B
Selection Compatibility Procedures and Macros

This section lists the XView Selection procedures and macros in alphabetical order. If you are creating new applications with XView Version 3, you should not use these functions (refer to Chapter 18, *Selections*, in the *XView Programming Manual* for further information).

selection_acquire()

Acquires the selection of a specified rank. This is typically used internally by XView packages. It is not used to *inquire* about the current selection.

```
Seln_rank
selection_acquire(server, client, asked)
        Xv_Server      server;
        Seln_client    client;
        Seln_rank      asked;
```

`client` is the opaque handle returned from `selection_create()`. The client uses this call to become the new holder of the selection of rank `asked`. `asked` should be one of `SELN_CARET`, `SELN_PRIMARY`, `SELN_SECONDARY`, `SELN_SHELF`, or `SELN_UNSPECIFIED`. If successful, the rank actually acquired is returned.

If `asked` is `SELN_UNSPECIFIED`, the client indicates it wants whichever of the primary or secondary selections is appropriate given the current state of the function keys; the one acquired can be determined from the return value.

selection_ask()

`selection_ask()` is a simplified form of `selection_request()` that looks and acts very much like `seln_query()`. The only difference is that it does not use a callback `proc` and so cannot handle replies that require more than a single buffer (e.g., long text selections). If it receives a reply consisting of more than one buffer, it returns the first buffer and discards the rest. The return value is a pointer to a static buffer; in case of error, this will be a valid pointer to a `NULL` buffer:

```
buffer->status = SELN_FAILED
```

The call looks like this:

```
Seln_request *
selection_ask(server, holder, attributes, ..., NULL)
        Xv_Server       server;
```

Selection Compatibility Procedures and Macros

```
        Seln_holder    *holder;
        Attr_union      attributes;
```

selection_clear_functions()

The server is told to forget about any function keys it thinks are down, resetting its state to all-up. If it knows of a current secondary selection, the server will tell its holder to yield.

```
void
selection_clear_functions()
```

selection_create()

The server is initialized for this client. `Client_data` is a 32-bit opaque client value which the server will pass back in callback procedures, as described above. The first two arguments are addresses of client procedures which will be called from the selection functions when client processing is required. These occasions occur when the server sees a function-key transition which may interest this client and when another process wishes to make a request concerning the selection this client holds.

```
Seln_client
selection_create(
        server, function_proc, request_proc, client_data)
                Xv_Server,        server;
                void            (*function_proc) ();
                Seln_result     (*request_proc) ();
                Xv_opaque         client_data;
```

selection_destroy()

A client created by `selection_create` is destroyed—any selection it may hold is released, and various pieces of data associated with the selection mechanism are freed. If this is the last client in this process using the Selection Service, the RPC socket is closed and its notification removed.

```
void
selection_destroy(server, client)
        Xv_Server      server;
        Seln_client    client;
```

selection_done()

Client indicates it is no longer the holder of the selection of the indicated rank. The only cause of failure is absence of the server. It is not necessary for a client to call this procedure when it has been asked by the server to yield a selection.

```
Seln_result
selection_done(server, client, rank)
        Xv_Server      server;
        Seln_client    client;
        Seln_rank      rank;
```

selection_figure_response()

Procedure to determine the correct response according to the standard user interface when `seln_inform()` returns `*buffer` or the client's `function_procs` called with it. The `addressee_rank` field in `Seln_function_buffer` will be modified to indicate the selection which should be affected by this client; `holder` will be set to point to the element of `*buffer` which should be contacted in the ensuing action, and the return value indicates what that action should be. Possible return values are `SELN_DELETE`, `SELN_FIND`, `SELN_IGNORE`, `SELN_REQUEST`, and `SELN_SHELVE`.

```
Seln_response
selection_figure_response(server, buffer, holder)
        Xv_Server              server;
        Seln_function_buffer  *buffer;
        Seln_holder           *holder;
```

selection_hold_file()

The server is requested to act as the holder of the specified rank, whose ASCII contents have been written to the file indicated by `path`. This allows a selection to persist longer than the application which made it can maintain it. Most commonly, this will be done by a process which holds the shelf when it is about to terminate.

```
Seln_result
selection_hold_file(server, rank, path)
        Xv_Server       server;
        Seln_rank       rank;
        char           *path;
```

selection_inform()

