



Ellisys Analyzer C# SDK Guide

Version 1.0, June 9, 2026

Table of Contents

1. Introduction	1
2. Installing the Remote Control plugin	1
3. Adding the SDK to your project	3
4. Getting started	3
5. Key concepts and design	3
5.1. The Analyzer and the connection	3
5.2. Overviews, scopes and handles	4
5.3. Cursors: never read everything by accident	4
5.4. The growth guard (loading and recording)	4
5.5. Product facets	4
5.6. Typed exports	5
5.7. Errors	5
6. How-to recipes	5
6.1. Extract a whole trace to CSV (constant memory)	5
6.2. Wait for a condition during a live capture	5
6.3. Annotate items found by a search	5
6.4. Build a table (column projection)	6
6.5. Preview the wire options without exporting	6
6.6. Survey an unknown trace	6
7. Overriding Connection Properties	6
7.1. By file	6
7.2. By command line	6
7.3. Priority order	6
8. Running the provided samples	7
8.1. C# / .net sample	7
8.2. Python sample	7
8.3. Using a different language or platform	7
9. Overview queries	7
9.1. Syntax	8
9.2. Examples	8
9.3. Behavior notes (important for automation)	8
10. Using with AI agents	9
10.1. Installing the agent skill	9
10.2. The MCP server	10
10.3. Choosing between the skill and the MCP server	11
11. API reference	11
11.1. Connecting and the Analyzer	11
11.1.1. Analyzer	11

11.2. Overview navigation	12
11.2.1. ScopeClosedException	12
11.2.2. BoundExceededException	12
11.2.3. OverviewIncompleteException	12
11.2.4. OverviewLimits	13
11.2.5. ItemField (enum)	13
11.2.6. XmlFilter	13
11.2.7. ItemRecord	14
11.2.8. ReleaseMode (enum)	14
11.2.9. OverviewItem	14
11.2.10. OverviewDefaults	16
11.2.11. OverviewItemList	16
11.2.12. Page	17
11.2.13. ChildCursor	18
11.2.14. SearchCursor	19
11.2.15. OverviewScope	20
11.2.16. Analyzer	21
11.3. Product facets (shared)	22
11.3.1. ProductFacet	22
11.3.2. DeviceFilterMode (enum)	22
11.4. Bluetooth facet	22
11.4.1. BluetoothChannelSummary	22
11.4.2. BluetoothChannelStat	23
11.4.3. BluetoothSpectrumSample	23
11.4.4. BluetoothSpectrumRange	23
11.4.5. BluetoothBdAddr	23
11.4.6. BluetoothFacet	23
11.4.7. Analyzer	25
11.5. Wi-Fi facet	25
11.5.1. WifiFacet	25
11.5.2. Analyzer	25
11.6. WPAN / 802.15.4 facet	25
11.6.1. WpanFacet	25
11.6.2. Analyzer	26
11.7. USB 3.0 facet	27
11.7.1. Usb30Facet	27
11.7.2. Analyzer	27
11.8. Exports	27
11.8.1. ExportOptionException	27

11.8.2. ExportMode	27
11.8.3. ExportModes	28
11.8.4. IExportOptions	28
11.8.5. FilteredTraceOptions	28
11.8.6. ThroughputOptions	29
11.8.7. ExportOptions	29
11.8.8. Analyzer	30
11.9. Bluetooth exports	31
11.9.1. BluetoothExportModes	31
11.9.2. BluetoothPacketLossMode (enum)	31
11.9.3. BluetoothAudioOptions	31
11.9.4. BluetoothChannelsOptions	32
11.9.5. BluetoothAirtimeOptions	32
11.10. Markers	33
11.10.1. MarkerColor (enum)	33
11.10.2. Marker	33
11.10.3. Analyzer	33
11.11. Session, trace files and info	34
11.11.1. MessageSeverity (enum)	34
11.11.2. AppInfo	34
11.11.3. RecordingStatus	35
11.11.4. RunningTask	35
11.11.5. TraceFileInfo	35
11.11.6. Analyzer	36
11.12. Logic signals	40
11.12.1. LogicSignalTransitionType (enum)	40
11.12.2. LogicSignalTransition	40
11.12.3. Analyzer	41
11.13. Errors	41
11.13.1. RemoteControlException	41
11.13.2. ConnectionFailedException	41
11.13.3. OperationException	42
11.13.4. LoadTimeoutException	42
11.13.5. NotAvailableException	42
Revisions History	42
Copyright and Intellectual Property	42

1. Introduction

`Ellisys.Analysis.Automation` is an idiomatic C# SDK for the Ellisys analyzer Remote Control automation API. The automation API itself is based on the ZeroC ICE library (see the [Analyzer Remote Control User Guide](#) for the raw, language-neutral API): the Ellisys analysis software is the server and your program is the client.

The SDK wraps that API so you write ordinary C# instead of dealing with Ice directly:

- `Analyzer.Connect(...)` returns an `Analyzer` you use with `using` — it hides all Ice setup (communicator, proxy, checked cast) and disconnects on dispose.
- The Overview is navigated through **bounded cursors** (`IEnumerable<T>` streams, indexers, capped materializers), so a trace with millions of items is safe to traverse by construction.
- Product-specific operations live on **facets** (`analyzer.Bluetooth`, `.Wifi`, `.Wpan`, `.Usb30`); exports are typed; times are `long` picoseconds (or `TimeSpan`); and **no Ice type ever crosses the SDK surface**.
- Errors are a small, catchable hierarchy rooted at `RemoteControlException`.

It targets .NET 10 and .NET Framework 4.8 (`net10.0` and `net48`) and depends on `zeroc.ice.net 3.7.x` — the same Ice runtime as the analyzer's own plug-ins, so the SDK drops straight into existing C# solutions.



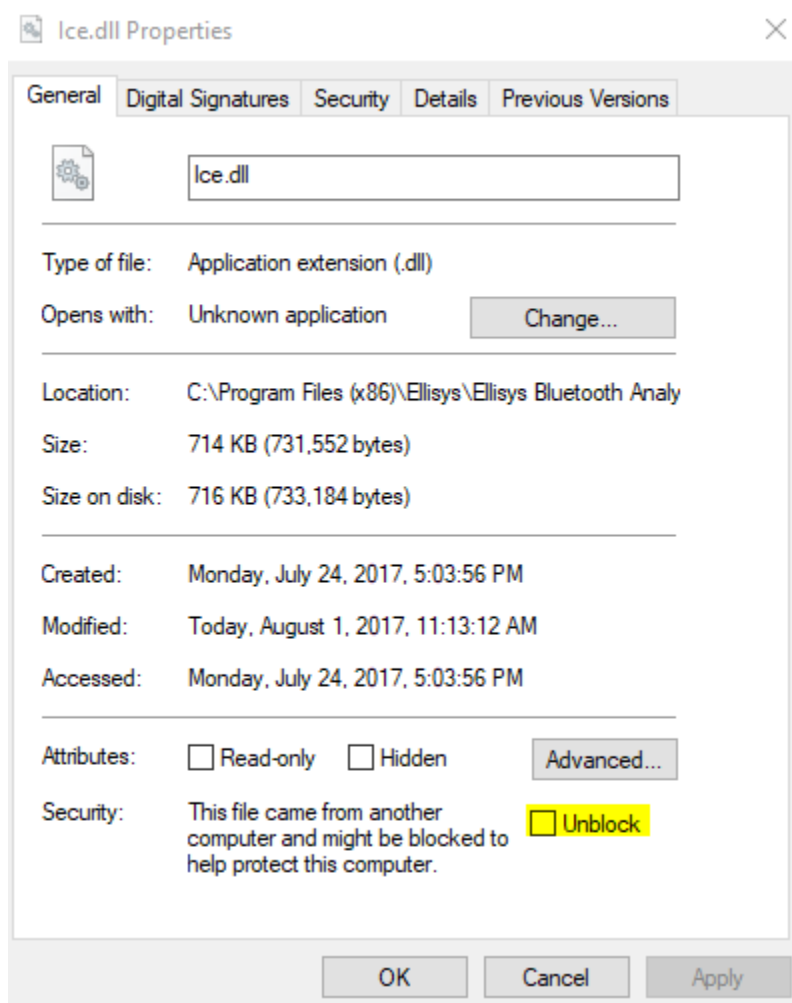
This guide is the recommended entry point for **C#** automation. For the underlying operation-level reference, or for other languages, see the [Analyzer Remote Control User Guide](#) and the product manuals.

