A
The Selection Service

This appendix describes the compatibility procedures and attributes that support the XView Selection Service. If you are creating a new application that uses selections, refer to Chapter 18, *Selections*. This appendix is provided only for those who need to use the selection service for compatibility reasons. It contains the text on selections that was found in older versions of this book. XView still supports the procedures and attributes described in this appendix. Details on these procedures and attributes may be found in appendices in the *XView Reference Manual*.

The X Window System provides several methods for separate applications to exchange information with one another. One of these methods is the use of the selection service. A *selection* transfers arbitrary information between two clients. An in-depth discussion of the selection mechanism that X provides is discussed in Volume One, *Xlib Programming Manual*. This chapter addresses XView's programmatic interface to the selection service provided by the X server.

While XView provides all the functions for applications to set and get selections of various sorts, OPEN LOOK applications must follow the conventions outlined in the *OPEN LOOK GUI Specification Guide*. All XView packages that have interactive text entry support the ability to make selections and to get selections from the server. In OPEN LOOK, you select objects in basically the same way that you select windows or icons—using the SELECT and ADJUST mouse buttons. OPEN LOOK describes three functions that operate on selected objects: CUT, COPY, and PASTE. These are *core functions* that are accessed from the keyboard.* CUT, COPY, and PASTE operations use the *clipboard* to keep track of selected objects. The clipboard temporarily stores items selected via the selection service; it does not store selected windows or icons since these are not considered selectable objects by the selection service.

*The function keys that are bound to these functions vary from keyboard to keyboard depending on the make and model of the computer.

Table A-1 summarizes text selection for the OPEN LOOK GUI.

Table A-1. Selecting Text

Action	Off Selection	On Selection
Click SELECT	Insert point is set at the pointer location.	When SELECT is released, insert point is set at pointer location and selection is cleared.
Drag SELECT	Text is highlighted as pointer is dragged (wipe-through selection).	Text move pointer is displayed. When you release SELECT, text is moved to pointer location if that location is outside the highlighted area.
Click ADJUST	Extends the highlighting to the pointer location extending either the beginning or the end of the current selection.	Moves the end of the highlighting to the pointer location. Beginning of selection is preserved.
Drag ADJUST	Adjusts an existing selection as the pointer is dragged (wipe-through). Beginning of selection is anchored at insert point.	Adjusts an existing selection as pointer is dragged (wipe-through). Beginning of selection is anchored at insert point.

Note that although OPEN LOOK specifies the selection of graphic objects, no XView objects currently support selection of graphical objects. A possible implementation is to have a canvas object, which has graphic objects displayed in it, set selections based on the event sequences outlined in Table A-1. A *draw* application might consider a drawn geometric shape as a graphic object, whereas a *paint* application might consider the pixels in an arbitrary area as the graphical object.

A.1 The XView Selection Model

The XView selection model is based upon the requestor/owner model of peer-to-peer communications. The *owner* has the data representing the value of its selection, and the *requestor* receives it.

In the X environment, *all* data transferred between an owner and a requestor must transfer via the server. An X client cannot assume that another client can open the same files or even communicate directly. Such assumptions might result in an application that does not work in all network configurations or across heterogeneous computer architectures.

X makes provisions for selections and therefore generates certain events such as `PropertyNotify` when a selection is acquired. For XView to implement its selection service, XView tracks all events that might be generated from selections. Because of this, you cannot use any of the selection mechanisms provided by Xlib. If you do, you will generate events that you will not be able to receive and that XView will be confused about.

The XView selection library deals with four discrete *ranks* under the general term *selection*. Those ranks are: primary, secondary, shelf, and caret. Most familiar is the *primary* selection, which is normally indicated on the screen by inverting (*highlighting*) its contents. Selections made while a function key is held down (usually indicated with an underscore under the selection) are considered *secondary* selections.* These selections are used when the primary selection must be left undisturbed. The *shelf* (or *clipboard*) selection is used by the CUT and COPY operations to load the selection, while the PASTE operation retrieves the selection. Finally, the *caret* (the insertion point for interactive text objects) is also treated as a selection even though it has no contents.

When a user interface element, such as a text subwindow, wants to allow the user to make a selection, it must have a selection *client*. Through this client, the text subwindow can acquire a selection rank (primary, secondary, etc) and provide it with data (text). An application can have many such clients, but there is typically one client per XView object. Each selection rank is associated with each client—a separate client need not be created just to utilize other selection ranks. However, only one client can be the holder of a particular selection rank at any one time. If a client wishes to acquire the selection, the current selection *holder* must *yield* the selection to the new requestor. The client becomes the new holder of the selection and might provide any data it chooses.

