Chapter 11
System Exclusive Protocol

K2500 System Exclusive Implementation

The MIDI System Exclusive capabilities of the K2500 allow you to manipulate objects in the K2500's memory from a computer system, another K2500, or a MIDI data recorder. The following is a reference to the SysEx protocol used by the K2500. This information can be used to build a simple object librarian software program. A word of advice—before you begin experimenting with SysEx, make sure you have saved anything of value in RAM to disk.

NOTE: *To support new features and changes in the K2500 line of products, the internal program structure has been changed from that of the K2000. Due to these changes, you cannot transfer a K2000 program to a K2500, or a K2500 program to a K2000 via MIDI system exclusive. The K2500 software will continue to be enhanced, and in the future the K2500 will be capable of accepting K2000 programs over MIDI. As a result of this, computer based K2000 editor/librarians will not currently work with the K2500, unless they have been revised to accommodate the changes.*

Common Format

In the following discussion, the fields of the K2500 System Exclusive Protocol messages are notated as…

field(length)

…where 'field' is the name of the particular information field in the message, and 'length' is either 1, 2, 3, or n, representing the number of sequential MIDI bytes that make up the field. A length of 'n' means that the field is of a variable length that is determined by its contents or sub-fields.

All K2500 SysEx messages have the common format:

sox(1) kid(1) dev-id(1) pid(1) msg-type(1) message(n) eox(1)

'sox' is always F0h, and represents start of System Exclusive.

'kid' must be 07h, and is the Kurzweil Manufacturer ID.

'dev-id' is Device ID. The K2500 will recognize a SysEx message if the 'dev-id' is the same is the SysX ID parameter from the MIDI Receive page (from the top level, press the MIDI mode button and the RECV soft button.) If the K2500's SysX ID parameter is set to 127, it will recognize SysEx messages no matter what the 'dev-id' is.

'pid' is the Product Identifier, and must be 78h (120 decimal), indicating the SysEx message is for the K2500.

'msg-type' is the identifier of one of the K2500 SysEx messages defined below, and 'message' is the variable-length message contents.

'eox' is always F7h, for end of System Exclusive.

Data Formats

K2500 SysEx messages are subdivided into fields that contain data in different formats. The various fields are shown in the Messages section below. Within a message, any fields for values that can be bigger than 7 bits are broken into 7 bit chunks. Thus two MIDI bytes gives 14 bits, three bytes gives 21 bits. The significant bits are right justified in the field. All bytes in a field must be present no matter what the value is. For example, an object type of 132 would be split into two MIDI bytes in a 'type' field as 01 04:

decimal:	132
binary:	10000100
binary encoding for type(2) field:	0000001 0000100
decimal encoding for type(2) field:	1 4

Object name fields are sent as a string of ASCII values in a 'name' field, with one MIDI byte of zero as a string terminator. For example, the name "Glass Kazoo" would be sent as letters:

	G	l	a	s	s	_	K	a	z	o	o	<null>
hex encoding for 'name' field:	47	6C	61	73	73	20	4B	61	7A	6F	6F	00

Data sizes and offsets are sent in the 'size' and 'offs' fields.These values refer to quantities of 8-bit bytes in the K2500's memory, which is packed in the 'data' field.

Binary data in the 'data' field is sent by in one of two formats, according to the value of the 'form' field. If the 'form' field equals zero, the data is transmitted as 4 bits or one "nibble" in every MIDI byte. If the 'form' field equals one, then the data is sent as a compressed bit-stream, with 7 bits per midi byte. The bit-stream format is more efficient for data-transmission, while the nibble format is easier to read (and write software for).

For example, to send the following four K2500 data bytes,

hex:	4F	D8	01	29
decimal:	79	216	1	41
binary:	01001111	11011000	00000001	00101001

eight MIDI bytes are sent in "nibble" format:

hex:	04	0F	0D	08	00	01	02	09
decimal:	4	15	13	8	0	1	2	9
binary:	0000100	0001111	0001101	0001000	0000000	0000001	0000010	0001001

five MIDI bytes are sent in bit-stream format:

hex:	27	76	0	12	48
decimal:	39	118	0	18	72
binary:	0100111	1110110	0000000	0010010	1001000

The bit-stream format can be thought of as taking the binary bits of the K2500 data and, starting from the left, slicing off groups of 7 bits. Note that the trailing bits are set to zero.

