Magellan Protocol and API Specification

John Tsiombikas[*]

June 26, 2007

1 Introduction

In order to use the 6dof input devices by *3Dconnexion* on UNIX systems, such as “space navigator”, “spaceball”, etc. Two components are provided by the vendor: a user space driver (i.e. a daemon), and a small library called “magellan,” to communicate with that daemon, and process input events transmitted by the daemon.

As part of the effort to replace the proprietary UNIX driver provided by the hardware vendor, it was deemed appropriate to also provide an accompanying free software implementation of the magellan library, without the shady licensing terms imposed on the original.

The purpose of this document is to describe the application programming interface of the magellan library, and the communication protocol between the library and the daemon, so that someone who hasn't seen the original code of the magellan library will be able to produce a cleanroom reimplementation to be released as free software.

2 Event Types and Structures

There are three types of events that can be received The actual event types are communicated through X11 atoms, so these values are only visible to the application and of no consequence to the protocol. However there is no reason to introduce incompatibilities with programs that may have foolishly hardcoded those values, so they should be kept the same.

```
enum {
    MagellanInputMotionEvent = 1,
```

[*]e-mail: nuclear@siggraph.org

```
    MagellanInputButtonEvent = 2,
    MagellanInputButtonReleaseEvent = 3
};
```

The following structures are used to store input events after being processed by `MagellanInputEvent` or `MagellanTranslateEvent`.

```
typedef union {
    int data[7];
    int button;
} MagellanIntUnion;

typedef struct {
    int type;
    MagellanIntUnion u;
} MagellanIntEvent;

typedef struct {
    int MagellanType;
    int MagellanButton;
    double MagellanData[6];
    int MagellanPeriod;
} MagellanFloatEvent;
```

Finally, in order to be able to acess the `data` and `MagellanData` values, in a meaningful symbolic manner, the following enumeration is also required:

```
enum {
    MagellanX,  /* X translation */
    MagellanY,  /* Y translation */
    MagellanZ,  /* Z translation */
    MagellanA,  /* rotation around X */
    MagellanB,  /* rotation around Y */
    MagellanC   /* rotation around Z */
};
```

3 Functions

3.1 Overview of Prototypes

The following functions must be visible to the user program:

```
int MagellanInit(Display *dpy, Window win);
int MagellanClose(Display *dpy);
int MagellanSetWindow(Display *dpy, Window win);
int MagellanApplicationSensitivity(Display *dpy, double sens)
   ;
int MagellanInputEvent(Display *dpy, XEvent *event,
   MagellanIntEvent *mag_event);
```

```
int MagellanTranslateEvent(Display *dpy, XEvent *event,
    MagellanFloatEvent *mag_event, double tscale, double
    rscale);
int MagellanRemoveMotionEvents(Display *dpy);
int MagellanRotationMatrix(double mat[4][4], double c, double
     b, double a);
int MagellanMultiplicationMatrix(double mat_a[4][4], double
    mat_b[4][4], double mat_c[4][4]);
```

3.2 Function Specifications

3.2.1 MagellanInit

Prototype:

```
int MagellanInit(Display *dpy, Window win);
```

This function initializes the library, and registers the user's event-handling window with the daemon. The return value is boolean, non-zero is returned in case of successful initialization, or zero otherwise.

During initialization the values of the following X server atoms must be determined and stored somewhere for reuse by the rest of the functions: `MotionEvent`, `ButtonPressEvent`, `ButtonReleaseEvent`, and `CommandEvent`. If these atoms are not already available on the X server, it means that the daemon is not running and has never done so during this X session, because the daemon interns those atoms upon connection to the X server. In that case of course the function must return zero.

Finally, `MagellanSetWindow` must be called in order to register the user's window with the daemon. `MagellanSetWindow` can only fail if the daemon is not running, in which case this function must return zero.

3.2.2 MagellanClose

Prototype:

```
int MagellanClose(Display *dpy);
```

In the original SDK, which probably assumes the daemon can only handle one client at a time, this function sets the client window to `InputFocus`. If we don't set a valid window before exit, possibly the proprietary daemon will exit with a `BadWindow` error next time it tries to send an event. However, the `InputFocus` window at this point makes no sense, and I suggest setting the root window, which can be detected and ignored by our free daemon easily.

3.2.3 MagellanSetWindow

Prototype:

```
int MagellanSetWindow(Display *dpy, Window win);
```

This function registers the application window which will handle magellan events, with the daemon. This is done by sending a `ClientMessage` event to the daemon's X window, which contains a `message_type` equal to the value of the atom `CommandEvent`, the most significant 16 bits of the window id in the first 16 bit data slot, the least significant 16 bits of the window id in the second 16 bit data slot, and the value 27695 (which is the identifier of the "set application window" command) in the third 16 bit data slot.

In order to retrieve the daemon's X window identifier, the root window must be examined for a `CommandEvent` property (see registered atoms), which contains the daemon's X window id as its value. If that property does not exist, it means that the daemon is not running, and this function must return zero.