A.2 Using the Selection Service

It is not necessary to create a selection service client just to query the holder of a selection or to get its contents. Since this is the most common usage, most of this chapter is dedicated to explaining this level of functionality. Creating a selection is typically used internally by packages in the XView library or by applications that wish to create their own objects that need to communicate via the selection service. Doing this is generally very intricate and complicated and is beyond the scope of this manual.

Nevertheless, to understand how to create a selection service client, it is best to learn how to request information from an existing client. Once you understand what to expect from a selection, you can understand how to create a client that provides information from other requests. Therefore, the bulk of the chapter addresses the process of querying for selections already provided by XView packages or other X-based applications.

OPEN LOOK assigns to function keys the special functions COPY, CUT, and PASTE. The function keys generate and correspond to the XView events `ACTION_CUT`, `ACTION_COPY`, and `ACTION_PASTE`. An OPEN LOOK application checks the state of these keys and modifies the selection rank accordingly. Note that it is the responsibility of the XView application (more specifically, each XView package) to set the state of the server's selection rank according to the state of some function keys. Unless you are writing your own XView package (a topic that this book does not address), you need not concern yourself about it.

*Which function key depends on your particular computer. By default the `L6` function key should work for Sun Workstations or the `F6` key for other computers.

Even though XView packages might query for the state of function keys, this does not interfere with normal event processing. All events that your application has registered to receive are not affected by the selection service. Note, however, that while the selection service might react to the state of these function keys, any action you take as a result of these keys might result in a *dual action* and might confuse the user. For example, if your application is coded to change the font of a text subwindow in the event of an `L8-up` event, you will not only get that event, but the selection service also gets it and will PASTE the contents of the shelf selection.

Whenever using the selection service either as a client or to query the selection from another client, application code must include the header file *<xview/seln.h>*. This file includes the files *<xview/sel_svc.h>* and *<xview/sel_attrs.h>*, which provide external declarations of available functions and data types.

A.3 Getting the Current Selection

Determining the current selection involves two steps: determining the holder of the current selection and getting the actual selection data from the holder. To determine who is holding the selection of a specified rank:

```
Seln_holder
selection_inquire(server, rank)
    Xv_Server server;
    Seln_rank rank;
```

The returned `holder` is used in other selection routines that allow you to access selection data for that rank. For example, the call `selection_ask()` asks for the data held by the holder of the selection. The form of the call is:

```
Seln_request *
selection_ask(server, holder, attrs)
    Xv_Server       server;
    Seln_holder    *holder;
    <attribute-value list>  attrs
```

The `holder` in this case is the address of the holder returned from the call to `selection_inquire()`.

The *server* is important to all the selection service routines because it identifies the X server in which the selection is associated. Because X applications can communicate with more than one server, specifying different servers can result in getting different selections.

The `Seln_rank` specifies which selection type you want. Its value corresponds with the four selection ranks outlined in the beginning of the chapter. However, there are six legal values in this enumerated data type:

`SELN_UNKNOWN` This is an error value, not a value you would use as a parameter or a legal or known rank.

`SELN_CARET` The caret selection is used with text subwindows and text panel items. It is usually used to describe where the insertion point is within the corresponding text stream of the object so there is no *data* associated with the caret selection. This selection type is not very widely used.

`SELN_PRIMARY` The primary selection is the most widely used and is usually the default type in most user interfaces and applications.

`SELN_SECONDARY` As noted earlier, the secondary selection is only used if the primary selection must not be removed or lost—or if it must remain clearly visible (visually selected) on the screen.

`SELN_SHELF` The selection buffers are stored in files. Clients of the selection service do not access these files; the selection service just uses them as temporary storage for data. As noted, this is not a good place to hold selections.

`SELN_UNSPECIFIED` When this rank is used, either the primary or the secondary selection is used depending on the state of the appropriate function keys. This is usually passed as the `rank` parameter to `selection_inquire()`.

The Selection Service

`selection_ask()` returns a pointer to a `Seln_request` data structure. The function goes out and asks the server for the selection associated with the rank described in the `holder` parameter. All the information about the selection is held in this data structure. The attribute-value list following the `holder` parameter describes the attributes of the selection you are interested in.

Before we go on, let's show an example program that illustrates what we have covered so far. The simple program in Example A-1 has a panel button that prints the current primary selection to the standard output. The selection can be held by any client on the server.