2. Installing the Remote Control plugin

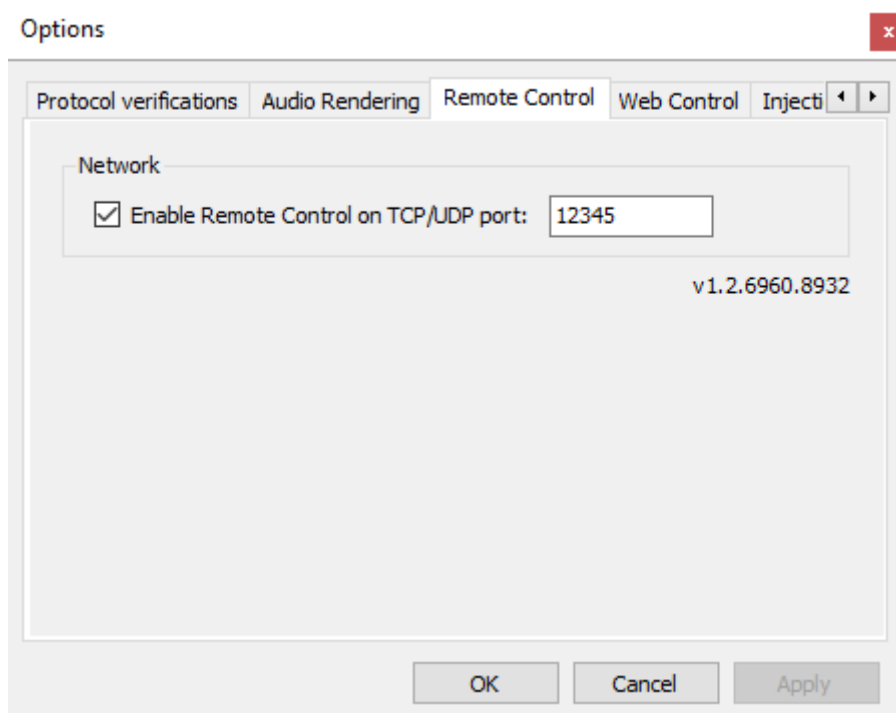
The remote control plugin is available on Ellisys website and is linked from the user manual of the analysis software. Please follow these steps to install the plugin:

1. Install the latest analysis software release.
2. Copy the `Binary\RemoteControl` folder and its content to the installation directory, typically `C:\Program Files (x86)\Ellisys\...Analyzer\RemoteControl`.

Please note that Windows tends to block files downloaded from the Internet. Make sure to unblock them in the file Properties dialog, as shown below:



3. Start the analysis application and go to menu "Tools", then "Options...". If step 2 above was successful you should see a panel named "Remote Control".



4. Check the box "Enable Remote Control on TCP/UDP port", and optionally change the port according to your network parameters. The software is now ready to accept incoming commands.

The port can be overridden by the command line with `/remote_control_port=54321`.

3. Adding the SDK to your project

Reference the SDK project (or its NuGet package) from your application; it pulls in the `zeroc.ice.net 3.7.x` runtime automatically:

```
dotnet add reference
..\Sdk\Analysis\CSharp\Ellisys.Analysis.Automation\Ellisys.Analysis.Automation.csproj
```

The SDK is fully XML-documented (IntelliSense) and ships an `llms.txt` usage guide for AI agents.

4. Getting started

Connect to a running analyzer (with the Remote Control plug-in enabled), load a trace, and read the first few Overview items:

```
using Ellisys.Analysis.Automation;

using var analyzer = Analyzer.Connect("localhost", 12345);
analyzer.Load(@"C:\traces\capture.btt"); // blocking by default; path is
on the analyzer

using var overview = analyzer.Overview("BR/EDR Overview");
foreach (var item in overview.Root.Cursor().Stream()) // bounded, batched, page-
released
{
    System.Console.WriteLine($"{item.Time} {item.Description}");
    if (item.TimePs > 5_000_000_000_000) // 5 s, in picoseconds
        break;
}
```

The `using` blocks matter: disposing the `analyzer` disconnects, and disposing the overview scope releases all of that overview's server-side item handles.

5. Key concepts and design

This is the mental model; the deeper internal rationale lives in the SDK's design notes.

5.1. The Analyzer and the connection

`Analyzer.Connect(...)` returns an `Analyzer` (an `IDisposable`). Every operation is synchronous and blocks until the analyzer replies. `Load(...)` waits for completion by default and throws `LoadTimeoutException` if it exceeds the timeout.

5.2. Overviews, scopes and handles

An **Overview** is a tree of protocol items. You open one as a scope:

```
foreach (var name in analyzer.Overviews()) System.Console.WriteLine(name);
using var overview = analyzer.Overview("BR/EDR Overview");
System.Console.WriteLine(overview.Root.ChildCount);      // cheap count, even at
millions
```

The server addresses items by opaque handles that live until released, and the only release primitive is `global` — so the SDK ties handle lifetime to the overview scope: disposing it frees every handle minted inside. Touching an item after its scope closed throws `ScopeClosedException` before any network call.

5.3. Cursors: never read everything by accident

You never get a plain list of children — that could be millions. You traverse through a **bounded cursor**:

```
var cursor = overview.Root.Cursor();                // page size defaults to 256
foreach (var item in cursor.Stream()) { /* one page at a time */ }

var page = overview.Root.Cursor().Page();           // a single bounded page
var first = overview.Root.Cursor().Take(50);        // explicit, bounded; throws
past the hard cap
```

- `Stream()` yields live items, paging transparently and (by default) releasing each page's handles — so do not hold a live item across a page boundary; snapshot it instead.
- `StreamRecords(...)` yields handle-free record snapshots — the safe full-scan extractor (constant memory, page-released). Ideal for CSV/data.
- `Take(n)`, `Page()`, and slicing are explicit bounded reads; `Materialize()` pulls everything (throwing `BoundExceededException` past the cap).

5.4. The growth guard (loading and recording)

A trace loads asynchronously and a live capture grows the Overview, so a full traversal that snapshotted the count once would stop short and silently drop later items. So the full traversals throw

`OverviewIncompleteException` while a trace is loading or recording — unless you opt in with `follow: true` (tail items as they arrive) or `snapshot: true` (only what exists now). Bounded reads (`Take/Page/slice`) are exempt.

