: Color Definitions for INPUT_REQ

INPREQ

This defines the location and color of parameters displayed, the prompt text and other messages.

```
typedef struct
   {
   int      event;
   int      taskId;
   INPUT_FIELD_TYPE *fp;
   byte     fi;
   char     *i_color;
   char     *t_color;
   char     *c_color;
   byte     c_row;
   byte     c_col;
   char     *s_color;
   byte     s_row;
   byte     s_col;
   }INPREQ;
```

event	INPUT_REQ
taskID	This bit is reserved. It is always set to 0.
*fp	A pointer to the INPUT_FIELD_TYPE structure.
fi	An index initializing the offset to the first selectable field. Set to zero if the cursor is to be on the first field when the INPUT_REQ is called.
*i_color	The color of the value field, specified according to Figure 4.4.
*t_color	The color of the description field, specified according to Figure 4.4.
*c_color	The color of the help prompt, specified according to Figure 4.4.
c_row/ c_col	The location of the help prompt. (row and column number)
*s_color	The color of the insert mode and invalid value messages, specified according to Figure 4.4.
s_row/ s_col	The location of the insert mode and invalid value message (row and column number)

INPUT_FIELD_TYPE

This structure defines a field on the screen. This includes the position on the screen, the title and input value, the field type and the allowed range.

To define a sequence of fields, an array of these structures is declared. The last entry of this array is defined as {0, 0, 0, 0 ,0 , ...} or zero for all values.

The arrow keys are set to go from field to field.

```
typedef struct
    {
    byte    row;
    byte    column;
    byte    len;
    byte    *buff;
    byte    type;
    byte    lf_flag;
    byte    arrow_flag;
    byte    c_row;
    byte    c_column;
    byte    *c_text;
    byte    *c_buff;
    byte    up;
    byte    down;
    byte    right;
    byte    left;
    byte    num_chk;
    unsigned int    min;
    unsigned int    max;
    FKEY_FIELD_TYPE *fk_ptr;
    } INPUT_FIELD_TYPE;
```

row/ column
: The screen location for the input field (row and column number)

len
: The maximum number of characters for input on the screen.

- For string input, this is equal to the string length.
- For integer input, where no range checking is configured, it corresponds to the maximum integer value.

*buff
: A pointer to the area where the result will be stored.

type	The input data type, where: 0 string 1 byte or int 2 hex (returns an ascii hex string) 3 binary (returns an ascii binary string)
lf_flag	Set to 1 to insert a line feed before the (\0) field terminator. This allows for easier file retrieval, for example, fgetstring() from the standard C library.
arrow_flag	Set to 1 to display a red arrow pointing to the selection. Otherwise, set to 0. Fields that toggle between values are automatically marked with a red arrow.
c_row/ c_column	The screen location for the field title (row and column number)
*c_text	Pointer to the field description or title text string
*c_buff	Pointer to the help field text string positioned through INPREQ (prompt text).
up	The next field in response to an up arrow
down	The next field in response to a down arrow
right	The next field in response to a right arrow
left	The next field in response to a left arrow
num_chk	Set to 1 to check that the input is within a specified valid range. This uses the next two parameters.
min	A minimum value for the valid range
max	A maximum value for the valid range
*fk_ptr	• A pointer to the FKEY_FIELD_TYPE, if used. These values are a fixed set, defined by this structure array. The red arrow is automatically displayed. With this selection, the user toggles between preset values with the space bar. • Set to NIL if it is not used.

FKEY_FIELD_TYPE

This structure defines the preset acceptable values for a field.

To set a sequence of values, define an array of these structures with the last entry equal to {0xff, "", "", NIL}. This will define the end of the value options.

```
typedef struct
    {
    byte    fkey;
    byte    *disp_text;
    byte    *value;
    byte    *link;
    }FKEY_FIELD_TYPE;
```

fkey — This is a flag indicating either the last entry, or an additional value option.

- ON display the text string
- 0xff end of toggle fields.

*disp_text — The text or description of the field. Note that each entry must be the same length.

This is the text seen when the value is toggled to his choice. This entry can be symbolic or numeric, for example "ON" or "1".

*value — The actual value associated with this entry. This value is specified as a string and converted according to the type of field.

*link — A pointer to a choice within another toggle field to be blocked as a result of this choice.
This field is set to NIL is no fields are to be blocked.

Returned Value

The key used to exit the input screen is returned in the byte sized parameter, ie an unsigned character.

For example, in the example shown on page 4-20, the return value declaration would be:

byte resultChar;

Example

The following example initializes a window with five fields and their associated help messages. Four of the five fields can be edited by entering a number within the range provided. The field **Encode** is changed by pressing the space-bar to toggle between the choices *NRZ* and *NRZI*.

```
FKEY_FIELD_TYPE    encode_fkey[] =
  {
  {ON,"NRZ ","0",NIL}, {ON,"NRZI","1",NIL},
  {0xff,"","",NIL}
  };

INPUT_FIELD_TYPE conf2_fields[] =
  {
      {10,16,3,&irWork.tei,  1,0,0, 10,5, "TEI    :",
      "TEI_COM",            4,1,4,3, 1,0,127,NIL},

      {12,16,2,&irWork.sapi,  1,0,0, 12,5, "SAPI     :",
      "SAPI_COM",           0,2,0,4, 1,0,63, NIL},

      {14,16,1,&irWork.encode, 1,0,0, 18,5, "Encode   :",
      "ENC_COM",            1,3,1,3, 0,0,0,encode_fkey},

      {10,38,3,&irWork.n201,   1,0,0, 10,28,"N201    :",
      "N201_COM",           2,4,0,4, 1,1,512,NIL},

      {12,38,4,&irWork.n200,   1,0,0, 12,28,"N200    :",
      "N200_COM",           3,0,1,0, 1,1,9999,NIL},

      {0,0,0,0,0, 0,0,0,0,0,0,0}
  };

INPREQ  conf2_input =
  { INPUT_REQ, conf2_fields, 0, CYAN, YELLOW, GREEN, 20, 23,
  MAGENTA, 20, 5};
```

REL_REQ

Description

This request is used to de-allocate the memory set aside for a window and releases the associated window number. This should be done when a window will not be used again.

You can then reinitialize the window number with new attributes.

Keys

None

Parameters

One structure is required to initiate a REL_REQ. It is used to identify the window that will be released. It is defined as follows:

```
typedef struct
   {
   int event;
   int taskId;
   int window;
   }RELREQ;
```

event	Defines the type of request, REL_REQ
taskId	This bit is reserved. It is always set to 0.
window	The window identification number, where $0 \leq n \leq$ NUM_OF_WINDOWS

Example

```
RELREQ exmp3   {REL_REQ, 0, WINDOW_NUM};

userInterface (&exmp3, NIL, dsp, box);
```

WINDOW_REQ

Description

The command WINDOW_REQ can be used in two ways. It can be used to initialize a window which will display information or it can display a frame around an input request.

There are three types of windows defined through WINDOW_REQ.

- SCROLLING

  A scrolling window displays information each time a DSP_REQ is made. The window is a fixed size, with the information scrolling either forward, with new information added at the bottom of the list, or backward, with new information added at the top.

- STATIC without field sequences

  A static window displays information at a fixed location within a window. The information does not remain in an allocated memory position, and requires a subsequent DSP_REQ to redisplay. This can be used to draw a frame.

- STATIC with field sequences

  A static window with field sequences displays information from an allocated memory position to a fixed location within a window. A field sequence is made up of several titles and values.

Parameters

The parameters for the WINDOW_REQ are incorporated into four structures. Each of these structures are shown below with a brief description of each of the internal parameters.

WINDOWREQ This identifies the window and sets up the basic attributes that determine how the window will be displayed.

```
typedef struct
    {
    int     event;
    int     tskId;
    int     window;
    int     type;
    int     len;
    int     strLen;
    byte    color;
    int     col;
    int     row;
    int     clear;
    int     minRow;
    int     maxRow;
    int     frame;
    int     back;
    byte    bcolor;
    FIELD   *title;
    FIELD   *output;
    }WINDOWREQ;
```

event — The event defines the type of request, in this case WINDOW_REQ

taskId — This bit is reserved. It is always set to 0.

window — The window identification number, where $0 < n \leq$ NUM_OF_WINDOWS

type — Either **STATIC** or **SCROLLING**, where;

- **STATIC**

  The information is displayed in a fixed location within the window as defined by the DSP_REQ made to this window number. This can be done with or without field sequence, where a field sequence is a list of titles, associated value fields and a flag indicating that the field has been displayed.

  - With a Field Sequence

    The field sequence is located in a storage area containing all of the information displayed within this window type. The contents can be modified before they are displayed. They are displayed using a DSP_REQ or WINDOW_REQ.

› Without a Field Sequence

With this type of window, no storage area is allocated. The location of each string is defined by the DSP_REQ. Once a string is erased, it cannot be redisplayed without a new DSP_REQ. This window type can be used to display a frame.

- **SCROLLING**

Each string of information is displayed either at the top or bottom of the existing information. This is determined by the parameter *back*. Note that all strings will be visible until the window frame is full.