Low-level, policy-independent procedure for informing the server that a function key has changed state. Most clients will prefer to use the higher-level procedure `seln_report_event`, which handles much of the standard interpretation required.

```
Seln_function_buffer
selection_inform(server, client, which, down)
        Xv_Server             server;
        Seln_client           client;
        Seln_function         which;
        int                   down;
```

selection_init_request()

Procedure used to initialize a buffer before calling `selection_request`. (It is also called internally by `selection_ask` and `seln_query`.) It takes a pointer to a request buffer, a pointer to a `struct` referring to the selection holder to which the request is to be addressed, and a list of attributes which constitute the request to be sent. The attributes are copied into `buffer->data`, and the corresponding size is stored into `buffer->buf_size`. Both elements of `requester_data` are zeroed; if the caller wants to handle long requests, consumer-proc and context pointers must be entered in these elements after `selection_init_request` returns.

```
void
selection_init_request(
        server, buffer, holder, attributes, ..., NULL)
```

Selection Compatibility Procedures and Macros

```
Xv_Server            server;
Selection_request   *buffer;
Seln_holder         *holder;
char                *attributes;
```

selection_inquire()

Returns a `Seln_holder` structure containing information which enables the holder of the indicated selection to be contacted. If the `rank` argument is `SELN_UNSPECIFIED`, the server will return access information for either the primary or the secondary selection holder, as warranted by the state of the function keys it knows about. The `rank` element in the returned struct will indicate which is being returned.

This procedure may be called without `selection_create()` having been called first.

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

selection_inquire_all()

Returns a `Seln_holders_all` struct from the Selection Service; it consists of a `Seln_holder` struct for each of the four ranks.

```
Seln_holders_all
selection_inquire_all()
```

selection_query()

Transmits a request to the selection holder indicated by the `holder` argument. `consume` and `context` are used to interpret the response and are described below. The remainder of the arguments to `selection_query` constitute an attribute-value list which is the request. (The last argument should be a `0` to terminate the list.) The procedure pointed to by `reader` will be called repeatedly with a pointer to each buffer of the reply. The value of the `context` argument will be available in `buffer->requester_data.context` for each buffer. This item is not used by the selection library; it is provided for the convenience of the client. When the reply has been completely processed (or when the `consume` proc returns something other than `SELN_SUCCESS`), `selection_query` returns.

```
Selection_result
selection_query(server,holder,reader,context,attributes,...,NULL)
                Xv_Server        server;
                Seln_holder     *holder;
                Seln_result    (*reader)();
                char *          *context;
                A-V list         attributes;
```

selection_report_event()

High-level procedure for informing the server of a function key transition which may affect the selection. It incorporates some of the policy of the standard user interface and provides a more convenient interface to `selection_inform`.

`Seln_client_node` is the client handle returned from `selection_create`; it may be `0` if the client guarantees it will not need to respond to the function transition.

`Event` is a pointer to the struct `inputevent` which reports the transition `seln_report_event`. `selection_report_event` generates a corresponding call to `seln_inform` and, if the returned struct is not `NULL`, passes it to the client's `function_proc` callback procedure.

```
void
selection_report_event(server, client, event)
        Xv_Server              server;
        Seln_client_node      *client;
        Event                 *event;
```

selection_request()

Low-level, policy-independent mechanism for retrieving information about a selection from the server. Most clients will access it only indirectly, through `selection_ask` or `selection_query`.

`selection_request` takes a pointer to a holder (as returned by `seln_inquire`) and a request constructed in `*buffer`. The request is transmitted to the indicated selection holder, and the buffer rewritten with its response. Failures in the RPC mechanism will cause a `SELN_FAILED` return; if the process of the addressed holder is no longer active, the return value will be `SELN_NON_EXIST`. Clients which call `selection_request` directly will find it most convenient to initialize the buffer by a call to `selection_init_request`.

Request attributes which are not recognized by the selection holder will be returned as the value of the attribute `SELN_UNRECOGNIZED`. Responses should be provided in the order requests were encountered.

```
Seln_result
selection_request(server, holder, buffer)
        Xv_Server       server
        Seln_holder    *holder;
        Seln_request   *buffer;
```

selection_yield_all()

Procedure that queries the holders of all selections and, for each which is held by a client in the calling process, sends a yield request to that client and a `Done` to the server. It should be called by applications which are about to exit or to undertake lengthy computations during which they will be unable to respond to requests concerning selections they hold.

```
void
selection_yield_all()
```

Selection Compatibility Procedures and Macros