5.5. Product facets

Product-specific operations are grouped on facets, reached as properties on the analyzer. They attach lazily and throw `NotAvailableException` if the connected analyzer is not that product:

```
var summaries = analyzer.Bluetooth.ChannelSummaries();
analyzer.Wifi.AddWifiKeyByApSsid("HomeNet", "secret");
analyzer.Wpan.AddZigbeeNetworkKey(0x1234, key16);
analyzer.Usb30.ConnectLink();
```


5.6. Typed exports

Export is one server operation selected by a mode; the SDK gives you typed wrappers and typed option carriers. Output paths are on the **analyzer machine**.

```
analyzer.ExportFilteredTraceTimeRange(@"C:\out\first5s.btt",
    maxDurationPs:
    ExportOptions.PicosecondsFromTimeSpan(System.TimeSpan.FromSeconds(5)));
analyzer.ExportThroughput(@"C:\out\tput.csv");

analyzer.Bluetooth.ExportAudio(@"C:\out\audio",
    new BluetoothAudioOptions { PacketLoss = BluetoothPacketLossMode.SilenceSmooth });
```

5.7. Errors

All SDK errors derive from `RemoteControlException`:

- `ConnectionFailedException` — transport/connection problems (or the endpoint is not an analyzer).
- `OperationException` — the analyzer rejected the operation; subclasses include `LoadTimeoutException`, `ScopeClosedException`, `BoundExceededException`, `OverviewIncompleteException`.
- `NotAvailableException` — a facet/feature is not present on this analyzer.
- `ExportOptionException` (an `ArgumentException`) — a bad export option, caught client-side.

6. How-to recipes

6.1. Extract a whole trace to CSV (constant memory)

```
using var overview = analyzer.Overview("BR/EDR Overview");
using var writer = new System.IO.StreamWriter("out.csv");
foreach (var rec in overview.Root.Cursor().StreamRecords(new[] { ItemField.Time,
    ItemField.Description }))
    writer.WriteLine($"{rec.TimeSeconds},{rec.Description}");
```

6.2. Wait for a condition during a live capture

```
using (analyzer.Recording(@"C:\traces\live.btt")) // saved when the block exits
using (var overview = analyzer.Overview("BR/EDR Overview"))
{
    foreach (var item in overview.Root.Cursor().Stream(follow: true))
        if (item.Description.Contains("Disconnect"))
            break; // got it; stop recording
}
```

6.3. Annotate items found by a search

```
using var overview = analyzer.Overview("BR/EDR Overview");
foreach (var hit in overview.SearchCursor(description: new[] { "*Error*" }).Stream())
    hit.AddMarker("error here", MarkerColor.Red);
```

6.4. Build a table (column projection)

```
using var overview = analyzer.Overview("BR/EDR Overview");
var rows = overview.Root.Cursor().ToTable(new[] { "Status", "Opcode" }, cap: 5000); //
raises past cap
foreach (var row in rows)
    System.Console.WriteLine($"{row["Status"]}, {row["Opcode"]}");
```

6.5. Preview the wire options without exporting

```
foreach (var (name, value) in analyzer.ExportDryRun(
    @"C:\out\x", ExportModes.Throughput, new ThroughputOptions {
RangeStartTimePs = 0 }))
    System.Console.WriteLine($"{name} = {value}");
```

6.6. Survey an unknown trace

Protocol layers are per-protocol *views* of an overview; switching is instant (no re-processing). Counting root items across overviews and layers fingerprints what traffic a trace contains:

```
using var analyzer = Analyzer.Connect();
foreach (var name in analyzer.Overviews())
{
    using var overview = analyzer.Overview(name);
    if (overview.Root.ChildCount == 0)
    {
        continue; // no traffic in this overview
    }
    Console.WriteLine($"{name}: {overview.Root.ChildCount} items");
    var original = overview.ProtocolLayer;
    foreach (var layer in overview.AvailableProtocolLayers())
    {
        overview.SetProtocolLayer(layer); // instant; invalidates handles
        Console.WriteLine($"    {layer}: {overview.Root.ChildCount}");
    }
    overview.SetProtocolLayer(original);
}
```

7. Overriding Connection Properties

7.1. By file

ZeroC ICE server offers plenty of parameters to optimize different applications. It is possible to override the default properties by placing a file named `Ice.props` in the `RemoteControl` folder. The file shall be of this format: <https://doc.zeroc.com/display/Ice35/Configuration+File+Syntax>

7.2. By command line

It is also possible to override properties with the command line. Any ICE property can be set on the command line, such as `/Ice.ThreadPool.Server.SizeMax=8`.

7.3. Priority order

The priority is given as follow:

1. Defaults in the application
2. File Ice.props
3. Command line

8. Running the provided samples

8.1. C# / .net sample

1. The Samples folder contains a solution compatible with Visual Studio 2017 or higher.
2. Follow these steps to run the sample:
 - a. Start the analysis software previously configured in the above section. Please ensure an analyzer unit is connected to this computer.
 - b. Compile and execute the provided sample from Visual Studio.
 - c. The analysis software should start recording and will wait for a key before stopping the recording.
3. The provided sample can be used as starting point for custom control software.

8.2. Python sample

1. Install the latest Python environment from <https://www.python.org/downloads/>
2. Install the sample's dependencies with the command: `python -m pip install -r requirements.txt`
3. Follow these steps to run the sample:
 - a. Start the analysis software previously configured in the above section. Please ensure an analyzer unit is connected to this computer.
 - b. Run the sample with the command: `python sample.py`
 - c. The analysis software should start recording and will wait for a key before stopping the recording.
4. The provided sample can be used as starting point for custom control software.

8.3. Using a different language or platform

The sample is provided with language-agnostic definition of the Ellisys API. The ICE files can be converted to the various supporting languages such as Ruby, JavaScript, and more, with a tool named slice provided by ZeroC. Please consult ZeroC documentation for creating a wrapper for your language and platform of choice: <https://doc.zeroc.com/ice/latest/the-slice-language/slice-compilation>

9. Overview queries

A query filters the overview to the lines matching criteria on **any column of the overview or any field of the Details view** — the filtering runs inside the analyzer, so it is the most efficient way to narrow millions of items down to the lines of interest before traversing them. Queries are the same feature as the application's query toolbar; through the API they are set with `SetOverviewQuery` (SDKs: Python `overview.set_query(...)`, C# `overview.SetQuery(...)`).

9.1. Syntax

```
Field or column name = [!]value[, value, ...], Another field = [!]value[, ...]
```

- A **term** is `Field = value[, value, ...]`. Values separated by commas are alternatives: the field must match **at least one** of them. An exclamation mark before a value creates a NOT condition. Terms separated by commas must all match; terms can also be put in parentheses and combined with `&&` (and) / `||` (or) instead.
- **Comparators**: `=`, `!=`, `<`, `>`, `<=`, `>=`.
- **Field and column names** are written as displayed, unquoted — multi-word names work (`RF Channel Number >= 40`). Fields that appear several times can be indexed: `Data[1]` is the second `Data` field.
- **Values**:
 - Text in double quotes; matches any run of characters: `"Text"`, `"error"`.
 - Numbers: decimal `123`, hex `0xABCD`, binary `0b010101`; inclusive ranges `7..10`; computations using operators and other fields (`4+5`, `0x0F << 2`).
 - Data patterns: `0x[A1 B2 C3]□□` bytes in hex, `##` matches any single byte, a trailing `*` matches any remaining bytes. Use a leaf data field (e.g. `Raw Data`), not a grouping field.
 - Regular expressions: `Regex("^[hc]at")`.
- **Functions**: `ByteAt(Field, n)` — the field's `n`-th byte (0-based) as a number.