Example A-1. The simple_seln.c program

```
/*
 * simple_seln.c -- print the primary selection by pressing the panel
 * button.  The selection may have originated from any window or
 * application on the server.
 */
#include <stdio.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/server.h>
#include <xview/seln.h>

Xv_Server server;

main(argc, argv)
char *argv[ ];
{
    Frame       frame;
    Panel       panel;
    void        exit();
    int         print_seln();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
```

Example A-1. The simple_seln.c program (continued)

```
    frame = (Frame) xv_create(NULL, FRAME,
        FRAME_LABEL,            argv[0],
        NULL);
    panel = (Panel)xv_create(frame, PANEL,
        WIN_WIDTH,              WIN_EXTEND_TO_EDGE,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Print Selection",
        PANEL_NOTIFY_PROC,      print_seln,
        NULL);
    window_fit(panel);
    window_fit(frame);

    server = (Xv_Server)xv_get(xv_get(frame, XV_SCREEN), SCREEN_SERVER);

    xv_main_loop(frame);
}

/*
 * Get the selection using selection_ask().  Note that if the
 * selection is bigger than about 2K, the whole selection will
 * not be gotten with one call, thus this method of getting
 * the selection may not be sufficient for all situations.
 */
int
print_seln(item, event)
Panel_item item;
Event *event;
{
    Seln_holder         holder;
    Seln_request       *response;
    char                text[BUFSIZ];

    /* get the holder of the primary selection */
    holder = selection_inquire(server, SELN_PRIMARY);
    response = selection_ask(server, &holder,
        SELN_REQ_CONTENTS_ASCII, NULL,
        NULL);

    strcpy(text, response->data + sizeof (SELN_REQ_CONTENTS_ASCII));
    printf("---selection---\n%s\n---end seln---\n", text);

    return XV_OK;
}
```

`selection_ask()` does not return until it has contacted the server and gotten a response back from it. This implies that if the server does not respond, the application *blocks* until either a time-out occurs or the selection is received. The attribute-value pair that is passed (`SELN_REQ_CONTENTS_ASCII, NULL`) indicates that we are interested in the ASCII contents of the selection. Whether the selection is successful or not, a pointer to a `Seln_request` structure is returned. If there was an error, the `status` field of the structure will indicate so. If it succeeded, then the selection contents will be in the data field of the structure. All this is clarified in the next section.

A.3.1 The Seln_request Structure

The Seln_request data structure returned from selection_ask() contains information about the selection requested. The pointer returned points to static data that is overwritten on each call. Thus, if you need to save any of this data, it should be copied. The Seln_request structure is defined as follows:

```
typedef struct {
    Seln_replier_data   *replier;
    Seln_requester       requester;
    char                *addressee;
    Seln_rank            rank;
    Seln_result          status;
    unsigned             buf_size;
    char                 data[SELN_BUFSIZE];
} Seln_request;
```

If there is no selection or if the selection fails in any way, the status field in the data structure is set to one of the values in the enumerated type Seln_result. If status is set to SELN_FAILED, then the data field should not be examined as it will not contain any reliable values.

On the other hand, if selection_ask() returns successfully, the same attributes that were passed into the function are copied into the data byte array along with the new values (see Figure A-1).

```
Seln_request  *request;

...

request = selection_ask(server, &holder,
    SELN_REQ_FIRST,          NULL,
    SELN_REQ_LAST,           NULL,
    SELN_REQ_CONTENTS_ASCII, NULL,
    NULL);
```

Figure A-1 shows what is returned assuming that the selection contained the string "Now is the time for all . . . ." The data field contains all the attributes passed in to selection_ask(), but the attributes and the values are all aligned to 4-byte boundaries. This includes the *string* returned from the selection. If the selection string is not a multiple of 4, then it is NULL-padded. The NULL-terminating byte of the string is required—if the last character of the string aligns to a 4-byte boundary, the NULL-terminator pushes it into the next 4-byte block and three more NULLs are required to align to the next boundary.

The value of buf_size is the number of bytes in the data array that is used by attribute-value pairs including the text selection and alignment padding. The attributes SELN_REQ_FIRST and SELN_REQ_LAST return the first and last indices into the object in which the selection resides.*

*Currently, the text subwindow is the only XView object that responds to these requests—panel text items do not.