Our free daemon removes these properties from the root windows upon exit, so there is no way to retrieve a window id which is reassigned to another X client after the exit of the daemon. However apparently the proprietary daemon doesn't do that, which is why the official SDK version of this function also verifies that the title of the retrieved window is: "Magellan Window" or fails otherwise. It is suggested for the sake of compatibility, that we do likewise.

Also in order to avoid crashing the application with a `BadWindow` error in case the retrieved window id is invalid, a custom X error handler must be installed before calling `XSendEvent`, to catch such errors.

Note that due to the asynchronous nature of the X protocol, the only way to make sure that the aforementioned error will occur *before* we remove the custom error handler, and thus won't go uncaught, is to call `XSync` before removing the handler[1].

3.2.4 MagellanApplicationSensitivity

Prototype:

```
int MagellanApplicationSensitivity(Display *dpy, double sens)
    ;
```

This function sets a non-persistent application sensitivity value, which is used by the daemon to scale the motion events, before they are sent to the

[1]The official SDK fails to call `XSync` as it should. It only calls `XFlush` which is not sufficient.

application. This is done by sending a `ClientMessage` event to the daemon's X window, which contains a `message_type` equal to the value of the atom `CommandEvent`. The sensitivity value is converted to a 32 bit floating point value before, the least significant 16 bit part of which is passed through the first 16 bit data slot of the event, and the most significant 16 bit part of is passed through the second 16 bit data slot. The third 16 bit data slot must contain the value 27696 (which is the identifier of the "application sensitivity" command).

For the details on how to extract the daemon's window id, see the specification of `MagellanSetWindow` above.

3.2.5 MagellanInputEvent

Prototype:

```
int MagellanInputEvent(Display *dpy, XEvent *event,
    MagellanIntEvent *mag_event);
```

This function processes a `ClientMessage` X event sent by the daemon containing one of the three types of possible magellan events, and fills in the `MagellanIntEvent` structure pointed to by the third argument.

This function returns zero for failure, if the X event is not a client message, or if the library isn't initialized. On success the type of magellan event is returned, which can be `MagellanInputMotionEvent`, `MagellanInputButtonPressEvent`, or `MagellanInputButtonReleaseEvent` (see event definitions at the beginning of this document).

The event type can be determined by the `message_type` field of the `ClientMessage`, which contains any of the following atoms: `MotionEvent`, `ButtonPressEvent`, or `ButtonReleaseEvent`.

For motion events, the `type` of the `MagellanIntEvent` structure must be set to `MagellanInputMotionEvent`. The six motion values must be extracted from the positions: 2, 3, 4, 5, 6, and 7 of the array of 16 bit data in the X `ClientMessage` event, and put into the first six positions of the `MagellanIntEvent` `u.data` array. Finally the ninth 16 bit slot of the X event data array contains the period that has elapsed from the previous motion event. This value must be multiplied by 1000, divided by 60, and placed into `u.data[6]`

For button press and button release events, the `type` of the `MagellanIntEvent` structure must be set to `MagellanInputButtonPressEvent` or `MagellanInputButtonReleaseEvent` accordingly. Then the button identifier must be extracted from the third 16 bit data slot of the X event and placed in `u.button` of the `MagellanIntEvent` structure.

3.2.6 MagellanTranslateEvent

Prototype:

```
int MagellanTranslateEvent(Display *dpy, XEvent *event,
    MagellanFloatEvent *mag_event, double tscale, double
    rscale);
```

Ok this is really strange ... This function does exactly what `MagellanInputEvent` does, but instead of filling in a `MagellanIntEvent`, it uses a `MagellanFloatEvent`, which has the following variations:

- The type field is called `MagellanType` instead of just `type`.
- The `u.data` array is replaced by the `MagellanData` array.
- The first three motion values (translation) are multiplied by `tscale`, and the next three (rotation) by `rscale`.
- The period now goes into `MagellanPeriod` instead of `u.data[6]`, and is not multipled or divided by anything.
- Button number goes into `MagellanButton` instead of `u.button`.

3.2.7 MagellanRemoveMotionEvents

Prototype:

```
int MagellanRemoveMotionEvents(Display *dpy);
```

This function discards all pending motion events from the X event queue. In order to remove all such events without disturbing any other events in the queue, the `XCheckIfEvent` function can be used, with a predicate function that matches motion events. Any `ClientMessage` events with a `message_type` field equal to the value of the `MotionEvent` atom, are considered motion events.

3.2.8 MagellanRotationMatrix

Prototype:

```
int MagellanRotationMatrix(double mat[4][4], double x, double
     y, double z);
```

This is a helper function, that constructs a rotation matrix out of three euler angles. Always returns non-zero.

3.2.9 MagellanMultiplicationMatrix

```
int MagellanMultiplicationMatrix(double mat_a[4][4], double
    mat_b[4][4], double mat_c[4][4]);
```

This function performs matrix multiplication between `mat_b` and `mat_c`, and stores the result in `mat_a`. Always returns non-zero.