9.2. Examples

Query	Matches lines where
<code>Item = "Text"</code>	the Item column starts with <code>Text</code>
<code>Item = !"Text"</code>	the Item column does not contain <code>Text</code>
<code>Status = "OK"</code>	the Status column is exactly <code>OK</code>
<code>Foo = 1, 0x03, 7..10</code>	<code>Foo</code> is 1, 3, 7, 8, 9 or 10
<code>Foo >= 0x0F << 2, 4+5 && (Bar = 1 Status != "OK")</code>	<code>Foo</code> is at least 9, and either <code>Bar</code> is 1 or <code>Status</code> is not <code>OK</code>
<code>Item = "AVDTP*" && Status = "OK"</code>	the Item starts with <code>AVDTP</code> and the Status is <code>OK</code>
<code>Payload = 0x[0A ## 0C *]</code>	the Payload's 1st byte is <code>0x0A</code> and its 3rd byte is <code>0x0C</code>
<code>ByteAt(Data[1], 3) = 0</code>	the second Data field's 4th byte is 0
<code>Item = Regex("^[hc]at")</code>	the Item starts with <code>hat</code> or <code>cat</code>

9.3. Behavior notes (important for automation)

- A line matches when any of its field occurrences match. Overview lines summarize the items beneath them, so a field can occur several times per line with different values (a line spanning several packets has several `RF Channel Number` values, and matches both `= 0..39` and `>= 40` if its packets span both ranges).
- The query is applied asynchronously. `SetOverviewQuery` returns at once and the overview re-filters in the background, reported as a running task (e.g. "Filtering data" in `GetRunningTasks` — which can take a moment to appear) while `IsLoading` stays false; existing item handles are invalidated, and counts read in the meantime may reflect the previous state. Raw-API users should poll `GetRunningTasks` until the analyzer

stays idle. The SDKs handle this: `set_query / SetQuery` block until the re-filtering finishes by default (pass `wait=False / wait: false` to return immediately and settle later with `wait_until_idle() / WaitUntilIdle()`).

- **Syntax errors are reported, unknown names are not.** A malformed query (e.g. unbalanced parentheses, missing value) fails the call with `Invalid query`; a misspelled field or column name is accepted and simply matches **nothing** (0 lines). If a query unexpectedly returns 0, verify the field name against the overview columns / Details view of a matching item first.
- An empty query ("") clears the filter and restores the full overview.

10. Using with AI agents

The Remote Control distribution is designed to be driven by AI coding agents as well as by humans. The release archive includes three layers of agent-oriented material:

- **Archive index** — `llms.txt` at the archive root: a Markdown link list describing what the archive contains and where everything is. An agent pointed at the unzipped archive discovers the rest from there.
- **SDK agent guides** — `Sdk/Analysis/Python/ellisys_analysis_automation/llms.txt` and `Sdk/Analysis/CSharp/Ellisys.Analysis.Automation/llms.txt`: dense, authoritative API guides with the rules the bounded API enforces and canonical code recipes. They are bundled inside the SDK packages, so they are also available wherever the SDK is installed.
- **Agent skill** — `Skill/ellisys-analysis/`: a ready-to-install skill for [Claude Code](#) packaging the complete workflow: `SKILL.md` (the mental model, a starter recipe and a troubleshooting table), `scripts/probe.py` (connectivity diagnosis with actionable hints) and `scripts/setup_env.py` (environment bootstrap).

10.1. Installing the agent skill

To enable the skill for a given project, copy the `Skill/ellisys-analysis` folder into the project's `.claude/skills/` directory:

```
<project>/
  .claude/
    skills/
      ellisys-analysis/
        SKILL.md
        scripts/           (probe.py, setup_env.py, explore.py)
        sdk/python/        (the bundled Python SDK source)
        reference/         (the Python SDK guide, Markdown)
```

To enable it for all projects of the current user instead, copy the same folder to `~/.claude/skills/` (`%USERPROFILE%\ .claude\skills\` on Windows). Claude Code activates the skill automatically whenever a task involves controlling an Ellisys analyzer; no further configuration is needed.

The skill folder is **self-contained**: it bundles the Python SDK source (`sdk/python/`) and the SDK guide (`reference/`), so it keeps working when copied on its own, away from this archive. Network access to PyPI is only needed for the `zeroc-ice` dependency at environment setup.

10.2. The MCP server

For AI hosts without shell access — chat applications such as Claude Desktop — Ellisys provides an MCP (Model Context Protocol) server, `ellisys-analysis-mcp`, distributed on PyPI. It exposes four tools: `analysis_guide` (the SDK agent guide and canonical recipes), `analysis_status` (connectivity diagnosis, application/trace state, markers), `analysis_explore` (the discovery pass: overview/protocol-layer survey, vocabulary, devices, anchors) and `analysis_run_python` (a persistent Python session with the SDK and a connected `analyzer` preloaded — all actions and analysis happen here). The session includes three helpers: `install("pkg", ...)` adds extra libraries (pandas, matplotlib, ...) on demand, `show(figure)` returns a matplotlib figure to the chat as an image, and `attach(path)` returns a small file (e.g. a CSV export; up to 4 MB) with the result. To have matplotlib and pandas preinstalled instead, configure the package with the `analysis` extra: `uvx "ellisys-analysis-mcp[analysis]@latest"`.

The only prerequisite is `uv`, a single small executable that provisions Python and the packages automatically — no Python installation is required. Install it once:

- **Windows:** `winget install astral-sh.uv`
- **macOS:** `brew install uv`
- **Linux (and macOS without Homebrew):** `curl -LsSf https://astral.sh/uv/install.sh | sh`

Then verify with `uvx --version` in a **new** terminal — the installer updates the PATH for new processes only — and (re)start the MCP host application afterwards so it sees the updated PATH.

The analyzer application must be running with the Remote Control plugin enabled; the server connects to `localhost:12345` by default (override with `--host/--port` arguments or the `ELLISYS_HOST/ELLISYS_PORT` environment variables).

Claude Code (terminal):

```
claude mcp add ellisys-analysis -- uvx ellisys-analysis-mcp@latest
```

Claude Desktop — add to `claude_desktop_config.json` (Settings > Developer > Edit Config):

```
{
  "mcpServers": {
    "ellisys-analysis": { "command": "uvx", "args": ["ellisys-analysis-mcp@latest"] }
  }
}
```

The `@latest` suffix makes `uvx` check for the newest release every time the server starts, so updating is just a restart of the host application. Without a suffix, `uvx` keeps reusing the first version it resolved until its cache is cleared manually (`uv cache clean ellisys-analysis-mcp`); a fixed version can be pinned with `ellisys-analysis-mcp@0.1.1`.



If the host reports the server as **failed** (often error `-32000`), the usual cause is that `uvx` is not on the PATH of the host process: verify `uvx --version` works in a **new** terminal, and restart the host application after installing `uv` (it inherits the PATH from its launch). Alternatively, configure the full path to `uvx` instead of the bare command.

Other MCP hosts (Cursor, VS Code, ChatGPT desktop, ...): register a custom/local MCP server with the command `uvx` and the argument `ellisys-analysis-mcp@latest` in the host's MCP configuration. Hosts that only accept **remote** (HTTP) MCP servers are not yet supported (a streamable-HTTP transport is planned).

10.3. Choosing between the skill and the MCP server

Both deliver the same capability — they share the SDK, its agent guide and its discovery module — so the choice is purely about the host:

- **Use the skill** with coding agents that have shell and filesystem access (Claude Code and similar). The agent works in its own environment: it can install extra packages alongside the SDK (pandas, matplotlib, ...), produce files in your project (CSV analyses, plots, reusable scripts) and integrate the analyzer into larger codebases and CI. This is the most capable option for engineering work.
- **Use the MCP server** with chat applications and other hosts that cannot run scripts (Claude Desktop, ChatGPT desktop, ...). Code runs inside the server's session and results come back as text — ideal for interactive exploration and Q&A about a trace.
- Hosts that support both (e.g. Claude Code) can use either; the skill offers more headroom there.