After the 'data' field, there is another field, 'xsum'. This is a checksum field which is calculated as the least significant 7-bits of the sum of all of the MIDI bytes that make up the 'data' field.

Messages

This section defines the K2500 System Exclusive message formats. Each message has a message type (that goes in the 'msg-type' field; see Common Format, above), followed by the field definitions of the message.

DUMP = 00h type(2) idno(2) offs(3) size(3) form(1)

…requests the K2500 to send a data dump of an object or portion thereof. 'type' and 'idno' identify the object. 'offs' is the offset from the beginning of the object's data and 'size' describes how many bytes should be dumped starting from the offset. 'form' indicates how the binary data is to transmitted (0=nibblized, 1=bit stream). The response is a LOAD message:

LOAD = 01h type(2) idno(2) offs(3) size(3) form(1) data(n) xsum(1)

…which writes data into the specified object, which must exist. Both load and dump operate on the object data only. The response to a load message will be

DACK = 02h type(2) idno(2) offs(3) size(3)

…meaning "load accepted", or

DNAK = 03h type(2) idno(2) offs(3) size(3) code(1)

…meaning "load not accepted." The 'code' field indicates the cause of the failure, as follows::

code	meaning
1	Object is currently being edited
2	Incorrect checksum
3	ID out of range (invalid)
4	Object not found (no object with that ID exists)
5	RAM is full

To request information about an object, use:

DIR = 04h type(2) idno(2)

The 'type' and 'idno' identify the object. The response is an INFO message:

INFO = 05h type(2) idno(2) size(3) ramf(1) name(n)

This is the response to DIR, NEW, or DEL. If object is not found, 'size' will be zero and 'name' will be null. 'ramf' is 1 if the object is in RAM.

NEW = 06h type(2) idno(2) size(3) mode(1) name(n)

…creates a new object and responds with an INFO message of the created object. The object's data will not be initialized to any default values. If 'idno' is zero, the first available object ID number will be assigned. If 'mode' is 0, the request will fail if the object exists. If 'mode' is 1, and the object exists in ROM, a RAM copy will be made. If 'mode' is 1, and the object exists in RAM, no action is taken.

DEL = 07h type(2) idno(2)

…deletes an existing object and responds with an INFO message for the deleted object. If there is only a RAM copy of the object, the response will indicate that the object doesn't exist anymore. However, if the deletion of a RAM object uncovers a ROM object, the INFO response will refer to the ROM object. A ROM object cannot be deleted.

CHANGE = 08h type(2) idno(2) newid(2) name(n)

…changes the name and/or ID number of an existing object. If 'newid' is zero or 'newid' equals 'idno', the ID number is not changed. If 'newid' is a legal object id number for the object's type , then the existing object will be relocated in the database at the new ID number. This will cause the deletion of any object which was previously assigned to the 'newid'. If the 'name' field is null, the name will not change. Otherwise, the name is changed to the (null-terminated) string in the 'name' field.

WRITE = 09h type(2) idno(2) size(3) mode(1) name(n) form(1) data(n) xsum(1)

…writes an entire object's data directly into the database. It functions like the message sequence DEL followed by NEW followed by a LOAD of one complete object data structure. It first deletes any object already existing at the same type/ID. If no RAM object currently exists there, a new one will be allocated and the data will be written into it. The object name will be set if the 'name' string is non-null. The response to this message will either be a DACK or a DNAK, as with the load message. The 'offs' field of the response will be zero. The K2500 will send a WRITE message whenever an object is dumped from the front-panel (using a "Dump" soft-button), or in response to a READ message.

The 'mode' field is used to determine how the 'idno' field is interpreted.

If 'mode' = 0:

The 'idno' specifies the absolute ID number to write to, which must exist.(must be valid)

If 'idno' equals zero, write to the first available ID number.

If 'mode' = 1

The object is written at the first available ID number after what is specified by 'idno'.