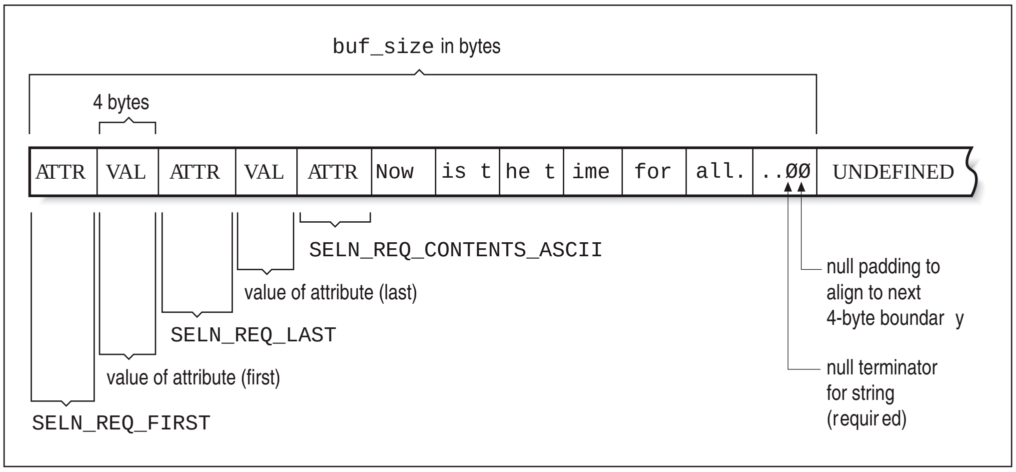

Figure A-1. Byte stream after selection_ask() returns the current text selection

To further demonstrate the use of the selection service, we will examine another program (see Example A-2) that is a little more intricate but that still follows the same principles outlined in the first program. The new program, *text_seln.c*, also helps explain some of the new concepts introduced in this section.

text_seln.c contains a text subwindow in which selections can be made. A panel button that prints the current primary selection is also provided. If the selection is made in the text subwindow provided, information is printed about the relationship between the selected text and the rest of the subwindow. The program makes more extensive use of the `selection_ask()` function.

Example A-2. The text_seln.c program

```
/*
 * text_seln.c -- print the primary selection from the server.  If the
 * selection is in a text subwindow, also print information about
 * the line number(s) the selection spans and the indexes of
 * the bytes within the textsw's buffer.
 */
#include <stdio.h>
#include <xview/xview.h>
#include <xview/textsw.h>
#include <xview/panel.h>
#include <xview/server.h>
#include <xview/seln.h>

Xv_Server       server;
Textsw          textsw;

char *get_selection();

main(argc, argv)
char *argv[ ];
{
    Frame       frame;
```

Example A-2. The text_seln.c program (continued)

```
    Panel       panel;
    void        exit();
    int         print_seln();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    frame = (Frame)xv_create(NULL, FRAME,
        FRAME_LABEL,            argv[0],
        NULL);
    panel = (Panel)xv_create(frame, PANEL,
        WIN_WIDTH,              WIN_EXTEND_TO_EDGE,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Get Selection",
        PANEL_NOTIFY_PROC,      print_seln,
        NULL);
    window_fit(panel);

    textsw = (Textsw)xv_create(frame, TEXTSW,
        WIN_X,                  0,
        WIN_BELOW,              panel,
        WIN_ROWS,               10,
        WIN_COLUMNS,            80,
        TEXTSW_FILE_CONTENTS,   "/etc/passwd",
        NULL);
    window_fit(frame);

    server = (Xv_Server)xv_get(xv_get(frame, XV_SCREEN), SCREEN_SERVER);

    xv_main_loop(frame);
}

int
print_seln()
{
    char *text = get_selection();

    if (text)
        printf("---selection---\n%s\n---end seln---\n", text);

    return XV_OK;
}

/*
 * Get the selection using selection_ask().  Note that if the
 * selection is bigger than about 2K, the whole selection will
 * not be gotten with one call, thus this method of getting the
 * selection may not be sufficient.
 */
char *
get_selection()
{
    long                sel_lin_num, lines_selected;
```