TRUE	Each new line is added to the top of the list, pushing existing information out of the bottom of the window.
FALSE	Each new line is added to the bottom of the list, pushing existing information out of the top of the window.

With the first WINDOW_REQ, an area in memory is allocated, corresponding to the size of the window. For subsequent calls to WINDOW_REQ for this window number, the complete contents of this area is displayed.

The size of the scroll area, or window, is determined by the number of lines in the window description and the length of a displayable string. The length of the displayable string is equal to the parameter *strLen*, the length of the window (*len*) plus the number of characters required by an optional escape sequence.

len	This determines the actual width of the window.
strLen	For STATIC windows, this should be set to 0. For SCROLLING windows, strLen is set to the parameter len plus the number of non-printing characters added to a string within this window. This, for example, allows you to change te color in the middle of a string.
color	The color of the text displayed within a window, specified according to Figure 4.2
col, row	The first row below the window frame and the first column to the left of the left frame.

clear — Determines if the area under the window will be erased before displaying. This is important with overlapping windows. The top window must erase the area before displaying to avoid overlap.

- TRUE — erase area before displaying
- FALSE — don't erase area before displaying (This can save time)

minRow — This parameter applies to scrolling windows only. It specifies the upper boundary of the scrolling area for a forward scrolling window, the lower boundary for a backward scrolling window.

This, combined with *maxRow*, defines the area of the window and screen that will contain scrolling data.

maxRow — The last row above the lower frame edge.

frame — A window can be bounded by a frame, with arrows on any of the sides. This parameter is used to select which portions of the border and which arrows will be displayed.

The following border and arrow selections are available:

FRAME COMMANDS	DESCRIPTION	ARROW COMMANDS	DESCRIPTION
FRM	A complete frame	ARS	All four arrows
TOPF	Top of frame	TAL	Top arrow only
BOTF	Bottom of frame	BAR	Bottom arrow only
RIGHTF	Right side of frame	RAR	Right arrow only
LEFTF	Left side of frame	LAR	Left arrow only
NOF	No frame		

Figure 4.5: Frame and Arrow Commands.

Multiple commands can be combined using +. For example, to display a complete border with arrows on the left and right, enter **FRM + LAR + RAR**.

back	This defines the direction of scrolling when the window type is SCROLLING. With other types of windows, it has no effect. TRUE scroll backward FALSE scroll forward
bcolor	The color of the outline, specified according to Figure 4.2.
*title	A pointer to the structure FIELD described below which can be used to initialize the title of a window. This title is redisplayed each time the window is redisplayed.
*output	This is used with static windows with field sequences only. It uses a FIELD_SEQ structure containing a sequence of FIELD_DEF structures, to display a window of information. For other window types, this field must be set to NIL.

FIELD This contains the location and text that make up the title of a window.

```
typedef struct
    {
    int  row;
    int  col;
    byte  *str;
    } FIELD;
```

row /col	The location of the text string containing the title. (Row and column information)
*str	A pointer to the text string

FIELD_DEF typedef struct

```
{
int changed;
int rowT;
int colT;
byte *title;
int rowO;
int colO;
byte *output;
} FIELD_DEF;
```

changed — This parameter, set to TRUE, indicates that the information for this field will be redisplayed when an additional WINDOW_REQ or a DSP_REQ is initiated on this window.

Note that this same structure is used to erase a field within the window simply by changing this parameter to ERASE_FIELD.

rowT/colT — The screen position of the title of the field description. (Row and column number.)

*title — A pointer to the text string containing the field title.

rowO/colO — The screen position of the field value. (Row and column number.)

*output — A pointer to the text string containing the value

FIELD_SEQ This contains pointers to the series of FIELD_DEF structures making up a static window. Note that a field sequence must be terminated with NIL.

```
typedef struct
   {
   FIELD_DEF *f[MAX_FIELDS];
   } FIELD_SEQ;
```

*f [MAX_FIELDS] — This contains the sequence of FIELD_DEF strings to be set up.

Examples

The following examples illustrate the use of WINDOW_REQ. The first sets up a static window, the second, a scrolling window.

Example 1

The first example sets up a static window with two field definitions in a field sequence. The outline of the box will be magenta and the text yellow.

```
FIELD conf2Title = {8,13,"EXAMPLE 1"};

FIELD_DEF e0 =
{TRUE, 5, 7,"MESSAGE:", 5, 16,"        "};

FIELD_DEF e1 =
{TRUE, 5, 40,"NAME:", 5, 47,"          "};

FIELD_SEQ ex1 = { &e0, &e1, NIL};

WINDOWREQ  conf2_win  =
{WINDOW_REQ,1, STATIC, 44,60,'3', 3, 8,
TRUE, 4, 20, FRM, FALSE, '5', &conf2Title, &ex1 };
```

Note: The output fields are initialized to blanks. They can be initialized to point to anything.

Example 2

The second example sets up a scrolling window with a title. It receives the text from a DSP_REQ which is not shown.

The initial DSP_REQ displays the text on line 10 (*minRow*). Each successive DSP_REQ displays the test on lines 11, 12, 13 and so on to line 20 (*maxRow*). Once line 20 is reached, further use of the DSP_REQ scrolls the text up and displays the new text on line 20.

The outline of the box is white and the text is green.

```
FIELD Title2 = {8,30,"EXAMPLE 2"};

WINDOWREQ  dsp_win  =
{WINDOW_REQ, 2, SCROLLING, 71,60,'2', 5, 5,
   FALSE, 10, 20, FRM, FALSE, '7', &Title2, NIL };
```

Chapter 5: APPLICATION PROGRAMMING INTERFACE EXAMPLES

Introduction

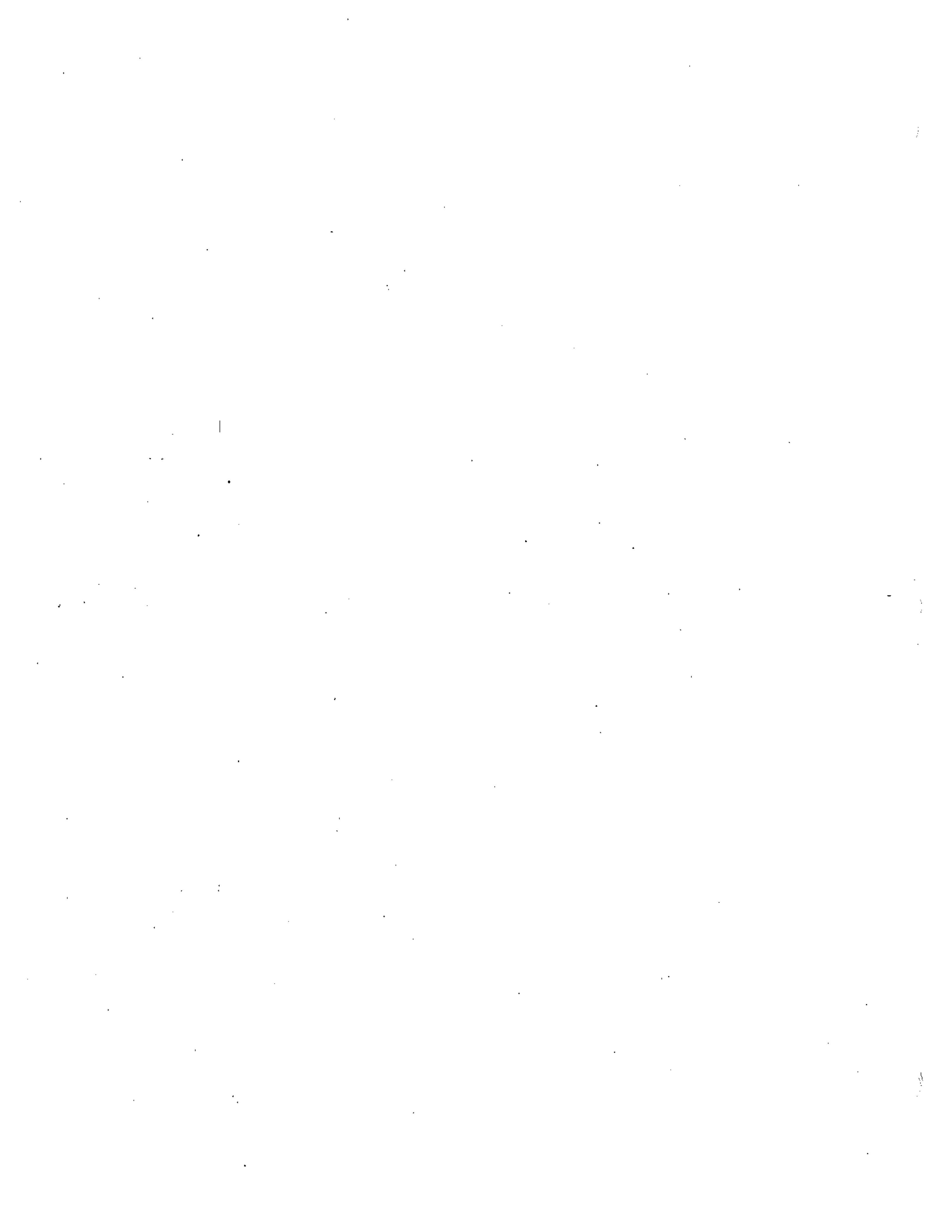
There are three examples provided with the application programming interface, each illustrating a different aspect of the interface. These examples, including

Section 5.1: Example 1, Pull Down Menu Logic

Section 5.2: Example 2, Parameter Input

Section 5.3: Example 3, Listing Files from a Directory

A sample display and brief description of each of the examples is provided at the beginning of the associated section.