The skill is plain Markdown plus two small Python scripts, so agents other than Claude Code can use `SKILL.md` directly as instructions; the scripts run with any Python 3.10+ on Windows, Linux or macOS.



The agent (the client side) can run on any platform, and the analyzer application runs on Windows natively and on Linux and macOS through the Ellisys-provided runtime — the whole chain is multiplatform. File paths passed to load, save and export operations are resolved by the analyzer application on its machine and must be valid for that installation.

11. API reference

The reference below is generated from the SDK's XML doc comments; edit those and run `python make.py build` to regenerate.

11.1. Connecting and the Analyzer

11.1.1. Analyzer

A connected Ellisys analyzer. Obtain one with `Connect` and dispose it (a `using` block) to disconnect. Every operation is synchronous.

This hides all ZeroC Ice setup (communicator, proxy, checked cast) and translates Ice exceptions into the `RemoteControlException` hierarchy. No ZeroC Ice type appears on the public surface.

```
static Analyzer Connect(string host = "localhost", int port = 12345, int messageSizeMaxMb = 256)
```

Connect to a running Ellisys analyzer that has Remote Control enabled.

Parameters:

- `host`: Analyzer host name or IP.

- `port`: Remote Control TCP port.
- `messageSizeMaxMb`: Raises Ice's max message size so large Overview reads succeed.

void Dispose()

Disconnect from the analyzer and release the Ice communicator.

11.2. Overview navigation

11.2.1. ScopeClosedException

A handle was used after its `OverviewScope` released it.

Thrown when an item/page is touched after its scope exited, after `ReleaseHandles`, or after a page-streaming cursor released the page that minted it. Subclasses `OperationException`, so existing `catch (OperationException)` handlers still catch it.

ScopeClosedException(string message, Exception? innerException = null)

Create a scope-closed failure.

11.2.2. BoundExceededException

A bounded read was asked to materialize more than its cap.

Thrown by `Take(n) / cursor[a..b] / Materialize(cap:) / search resultCap` when the requested or discovered size would exceed the limit, and by constructing a `Page` larger than `HardCap`. Fail-loud by design: there is no silent truncation.

BoundExceededException(string message, Exception? innerException = null)

Create a bound-exceeded failure.

11.2.3. OverviewIncompleteException

A full traversal was attempted while the overview is still being populated.

The overview grows while a trace is `loading` or while the analyzer is `recording`, so a plain stream / record-stream / walk / iterate-children / materialize would stop at whatever count happened to exist when it started and silently drop the rest. Rather than truncate quietly, those operations throw this unless the caller says how to handle the growth: wait until it is complete first (`WaitUntilLoaded`), pass `follow:true` to keep traversing as items arrive (live-tail), or pass `snapshot:true` to deliberately traverse only what exists right now. Bounded reads (`Take(n) / Page() / cursor[a..b]`) are exempt — they are explicitly partial by request, so nothing is lost silently. Subclasses `OperationException`.

OverviewIncompleteException(string message, Exception? innerException = null)

Create an overview-incomplete failure.

11.2.4. OverviewLimits

The structural caps that make bounded access a guarantee rather than advice.

const int PageDefault = 256

Per-round-trip page for cursors/streams.

const int PageMax = 4096

Hard ceiling on any single page / batch round-trip (clamps caller-supplied page sizes).

const int HardCap = 100_000

Ceiling on an explicit materialize (Take / finite slice / Materialize). Exceeding it throws.

const int DefaultSearchDepth = 8

Default recursion depth for a search cursor (finite on purpose — NOT the whole subtree).

const int DefaultSearchFanout = 16

How many of the base's direct children a search window covers per round-trip.

const int DefaultResultCap = 100_000

Default ceiling on total search matches before a search throws.

static readonly TimeSpan FollowPoll = TimeSpan.FromMilliseconds(250)

Default poll interval for `follow` traversals while the overview is still growing.

11.2.5. ItemField (enum)

Selects which batch operation a reader issues. One member == one plural Ice op.

- `Description` — Text of the Item column (-> `GetOverviewItemsDescription`).
- `Time` — Timestamp in picoseconds (-> `GetOverviewItemsTimeInPicoseconds`).
- `Data` — Bytes shown in the Raw data view (-> `GetOverviewItemsData`).
- `Xml` — Details view as XML (-> `GetOverviewItemsXmlReport`).

11.2.6. XmlFilter

A request for the filtered Details XML of the given field names (-> `...XmlReportFiltered`).

IReadOnlyList<string> FieldNames

The Details field names to include in the filtered XML report.

11.2.7. ItemRecord

A pure-data, handle-FREE snapshot of one item — safe to keep after the scope closes.

Only the fields you requested are populated; the rest stay `null`. `Position` is the item's index within its parent / result set (for traceability) — it is **not** a handle.

ItemRecord(int position, string? description = null, long? timePs = null,

Create a handle-free snapshot. Unset fields stay `null`.

int Position

Index of this item within its parent / result set (traceability only; not a handle).

string? Description

The Item-column text, or `null` if not requested.

long? TimePs

The timestamp in picoseconds, or `null` if not requested.

byte[]? Data

The raw-data-view bytes, or `null` if not requested.

string? Xml

The Details XML report, or `null` if not requested.

TimeSpan? Time

`TimePs` as a `TimeSpan` (100-nanosecond resolution), or `null`.

double? TimeSeconds

`TimePs` in seconds as a `double`, or `null`.

11.2.8. ReleaseMode (enum)

Whether a page-streaming cursor releases each page's handles at the page boundary.

- `Page` — Free each page's handles at the page boundary (default) — a full scan keeps server memory at $O(\text{pageSize})$, at the cost that a live item must not be held across a page boundary (touching it throws `ScopeClosedException`).
- `Defer` — Retain handles (e.g. while holding live items or running concurrent cursors).

11.2.9. OverviewItem

A lazy, scope-bound wrapper around one opaque handle.

Constructing one is free. Each single-item field property does one Ice call on first access then caches it. For many items, navigate with `Cursor` (bounded) or stream with `IterateChildren` / `Walk`. The internal index path lets a page-releasing cursor re-acquire the item after a global release; items whose path is unknown (search results, the selected item) cannot be re-seated.

int Position

Index of this item within its parent / result set.

int Depth

Depth relative to where a `Walk` started (0 for that level's children).

TimeSpan Time

Timestamp as a `TimeSpan` (100-nanosecond resolution).

string XmlReport()

Details view as XML (-> `GetOverviewItemXmlReport`, cached).

string XmlReport(params string[] fields)

Filtered Details view as XML for the given field names (-> `GetOverviewItemXmlReportFiltered`).

ChildCursor Cursor(int pageSize = OverviewLimits.PageDefault,

A bounded cursor over this item's children (replaces an unbounded children list).

The cursor yields one `Page` at a time. `pageSize` is clamped to `PageMax`. With `Page` (default) each page's handles are freed at the page boundary, so a full scan keeps server memory at $O(\text{pageSize})$ — at the cost that a live item must not be held across a page boundary (it throws `ScopeClosedException`; use `StreamRecords` to keep data). Use `Defer` to retain handles (e.g. while holding live items or running concurrent cursors).

OverviewItem Child(int index)

The single child at `index` (-> `GetOverviewItemChild`).

IEnumerable<OverviewItem> IterateChildren(int batchSize = OverviewLimits.PageDefault,

Lazily page through direct children (`ceil(N/batchSize)` calls, not `N`).