Example A-2. The text_seln.c program (continued)

```
Textsw_index        first, last;
Seln_holder         holder;
Seln_result         result;
int                 len;
Seln_request        *response;
static char         selection_buf[BUFSIZ];
register char       *ptr;

/* get the holder of the primary selection */
holder = selection_inquire(server, SELN_PRIMARY);

/* If the selection occurs in the text subwindow, print lots of
 * info about the selection.
 */
if (seln_holder_same_client(&holder, textsw)) {
    /* ask for information from the selection service */
    response = selection_ask(server, &holder,
        /* get index of the first and last chars in the textsw */
        SELN_REQ_FIRST,             NULL,
        SELN_REQ_LAST,              NULL,
        /* get the actual selection bytes */
        SELN_REQ_CONTENTS_ASCII,    NULL,
        /* Now fool the textsw to think entire lines are selected */
        SELN_REQ_FAKE_LEVEL,        SELN_LEVEL_LINE,
        /* Get the line numbers of beginning and ending of the
         * selection */
        SELN_REQ_FIRST_UNIT,        NULL,
        SELN_REQ_LAST_UNIT,         NULL,
        NULL);
    /* set the ptr to beginning of data -- SELN_REQ_FIRST */
    ptr = response->data;
    /* "first" is data succeeding SELN_REQ_FIRST -- skip attr */
    first = *(Textsw_index *)(ptr += sizeof(SELN_REQ_FIRST));
    ptr += sizeof(Textsw_index); /* skip over value of "first" */
    /* "last" is data succeeding SELN_REQ_LAST -- skip attr */
    last  = *(Textsw_index *)(ptr += sizeof(SELN_REQ_LAST));
    ptr += sizeof(Textsw_index); /* skip over value of "last" */

    /* advance pointer past SELN_REQ_CONTENTS_ASCII */
    ptr += sizeof(SELN_REQ_CONTENTS_ASCII);
    len = strlen(ptr); /* length of string in response */
    (void) strcpy(selection_buf, ptr);
    /*
     * advance pointer past length of string.  If the string length
     * isn't aligned to a 4-byte boundary, add the difference in
     * bytes -- then advance pointer passed "value".
     */
    if (len % 4)
        len = len + (4 - (len % 4));
    ptr += len + sizeof(Seln_attribute); /* skip over "value" */

    /* advance pointer past SELN_REQ_FAKE_LEVEL, SELN_LEVEL_LINE */
    ptr += sizeof(SELN_REQ_FAKE_LEVEL) + sizeof(SELN_LEVEL_LINE);

    sel_lin_num = *(long *)(ptr += sizeof(SELN_REQ_FIRST_UNIT));
    ptr += sizeof(long);
```

Example A-2. The text_seln.c program (continued)

```
        lines_selected = *(long *)(ptr += sizeof(SELN_REQ_LAST_UNIT));
        ptr += sizeof(long);

        /* hack to workaround bug with SELN_REQ_LAST_UNIT always
         * returning -1.  Count the lines explicitly in the selection.
         */
        if (lines_selected < 0) {
            register char *p;
            lines_selected++;
            for (p = selection_buf; *p; p++)
                if (*p == '\n')
                    lines_selected++;
        }
        printf("index in textsw: %d-%d, line number(s) = %d-%d\n",
            first+1, last+1, sel_lin_num+1,
            sel_lin_num + lines_selected + 1);
    } else {
        /* the selection does not lie in our text subwindow */
        response = selection_ask(server, &holder,
            SELN_REQ_CONTENTS_ASCII, NULL,
            NULL);
        if (response->status != SELN_SUCCESS) {
            printf("selection_ask() returns %d\n", response->status);
            return NULL;
        }
        (void) strcpy(selection_buf,
            response->data + sizeof(SELN_REQ_CONTENTS_ASCII));
    }
    return selection_buf;
}
```

There are several points of interest here. In the function `get_selection()`, once the holder of the client has been obtained, it is tested to see if the holder is the text subwindow using `seln_holder_same_client()`. If so, `selection_ask()` is called requesting information specific to the text subwindow. If the text subwindow is not the holder of the selection, then `selection_ask()` is called requesting only the ASCII contents. If there is no selection, then the `status` field of the structure is set to `SELN_FAILED`.

In the case where the holder is the text subwindow, we ask it for the first and last indices of the selection relative to the beginning of the text stream. Note that we might not be able to request this information from any object. For example, if the selection were inside an *xterm*, then this information would not be available and the *xterm*'s selection client would not respond to such requests.

The next attribute (`SELN_REQ_CONTENTS_ASCII`) requests the ASCII contents of the selection actually made. Following that, the attribute-value pair:

```
SELN_REQ_FAKE_LEVEL, SELN_LEVEL_LINE
```

fools the text subwindow into thinking that the user selected an entire line of text (in OPEN LOOK, this would have meant a triple-click with the SELECT mouse button). Had this attribute-value pair been listed *before* the request for ASCII contents, the text returned by the request would have contained the entire line of text on which the selection occurred *regardless of whether the selection began at the beginning of the line*.

The reason we fake the text window into thinking the entire line has been selected is: the attributes SELN_REQ_FIRST_UNIT and SELN_REQ_LAST_UNIT request the line numbers that the selection spans. As the names of the attributes imply, the request is for the first and last *units* selected. Setting the SELN_REQ_FAKE_LEVEL attribute to SELN_LEVEL_LINE indicates that the unit type should be line. Note that we fake the fact that the selection unit is set to line just to get the start and end line numbers of the selection. If we wanted to actually set the level, we would have used SELN_REQ_SET_LEVEL.