EXAMPLE ONE: PULL DOWN MENU LOGIC

Introduction

This appendix contains the files for example 1. This example is made up of three files:

- Select.c
- uitab.c
- uitab.h

This example is composed of 5 boxes or list selectors. These provide the menu strip along the top edge and the four pull down menus.

A window, as shown in Figure 5.1-1, is displayed when a selection if made from one of the boxes.

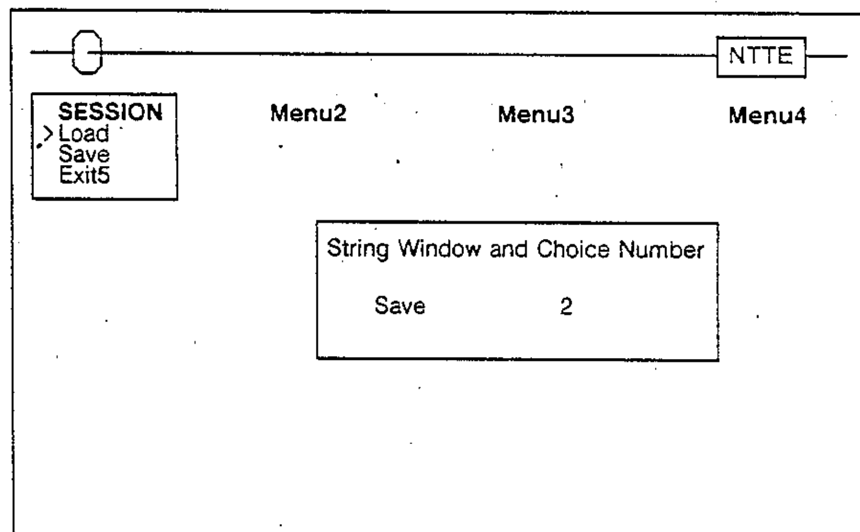


Figure 5.1-1: Example 1

```
/***********************************************************************
 *
 *   File name:      select.c
 *
 *   Description:    Display four list boxes to choose from.  Once a
 *                   selection has been made, show the string and choice
 *                   number.
 *
 *      Version       Date       ID      Comment
 *   ---------------------------------------------------------------
 *      1.0          030790    RHT    Created.
 ***********************************************************************/

#include "mainsym.h"
#include "ui.h"
#include "uitab.h"


main()
{
    BOXCONF  conf;
    int i;

    /*************************/
    /* INITIALIZE MENU SYSTEM */
    /*************************/
    initUI(dsp,box,&error_win,NUM_OF_WINDOWS,NUM_OF_BOXES);

    /**************************************************/
    /* MOVE STRING FIELDS INTO THEIR ASSOCIATED ARRAYS */
    /**************************************************/
    i=0;
    while (fillboxes[i].box) {
      fillBoxArea(fillboxes[i].box,fillboxes[i].list);
      i++;
    }

    /**********************************/
    /* DISPLAY THE OPTIONS HEADERS ONLY */
    /**********************************/
    userInterface(&box_Titles,&conf,dsp,box);
```

```
     /**********************************************************/
     /* DISPLAY EACH LIST BOX UNLESS ESC OR CANCEL IS SELECTED */
     /**********************************************************/
     i=0;
     for(;;) {
        all_box[i].box->event = BOX_REQ;                  /* init. req. type */
        all_box[i].box->choice = TRUE;                    /* set re-display */
        userInterface(all_box[i].box,&conf,dsp,box);      /* display list box */
        all_box[i].box->event = ERASEB_REQ;               /* set erase flag */
        userInterface(all_box[i].box,&conf,dsp,box);      /* erase list box */
        userInterface(all_box[4].box,NIL,dsp,box);        /* display titles */
        switch(conf.exit) {

                /*********************************************/
                /* GO TO THE NEXT LIST BOX, ERASE THE WINDOW */
                /* AND RESET THE HIGHLIGHT BACK TO CHOICE 1  */
                /*********************************************/
          case GO:
          case RIGHT: if(i++ > 2) i = 0;
                      choice_win.event = ERASEW_REQ;
                      userInterface(&choice_win,NIL,dsp,box);
                      all_box[i].box->offset = 0;
                      all_box[i].box->setRow = 2;
                      break;

                /*************************************************/
                /* GO TO THE PREVIOUS LIST BOX, ERASE THE WINDOW */
                /* AND RESET THE HIGHLIGHT BACK TO CHOICE 1      */
                /*************************************************/
          case LEFT: if(i-- == 0) i = 3;
                     choice_win.event = ERASEW_REQ;
                     userInterface(&choice_win,NIL,dsp,box);
                     all_box[i].box->offset = 0;
                     all_box[i].box->setRow = 2;
                     break;

                /*****************************************/
                /* CALL THE APPROPRIATE FUNCTION AND SET */
                /* THE HIGHLIGHT TO THE CHOICE MADE      */
                /*****************************************/
          case RTN: (*all_box[i].func)(conf.choice,conf.str); /* handle_boxN */
                    all_box[i].box->offset = conf.choice; /* highlight the */
                    all_box[i].box->setRow = conf.row;  /* selec. position */
                    break;

                /*****************************/
                /* RESET AND CLEAR THE SCREEN, */
                /* THEN EXIT THE PROGRAM       */
                /*****************************/
          case ESC:
          case CAN:
          default: end_program();
        }
     }
}
```

```
/* EACH OF THE FOLLOWING FUNCTIONS ARE IDENTICLE.  THE ONLY REASON THEY
   ARE PRESENT IS TO SHOW HOW 4 SEPARATE FUNCTIONS CAN BE ORGANIZED AND
   ACCESSED.  EACH LIST BOX AND SELECTION MADE WITHIN THE LIST BOX, IN
   AN ACTUAL APPLICATION, WILL BE HANDLE DIFFERENTLY.
*/

/*
 * Display a window containing the string and choice number selected.
 */
handle_box1(choice,pstr)
int choice;
byte *pstr;
{
    if(choice == 3) end_program();        /* exit program */
    /************************************/
    /* SET THE STRING AND CHOICE NUMBER */
    /************************************/
    Choice_Conf.title = pstr;
    *Choice_Conf.output = choice + 0x30;

    /*********************/
    /* DRAW THE OUTLINE */
    /*********************/
    choice_win.event = WINDOW_REQ;
    userInterface(&choice_win,NIL,dsp,box);

    /***********************************/
    /* INSERT THE TEXT INTO THE WINDOW */
    /***********************************/
    choice_win.event = DSP_REQ;
    Ch_C.f[0]->changed = TRUE;
    userInterface(&choice_win,NIL,dsp,box);
}
```

```
/*
 * Display a window containing the string and choice number selected.
 */
handle_box2(choice,pstr)
int choice;
byte *pstr;
{
    /************************************/
    /* SET THE STRING AND CHOICE NUMBER */
    /************************************/
    Choice_Conf.title = pstr;
    *Choice_Conf.output = choice + 0x30;

    /*******************/
    /* DRAW THE OUTLINE */
    /*******************/
    choice_win.event = WINDOW_REQ;
    userInterface(&choice_win,NIL,dsp,box);

    /**********************************/
    /* INSERT THE TEXT INTO THE WINDOW */
    /**********************************/
    choice_win.event = DSP_REQ;
    Ch_C.f[0]->changed = TRUE;
    userInterface(&choice_win,NIL,dsp,box);
}


/*
 * Display a window containing the string and choice number selected.
 */
handle_box3(choice,pstr)
int choice;
byte *pstr;
{
    /************************************/
    /* SET THE STRING AND CHOICE NUMBER */
    /************************************/
    Choice_Conf.title = pstr;