A true generator for bounded exploration: break early and only fetched pages cost round-trips. It does NOT release handles per page (it yields live items), so for a full multi-million-item scan prefer `StreamRecords`, which is page-released. While the overview is growing (loading/recording) this throws `OverviewIncompleteException` unless you pass `follow` (live-tail the children as they arrive) or `snapshot` (only the children that exist now).

IEnumerable<OverviewItem> Walk(int? maxDepth = null,

Lazy depth-first walk of descendants; sets each item's `Depth`.

Yields this item's children at depth 0, their children at depth 1, etc. `maxDepth` limits the levels (`null` = unbounded). Built on `IterateChildren` (no per-page release, since it holds ancestors while descending) — intended for bounded exploration with an early break, not for materializing a whole huge tree. On a growing overview this throws `OverviewIncompleteException` unless you pass `snapshot` (walk what exists now) or `follow` (wait until loading / recording stops, then walk the complete tree — live-tailing a recursive walk is not well-defined, so `follow` here means "let it finish first").

void Select()

Highlight this item in the analyzer GUI (-> `SelectOverviewItem`).

void AddMarker(string text, MarkerColor color = MarkerColor.Yellow)

Select this item and drop a marker on it (-> `SelectOverviewItem` + `AddMarkerOnSelectedOverviewItem`).

ItemRecord Record(params ItemField[] fields)

Snapshot this item to a handle-free `ItemRecord` (single-item ops).

ItemRecord Record(XmlFilter xmlFilter, params ItemField[] fields)

Snapshot this item including a filtered-XML column (single-item ops).

11.2.10. OverviewDefaults

Fields a page prefetches on construction (the two columns analysts read most).

static readonly IReadOnlyList<ItemField> Prefetch =

The default prefetch columns (Description + Time). Override per call.

11.2.11. OverviewItemList

An ordered, scope-bound, batch-backed sequence of `OverviewItem` (base of `Page`).

Not constructed directly by users — a bounded `Page` is what cursors hand out. Count / indexing / slicing / iteration are pure-local; each batch reader is one bounded round-trip over the list's handles. Construction is capped at `HardCap`.

int Count

Number of items in this list (pure-local; no round-trip).

IEnumerator<OverviewItem> GetEnumerator()

ReadOnlyList<string> Descriptions()

Item-column text for every item (-> GetOverviewItemsDescription).

ReadOnlyList<long> TimesPs()

Timestamps in picoseconds (-> GetOverviewItemsTimeInPicoseconds).

ReadOnlyList<TimeSpan> Times()

Timestamps as `TimeSpan`s (100-nanosecond resolution).

ReadOnlyList<byte[]> Data()

Raw-data-view bytes for every item (-> GetOverviewItemsData).

ReadOnlyList<string> XmlReports()

Details XML for every item (-> GetOverviewItemsXmlReport).

ReadOnlyList<string> XmlReports(params string[] fields)

Filtered Details XML for every item (-> GetOverviewItemsXmlReportFiltered).

ReadOnlyList<ItemRecord> Records(params ItemField[] fields)

Snapshot the whole list to handle-free records (one batch per requested field).

ReadOnlyList<ItemRecord> Records(XmlFilter xmlFilter, params ItemField[] fields)

Snapshot the whole list including a filtered-XML column (one batch per field).

ReadOnlyList<ReadOnlyDictionary<string, string>> ToRows(params string[] fieldNames)

Filtered Details fields parsed into dictionaries (for tabular export).

OverviewItemList Filter(Func<OverviewItem, bool> predicate)

A new list of the items matching `predicate` (reuses the items).

11.2.12. Page

One bounded window of children/matches — the only object carrying whole-set readers.

A `Page` is an `OverviewItemList` (so it has the batch readers / slicing / filter) whose length is capped by construction (`<= PageMax` for a streamed page, `<= HardCap` for a Take/slice result). It also knows its `Position` in the parent and whether more follow (`HasMore`).

int Position

Index of this page's first item within the parent / result set.

int PageSize

The page size used to fetch this window.

int Total

Total number of children/matches in the parent / search space.

bool HasMore

Whether more items follow this page.

Page? NextPage()

Advance the owning cursor and return the next page (`null` at the end).

11.2.13. ChildCursor

A bounded, page-at-a-time cursor over one item's children. Built by `Cursor`.

Streaming (`Stream`, `StreamRecords`) and manual paging (`Page`, `NextPage`, `Seek`, `Reset`) page in windows of `pageSize` (clamped to `PageMax`). Materializing (`Take`, `cursor[a..b]`, `Materialize`) is bounded and throws `BoundExceededException` past `HardCap`. It is NOT iterable directly; `Total` is the cheap count, not a fetch of items.

int Position

Index of the next unread child.

Page Page(int? size = null)

The page at the current position (peek; does not advance). `size` clamped to `PageMax`.

Page? NextPage()

Advance past the current page (releasing it if release mode is `Page`); `null` at end.

void Reset()

Release any live page and seek back to the start.

void Seek(int index)

Release any live page and jump the position to `index`.

IEnumerable<OverviewItem> Stream(IReadOnlyList<ItemField>? prefetch = null,

Yield children one at a time, paging under the hood (batched, bounded, page-released).

The obvious loop on a 10M-child node: $O(\text{pageSize})$ handles in flight, ~3 ops/page, an early break stops fetching. With `Page` an item must be consumed within its page — don't stash live items across iterations (use

`StreamRecords`). While the overview is still growing this throws `OverviewIncompleteException` unless you pass `follow` (re-read the count at the end and keep yielding new items as they arrive) or `snapshot` (traverse only what exists now).

`IEnumerable<ItemRecord> StreamRecords(IReadOnlyList<ItemField>? fields = null,`

Yield handle-free `ItemRecord`'s, page by page, releasing each page.

The safe full-scan extractor: constant client memory, $O(\text{pageSize})$ server handles, batched (one round-trip per field per page). Records survive the per-page release. Defaults to description + time. `follow` / `snapshot` behave as in `Stream` (a growing overview otherwise throws `OverviewIncompleteException`).

`Page Take(int n)`

A `Page` of the first `min(n, total)` children. Throws if `n > HardCap`.

`IReadOnlyList<ItemRecord> Materialize(int cap = OverviewLimits.HardCap,`

All children as handle-free records (page-released). Throws if `total > cap`.

On a growing overview this throws `OverviewIncompleteException` unless you pass `snapshot` (materialize what exists now) or `follow` (wait until loading / recording stops, then materialize the complete set — so do not follow an open-ended live recording here; use `Stream` and break for that).

`IReadOnlyList<IReadOnlyDictionary<string, string>> ToTable(string[] fieldNames,`

All children's filtered Details fields as dictionaries (for tabular export). Throws if over `cap`.

11.2.14. SearchCursor

A bounded cursor over search matches. Built by `SearchCursor`.

`SearchOverviewItems` has no result paging, so this bounds the search SPACE: it windows the base's DIRECT children in steps of `searchFanout`, issuing one search per window and (by default) releasing each window's match handles before the next. `maxDepth` is finite by default; `resultCap` throws `BoundExceededException` if total matches exceed it. `Scanned` / `Found` expose progress. Residual risk (inherent to the server): a single windowed child whose subtree alone holds millions of matches still returns them in one array; `maxDepth` and `resultCap` cap the damage to a loud throw rather than an OOM, but cannot subdivide one subtree.

`int Scanned`

Direct children of the base searched so far.

`int Found`

Matches yielded so far.

`Page? NextWindow()`

The next window's matches as a `Page` (`null` when the search space is exhausted).

