# Writing ETW listener in C#

1. Create a new .NET Framework 4.5 (or newer) Console App project in Visual Studio
2. To be able to collect and process ETW events we need to add the NuGet package Microsoft.Diagnostics.Tracing.TraceEvent:
   1. Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution
   2. Find Microsoft.Diagnostics.Tracing.TraceEvent (from nuget.org) in the Browse tab
   3. Select the package, add it to the solution and click Install
3. Only elevated processes can read ETW events so we must run our program as Administrator (either from elevated Visual Studio or cmd). You can use TraceEventSession.IsElevated() to check if your program is being run as Administrator and exit the program if it isn’t.
4. The 3 main classes we are going to work with:
   * Trace session class: TraceEventSession
   * ETW Events processor class: TraceEventSource
   * Event parser class: TraceEventParser
5. Let’s start by creating a session:

using (var session = new TraceEventSession("my first trace event session")) {}

* TraceEventSession implements IDisposable thus we can create it inside a using block, so it is naturally disposed at the end of the block.
* Sessions can outlive the process that created them, so if not setting StopOnDispose property of the session to false, the ETW session will be stopped when the object is disposed.
* TraceEventSession c’tor receives the session name. These names are unique across the machine. Since sessions can outlive the process, the name can be used to connect to the session from another process.
* Unless given additional parameters, the c’tor starts a real-time session, which is what we are going to use. Otherwise you can write the session output to an .etl file for later processing.

1. Naturally if the process that created the session is terminated, it won’t dispose the session object and the ETW session on the machine will live. We want to prevent that and close the ETW session when killing the process by pressing Ctrl+C, so inside the using block we’ll add:

Console.CancelKeyPress += (object sender, ConsoleCancelEventArgs e) => session.Dispose();

1. You can use logman query -ets to see all active sessions, and logman stop <session-name> -ets to stop specific sessions
2. We would like to enable the kernel provider:

session.EnableKernelProvider(

KernelTraceEventParser.Keywords.ImageLoad | KernelTraceEventParser.Keywords.Process);

The first parameter to this function specifies which events will be turned on. See KernelTraceEventParser.Keywords for all available keywords and KernelTraceEventParser for all available events.

*Note for Windows 7 and earlier users: to be able to add the kernel provider to a session you must name the session KernelTraceEventParser.KernelSessionName. Notice that only one session with this name can exist on the machine at a time, and no other providers can be enabled for that session.*

1. Now let’s hook events processor, parser and a callback to be called once events are fired:

session.Source.Kernel.ImageLoad += DllLoad;

* Real-time sessions have the Source property of type ETWTraceEventSource. Accessing it attaches the events processor to this session. It in turn has properties for the most common event parsers: Registered, Dynamic, Clr, Kernel. Accessing any of them attaches events parsers to this source.
* ImageLoad is the event we are subscribing to. Go ahead and implement DllLoad. It should return void, and for its parameter type see ImageLoad definition.
* Now go ahead and find the events enabled by KernelTraceEventParser.Keywords.Process keyword and add callbacks for them as well.

1. Finally, we would like to start the session:

session.Source.Process();

Notice that this call returns only when stopped prematurely, for example when pressing Ctrl+C

# Further exploration

1. Use logman query providers to discover available providers. Those providers’ events can be parsed by the session.Source.Dynamic parser.
2. To add providers other than the kernel providers to the session you have created and parse their events:

session.EnableProvider(string providerName, TraceEventLevel providerLevel);

session.Source.Dynamic.All += HandleDynamicEvents;

1. You can use the above to discover available providers and events, by for example printing the event name and ToString() representation from the assigned callback. After you have a specific provider and event you want to process, you can use the following template:

session.Source.Dynamic.AddCallbackForProviderEvent(<provider-name>, <event-name>, callback);

The callback should take single parameter of type TraceEvent, and inside the callback you should access the event properties using the PayloadByName method of TraceEvent.

*Note for Windows 7 and earlier users: The above will only work on Windows 8 and 10. Before that, to enable the kernel provider you must have called the session KernelTraceEventParser.KernelSessionName which prevents you from adding non-kernel providers to this session – so you in a single session you must choose between kernel and non-kernel providers but you can’t mix.*

*One solution for that can be to create 2 different sessions, one for the kernel provider and the other one for all other providers. To do this your program must be multi-threaded and adopt thread-safety mechanisms. It is ok to dispose sessions and enable providers from multiple threads, but accesses to the Source property, the providers of a session and calling Process() must be done from a single thread only.*

*An alternative solution can be to create a single non-kernel session, adding only non-kernel providers, and getting the events we got earlier from the kernel provider from other providers that also provide them. Use* logman query providers *and look for the providers with their name starting with Microsoft-Windows-Kernel. For example, Microsoft-Windows-Kernel-Process provides the events we used before from the kernel provider.*