# 12 Events.

The event mechanism is primarily provided by the Kernel to support the handling of interrupts and other external events. However, the mechanism may also be used to handle internal events in complicated programs (such as a simulation, for example). An event is characterised by the following:

**a. Event Class (see section 12.1)**

Events may be synchronous or asynchronous, express or normal.

**b. Event Priority (see section 12.1)**

Synchronous events have an associated priority.

**c. Event Count (see section 12.2)**

Each time an event occurs the count is incremented.
Each time an event is processed the count is decremented.
The event may be disarmed by setting the count negative.

**d. Event Routine. (see section 12.3)**

The address of the routine which is called to process the event.

An event appears to the Kernel as a data block containing the above values (see Appendix X for the exact layout of an event block). The block must be in the central 32K bytes of memory, so that the Kernel can access it without worrying about the ROM enable state.

When an event occurs the associated event block is kicked by calling KL EVENT. If the event count is negative, the 'kick' is ignored, otherwise the event count is incremented (up to a maximum of 127) and the event routine will be called at some time in the future - depending on the event class. When the event routine returns the event count is decremented, unless it has been set to zero or negative in the meantime.

## 12.1 Event Class.

Events are either synchronous or asynchronous. Asynchronous events are intended for the processing of external events which require almost immediate service. The processing of asynchronous events pre-empts the main program. The processing of synchronous events is under the complete control of the main program, which will, in general, deal with them when it is convenient to do so.

**a. Asynchronous Events.**

An asynchronous event is processed immediately the event is kicked - or almost immediately if the kick occurs in the interrupt path – see section 11 on interrupts. The Kernel does not provide any interlocks between asynchronous events and the main program or other events, so care must be exercised to avoid interactions. It is most unwise to call routines that are not re-entrant - for example, the firmware screen driving routines.

If the event count is still greater than zero when the event routine returns, it is decremented. If the count remains greater than zero then the process is repeated (the event routine is called again and the event count is decremented) until the count becomes zero or is set negative (see 12.2 below).

**b. Synchronous Events.**

Synchronous events are not processed when the event is kicked, but are placed on the synchronous event queue, waiting to be processed. Events are queued in descending order of priority - equal priority events after those already on the queue.

The foreground program should poll the synchronous event queue regularly, to see if there are any events outstanding. If there are then it should process them. The difference between synchronous and asynchronous events is, therefore, that the foreground program decides when synchronous events should be processed, but the event 'kicker' decides when asynchronous events are processed. Provided that the foreground program takes suitable care, there should be no difficulty in handling the interactions and resource sharing between synchronous events and the foreground program.

When the foreground program finds the synchronous event queue is not empty it should (but is not constrained to) instruct the Kernel to process the first event on the queue. When a synchronous event routine is run the Kernel remembers the priority of the event. In the event routine the synchronous event queue may be polled, but the Kernel hides any event whose priority is less than or equal to that event currently being processed. When the event routine returns the previous event priority is restored - so the processing of events may be nested.

The synchronous event priorities are split into two ranges, express and normal. All express events have higher priorities than all normal events. The Kernel provides a mechanism to disable the processing of normal events, without affecting express events. This may be used to implement 'critical regions' through which normal events may interact. The synchronous event 'kicked' by the Key Manager break handling mechanism is an example of an express synchronous event.

## 12.2 Event Count.

The main purpose of the event count is to keep track of the difference between the number of times the event has been kicked, and the number of times the event has been processed. This ensures that a kick is not missed if it occurs before the previous kick has been processed. The event count is normally incremented when the event is kicked and decremented when the event routine returns. However the exact action depends on the event count as follows:

Increment.

| | |
|---|---|
| -128..-2: | The count is not changed - the event is ignored. |
| -1: | This value is illegal. |
| 0: | The count is incremented and event processing is initiated as required by the even class. |
| 1..126: | The count is incremented but no further action is taken. The event is waiting for a previous kick to be processed or for processing to complete. |
| 127: | The count is not changed - the kick is ignored. |

Decrement.

| | |
|---|---|
| -128: | This value is illegal. |
| -127..0: | The count is not changed - the event has been disarmed. |
| 1: | The count is decremented and the event processing is terminated. |
| 2..127: | The count is decremented and the event processing is continued. |

Note that the event routine may disarm itself by setting the count negative (by convention to -64) and can discard unwanted kicks by setting its count to one.

## 12.3 Event Routine.

In general the address of the event routine is given as a 3 byte 'far address' (see section 2 on the memory layout). This allows the routine to be located in any ROM or anywhere in RAM.

A special form of the address class may specify the routine as at a 'near address'. This does not change the ROM state and so the routine must be located either in the lower ROM or in the central 32K of RAM. The ROM select byte of the 'far address' is ignored and the other two bytes taken as the address of the routine. Calling a 'near address' event routine requires a little less work than calling a full 'far address', and is used by the firmware itself.

## 12.4 Disarming and Reinitializing Events.

Before an event block may be reinitialized the event must be disarmed. This ensures that the event is removed from the various event pending queues and prevents the event queues being corrupted when the event block is initialized. An asynchronous event must not be reinitialised from inside its asynchronous event routine (because in this case disarming the event does not remove the event from the interrupt event pending queue).

Synchronous and asynchronous events are disarmed in different manners.

**a. Asynchronous Events.**

An asynchronous event should be disarmed by calling KL DISARM EVENT. This sets the event count to a negative value (-64) and thus prevents kicks having any effect. If the event is on the interrupt event pending queue then it will be discarded only when an attempt is made to process the event and not immediately that the event is disarmed.

**b. Synchronous Events.**

A synchronous event should be disarmed by calling KL DEL SYNCHRONOUS. This sets the event count to a negative value (-64) and removes the event block from the synchronous event pending queue (if it is on the queue).

The above procedures prevent the event being successfully kicked, they do not prevent attempts being made to kick the event. A fast ticker, frame flyback or ticker event (see section 11.5) will still be on its appropriate queue and will still be receiving regular attempts to kick it. To prevent time being wasted (and the system from being slowed down because of it) the event should be removed from the interrupt queue by calling KL DEL FAST TICKER, KL DEL FRAME FLY or KL DEL TICKER.