After selection_ask() returns a pointer to a Seln_request structure, the values of the requested attributes are found in data, the byte stream. As demonstrated in Example A-2 above, the way to retrieve these values is by moving a pointer along the array:

```
Seln_request *response;
char         *ptr;
long          value;
 ...
response = selection_ask(server, &holder,
    ATTR1,      NULL,
    ATTR2,      NULL,
    ...
    NULL);
 ...
/* set the ptr to beginning of data response -- first attribute */
ptr = response->data;
/* value is data succeeding first attribute -- skip over attr */
value = *(long *)(ptr += sizeof(Seln_attribute));
ptr += sizeof(long); /* skip over the size of the type of value */
```

There is no need to test the attributes as you scan data; they are the same attributes that you used in selection_ask() and they remain in the same order. The values you get back are almost always long.* The exception to this is the text string returned when SELN_REQ_CONTENTS_ASCII is specified. However, the text string is padded to a 4-byte boundary to make sure that the alignment is correct. *text_seln.c* demonstrates how this is done.

A.4 Using selection_query()

One problem with using selection_ask() is handling large selections. In this context, “large” means a text string that is long enough so that it, along with all its attributes and values, does not fit in the data byte-stream. Of course, the fewer attributes that are requested, the more text is returned from the selection.

In this case, the problem is that there is an upper limit to the number of bytes that can be retrieved from the selection. There is no guarantee that the user is not going to select a large number of bytes from some arbitrary application on the screen. However, there is another way to get the selection, regardless of how large it is, by using selection_query():

*Architectures whose int type does not equal its long type must be sure to compensate for this.

```
Seln_result
selection_query(server, holder, reader, context, attrs)
    Xv_Server     server;
    Seln_holder   *holder;
    Seln_result   (*reader)();
    char          *context;
    <attribute-value list> attrs
```

The primary feature of this routine is that you provide it with a pointer to a function that does the scanning of the `data` array in the `Seln_request` structure, as demonstrated earlier. Your `reader` function is called by `selection_query()`, and it gets the `Seln_request` structure as the sole parameter to your function. Your function takes the form of:

```
Seln_result
reader(request)
    Seln_request *request;
```


Your function should return `SELN_SUCCESS` provided that you encountered no problems with scanning `request->data`. `selection_query()` returns the same `Seln_result` that your `reader` function returns. Your `reader` function is called by `selection_query()` for each *chunk* of data in the selection.* The flowchart in Figure A-2 shows the sequence of operations.

The program in Example A-3 demonstrates the use of `selection_query()`. It is similar to *text_seln.c*, but this new program also provides for selections from one of three selection ranks. The user chooses the selection rank from the panel choice item. When the Get Selection button is pressed, the current selection from that selection rank is displayed.

The point of the program is to demonstrate the flow of control between `selection_query()` and the client-installed `reader` procedure. The text subwindow in the application loads the file */etc/termcap*. When a selection is made, `selection_query()` is called, which in turn calls the reader procedure. You can make an arbitrarily large selection to show how `read_proc` is called many times. Start by initializing the selection and then *scrolling* the window and *extending* the selection by using the ADJUST mouse button on later text. Select the Get Selection panel button and the output is directed to `stdout`. Because the selection size can be large, the output text is truncated to the first 20 characters of the selection.

Example A-3. The long_seln.c program

```
/*
 * long_seln.c shows how to get an arbitrarily large selection by
 * providing a reading procedure to selection_query().  The panel
 * items allow the user to choose between 3 selection ranks.
 */
#include <xview/xview.h>
#include <xview/textsw.h>
#include <xview/panel.h>
#include <xview/seln.h>
```

*A *chunk* is the largest text string that will fit in the `data` field of the `Seln_request` structure.