    *Choice_Conf.output = choice + 0x30;

    /*******************/
    /* DRAW THE OUTLINE */
    /*******************/
    choice_win.event = WINDOW_REQ;
    userInterface(&choice_win,NIL,dsp,box);

    /**********************************/
    /* INSERT THE TEXT INTO THE WINDOW */
    /**********************************/
    choice_win.event = DSP_REQ;
    Ch_C.f[0]->changed = TRUE;
    userInterface(&choice_win,NIL,dsp,box);
}
```

```
/*
 * Display a window containing the string and choice number selected.
 */
handle_box4(choice,pstr)
int choice;
byte *pstr;
{
    /************************************/
    /* SET THE STRING AND CHOICE NUMBER */
    /************************************/
    Choice_Conf.title = pstr;
    *Choice_Conf.output = choice + 0x30;

    /********************/
    /* DRAW THE OUTLINE */
    /********************/
    choice_win.event = WINDOW_REQ;
    userInterface(&choice_win,NIL,dsp,box);

    /**********************************/
    /* INSERT THE TEXT INTO THE WINDOW */
    /**********************************/
    choice_win.event = DSP_REQ;
    Ch_C.f[0]->changed = TRUE;
    userInterface(&choice_win,NIL,dsp,box);
}



end_program() /* exit the program */
{
  printf(RESET);
  printf(CLEAR);
  enablecur(_stdvt);
  exit(0);
}
```

```
/**********************************************************************
 *
 *   File name:     uitab.c
 *
 *   Description:   Initialize all user interface windows and boxes.
 *
 *      Version      Date      ID      Comment
 *   ----------------------------------------------------------------
 *      1.0         030790    RHT   Created.
 *********************************************************************/

#include   "mainsym.h"
#include "ui.h"
#include "uitab.h"

DISPLAY dsp[NUM_OF_WINDOWS];    /* System configuration. */
BOX     box[NUM_OF_BOXES];      /* System configuration. */

/*********************************/
/* DISPLAY ENTIRE BOX OF CHOICES */
/*********************************/
byte *Box1[] =
   {"SESSION   ",
    "Load      ",
    "Save      ",
    "Exit5     ",
    ""};

byte *Box2[] =
   {"MENU2     ",
    "Choice1   ",
    "Choice2   ",
    "Choice3   ",
    "Choice4   ",
    "Choice5   ",
    "Choice6   ",
    "Exit7     ",
    ""};

byte *Box3[] =
   {"MENU3     ",
    "Choice1   ",
    "Choice2   ",
    "Choice3   ",
    "Choice4   ",
    "Exit5     ",
    ""};

byte *Box4[] =
   {"MENU4     ",
    "Choice1   ",
    "Choice2   ",
    "Choice3   ",
    "Choice4   ",
    "Exit5     ",
    ""};
```

```
BOXREQ  box_1  =
    {   BOX_REQ,0,11,'3', 5, 2, TRUE,
        TRUE, 5, FRM, '6', '5', 5, NIL,0,2};

BOXREQ  box_2  =
    {   BOX_REQ,1,11,'3', 26, 2, TRUE,
        FALSE, 7, FRM, '6', '5', 9, NIL,0,2};

BOXREQ  box_3  =
    {   BOX_REQ,2,11,'3', 47, 2, TRUE,
        FALSE, 5, FRM, '6', '5', 7, NIL,0,2};

BOXREQ  box_4  =
    {   BOX_REQ,3,11,'3', 68, 2, TRUE,
        FALSE, 6, FRM, '6', '5', 7, NIL,0,2};

/*****************************/
/* DISPLAY TITLE OF BOX ONLY */
/*****************************/
byte *BoxTitles[] =
   {"SESSION              MENU2                 MENU3                 MENU4",
    ""};

BOXREQ  box_Titles  =
    {   BOX_REQ,4,72,'3', 5, 2, FALSE,
        FALSE, 2, NOF, '6', '5', 2, NIL,0,2};

/************************************/
/* ORGANIZE THE FILLING OF EACH BOX */
/************************************/
FILLBOXES fillboxes[] =
    { { &box_1,      Box1 },
      { &box_2,      Box2 },
      { &box_3,      Box3 },
      { &box_4,      Box4 },
      { &box_Titles, BoxTitles },
      { NIL,         NIL }
   };

/************************************/
/* ORGANIZE THE ADDRESS OF EACH BOX */
/************************************/
ALLBOX all_box[] =
    { { &box_1, handle_box1 },
      { &box_2, handle_box2 },
      { &box_3, handle_box3 },
      { &box_4, handle_box4 },
      { &box_Titles, NIL }
   };
```

```
/***************************************************/
/* NEEDED FOR THE INITIALIZATION OF THE MENU SYSTEM */
/***************************************************/
FIELD errStr = {22,1,NIL};

DSPREQ   errDsp =
  {
  DSP_REQ,
  0,
  ERROR_WIN,
  (byte *) &errStr
  };

FIELD noTitle = {1,1,"" }; /* empty string */

WINDOWREQ  error_win  =
    {  WINDOW_REQ,0,ERROR_WIN, STATIC, 40,40, '2', 1, 20,
       FALSE, 20, 20, NOF, FALSE, '7', &noTitle, NIL };

/*****************************************/
/* USED TO DISPLAY WHICH CHOICE WAS MADE */
/*****************************************/
FIELD Title = {10,26,"String Window and Choice Number" };

byte p_ch[2] = {0x00, 0x00};

FIELD_DEF Choice_Conf =
 {TRUE, 12, 26, NIL, 12, 46, p_ch };

FIELD_SEQ Ch_C = { &Choice_Conf, NIL };

DSPREQ Choice_Str =
   {    DSP_REQ, 0, CHOICE_WIN, "" };

WINDOWREQ  choice_win  =
    {  WINDOW_REQ,0,CHOICE_WIN, STATIC, 35,35, '2', 26, 10,
       TRUE, 12, 15, FRM, FALSE, '7', &Title, &Ch_C };

/*-------------------       end uiTab.c        ----------------*/
```

```
/**********************************************************************
 *
 * File name:      uiTab.h
 *
 * Description:    Definitions used for the user interface.
 *
 *     Version       Date       ID     Comment
 * ---------------------------------------------------------------
 *     1.0         030790    RHT   Created.
 **********************************************************************/

#define  NUM_OF_WINDOWS 30     /* Number of windows */
#define  NUM_OF_BOXES   30     /* Number of structures */

/*
 *    System arrays.
 */

extern DISPLAY dsp[];
extern BOX     box[];

#define  ERROR_WIN          0
#define  CHOICE_WIN         1

/*
 *   External declaration of box structures. (declared in uiTab.c)
 */

typedef struct
    {
    BOXREQ *box;
    int    (*func)(); /* Pointer to a specific function */
    } ALLBOX;

typedef struct
    {
    BOXREQ *box;
    byte   *list;
    } FILLBOXES;
```

```
/*
 *    External declaration of boxes.
 */

extern BOXREQ  box_1;
extern BOXREQ  box_2;
extern BOXREQ  box_3;
extern BOXREQ  box_4;
extern BOXREQ  box_Titles;

extern handle_box1();
extern handle_box2();
extern handle_box3();
extern handle_box4();

extern ALLBOX     all_box[];
extern FILLBOXES  fillboxes[];

extern WINDOWREQ  error_win;
extern WINDOWREQ  choice_win;

extern DSPREQ  Choice_Str;
extern FIELD_DEF Choice_Conf;
extern FIELD_SEQ Ch_C;

extern byte *Box1[];
extern byte *Box2[];
extern byte *Box3[];
extern byte *Box4[];
extern byte *BoxTitles[];

/*--------------------    end  uiTab.h   ----------------------*/
```

