How WaveMix works.

1. Mixing Sounds

The low-level multimedia wave out put function (waveOutWrite) does not support multiple simultaneous channels of output. So in order to accomplish this we must fake out the output device. This is done by premixing the wave output and then sending this new wave to the wave output device. The core concept which wavemix uses to mix files at run time is very simple. What it does is take a small slice off of each input wave file (each input is referred to as an "channel"), mix them together into an extra buffer which is the same length of the slice and then submit this wave buffer to the multimedia function waveOutWrite.

µ §

A digital wave is essentially an array of wave samples. At 11Khz 8bit Mono (which is used by the Wave Mixer) a wave array (or buffer) will contain 11025 byte elements per sec. of the wave duration. (i.e. a one second wave will contain 11025 data samples, a 1 minute data wave will contain 60*11025=661500 samples). To mix the waves in real time the input waves are summed together into another destination buffer. The destination buffer is typically a small size so that we can achieve real time results. A typical length is 1/8 th of a second = 11025/8=1378 samples.

µ §

e.g. Assuming that we have 4 waves playing and the slice length = 1378 samples. we could call a mixing routine:
mixit(lpDest,rgpCDdata,4,1378);

```
void mixit(LPSAMPLE lpDest, LPSAMPLE rgWaveSrc[], int iNumWaves, WORD wLen)
{
  int i,iSum;
  WORD ctr;
  ctr = 0;
  while (wLen)
  {
    iSum=128; /* 128 is the "normal" value (silence) for 8 bit 11KHz */
    for (i=0;i<iNumWaves;i++)
      iSum = iSum + *(rgWaveSrc[i]+ctr) -128;
    PEG((int)0,iSum,(int)255);
    *lpDest++=iSum;
    ctr++;
    wLen--;
  }
}
```

The above routine will visit the i th element in each source array, sum them into a destination variable (iSum) that can accommodate any possible overflow (i.e. 8 bit elements are summed into a 16 bit destination) and then the destination variable is converted back to the original magnitude (saturation is done after the additions to minimize distortion).

The destination buffer can now be submitted to the wave output device (using waveOutWrite) and the user will hear all four sounds played simultaneously. When the wave output device completes playing the buffer it will return it to the client (WaveMix.DLL) with a request for more output data.

Since the wave output device can only play data that has been submitted to it and it takes a finite amount of time to mix the wave data, as soon as we submit the destination wave to the output device we must mix another buffer with the next slice of data and submit it to the wave output device to avoid an interruption in the audio output. This buffer will get queued by the device and it will then play the data in it when it finishes playing the data in the first buffer.

While the output device is playing this second buffer we can mix the third slice into the first buffer and then submit it back to the device. We continue this "ping-pong" procedure of mixing the data and submitting the buffer to the wave output device until we have submitted all the wave data, at that point since we no longer submit data to the device it will stop playing. Note: In practice we often use more than two buffers (i.e. 3 or 4 is common) We then replace the ping-pong action with a "juggling" system:

µ §

The above text and diagrams describe the mixing process. There are, however, other situations which occur that greatly complicate the mixing process: Playing multiple files that are of uneven length, A request to start a new wave playing while others are already playing, A request to terminate a specific wave that is simultaneously playing with other wave files, A request to play multiple wave files and start them together. A request to pause the wave output, but not lose the current location.

Playing multiple files of uneven length:

µ §

The Wave Mixer has to take into account the possibility that the current slice will be shorter than the wave slice which it is attempting to fill. That is: one of the waves which it is mixing does not contain sufficient data to have a sample mixed into all the elements of the destination buffer. To handle the situation the wave mixer must first determine if a wave like this exists. If it does then it must determine how many samples it can mix from that wave. It will then mix from all the waves for that many samples. Then it will stop mixing that wave and mix the remaining waves for the second part of the destination buffer. Since it is possible that the same situation can occur again while mixing the remaining waves into the remaining part of the buffer it must repeat the above process again. This process will continue until the destination buffer gets completely filled up or their is no more wave data remaining to be mixed. At this point the destination buffer is submitted to the wave output device.

A request to start a new wave playing while others are already playing: "remixing"

A complex situation that the real time wave mixer must handle is a request to play a new wave file while there is currently other waves being played. The reason that this is difficult is that the wave mixer is actually mixing wave data a finite amount of time before the wave output that is currently being played by the wave driver. The wave mixer must determine the position at which the wave output device is actually playing wave data, "remix" the wave data to include the new wave, notify the output device that the data it is currently playing is no longer valid (waveOutReset( )) and submit new wave data to it that contains the new wave too. All of this must be done very quickly so that the user does not hear an interruption in the audio.

A second algorithm is also employed to achieve the above effect on hardware configurations where doing a waveOutReset can cause an audible click or cause the hardware to slow down. In these situations the wave mixer queues the new wave but does not interrupt the wave output device. When it mixes the next slice of data it will include the new wave also. This method can have a small delay which is the amount of time from which the wave data is submitted to when it is played: The driver must finish playing the current buffer, play all the previously queued buffers, and then play the new buffer. If there are three buffers being juggled with the wave output device the delay can be the amount of time required to play two to three buffers before the new wave data can be heard. If the wave buffers are very short then it is difficult but not impossible to hear the delay.

A request to terminate a specific wave that is simultaneously playing with other wave files:

Occasionally while playing multiple files the WaveMix.DLL will be requested to stop playing a wave on a particular channel before that wave has completed playing, i.e. "flush" a channel. This is handled in a similar manner to starting a new wave while others are playing. The wave mixer first determines the wave output position. It then removes the desired wave from the specified channel and "remixes" the remaining waves from the obtained output position.