It doesn't matter if 'idno' is a legal ID number. Remember that for certain object types, the 100s through 900s banks allow fewer than 100 objects to be stored (for example, the 100s bank will store preset effects at IDs 100—109 only). In this mode, if 'idno' was 313, the object would be written to ID 400 if available.

READ = 0Ah type(2) idno(2) form(1)

…requests the K2500 to send a WRITE message for the given object. No response will be sent if the object does not exist.

READBANK = 0Bh type(2) bank(1) form(1) ramonly(1)

…requests the K2500 to send a WRITE message for multiple objects within one or all banks.

'type' and 'bank' specify the group of objects to be returned in WRITE messages. The 'type' field specifies a single object type, unless it is zero, in which case objects of all user types will be returned (see object type table below). The 'bank' field specifies a single bank, 0-9, unless it is set to 127, in which case objects from all banks will be returned.

'form' requests the format of the binary data in the WRITE messages. If 'ramonly' is one, only objects in RAM will be returned. If 'ramonly' is zero, both RAM and ROM objects are returned.

The responses, a stream of complete WRITE messages, will come out in order of object type, while objects of a given type are in order by ID number, from lowest to highest. If no objects are found that match the specifications, no WRITE messages will be returned. After the last WRITE message, an ENDOFBANK message (defined below) is sent to indicate the completion of the bank dump.

The K2500 will insert a small delay (50ms) between WRITE messages that it issues in response to a READBANK message.

A bank dump can be sent in its entirety to another K2500, which will add all of the objects contained in the dump to its own object database. IMPORTANT: If the K2500 receives a large bank dump for a bank or banks that already contain objects, errors may result unless the sender waits for the DACK message before sending the next object's WRITE message. One way to avoid transmission errors such as this is to make sure that the bank being dumped is clear in the K2500 before sending the dump, so that the K2500 will not miss parts of the dump while its CPU is busy deleting already existing objects. This can be done using the DELBANK message (defined below). If the destination bank in the K2500 is pre-cleared, it is not necessary to wait for the DACK before sending. Even if the sender chooses not to wait for the DACK before sending the next message, it may be necessary to preserve the 50ms delay between the WRITE messages.

Due to the large amount of incoming data during a bank dump containing many objects, the receiving K2500 may have a more sluggish response to front-panel use and keyboard playing during the data transfer. This is normal behavior and the machine will become fully responsive as soon as the dump is finished.

DIRBANK = 0Ch type(2) bank(1) ramonly(1)

This is similar to the READBANK message. The DIRBANK message requests an INFO message (containing object size, name, and memory information) be returned for each object meeting the specifications in the 'type' and 'bank' fields. Following the last INFO response will be an ENDOFBANK message.

ENDOFBANK = 0Dhtype(2) bank(1)

This message is returned after the last WRITE or INFO response to a READBANK or DIRBANK message. If no objects matched the specifications in one of these messages, ENDOFBANK will be the only response.

DELBANK = 0Eh type(2) bank(1)

This message will cause banks of objects (of one or all types) to be deleted from RAM. The 'type' and 'bank' specifications are the same as for the READBANK message. The deletion will take place with no confirmation. Specifically, the sender of this message could just as easily delete every RAM object from the K2500 (e.g. 'type' = 0 and 'bank' = 127) as it could delete all effects from bank 7 (e.g. 'type' = 113, 'bank' = 7.)

MOVEBANK = 0Fh type(2) bank(1) newbank(1)

This message is used to move entire banks of RAM objects from one bank to another. A specific object type may be selected with the "type" field. Otherwise, if the "type" field is unspecified (0), all object types in the bank will be moved. The "bank" and "newbank" fields must be between 0 and 9. The acknowledgement is an ENDOFBANK message, with the "bank" field

equal to the new bank number. If the operation can't be completed because of a bad type or bank number, the ENDOFBANK message will specify the old bank number.

LOADMACRO = 10h

…tells K2500 to load in the macro currently in memory.

MACRODONE = 11h code(1)

…acknowledges loading of macro. Code 0 indicates success; code 1 means failure.