EXAMPLE TWO: PARAMETER INPUT

Introduction

This appendix contains the files for example 2. There are three required files.

- ex2.c
- uitab.c
- uitab.h

Note that the uitab.c and uitab.h files are not the same files as used for example 1. These files contain the text to be displayed in the box and window.

This example consists of one box, the list selector shown in Figure 5.2-1, and two windows. Only one of the windows will be displayed at any given time, depending on the selection made.

If *Load* or *Save* are selected, a scrolling window is displayed containing information or which selection was made.

If *Setup* is selected, the Layer 2 configuration window, Figure 5.2-1, is displayed. This window utilizes an INPUT_REQ to allow the user to change the configuration parameters. This also illustrates overlaying windows.

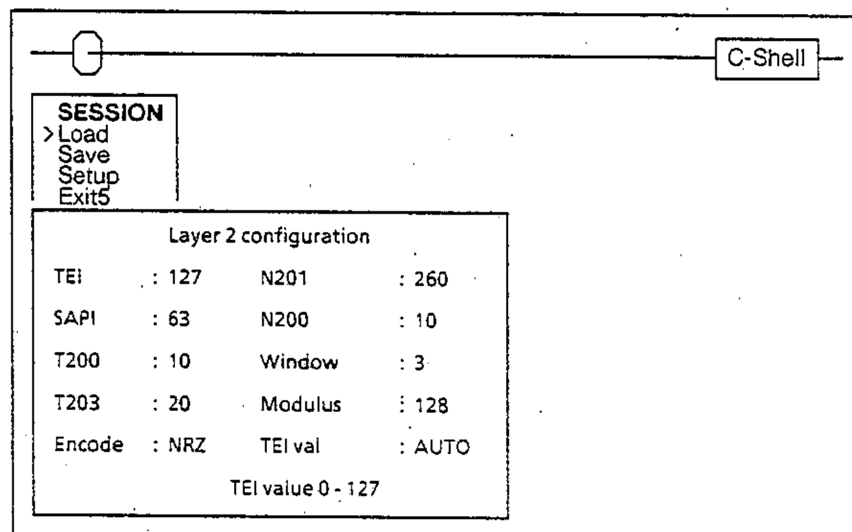


Figure 5.2-1: Example 2

```
/*********************************************************************
 *
 *   File name:      ex2.c
 *
 *   Description:    Depending on the choice made within a list box,
 *                   either display a scrolling window or an input request
 *                   allowing fields to be edited.
 *
 *      Version       Date       ID       Comment
 *   ---------------------------------------------------------------
 *      1.0          030790    RHT    Created.
 ********************************************************************/

#include "mainsym.h"
#include "ui.h"
#include "uitab.h"

main()
{
    BOXCONF  conf;

    /**************************/
    /* INITIALIZE MENU SYSTEM */
    /**************************/
    initUI(dsp,box,&error_win,NUM_OF_WINDOWS,NUM_OF_BOXES);

    /*************************************************/
    /* MOVE STRING FIELDS INTO THEIR ASSOCIATED ARRAY */
    /*************************************************/
    fillBoxArea(&box_1,Box1);

    /**********************************************************/
    /* DISPLAY EACH LIST BOX UNLESS ESC OR CANCEL IS SELECTED */
    /**********************************************************/
    for(;;) {
       box_1.choice = TRUE;                        /* set re-display */
       userInterface(&box_1,&conf,dsp,box);        /* display list box */
       switch(conf.exit) {

         case GO:    /***********************/
         case RIGHT:/* IGNORE THESE ENTRIES */
         case LEFT:  /***********************/
                     break;

                /**************************************/
                /* HANDLE THE CHOICE MADE AND SET     */
                /* THE HIGHLIGHT TO THE SAME CHOICE */
                /**************************************/
         case RTN: handle_choice(conf.choice);
                   box_1.offset = conf.choice;   /* highlight the */
                   box_1.setRow = conf.row;      /* selec. position */
                   break;
```

```
            case ESC: /*********************************/
            case CAN: /* RESET, CLEAR SCREEN THEN EXIT */
            default:  /*********************************/
                     printf(RESET);
                     printf(CLEAR);
                     enablecur(_stdvt);
                     exit(0);
        }
    }
}


handle_choice(choice)
int choice;
  {
    byte ch;

    switch (choice ) {

       case 1: userInterface(&dsp_win,NIL,dsp,box);  /* display window */
               message.text = msg1;
               userInterface(&message,NIL,dsp,box);  /* display message */
               break;

       case 2: userInterface(&dsp_win,NIL,dsp,box);  /* display window */
               message.text = msg2;
               userInterface(&message,NIL,dsp,box);  /* display message */
               break;

       case 3: userInterface(&dspwERA,NIL,dsp,box);  /* erase dsp_win */
               userInterface(&conf2_win,NIL,dsp,box);/* display window */
               irWork = ir;
               userInterface(&conf2_input, &ch);     /* display contents */
               if ( ch == GO ) ir = irWork;          /* save choices on GO */
               conf2_win.event = ERASEW_REQ;
               userInterface(&conf2_win,NIL,dsp,box);/* erase window */
               conf2_win.event = WINDOW_REQ;
               break;

       case 4: printf(RESET);                      /* exit program */
               printf(CLEAR);
               enablecur(_stdvt);
               exit(0);
    }

}
```

```
/**********************************************************************
 *
 *   File name:      uitab.c
 *
 *   Description:   Initialize all user interface windows and boxes.
 *
 *      Version      Date      ID      Comment
 *   ----------------------------------------------------------------
 *      1.0         030790    RHT   Created.
 **********************************************************************/

#include "mainsym.h"
#include "ui.h"
#include "uitab.h"

DISPLAY dsp[NUM_OF_WINDOWS];   /* System configuration. */
BOX     box[NUM_OF_BOXES];     /* System configuration. */

byte msg1[] =  "The text for choice 1 - Load.";
byte msg2[] =  "The text for choice 2 - Save.";

/*********************************/
/* DISPLAY ENTIRE BOX OF CHOICES */
/*********************************/
byte *Box1[] =
   {"SESSION  ",
    "Load     ",
    "Save     ",
    "Setup    ",
    "Exit5    ",
    ""};

BOXREQ  box_1  =
    {   BOX_REQ,0,0,11,'3', 5, 2, TRUE,
        TRUE, 6, FRM, '6', '5', 6, NIL,0,2};

/****************************************************/
/* NEEDED FOR THE INITIALIZATION OF THE MENU SYSTEM */
/****************************************************/
FIELD errStr = {22,1,NIL};

DSPREQ   errDsp =
  {
  DSP_REQ,
  0,
  ERROR_WIN,
  (byte *) &errStr
  };

FIELD noTitle = {20,1,NIL}; /* empty string */
```

```
WINDOWREQ  error_win  =
    {  WINDOW_REQ,0,ERROR_WIN, STATIC, 40,40, '2', 1, 20,
       FALSE, 20, 20, NOF, FALSE, '7', &noTitle, NIL };

/**************************************/
/* DISPLAY MESSAGE FOR SELECTION MADE */
/**************************************/
FIELD Title = {6,45,"\033[36mScrolling Window Title" };

WINDOWREQ  dsp_win  =
    {  WINDOW_REQ, 0, DSP_WIN, SCROLLING, 35,35,'2', 40, 6,
       FALSE, 8, 10, FRM, FALSE, '7', &Title, NIL };

DSPREQ   message =
  {
  DSP_REQ,
  0,
  DSP_WIN,
  NIL};

ERABREQ  dspwERA =
  {
  ERASEW_REQ,
  0,
  DSP_WIN};

/*****************************/
/* WINDOW USED FOR USER INPUT */
/*****************************/
FIELD conf2Title = {8,13,"Layer 2 configuration"};

WINDOWREQ  conf2_win  =
    {  WINDOW_REQ,0,CONF2_WIN, STATIC, 44,60,'3', 3, 8,
       TRUE, 4, 20, FRM, FALSE, '5', &conf2Title, NIL };


/*********************************************************************/
/* ALLOW FOR SPACE-BAR TOGGLE OF FIELD WITH DEVEOLPER DEFINED RESULTS */
/*********************************************************************/
SETUP_TYPE irWork, ir; /* initial and saved values */

FKEY_FIELD_TYPE   encode_fkey[] =
{ {ON,"NRZ ","0",NIL}, {ON,"NRZI","1",NIL}, {0xff,"","",NIL} };

FKEY_FIELD_TYPE   mod_fkey[] =
{ {ON,"8  ","0",NIL}, {ON,"128","1",NIL}, {0xff,"","",NIL} };

FKEY_FIELD_TYPE   tei_fkey[] =
{ {ON,"AUTO ","0",NIL}, {ON,"FIXED","2",NIL}, {0xff,"","",NIL} };
```

```
/***********************************************/
/* WHERE AND HOW TO DISPLAY THE USER I/O WINDOW */
/***********************************************/
INPUT_FIELD_TYPE conf2_fields[] =
{
{10,16,3,&irWork.tei,    1,0,0, 10,5, "TEI       :","TEI_COM ", 9,1,5,5,
        1,0,127,0},
{12,16,2,&irWork.sapi,   1,0,0, 12,5, "SAPI      :","SAPI_COM",0,2,6,6,
        1,0,63, 0},
{14,16,4,&irWork.t200,   1,0,0, 14,5, "T200      :","T200_COM",1,3,7,7,
        0,0,0, 0},
{16,16,4,&irWork.t203,   1,0,0, 16,5, "T203      :","T203_COM",2,4,8,8,
        0,0,0, 0},
{18,16,1,&irWork.encode,1,0,0, 18,5, "Encode    :","ENC_COM ", 3,5,9,9,
        0,0,0,encode_fkey},

{10,38,3,&irWork.n201,    1,0,0, 10,28,"N201    :","N201_COM",4,6,0,0,
        1,1,512,0},
{12,38,4,&irWork.n200,    1,0,0, 12,28,"N200    :","N200_COM",5,7,1,1,
        1,1,9999,0},
{14,38,1,&irWork.window, 1,0,0, 14,28,"Window  :","WIN_COM ", 6,8,2,2,
        1,1,7,  0},
{16,38,1,&irWork.modulus,1,0,0, 16,28,"Modulus :","MOD_COM ", 7,9,3,3,
        0,0,0,mod_fkey},
{18,38,1,&irWork.tei_flag,1,0,0,18,28,"TEI val :","TEIA_COM",8,0,4,4,
        0,0,0,tei_fkey},

{0,0,0,0,0, 0,0,0,0,0,0,0}
};

INPREQ   conf2_input =
   {  INPUT_REQ,0,conf2_fields,0,CYAN,YELLOW,GREEN,20,23,
              MAGENTA,20,5};

/*-------------------      end uiTab.c       ----------------*/
```

```
/**********************************************************************
 *
 *   File name:      uiTab.h
 *
 *   Description:    Definitions used for the user interface.
 *
 *     Version      Date      ID      Comment
 *   ----------------------------------------------------------------
 *     1.0         030790   RHT   Created.
 **********************************************************************/

#define  NUM_OF_WINDOWS  30   /* Number of windows */
#define  NUM_OF_BOXES    30   /* Number of structures */

extern   DISPLAY dsp[];       /* System arrays. */
extern   BOX     box[];       /* System arrays. */

#define  ERROR_WIN          0
#define  DSP_WIN            1
#define  CONF2_WIN          2

/*
 *    External declaration of boxes.
 */

extern BOXREQ     box_1;
extern byte       *Box1[];

extern WINDOWREQ  error_win;
extern WINDOWREQ  dsp_win;
extern WINDOWREQ  conf2_win;

extern DSPREQ     message;

extern ERABREQ    dspwERA;

extern INPREQ     conf2_input;
extern INPUT_FIELD_TYPE conf2_fields[];
```

```
typedef   struct
   {
   int   portnum;
   int   tei;
   int   sapi;
   int   mode;
   int   D_chan;
   int   B1_chan;
   int   B2_chan;
   int   interface;
   int   station;
   int   encode;
   unsigned int   bitrate;
   int   tei_flag;
   int   t200;
   int   t203;
   int   n201;
   int   n200;
   int   window;
   int   modulus;
   int   config;
   int   phys_setup;
   int   bit_inv;
   int   nt_power;
   }   SETUP_TYPE;

/* Structure for setup layer 1 and 2 */

extern SETUP_TYPE ir;
extern SETUP_TYPE irWork;

/* Text message output */
extern byte msg1[];
extern byte msg2[];

/*--------------------   end  uiTab.h   ----------------------*/
```

EXAMPLE THREE:
LISTING FILES FROM A DIRECTORY

Introduction

This appendix contains the files for example 3. There are three required files.

