```
WaveTracking/FREE counter
```

Turning wave tracking on resets the counter even if it was already on.

This command demonstrates that no free waves were leaked since counting was turned on:

```
WaveTracking/FREE count; Print V_numWaves
  0
```

Now we run the Leaks function and check the count again:

```
Leaks(); WaveTracking/FREE count; Print V_numWaves
  2
```

After running the Leaks function, there are two free waves that did not exist before. There is no way at this point to get rid of them, but at least now you know about it, and perhaps you can figure out what leaked them.

The counter tracking mode does a simple increment when a wave of a given category is created, and a decrement when a wave is killed. If you kill a wave that was created before the tracking counter was started, the count can go negative.

Naturally the Leaks function is much less complex than typical code. It can be quite difficult to figure out where the leaked waves are created. To help with that, you can use the tracker mode and give your free waves names. If you want to follow along with the example, copy these functions to your procedure window and then close it:

```
Function/WAVE GenerateFreeNamed()
   return NewFreeWave(2,3, "MadeByGenerateFree")
End

Function LeaksNamed()
   Duplicate/O GenerateFreeNamed(),dummy           // This leaks
   Variable maxVal = WaveMax(GenerateFreeNamed())  // So does this
End
```

We now run LeaksNamed with WaveTracking in tracker mode:

```
WaveTracking/FREE tracker
LeaksNamed()
WaveTracking/FREE dump
Since tracking began, 2 free wave(s) have been created and not killed.
Wave 'MadeByGenerateFree'; refcount: 0
Wave 'MadeByGenerateFree'; refcount: 0
```

This time, instead of printing the count of free waves, we used "dump" to get a list of the free waves. The dump keyword prints a list in the history area showing the name of each free wave and its reference count. The names tell you that those were waves made by the GenerateFreeNamed function.

In the example in **Wave Reference MultiThread Example** on page IV-327, a WAVE wave, named ww, is created at the start of the Demo function. ww is filled with free waves by the MultiThread call. At the end of the function, KillWaves is called to kill ww which automatically kills the free waves. If you omitted the KillWaves call at the end, this would create a large leak. You could investigate it using `WaveTracking/FREE tracker` and `WaveTracking/FREE dump`.

Creating Igor Extensions

Igor includes a feature that allows a C or C++ programmer to extend its capabilities. Using Apple's Xcode or Microsoft Visual C++, a programmer can add command line operations and functions to Igor. These are called external operations and external functions. Experienced programmers can also add menu items, dialogs and windows.

A file containing external operations or external functions is called an "XOP file" or "XOP" (pronounced "ex-op") for short.

Here are some things for which you might want to write an XOP:

- To do number-crunching on waves.
- To import data from a file format that Igor does not support.
- To acquire data directly into Igor waves.
- To communicate with a remote computer, instrument or database.

The main reasons for writing something as an XOP instead of writing it as an Igor procedure are:

- It needs to be blazing fast.
- You already have a lot of C code that you want to reuse.
- You need to call drivers for data acquisition.
- It requires programming techniques that Igor doesn't support.
- You want to add your own dialogs or windows.

Writing an XOP that does number-crunching is considerably easier than writing a stand-alone program. You don't need to know anything about Macintosh or Windows APIs.

If you are a C or C++ programmer and would like to extend Igor with your own XOP, you need to purchase the Igor External Operations Toolkit from WaveMetrics. This toolkit contains documentation on writing XOPs as well as the source code for several WaveMetrics sample XOPs. It supplies a large library of routines that enable communication between Igor and the external code. You will also need the a recent version of Xcode, or Microsoft Visual C++.