About Pulses and Pulse Events

Authored by: **Steve Dufresne**
Updated by: **Thomas Fletcher**

A pulse is a special notification mechanism whereby a sender sends a message to a receiving process without waiting for acknowledgment. Contrast this with the Send-Receive-Reply protocol, where the sender blocks waiting until the receiver receives the message and then replies.

When would you use a pulse?

Sometimes the sender of a message can't afford to block.

One example of the use of a pulse would be where a driver needs to notify a client that data is available. The driver would buffer the data and then send a pulse to the client. At its leisure, the client receives the pulse message and sends a message to the driver asking for the data. In this situation, at no time is the driver blocked waiting for the client. It was blocked on the **MsgReceive***() call, but that is where it wanted to be while waiting for notification from an ISR, the kernel, or an interrupt-handling thread.

Delivery isn't guaranteed

Pulse delivery isn't guaranteed. If sending a pulse from one thread to another, where both threads are on the same machine, then your only guarantee is that the connection that the pulse sent over is valid. If the receiver is in another process, then this implies that the receiver process is still alive. However, if the receiver is not sitting in its receive function (**MsgReceive***()) waiting for the pulse, then the pulse is added to the receiver's receive queue. If the receiver dies before receiving the pulse, then it simply never gets there. In sending a pulse across the native Neutrino network, you can't be guaranteed even that the receiver is still alive.

What's in a pulse?

When sending a pulse, you can include an 8-bit code and a 32-bit value. The code is typically used as a pulse message type, since you may be using pulses for a variety of things and will need some way of distinguishing between them. This means that you really have only the 32-bit value for the data that may vary from time to time. The pulse code should be in the range **_PULSE_CODE_MINAVAIL** to **_PULSE_CODE_MAXAVAIL**.

Pulses and priorities

A pulse has a priority. This priority specifies where the pulse will be inserted in the receiver's receive queue. Let's say the receiver isn't currently sitting RECEIVE-blocked and a priority 13 thread sends a message via **MsgSend**() to the receiver. This message is added to the receiver's receive queue. If a priority 17 pulse is then sent to the receiver, this pulse is inserted into the receive queue in front of the message from the priority 13 thread. When the receiver next gets to its **MsgReceive**(), the pulse will be received.

By default, a receiver floats to the priority of the sender. This means that if the receiver is currently at priority 15, and if a priority 17 pulse arrives, then the receiver will be boosted to priority 17. This is okay, because the receiver will do the work at the pulse's priority and then go back to its RECEIVE-blocked state. It will still be at priority 17, but who cares -- it's now blocked.

When another message arrives, it will then run at the priority of that message's sender. This priority-floating can be suppressed with the **_NTO_CHF_FIXED_PRIORITY** channel flag.

Using MsgDeliverEvent() to send a pulse

You can use either of these two functions to send a pulse:

- **MsgSendPulse**()
- **MsgDeliverEvent**()

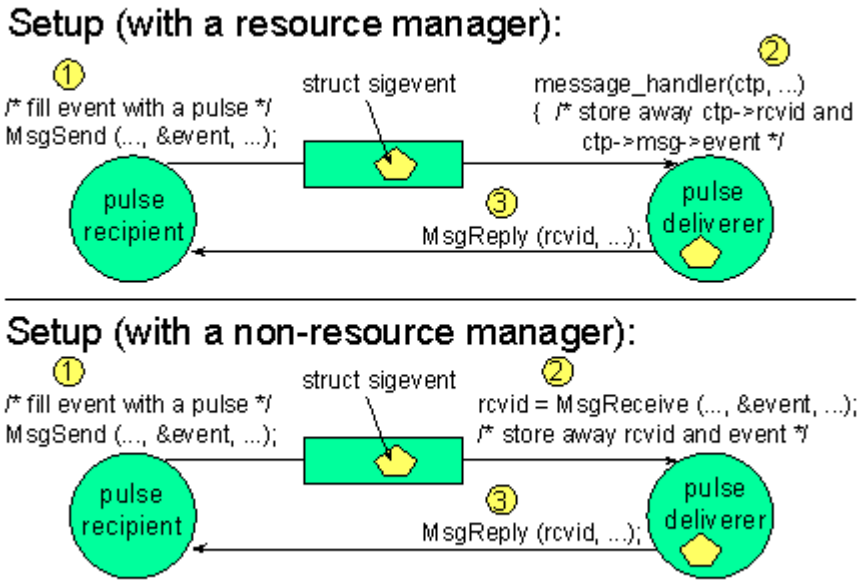
We'll discuss the **MsgDeliverEvent**() function here.

The idea behind events, and hence **MsgDeliverEvent**(), is twofold. First, it provides a common mechanism for the delivery of pulses, signals, and other types. Second, it allows the recipient to choose how it gets notified (i.e. via a pulse, signal, etc...).

There are two parts to dealing with events: setup and delivery. The setup involves the pulse's recipient filling a **struct sigevent** with a pulse event, and then giving it to the delivering thread (which could be a thread in a separate process). You'll find some handy macros for filling in the **sigevent** structure in the *Neutrino Library Reference* manual (under the **sigevent** entry).

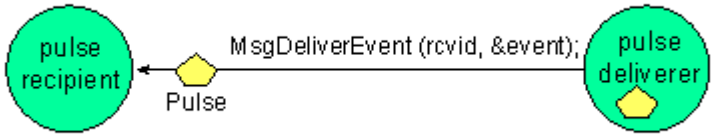
The setup is typically done only once. Then, whenever the delivering thread wants to notify the recipient that something has happened, it delivers the event by calling **MsgDeliverEvent**(). At no time does the delivering thread need to know that the event contains a pulse.

The following diagram illustrates two setups: one where the pulse deliverer is a resource manager and one where it's not.



In both cases, the pulse deliverer has to save away the **rcvid** and the event structure for later use with the **MsgDeliverEvent**() function.

The following diagram illustrates the delivery of the pulse.



Notice that the call to **MsgDeliverEvent**() is passed both the **rcvid** and the event that had previously been saved away.