IEnumerable<OverviewItem> Stream(bool snapshot = false)

Yield matches one at a time, windowing the search space (consume within each window). Throws `OverviewIncompleteException` while the overview is loading or recording (the searched child count is captured once, so a live search would silently miss late arrivals); pass `snapshot = true` to search just what exists now.

void Reset()

Release matches and restart the search from the first window.

Page Take(int n)

A Page of the first `min(n, matches)` matches. Throws if `n > HardCap`.

IReadOnlyList<ItemRecord> Materialize(int cap = OverviewLimits.HardCap,

All matches as handle-free records (window-released). Throws if matches exceed `cap`, or `OverviewIncompleteException` while loading/recording (pass `snapshot = true` to materialize just what exists now).

11.2.15. OverviewScope

Owns one server-side handle session for one active overview.

Constructed only by `Overview` — never directly. It is the single place that calls

`ReleaseAllOverviewItemHandles` (on dispose, via `ReleaseHandles`, or at a page-streaming cursor's page boundaries). Every item / page / cursor it mints checks the scope is live before any network call. Dispose it (a `using` block) to release its handles and, unless requested otherwise, re-select the previously active overview.

string? Name

The overview this scope is bound to (`null` for the already-active one).

bool Closed

`true` once the scope has been disposed.

string Query

The active overview's filter query (-> `GetOverviewQuery`).

void SetQuery(string query, bool wait = true, TimeSpan? timeout = null)

Apply a filter query (-> `SetOverviewQuery`). Invalidates existing handles.

Queries filter the overview, inside the analyzer, on any overview column or Details field — e.g. `Item = "AVDTP*" && Status != "OK"`. Comparators = `!= < > <= >=`; comma-separated alternative values; `!` before a value for NOT; `&&/||` and parentheses; numbers (123, 0xABCD, 0b0101, inclusive ranges 7..10); data patterns `0x[A1] (` (= any byte, trailing = any rest); `Regex("...")`; `ByteAt(Field, n)`. The analyzer re-filters in the background (a "Filtering data" running task; `IsLoading` stays `false`): by default this call BLOCKS until that

finishes (`wait: false` returns immediately; settle later via `WaitUntilIdle`). A malformed query throws `OperationException ("Invalid query")`; an unknown field name silently matches nothing. `""` clears the filter. See the SDK guide's "Overview queries" section for the full syntax.

string ProtocolLayer

The active overview's protocol layer (-> `GetSelectedProtocolLayer`).

void SetProtocolLayer(string layer)

Switch the protocol layer (-> `SelectProtocolLayer`). Invalidates existing handles.

Protocol layers are per-protocol views of the overview; switching is instant (no background re-processing, unlike queries/device filters). Iterating `AvailableProtocolLayers` and reading `Root.ChildCount` per layer is the quickest way to fingerprint what traffic a trace contains.

ReadOnlyList<string> AvailableProtocolLayers()

Protocol layers selectable on the active overview (-> `GetAvailableProtocolLayers`).

SearchCursor SearchCursor(

A bounded cursor over matches under `within` (default: the scope root).

Filters are **glob patterns** (`*` / `?`) or a .NET regex prefixed `regex::`; a bare literal must match the *whole* value. `fieldName / fieldValue` filter the Details fields, pairing by index when both are given. `maxDepth` defaults to a finite `DefaultSearchDepth` (pass `null` for the whole subtree); `resultCap` bounds the total matches before `BoundExceededException`.

void Select(OverviewItem item)

Highlight `item` in the GUI (-> `SelectOverviewItem`).

void ReleaseHandles()

Free **all** item handles now without closing the scope (-> `ReleaseAllOverviewItemHandles`).

Global by nature — there is no per-handle release. Every live item/page minted in this scope throws `ScopeClosedException` on next use; already-extracted `ItemRecord`'s stay valid. This is also the primitive a page-streaming cursor uses at page boundaries.

void Dispose()

Close the scope: release all item handles once and (unless requested otherwise) re-select the previously active overview (restored on `Dispose`).

11.2.16. Analyzer

Analyzer-level overview entry points (reopens the foundation class).

IReadOnlyList<string> Overviews()

Names of the available overviews (-> GetAvailableOverviews). No scope needed.

string? ActiveOverview()

Name of the currently selected overview, or `null` (-> GetSelectedOverview).

OverviewScope Overview(string? name = null, string? query = null,

Open a navigation scope on `name` (or the already-active overview if `null`).

Mirrors Recording: on entry it snapshots the active overview, selects `name`, and optionally applies `query` / `protocolLayer`; it returns an `OverviewScope` (dispose it, e.g. with a `using` block, to close it). On disposal it frees all item handles once (`ReleaseAllOverviewItemHandles`) and, unless `restoreActive` is `false`, re-selects the previously active overview. Only one scope may be open at a time — handle release is global, so a nested scope would free this one's handles. Opening a second throws `OperationException`.

11.3. Product facets (shared)**11.3.1. ProductFacet**

Shared plumbing for the product facets: lazily attaches the product proxy and maps a missing/wrong-product interface to `NotAvailableException`.

11.3.2. DeviceFilterMode (enum)

Which devices a capture keeps or excludes (→ `ConfigureDeviceFilter`). Shared by the Bluetooth and Wi-Fi device filters.

- `KeepAll` — Keep traffic from every device.
- `ExcludeBackground` — Exclude background/uninvolved devices.
- `KeepOnly` — Keep only the listed devices.
- `KeepInvolving` — Keep traffic involving the listed devices.

11.4. Bluetooth facet**11.4.1. BluetoothChannelSummary**

Per-RF-channel packet tallies (→ Slice struct `ChannelSummary`).

long Errors

Header + payload errors.

long Total

Classified packets on this channel (excludes `NotApplicable`).

double? ErrorRate

Fraction of classified packets in error, or `null` if the channel had no data.

11.4.2. BluetoothChannelStat

An (RF channel, summary) pair — the explicit-index view of channel summaries.

11.4.3. BluetoothSpectrumSample

One RSSI sample (→ `GetSpectrumRssi`).

TimeSpan Window

Length of the sampling window as a `TimeSpan`.

11.4.4. BluetoothSpectrumRange

A series of RSSI samples over a window (→ `GetSpectrumRssiRange`).

int Count

Number of samples.

TimeSpan Window

Length of the sampling window as a `TimeSpan`.

double? MeanDbm

Mean RSSI across the samples in dBm, or `null` if there are none.

double? MinDbm

Lowest RSSI sample in dBm, or `null` if there are none.

double? MaxDbm

Highest RSSI sample in dBm, or `null` if there are none.

11.4.5. BluetoothBdAddr

Helper to format a 48-bit BDADDR.

static string Format(long address)

Format a 48-bit BDADDR as "AA:BB:CC:DD:EE:FF".

11.4.6. BluetoothFacet

Bluetooth-only operations on a connected analyzer. Obtain via `analyzer.Bluetooth`.

void SplitTraceAndContinue(string filename)

Save the current trace and keep recording — "Save & Continue".

BluetoothSpectrumSample SpectrumRssi(long timePs, int rfChannel)

RSSI at `timePs` on `rfChannel` (0-78).

BluetoothSpectrumRange SpectrumRssiRange(long fromTimePs, long toTimePs, int rfChannel)

RSSI samples across `[fromTimePs, toTimePs]` on a channel.

IReadOnlyList<BluetoothChannelSummary> ChannelSummaries(long fromTimePs = 0, long? toTimePs = null)

Per-RF-channel summaries; list index == RF channel. Defaults to the whole trace.