PANEL = 14h buttons(3n)

…sends a sequence of front-panel button presses that are interpreted by the K2500 as if the buttons were pressed at its front-panel. The button codes are listed in a table at the end of this chapter. The K2500 will send these messages if the Buttons parameter on the XMIT page in MIDI mode is set to On. Each button press is 3 bytes in the message. The PANEL message can include as many 3-byte segments as necessary.

Byte 1	Button event type:
08h	Button up
09h	Button down
0Ah	Button repeat
0Dh	Alpha Wheel
Byte 2	Button number (see table)
Byte 3	Repeat count (number of clicks) for Alpha Wheel; the count is the delta (difference) from 64—that is, the value of the byte minus 64 equals the number of clicks. A Byte 3 value of 46h (70 dec) equates to 6 clicks to the right. A Byte 3 value of 3Ah (58 dec) equates to six clicks to the left. For example, the equivalent of 6 clicks to the right would be the following message: (header) 14h 0Dh 40h 46 (eox)

For efficiency, multiple button presses should be handled by sending multiple Button down bytes followed by a single Button up byte (for incrementing with the '+' button, for instance.)

Object Types

These are the object types and the values that represent them in 'type' fields:

Type	ID (decimal)	ID (hex)	ID(hex, 'type' field)
Program	132	84h	01h 04h
Keymap	133	85h	01h 05h
Effect	113	71h	00h 71h
Song	112	70h	00h 70h
Setup	135	87h	01h 07h
Soundblock	134	86h	01h 06h
Velocity Map	104	68h	00h 68h
Pressure Map	105	69h	00h 69h

Quick Access Bank	111	6Fh	00h 6Fh
Intonation Table	103	67h	00h 67h

Master Parameters

The Master parameters can be accessed as type 100 (00h 64h), ID number 16. Master parameters cannot be accessed with any of the Bank messages.

Button Press Equivalence Table

Button	Code (hex)	Button	Code(hex)
Alphanumeric pad		***Soft-Buttons 'A-F'***	
zero	00	A '(leftmost)	22
one	01	B	23
two	02	C	24
three	03	D	25
four	04	E	26
five	05	F (rightmost)	27
six	06	AB	28
seven	07	CD (two center)	29
eight	08	EF	2A
nine	09	YES	26
+/-	0A	NO	27
Alphanumeric pad		***Edit/Exit***	
CANCEL	0B	EDIT	20
CLEAR	0C	EXIT	21
ENTER	0D		
Navigation		***Mode Selection***	
Plus (+)	16	PROGRAM	40
Minus (-)	17	SETUP	41
Plus and Minus	1E	QUICK ACCESS	42
CHAN/BANK Inc	14	EFFECTS	47
CHAN/BANK Dec	15	MIDI	44
CHAN/BANK Inc/Dec	1C	MASTER	43
Cursor Left	12	SONG	46
Cursor Right	13	DISK	45
Cursor Left/Right	1A		
Cursor Up	10		
Cursor Down	11		

Cursor Up/Down	18

The next four commands allow you to read the screen display, both text and graphics layers.

ALLTEXT = 15h

…requests all text in the K2500's display.

PARAMVALUE = 16h

…requests the current parameter value.

PARAMNAME = 17h

…requests the current parameter name.

GETGRAPHICS = 18h

…requests the current graphics layer.

SCREENREPLY = 19h

This is the reply to ALLTEXT, PARAMVAL, PARAMNAME, GETGRAPHICS, or SCREENREPLY.

The reply to ALLTEXT will be 320 bytes of ASCII text (the display has 8 rows of 40 characters each). If you receive less than that, then the screen was in the middle of redrawing and you should request the display again.

The reply to PARAMVALUE will be a variable length ASCII text string. Some values (like keymaps, programs, samples, etc.) include their ID number in the text string (e.g., "983 OB Wave 1"). Some messages are also padded with extra spaces.

The reply to PARAMNAME will be a variable length ASCII text string. In cases where there is no parameter name (like on the program page) there will just be the single 00 null terminator.

The reply to GETGRAPHICS will be 2560 bytes of information. The 6 least significant bits of each byte indicate whether a pixel is on or off. If pixels are on over characters, the text becomes inverted. Characters on the K2500 display are a monospaced font with a height of 8 pixels and a width of 6 pixels.