- ex3.c
- uitab.c
- uitab.h

Note that the uitab.c and uitab.h files are not the same files as used for examples 1 and 2. These files contain the text to be displayed in the box and windows.

This example consists of two box, the two list selectors shown in Figure 5.3-1 and two windows. The boxes and windows displayed depend on the selection made.

If *Load* is selected, the display will appear as shown in Figure 5.3-1. The second box or list selector is displayed using *getFileChoice* to display the list of files. Once a file is selected, it is marked with an asterisk, the border of the box changes to include an arrow and a static window showing the choice is displayed.

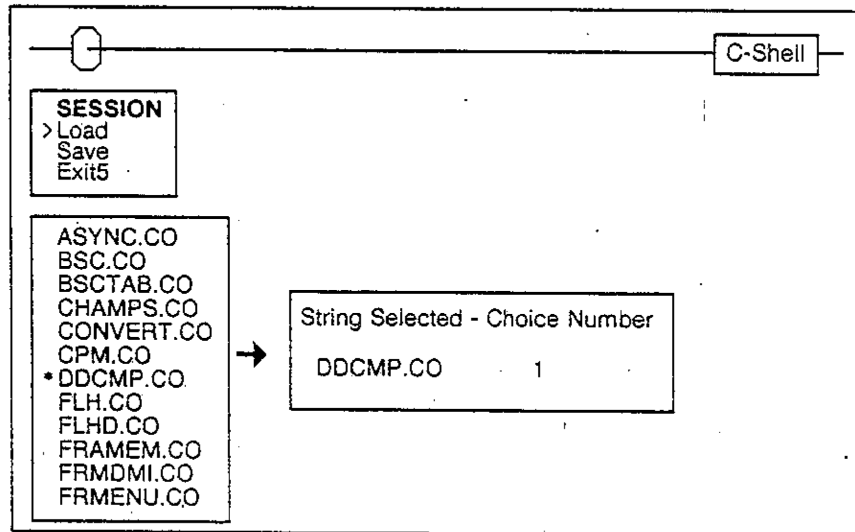


Figure 5.3-1: Example 3

If *Save* is selected, a scrolling window is displayed with the message *The choice made was SAVE.*

```
/**********************************************************************
 *
 *   File name:      ex3.c
 *
 *   Description: Depending on the choice made from a list box, display
 *                either a scrolling window or another box.  This other
 *                box displays a list of directory files.  When a selection
 *                has been made it is marked and a static window displays
 *                the filename and choice number.
 *
 *      Version       Date       ID      Comment
 *   ----------------------------------------------------------------
 *      1.0         030790    RHT    Created.
 **********************************************************************/

#include "mainsym.h"
#include "ui.h"
#include "uitab.h"

main()
{
    BOXCONF  conf;

    /**************************/
    /* INITIALIZE MENU SYSTEM */
    /**************************/
    initUI(dsp,box,&error_win,NUM_OF_WINDOWS,NUM_OF_BOXES);

    /**************************************************/
    /* MOVE STRING FIELDS INTO THEIR ASSOCIATED ARRAY */
    /**************************************************/
    fillBoxArea(&box_1,Box1);

    /*********************************************************/
    /* DISPLAY EACH LIST BOX UNLESS ESC OR CANCEL IS SELECTED */
    /*********************************************************/
    for(;;) {
       box_1.choice = TRUE;                         /* set re-display */
       userInterface(&box_1,&conf,dsp,box);         /* display list box */
       switch(conf.exit) {

         case GO:     /************************/
         case RIGHT:  /* IGNORE THESE ENTRIES */
         case LEFT:   /************************/
                      break;
```

```
                /************************************/
                /* HANDLE THE CHOICE MADE AND SET   */
                /* THE HIGHLIGHT TO THE SAME CHOICE */
                /************************************/
            case RTN: handle_choice(conf.choice);
                box_1.offset = conf.choice;    /* highlight the */
                box_1.setRow = conf.row;       /* selec. position */
                break;

            case ESC: /**************************/
            case CAN: /* RESET, CLEAR THEN EXIT */
            default:  /**************************/
                end_program();
        }
    }
}


/*
 * Determine how to handle the choice made.
 */
handle_choice(choice)
int choice;
{

    switch (choice ) {

       case 1: /* load */
          handle_load();
          break;

       case 2: /* save */
          userInterface(&excbERA,NIL,dsp,box);      /* erase excF_box */
          userInterface(&choicewERA,NIL,dsp,box);   /* erase choice_win */
          userInterface(&dsp_win,NIL,dsp,box);      /* display window */
          userInterface(&message,NIL,dsp,box);      /* display message */
          break;

       case 3: /* exit program */
          end_program();
    }
}
```

```
/*
 * Display a box containing a list of the files within \tekelec\system
 * which end with the extension 'co'.
 */
handle_load()
{
    byte *pfile;
    BOXCONF  conf;

          userInterface(&dspwERA,NIL,dsp,box);         /* erase dsp_win */
          userInterface(&choicewERA,NIL,dsp,box);      /* erase choice_win */
          pfile = getFileChoice(&excF_box,EXC_PATH,"co",
                   excFtitle,excErr,FALSE,NIL,0,&conf);
          if(conf.exit == CAN || conf.exit == ESC) return;
          cSToggle(excF_box.p,conf.choice,0,'*',' ');   /* mark choice */
          excF_box.choice = FALSE;                      /* highlight off */
          excF_box.frame = FRM+RAR;                     /* add arrow */
          excF_box.setRow = conf.row;                   /* keep the list */
          excF_box.offset = conf.choice;                /* displayed */
          userInterface(&excF_box,&conf,dsp,box);       /* display box */
          excF_box.choice = TRUE;                       /* highlight on */
          excF_box.frame = FRM;                         /* remove arrow */
          excF_box.setRow = excF_box.row;               /* reset the list */
          excF_box.offset = 0;                          /* displayed */
          handle_box(conf.choice,pfile);                /* display choice */
}

/*
 * Display a window containing the string and choice number selected.
 */
handle_box(choice,pstr)
int choice;
byte *pstr;
{
    /************************************/
    /* SET THE STRING AND CHOICE NUMBER */
    /************************************/
    Choice_Conf.title = pstr;
    itoa(choice,Choice_Conf.output);

    /********************/
    /* DRAW THE OUTLINE */
    /********************/
    choice_win.event = WINDOW_REQ;
    userInterface(&choice_win,NIL,dsp,box);

    /***********************************/
    /* INSERT THE TEXT INTO THE WINDOW */
    /***********************************/
    choice_win.event = DSP_REQ;
    Ch_C.f[0]->changed = TRUE;
    userInterface(&choice_win,NIL,dsp,box);
}
```

```
/*
 * Convert the integer to an ascii string.
 */
itoa(n, s)
int n;
byte s[4];
{
   int i=0,j,tmp;

   do {                        /* convert integers to ascii in reverse order */
      s[i++] = n % 10 + '0';
   } while( (n /= 10) > 0);

   s[i] = '\0';

   for(i=0, j=strlen(s)-1; i<j; i++, j--) {    /* fix the order of digits */
      tmp = s[i];
      s[i] = s[j];
      s[j] = tmp;
   }
}


/*
 * Reset attributes, turn cursor on and then exit to the C-shell.
 */
end_program()
{
   printf(RESET);
   printf(CLEAR);
   enablecur(_stdvt);
   exit(0);
}
```

```
/***********************************************************************
 *
 *   File name:     uitab.c
 *
 *   Description:   Initialize all user interface windows and boxes.
 *
 *      Version      Date      ID      Comment
 *   ----------------------------------------------------------------
 *      1.0         030790   RHT   Created.
 ***********************************************************************/

#include "mainsym.h"
#include "ui.h"
#include "uitab.h"

DISPLAY dsp[NUM_OF_WINDOWS];  /* System configuration. */
BOX     box[NUM_OF_BOXES];    /* System configuration. */

/**********************************/
/* DISPLAY ENTIRE BOX OF CHOICES */
/**********************************/
byte *Box1[] =
   {"SESSION  ",
    "Load     ",
    "Save     ",
    "Exit5    ",
    ""};

BOXREQ  box_1  =
    {   BOX_REQ,0,SESSION_BOX,11,'3', 5, 2, TRUE,
        TRUE, 5, FRM, '6', '5', 5, NIL,0,2};

/**********************************/
/* DISPLAY CHOICES OF FILE NAMES */
/**********************************/
BOXREQ  excF_box  =
    {   BOX_REQ, 0, FILE_BOX, 17, '2', 5,  8, TRUE,
        TRUE, 19, FRM, '4', '5', 100, NIL,1,9};

byte excFtitle[16] =  "DIRECTORY FILES";
byte excErr[41] =  "NO '*.co' FILES, PRESS CANCEL TO CONTINUE";
```

```
/***************************************************/
/* NEEDED FOR THE INITIALIZATION OF THE MENU SYSTEM */
/***************************************************/
FIELD errStr = {22,1,NIL};

DSPREQ   errDsp =
  {
  DSP_REQ,
  0,
  ERROR_WIN,
  (byte *) &errStr
  };


FIELD noTitle = {20,1,NIL}; /* empty string */

WINDOWREQ  error_win  =
    {  WINDOW_REQ,0,ERROR_WIN, STATIC, 40,40, '2', 1, 20,
       FALSE, 20, 20, NOF, FALSE, '7', &noTitle, NIL };


/*******************************************/
/* DISPLAY A MESSAGE FOR THE SAVE SELECTION */
/*******************************************/
FIELD Title = {6,45,"\033[36mScrolling Window Title" };

WINDOWREQ  dsp_win  =
    {  WINDOW_REQ, 0, DSP_WIN, SCROLLING, 35,35,'2', 40, 6,
       FALSE, 8, 10, FRM, FALSE, '7', &Title, NIL };

/****************************************/
/* USED TO DISPLAY WHICH CHOICE WAS MADE */
/****************************************/
FIELD Title2 = {10,30,"String Selected - Choice Number" };

byte p_ch[4] = {0x00, 0x00, 0x00, 0x00};

FIELD_DEF Choice_Conf =
 {TRUE, 12, 32, NIL, 12, 52, p_ch };

FIELD_SEQ Ch_C = { &Choice_Conf, NIL };

DSPREQ Choice_Str =
   {  DSP_REQ, 0, CHOICE_WIN, NIL };

WINDOWREQ  choice_win  =
    {  WINDOW_REQ,0,CHOICE_WIN, STATIC, 35,35, '2', 30, 10,
       TRUE, 12, 15, FRM, FALSE, '7', &Title2, &Ch_C };
```

```
/********************************/
/* RE-DISPLAY AND ERASE WINDOWS */
/********************************/
DSPREQ   message =
  {
  DSP_REQ,
  0,
  DSP_WIN,
  "The choice made was SAVE"};

ERABREQ  dspwERA =
  {
  ERASEW_REQ,
  0,
  DSP_WIN};

ERABREQ  excbERA =
  {
  ERASEB_REQ,
  0,
  FILE_BOX};

ERABREQ  choicewERA =
  {
  ERASEW_REQ,
  0,
  CHOICE_WIN};


/*-------------------        end uiTab.c        ----------------*/
```

```
/**********************************************************************
 *
 *   File name:      uiTab.h
 *
 *   Description:    User interface definitions.
 *
 *   Version      Date      ID      Comment
 *   ---------------------------------------------------------------
 *      1.0         030790   RHT   Created.
 **********************************************************************/

#define  NUM_OF_WINDOWS 30   /* Number of windows */
#define  NUM_OF_BOXES   30   /*  Number of structures */

#define EXC_PATH    "\\tekelec\\system\\"

#define  ERROR_WIN          0
#define  DSP_WIN            1
#define  CHOICE_WIN         2

#define  SESSION_BOX        0
#define  FILE_BOX           1

extern BOXREQ     box_1;
extern BOXREQ     excF_box;

extern byte       *Box1[];

extern WINDOWREQ  error_win;
extern WINDOWREQ  dsp_win;
extern WINDOWREQ  choice_win;

extern DSPREQ     message;
extern DSPREQ     Choice_Str;

extern FIELD_DEF  Choice_Conf;
extern FIELD_SEQ  Ch_C;

extern ERABREQ    dspwERA;
extern ERABREQ    excbERA;
extern ERABREQ    choicewERA;

extern byte       *getFileChoice();

extern byte       excFtitle[];
extern byte       excErr[];

/*--------------------    end  uiTab.h   ----------------------*/
```

