```
COMMENT
This mechanism emits spike events at the times given in a supplied Vector.

Example usage:
    objref vs
    vs = new VecStim(.5)
    vs.play(spikevec)

This is a modified version of the original vecstim.mod (author unknown?) which
allows multiple vectors to be used sequentially. This saves memory in a long
simulation, as the same storage can be reused.

The mechanism checks at intervals 'ping' whether a new vector has been provided
using the play() procedure and if so resets its pointer to the first element
in the new vector. Note that any spikes remaining in the first vector will be
lost. Any spiketimes in the new vector that are earlier than the current time
are ignored.

The mechanism actually checks slightly after the ping interval, to avoid play()
and the ping check occurring at the same time step but in the wrong order.


Extracts from the comments on the original vecstim:

The idiom for getting a  Vector argument in a model description is encapsulated
in the "play" procedure. There are potentially many VecStim instances and so the
Vector pointer must be stored in the space allocated for the particular instance
when "play" is called. The assigned variable "space" gives us space for a double
precision number, 64 bits, which is sufficient to store an opaque pointer.
The "element" procedure uses this opaque pointer to make sure that the requested
"index" element is within the size of the vector and assigns the "etime" double
precision variable to the value of that element. Since index is defined at the
model description level it is a double precision variable as well and must be
treated as such in the VERBATIM block. An index value of -1 means that no
further events should be sent from this instance. Fortunately, space for model
data is cleared when it is first allocated. So if play is not called, the
pointer will be 0 and the test in the element procedure would turn off the
VecStim by setting index to -1. Also, because the existence of the first
argument is checked in the "play" procedure, one can turn off the VecStim with
    vs.play()
No checking is done if the stimvec is destroyed (when the reference count for
the underlying Vector becomes 0). Continued use of the VecStim instance in this
case would cause a memory error. So it is up to the user to call vs.play() or to
destroy the VecStim instance before running another simulation.

The strategy of the INITIAL and NET_RECEIVE blocks is to send a self event
(with flag 1) to be delivered at the time specified by the index of the Vector
starting at index 0. When the self event is delivered to the NET_RECEIVE block,
it causes an immediate input event on every NetCon which has this VecStim as its
source. These events, would then be delivered to their targets after the
appropriate delay specified for each NetCon.
ENDCOMMENT


: Vector stream of events

NEURON {
    ARTIFICIAL_CELL VecStim
    RANGE ping
}

PARAMETER {
    ping = 1 (ms)
}

ASSIGNED {
    index
    etime (ms)
    space
}

INITIAL {
    index = 0
    element()
    if (index > 0) {
        net_send(etime - t, 1)
    }
    if (ping > 0) {
        net_send(ping, 2)
    }
}

NET_RECEIVE (w) {
    if (flag == 1) {
        net_event(t)
        element()
        if (index > 0) {
            if (etime < t) {
                printf("Warning in VecStim: spike time (%g ms) before current time (%g ms)\n",etime,t)
            } else {
                net_send(etime - t, 1)
            }
        }
    } else if (flag == 2) { : ping - reset index to 0
        :printf("flag=2, etime=%g, t=%g, ping=%g, index=%g\n",etime,t,ping,index)
        if (index == -2) { : play() has been called
            :printf("Detected new vector\n")
            index = 0
            : the following loop ensures that if the vector
            : contains spiketimes earlier than the current
            : time, they are ignored.
            while (etime < t && index >= 0) {
                element()
                :printf("element(): index=%g, etime=%g, t=%g\n",index,etime,t)
            }
            if (index > 0) {
                net_send(etime - t, 1)
            }
        }
        net_send(ping, 2)
    }
}


VERBATIM
extern double* vector_vec();
extern int vector_capacity();
extern void* vector_arg();
ENDVERBATIM

PROCEDURE element() {
VERBATIM
    { void* vv; int i, size; double* px;
        i = (int)index;
        if (i >= 0) {
            vv = *((void**)(&space));
            if (vv) {
                size = vector_capacity(vv);
                px = vector_vec(vv);
                if (i < size) {
                    etime = px[i];
                    index += 1.;
                } else {
                    index = -1.;
                }
            } else {
                index = -1.;
            }
        }
    }
ENDVERBATIM
}

PROCEDURE play() {
VERBATIM
```

```
    void** vv;
    vv = (void**)(&space);
    *vv = (void*)0;
    if (ifarg(1)) {
        *vv = vector_arg(1);
    }
    index = −2;
ENDVERBATIM
}
```
end