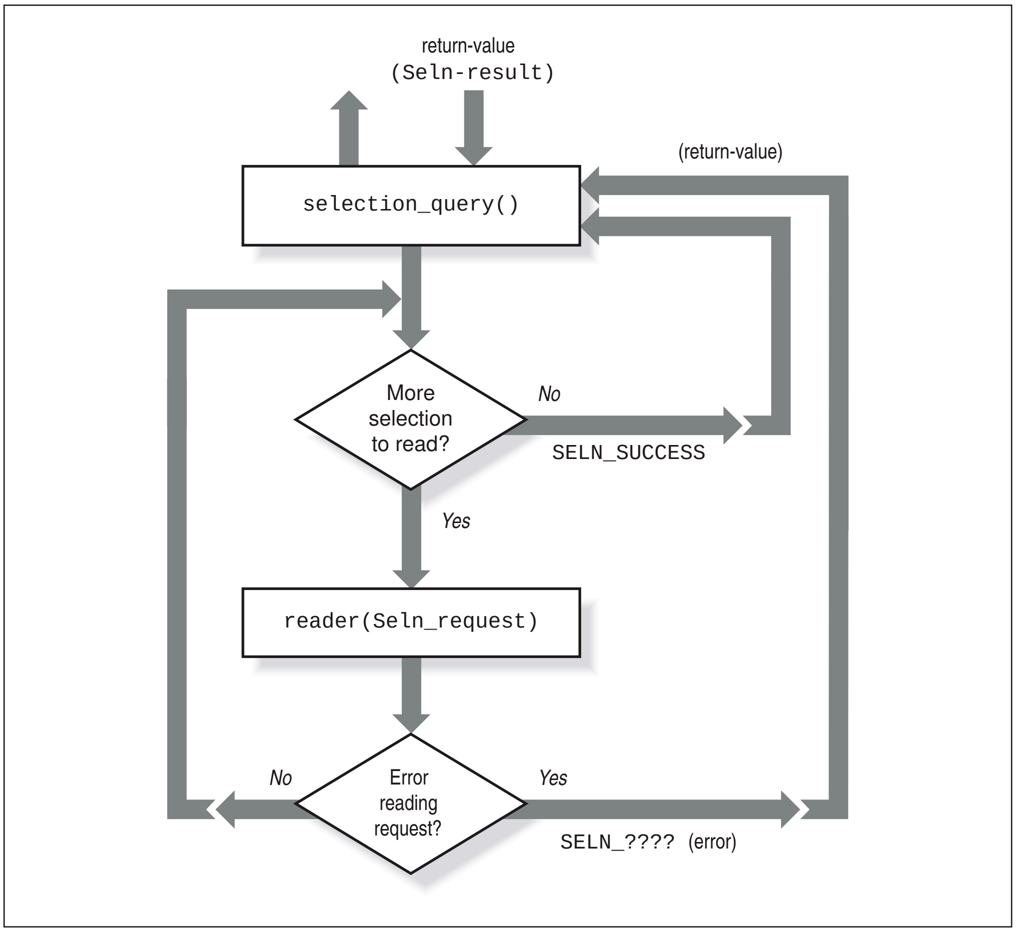

Figure A-2. How selection_query() is used

Example A-3. The long_seln.c program (continued)

```
extern char *malloc();

Seln_rank seln_type = SELN_PRIMARY;

#define FIRST_BUFFER            0
#define NOT_FIRST_BUFFER        !FIRST_BUFFER

char *seln_bufs[3];     /* contents of each of the three selections */

Seln_result read_proc(); /* supplied to selection_query() as reader */

Textsw          textsw;  /* select from this textsw */
Xv_Server       server;
char *get_selection();

void
change_selection(item, value)
```

Example A-3. The long_seln.c program (continued)

```
Panel_item item;
int value;
{
    if (value == 0)
        seln_type = SELN_PRIMARY;
    else if (value == 1)
        seln_type = SELN_SECONDARY;
    else
        seln_type = SELN_SHELF;
}

main(argc, argv)
char *argv[ ];
{
    Frame       frame;
    Panel       panel;
    void        print_seln(), exit();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    frame = (Frame) xv_create(NULL, FRAME,
        FRAME_LABEL, argv[0],
        NULL);

    panel = (Panel)xv_create(frame, PANEL,
        WIN_WIDTH,              WIN_EXTEND_TO_EDGE,
        NULL);

    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Get Selection",
        PANEL_NOTIFY_PROC,      print_seln,
        NULL);
    (void) xv_create(panel, PANEL_CHOICE,
        PANEL_LABEL_STRING,     "Selection Type",
        PANEL_CHOICE_STRINGS,   "Primary", "Secondary", "Shelf", NULL,
        PANEL_NOTIFY_PROC,      change_selection,
        NULL);
    window_fit(panel);

    textsw = (Textsw)xv_create(frame, TEXTSW,
        WIN_X,                  0,
        WIN_BELOW,              panel,
        WIN_ROWS,               10,
        WIN_COLUMNS,            80,
        TEXTSW_FILE_CONTENTS,   "/etc/termcap",
        NULL);
    window_fit(frame);
    server = (Xv_Server)xv_get(xv_get(frame, XV_SCREEN), SCREEN_SERVER);
    xv_main_loop(frame);
}

void
print_seln()
```