Appendix A:
INCLUDE FILE UI.H

Introduction

This appendix contains the file ui.h. This file must be included in all applications using the user interface as describe in this document. This file contains the structure definitions and global variable definitions.

```
/**********************************************************************
*
*  File name:      ui.h
*
*  Description:    This file contains the definition of the
*                  structures and contants used when interacting
*                  with the windowing user interface.
*
*  Functions:
*
*
*     Version      Date        ID             Comment
*  ---------------------------------------------------------------
*     0.1          121288      TEKELEC/IA     First created
*     1.0          021188      TEKELEC/IA     Switch Release 1.0
*     1.1          021489      TEKELEC/IA     Ntsim development
*     1.2          022189      TEKELEC/EvdM   Input
*     1.3          022289      TEKELEC/EvdM   Input Field Struct
*     1.4          022289      TEKELEC/IA     choice in BOX_REQ
*     1.5          030189      TEKELEC/IA     EDIT_REQ & EDITREQ
*     1.6          030289      TEKELEC/IA     ERASE_FIELD & BINARY_INPUT
*     1.7          030689      TEKELEC/IA     SCROLL_INPUT
*     1.8          030789      TEKELEC/IA     scrollBuf in EDITREQ
*     1.9          030889      TEKELEC/IA     SCROLL_REQ & SCROLLREQ
*     1.10         032989      TEKELEC/IA     offset parameter in list box
*     1.11         032989      TEKELEC/IA     row in BOXCONF
*     2.0          041389      TEKELEC/IA     NT/TE SIMULATOR
*     2.1          112089      TEKELEC/IA     BOXREQ edit sim 3.0
*
**********************************************************************/
```

```
/*
 *    Window types.
 */

#define  STATIC             0
#define  SCROLLING          1


/*
 *    Frame Flags.
 */

#define  NOF               0x0000
#define  LEFTF             0x0001
#define  TOPF              0x0002
#define  RIGHTF            0x0004
#define  BOTF              0x0008
#define  FRM               LEFTF+TOPF+RIGHTF+BOTF

/*
 *    Arrow Flags.
 */

#define  LAR               0x0100
#define  TAR               0x0200
#define  RAR               0x0400
#define  BAR               0x0800
#define  ARS               LAR+TAR+RAR+BAR

/*
 *    Edit modes.
 */

#define     OW_MODE         0
#define     INS_MODE        1


typedef struct
    {
    byte *area;
    byte *next;
    byte *previous;
    }LINE;

typedef struct
    {
     LINE *free;
     LINE *used;
     LINE *last;
     } SCRAREA;
```

```
/*
 *    Window administration structure.
 */

typedef struct
    {
    byte     pos[8];
    byte     color[6];
    byte     text[500];
    int      col;
    int      row;
    int      minRow;
    int      maxRow;
    int      len;
    int      strLen;
    int      frame;
    byte     bcolor;
    int      init;
    SCRAREA  *p;
    int      type;
    int      back;
    int      oldRow;
    }DISPLAY;


/*
 *   Box administration structure.
 */


typedef struct
    {
    byte     pos[8];
    byte     color[6];
    byte     text[500];
    int      col;
    int      row;
    int      minRow;
    int      maxRow;
    int      len;
    int      strLen;
    int      frame;
    byte     bcolor;
    int      init;
    SCRAREA  *p;
    byte     rev[5];
    byte     dummy;
    int      lines;
    int      offset;                        /*1.10*/
    int      setRow;                        /*1.12*/
/*    int        edit; */
    }BOX;
```

```
/*
 *
 *    System arrays.
 *
 */

extern DISPLAY dsp[];
extern BOX     box[];


/*
 *
 *  STATIC WINDOW DEFINITIONS.
 *
 */

#define MAX_FIELDS  30

typedef struct
    {
    int   changed;
    int      rowT;
    int      colT;
    byte   *title;
    int      row0;
    int      col0;
    byte  *output;
    }  FIELD_DEF;

typedef struct
    {
    FIELD_DEF   *f[MAX_FIELDS];
    } FIELD_SEQ;



/*
 *    Event definitions.
 */

#define WINDOW_REQ      10    /* user interface signals  */
#define DSP_REQ         11
#define REL_REQ         12
#define LED_REQ         13
#define BOX_REQ         14
#define BOX_CONF        15
#define ERASEB_REQ      16
#define ERASEW_REQ      17
#define INPUT_REQ       18
#define EDIT_REQ        19
#define STR_EDIT        20
#define ERASE_FIELD     21
#define BINARY_INPUT    22
#define SCROLL_INPUT    23   /*1.7*/
#define SCROLL_REQ      24   /*1.9*/
#define BOX_INPUT       25   /*1.9*/
```

```
/*
 *   Event structure definitions.
 */


typedef struct
    {
    int event;
    int taskId;
    int window;
    byte *text;
    }DSPREQ;


typedef struct
    {
    int  event;
    int  taskId;
    int  window;
    byte *scrollBuf;
    byte *max;
    }EDITREQ;       /*1.8*/


typedef struct
    {
    int  event;
    int  taskId;
    int  window;
    byte *scrollBuf;
    byte *max;
    }SCROLLREQ;     /*1.9*/


typedef struct
    {
    int event;
    int taskId;
    int window;
    }ERASEFIELD;


typedef struct
    {
    int event;
    int taskId;
    int window;
    }RELREQ;


typedef struct
    {
    int event;
    int taskId;
    int box;
    }ERABREQ;
```

```
typedef struct
    {
    int   row;
    int   col;
    byte  *str;
    } FIELD;


typedef struct
    {
    int   event;
    int   taskId;
    int   window;
    int   type;
    int   len;
    int   strLen;
    byte  color;
    int   col;
    int   row;
    int   clear;
    int   minRow;
    int   maxRow;
    int   frame;
    int   back;
    byte  bcolor;
    FIELD *title;
    FIELD *output;
    }WINDOWREQ;


typedef struct
    {
    int       event;
    int       taskId;
    int       box;
    int       len;
    byte      color;
    int       col;
    int       row;
    int       clear;
    int       choice;
    int       maxRow;
    int       frame;
    byte      bcolor;
    byte      rev;
    int       lines;
    SCRAREA   *p;
    int       offset;                  /*1.10*/
    int       setRow;                  /*1.12*/
/*    int       edit; */
    }BOXREQ;
```

```
typedef struct
    {
    int         event;
    int         taskId;
    long int    vtnum;
    byte        *pleds;
    long int    ledword;
    }LEDREQ;


typedef struct
    {
    int    event;
    int    exit;
    int    choice;
    byte   *str;
    int    row;   /*2.11*/
    }BOXCONF;


typedef struct
    {
    byte     fkey;
    byte     *disp_text;
    byte     *value;
    byte     *link;
    }   FKEY_FIELD_TYPE;


typedef struct
    {
    byte     row;                        /* x position for field input   */
    byte     column;                     /* y position for field input   */
    byte     len;                        /* Max len of field             */
    byte     *buff;                      /* Input buffer                 */
    byte     type;                       /* If 1, input = byte or int    */
                                         /* If 0, input = string         */
    byte     lf_flag;                    /* If 1, insert lf before null  */
                                         /* for string input             */
    byte     arrow_flag;                 /* If 1, display arrow          */
    byte     c_row;                      /* x position for field label   */
    byte     c_column;                   /* y position for field label   */
    byte     *c_text;                    /* Field label                  */
    byte     *c_buff;                    /* Field comment                */
    byte     up;                         /* Field no to go to for UP     */
    byte     down;                       /*                   for DOWN   */
    byte     right;                      /*                   for RIGHT  */
    byte     left;                       /*                   for LEFT   */
    byte     num_chk;                    /*  If 1, check range of input  */
    unsigned int     min;                /* Min value of input           */
    unsigned int     max;                /* Max value of input           */
    FKEY_FIELD_TYPE      *fk_ptr;        /* If not 0, ptr to key field   */
    }   INPUT_FIELD_TYPE;
```

```
typedef struct
    {
    int        event;
    int        taskId;
    INPUT_FIELD_TYPE *fp;
    byte       fi;          /* Index of start field   */
    char       *i_color;    /* Color of field content */
    char       *t_color;    /* Color of field label   */
    char       *c_color;    /* Color of commnet       */
    byte       c_row;
    byte       c_col;
    char       *s_color;    /* Color of status info   */
    byte       s_row;
    byte       s_col;
    }INPREQ;


extern DSPREQ      errDsp;

extern FIELD      errStr;



/*--------------------    end  ui.h   ----------------------*/
```

