

dotNet Protector® Trace Decoder

How it works

TraceDecoder helps you in handling runtime errors in your application.

The goal is to get the source line where an error occurs. This is usually done by loading the program database (pdb debug information) in addition to an assembly. Once protected, pdb are not valid anymore, and could give sensitive information.

Instead of deploying pdb's with an application, dotNet Protector® can build decodable stacktraces at run-time. These encoded stacktraces can be decoded only with a matching traceinfo file (data contained in the encoded stacktrace is what available at runtime – obfuscated names).

If you collect encoded stacktraces in an error report module, you can then decode them with dotNet Protector's tracedecoder library and your stored traceinfos.

To enable traceinfo production, you just need to check 'Build trace decoding informations'.

Encoded trace format

An encoded trace is a multiline string (one line per encoded frame).

And encoded frame has 2 sections separated by a NUL character ('\0'). The first section is the traceinfo ID (actually the module MVID) followed by frame location informations. The second section is the method name recovered from reflection in the following format:

[module_Name]Full_Type_Name::Method_Name(Parameter_types)

If you wish to collect encoded traces, it is important that you keep the string formatting intact (a BinaryWriter can do the job).

The TraceDecoder Library (PvLogiciels.dotNetProtector.TraceDecoder.dll)

As you may expect, TraceDecoder is protected by dotNet Protector, thus it needs dotNet Protector's runtime.

The main class is the TraceDecoder class (PvLogiciels.dotNetProtector.TraceDecoder). The constructor instantiates a new TraceDecoder which by default looks for traceinfo files in dotNet Protector's programdata directory. In fact, it registers a DirectoryTraceInfoStore to load traceinfos from their default location. You can implement your own TraceInfoStore derived classes by implementing the only method : GetModuleDebugInfoReader(Guid); this methods take the traceinfo ID as argument and returns a BinaryReader to the traceinfo data (or null if traceinfo not found). The underlying stream is closed by the TraceDecoder when the data have been read.

To register you custom TraceInfoStore, call the RegisterStore Method.

To decode a trace, simply call the DecodeTrace Method, which takes an encoded trace as argument and returns a decoded trace.

The TraceDecoderSample sample solution

This solution is straightforward:

FaultyApplication : a dummy winforms application that throws exception each time a button is clicked, with a ThreadException handler that catches exceptions, displays the stacktrace (obfuscated one, given by the framework), and records the encoded trace in a 'Errors.log' file.

TraceReader : a winform application that loads the log files and decodes the encoded traces.

Protect FaultyApplication using FaultyApplication.dpp and run FaultyApplication. Launch TraceReader to decode the log.