Example A-3. The long_seln.c program (continued)

```
{
    char *text = get_selection();

    if (text)
        printf("---seln---\n%.*s [...]\n---end seln---\n", 20, text);
}

/*
 * return the text selected in the current selection rank.  Use
 * selection_query() to guarantee that the entire selection is
 * retrieved.  selection_query() calls our installed routine,
 * read_proc() (see below).
 */
char *
get_selection()
{
    Seln_holder   holder;
    Seln_result   result;
    Seln_request  *response;
    char          context = FIRST_BUFFER;

    holder = selection_inquire(server, seln_type);
    printf("selection type = %s\n",
        seln_type == SELN_PRIMARY? "primary" :
        seln_type == SELN_SECONDARY? "secondary" : "shelf");

    /* result is based on the return value of read_proc() */
    result = selection_query(server, &holder, read_proc, &context,
        SELN_REQ_BYTESIZE,              NULL,
        SELN_REQ_CONTENTS_ASCII,        NULL,
        NULL);
    if (result == SELN_FAILED) {
        puts("couldn't get selection");
        return NULL;
    }

    return seln_bufs[seln_type];
}

/*
 * Called by selection_query for every buffer of information received.
 * Short messages (under about 2000 bytes) will fit into one buffer.
 * For larger messages, read_proc is called for each buffer in the
 * selection.  The context pointer passed to selection_query is
 * modified by read_proc so that we know if this is the first buffer
 * or not.
 */
Seln_result
read_proc(response)
Seln_request *response;
{
    char *reply;  /* pointer to the data in the response received */
    long seln_len; /* total number of bytes in the selection */
    static long seln_have_bytes;
        /* number of bytes of the selection
         * which have been read; cumulative over all calls for
```

Example A-3. The long_seln.c program (continued)

```
         * the same selection (it is reset when the first
         * response of a selection is read)
         */

    printf("read_proc status: %s (%d)\n",
        response->status == SELN_FAILED? "failed" :
        response->status == SELN_SUCCESS? "succeeded" :
        response->status == SELN_CONTINUED? "continued" : "???",
        response->status);
    if (*response->requester.context == FIRST_BUFFER) {
        reply = response->data;

        /* read in the length of the selection -- first attribute.
         * advance "reply" passed attribute to point to actual data.
         */
        reply += sizeof(SELN_REQ_BYTESIZE);
        /* set seln_len to actual data now. (bytes selected) */
        seln_len = *(int *)reply;
        printf("selection size is %ld bytes\n", seln_len);
        /* advance "reply" to next attribute in list */
        reply += sizeof(long);

        /* create a buffer large enough to store entire selection */
        if (seln_bufs[seln_type] != NULL)
            free(seln_bufs[seln_type]);
        if (!(seln_bufs[seln_type] = malloc(seln_len + 1))) {
            puts("out of memory");
            return(SELN_FAILED);
        }
        seln_have_bytes = 0;

        /* move "reply" passed attribute so it points to contents */
        reply += sizeof(SELN_REQ_CONTENTS_ASCII);
        *response->requester.context = NOT_FIRST_BUFFER;
    } else {
        /* this is not the first buffer, so the contents of the
         * response is just more of the selection
         */
        reply = response->data;
    }

    /* copy data from received to the seln buffer allocated above */
    (void) strcpy(&seln_bufs[seln_type][seln_have_bytes], reply);
    seln_have_bytes += strlen(reply);

    return SELN_SUCCESS;
}
```

The Selection Service

A.5 Selection Package Summary

Table A-2 lists the procedures and macros in the Selection Service. Table A-3 lists the Selection Service Attributes.

Table A-2. Selection Service Procedures

Selection Procedures	
selection_acquire()	selection_inform()
selection_ask()	selection_init_request()
selection_clear_functions()	selection_inquire()
selection_create()	selection_inquire_all()
selection_destroy()	selection_query()
selection_done()	selection_report_event()
selection_figure_response()	selection_request()
selection_hold_file()	selection_yield_all()

Table A-3. Selection Service Attributes

Selection Attributes	Advanced Selection Attributes
SELN_REQ_BYTESIZE	SELN_REQ_COMMIT_PENDING_DELETE
SELN_REQ_CONTENTS_ASCII	SELN_REQ_CONTENTS_PIECES
SELN_REQ_DELETE	SELN_REQ_FAKE_LEVEL
SELN_REQ_END_REQUEST	SELN_REQ_FIRST
SELN_REQ_FILE_NAME	SELN_REQ_FIRST_UNIT
SELN_REQ_YIELD	SELN_REQ_LAST
	SELN_REQ_LAST_UNIT
	SELN_REQ_LEVEL
	SELN_REQ_RESTORE
	SELN_REQ_SET_LEVEL