Appendix B: INCLUDE FILE MAINSYM.H

Introduction

This appendix contains the file *mainsym.h*. It must be included in all applications using the user interface as described in this document. It includes the following information:

- General symbols
- Keyboard codes
- Color commands
- Screen attributes and commands
- Port Definitions
- Definitions for UI frame and FSearch
- Record structure

```
/*************************************************************************
 *
 *   File name:      mainSym.h
 *
 *   Description:   This file contains the global symbols.
 *
 *      Version       Date       ID      Comment
 *   ------------------------------------------------------------
 *      1.0          030790    RHT   Created.
 *************************************************************************/

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

```
/*
 *   GENERAL USEFUL SYMBOLS
 */

#define byte unsigned char

extern byte     orgC;
extern byte     termC;
extern byte     *interface, *device;
extern int      stand_type;
extern byte     *mallocRe();
extern byte     *findChar();
extern long int get_dest();
extern int      rxlen;
extern long     _stdvt,getch();
extern char     *malloc();
extern byte     *findElement();
extern int      Semafore;

#define     STOP            0
#define     CONT            1
#define     NIL             0L
#define     TRUE            -1
#define     FALSE           0
#define     YES             0
#define     NO              1
#define     OFF             1
#define     ON              0
#define     MOD(x,y)        ( x % y )
#define     AND             &&
#define     OR              ||
#define     NONE            -1

#define  MAX_NUM             51

#define  ELE_BUF_LEN          257

#define  MSG_BUF_LEN          256

#define  MAX_MSG_LEN        MSG_BUF_LEN

#define  NUM_OF_MESSAGES    21

#define  NUM_OF_CCITT       30
```

```
/*
 *    KEYBOARD CODE DEFINITIONS
 */

#define     F1    0x81
#define     F2    0x82
#define     F3    0x83
#define     F4    0x84
#define     F5    0x85
#define     F6    0x86
#define     F7    0x87
#define     F8    0x88
#define     F9    0x89
#define     F10    0x8a
#define     key0    0x30
#define     key1    0x31
#define     key2    0x32
#define     key3    0x33
#define     key4    0x34
#define     key5    0x35
#define     key6    0x36
#define     key7    0x37
#define     key8    0x38
#define     key9    0x39
#define     UP      0x0b
#define     DOWN    0x0a
#define     RIGHT   0x0c
#define     LEFT    0x08
#define     RTN     0x0d
#define     DELETE  0x7f
#define     ESC     0x1b
#define     CAN     0x18
#define     GO      0x19
#define     CTRL_A    0x01
#define     CTRL_B    0x02
#define     CTRL_C    0x03
#define     CTRL_D    0x04
#define     CTRL_E    0x05
#define     CTRL_I    0x09
#define     CTRL_N    0x0e
#define     CTRL_P    0x10
#define     CTRL_Q    0x11

/*
 *    SCREEN COMMAND MACRO
 */

#define     setScr(x)    printf(x);fflush(stdout);
```

```
/*
 *    COLOR COMMANDS
 */

#define      BLACK       "\033[30m"
#define      RED         "\033[31m"
#define      GREEN       "\033[32m"
#define      YELLOW      "\033[33m"
#define      BLUE        "\033[34m"
#define      MAGENTA     "\033[35m"
#define      CYAN        "\033[36m"
#define      WHITE       "\033[37m"
#define      BBLACK      "\033[40m"
#define      BRED        "\033[41m"
#define      BGREEN      "\033[42m"
#define      BYELLOW     "\033[43m"
#define      BBLUE       "\033[44m"
#define      BMAGENTA    "\033[45m"
#define      BCYAN       "\033[46m"
#define      BWHITE      "\033[47m"


/*
 *   SCREEN ATTRIBUTES
 */

#define      RESET       "\033[0m"
#define      HIGHLIGHT   "\033[1m"
#define      UNDERLINE   "\033[4m"
#define      BLINK       "\033[5m"
#define      REVERSE     "\033[7m"


/*
 *    SCREEN COMMANDS
 */

#define      POS_CUR     "%c[%d;%df",0x1b
#define      DEL_EOL     "%c[0K",0x1b
#define      DEL_EOS     "%c[0J",0x1b
#define      CLEAR       "%c[2J",0x1b


/*
 *    PORT DEFINITIONS
 */

#define      PORTA       0
#define      PORTB       1
```

```
/*
 *    DEFINITION FOR UI FRAME
 */

#define     UI          0x03
#define     MEI         0x0f
#define     IDREQ       0x01
#define     IDASS       0x02
#define     IDDENY      0x03
#define     IDCHK       0x04
#define     IDCHKACK    0x05
#define     IDREL       0x06
#define     IDCONF      0x07


/*
 *    DEFINITION FOR FSearch
 */

/* Directory record data lengths */
#define FN_LEN              8               /* filename length */
#define EX_LEN              3               /* extension length */
#define AT_LEN              1               /* file attributes length */
#define RS_LEN              10              /* reserved bytes length */


/* Directory Record Structure */
struct DREC
        {
        char                dc_fn[FN_LEN];      /* file name */
        char                dc_ex[EX_LEN];      /* file extension */
        char                dc_at;              /* file attributes */
        char                dc_rs[RS_LEN];      /* reserved bytes */
        unsigned short        dc_tim;         /* time file was created */
        unsigned short        dc_dat;         /* date of file creation */
        unsigned short        dc_str;         /* starting cluster number */
        unsigned long         dc_fsz;         /* file size (bytes) */
        };

extern exit_pgm();
```