IReadOnlyList<BluetoothChannelStat> Channels(long fromTimePs = 0, long? toTimePs = null)

Per-channel summaries as explicit (RF channel, summary) pairs.

void ConfigureDeviceFilter(DeviceFilterMode mode, IReadOnlyList<long>? deviceAddresses = null,

Set which devices are kept/excluded.

void ConfigureDeviceFilter(DeviceFilterMode mode, IReadOnlyList<string> deviceAddresses,

Set which devices are kept/excluded, with addresses as the analyzer displays them (e.g. from the Overview's Communication column: "AA:BB:CC:DD:EE:FF").

void AddLinkKey(long bdAddr1, long bdAddr2, byte[] linkKey)

Add a 16-byte link key for the `bdAddr1` ↔ `bdAddr2` link.

void AddLinkKey(string bdAddr1, string bdAddr2, byte[] linkKey)

Add a 16-byte link key, with the BDADDRs as the analyzer displays them ("AA:BB:CC:DD:EE:FF").

void ExportAudio(string output, BluetoothAudioOptions? options = null)

Export Bluetooth audio (mode `bluetooth_audio`; the supported audio export).

void ExportChannels(string output, BluetoothChannelsOptions? options = null)

Export per-channel statistics (mode `bluetooth_channels`).

void ExportAirtime(string output, BluetoothAirtimeOptions? options = null)

Export airtime usage (mode `airtime`).

void ExportMobilePhoneData(string output)

Export Bluetooth mobile-phone data (mode `bluetooth_mobile_phone_data`).

11.4.7. Analyzer

BluetoothFacet Bluetooth

Bluetooth-only operations. Throws `NotAvailableException` if absent.

11.5. Wi-Fi facet

11.5.1. WifiFacet

Wi-Fi-only operations on a connected analyzer. Obtain via `analyzer.Wifi`.

void AddWifiKeyByApSsid(string apSsid, string key)

Add a Wi-Fi key for an access point identified by SSID.

void AddWifiKeyByApMac(long apMacAddress, string key)

Add a Wi-Fi key for an access point identified by its 48-bit MAC address.

void AddWifiKeyByApMac(string apMacAddress, string key)

Add a Wi-Fi key for an access point, with the MAC as the analyzer displays it ("AA:BB:CC:DD:EE:FF").

void ConfigureDeviceFilter(DeviceFilterMode mode, IReadOnlyList<long>? deviceAddresses = null,

Set which devices are kept/excluded.

void ConfigureDeviceFilter(DeviceFilterMode mode, IReadOnlyList<string> deviceAddresses,

Set which devices are kept/excluded, with addresses as the analyzer displays them ("AA:BB:CC:DD:EE:FF").

11.5.2. Analyzer

WifiFacet Wifi

Wi-Fi-only operations. Throws `NotAvailableException` if absent.

11.6. WPAN / 802.15.4 facet

11.6.1. WpanFacet

WPAN / 802.15.4-only operations on a connected analyzer. Obtain via `analyzer.Wpan`.

void AddThreadMasterKey(int panId, byte[] key, int sequenceCounter)

Add a Thread network master key for a PAN.

void RemoveAllThreadMasterKeys()

Remove every Thread network master key.

void RemoveThreadMasterKeys(int panId)

Remove all Thread master keys for a PAN.

void RemoveThreadMasterKey(int panId, byte[] key)

Remove a specific Thread master key.

void AddZigbeeApsKey(byte[] destinationAddress64, byte[] sourceAddress64, byte[] key)

Add a Zigbee APS link key between two 64-bit addresses.

void RemoveAllZigbeeApsKeys()

Remove every Zigbee APS key.

void RemoveZigbeeApsKeys(byte[] destinationAddress64, byte[] sourceAddress64)

Remove all Zigbee APS keys between two 64-bit addresses.

void RemoveZigbeeApsKey(byte[] destinationAddress64, byte[] sourceAddress64, byte[] key)

Remove a specific Zigbee APS key.

void AddZigbeeNetworkKey(int panId, byte[] key)

Add a Zigbee network key for a PAN.

void RemoveAllZigbeeNetworkKeys()

Remove every Zigbee network key.

void RemoveZigbeeNetworkKeys(int panId)

Remove all Zigbee network keys for a PAN.

void RemoveZigbeeNetworkKey(int panId, byte[] key)

Remove a specific Zigbee network key.

11.6.2. Analyzer

WpanFacet Wpan

WPAN-only operations. Throws `NotAvailableException` if absent.

11.7. USB 3.0 facet

11.7.1. Usb30Facet

USB 3.0-only operations on a connected analyzer. Obtain via `analyzer.Usb30`.

void ConnectLink(bool disconnectSuperSpeed = false)

Connect the analyzer link.

void DisconnectLink()

Disconnect the analyzer link.

void SaveUsb20Packets(string filename)

Save captured USB 2.0 packets to a file (on the analyzer machine).

void SaveUsb30Symbols(string filename, bool upstreamSymbols)

Save captured USB 3.0 symbols to a file (on the analyzer machine).

11.7.2. Analyzer

Usb30Facet Usb30

USB 3.0-only operations. Throws `NotAvailableException` if absent.

11.8. Exports

11.8.1. ExportOptionException

A client-side export-option problem (a bad value type, or a failed validation such as a range whose end precedes its start).

Distinct from `OperationException` (a server-side failure, e.g. a wrong-product mode) and `ConnectionFailedException`; derives from `ArgumentException` so it reads as a bad-argument error to callers.

11.8.2. ExportMode

An export mode: the exact `exportName` wire token plus its `Product` and a `Description`. Named modes live in `ExportModes` (base) and each product's holder (e.g. `BluetoothExportModes`); a raw string can also be passed wherever a mode is expected, so modes newer than this SDK remain reachable. Equality is by token only.

string Value

The exact wire token (the server `exportName`).

string Product

The product that supports the mode ("base" or e.g. "bluetooth"; empty for a raw token).

string Description

A human-readable description (empty for a raw token).

static implicit operator ExportMode(string value)

Implicitly wrap any raw token, so a string can be passed wherever a mode is expected.

override string ToString()

The wire token.

bool Equals(ExportMode other)**override bool Equals(object? obj)****override int GetHashCode()****static bool operator ==(ExportMode left, ExportMode right)**

Token equality.

11.8.3. ExportModes

The base export modes (valid on any analyzer). There is no server operation to enumerate modes, so these are also the documentation catalogue. Product modes live on their product's holder (e.g. `BluetoothExportModes`).

static ExportMode FilteredTraceTimeRange

Filtered trace, by time range.

static ExportMode FilteredTraceActiveOverview

Filtered trace, by active overview.

static ExportMode Throughput

Throughput over a time range.

static IReadOnlyList<ExportMode> All

All base export modes.

11.8.4. IExportOptions

Anything that can supply server-spelled export-option pairs. Every typed option carrier implements it; the snake-to-Pascal name mapping lives in each carrier's `AsOptions`.

11.8.5. FilteredTraceOptions

Options for `FilteredTraceTimeRange` and `FilteredTraceActiveOverview`.

*Ps values are picoseconds (use `PicosecondsFromTimeSpan` to convert a `TimeSpan`). `null` omits the option, applying the server default (`StartTime 0`; the rest = full trace).

long? StartTimePs

Start of the range, in picoseconds (`StartTime`; default 0).

long? MaxSizeBytes

Maximum output size, in bytes (`MaxSize`; default = full).

long? MaxDurationPs

Maximum duration, in picoseconds (`MaxDuration`; default = full).

long? MaxItems

Maximum number of items (`MaxItems`; default = full).

ReadOnlyDictionary<string, object?> AsOptions()

void Validate()

No range constraints to check.

11.8.6. ThroughputOptions

Options for `Throughput`.

long? RangeStartTimePs

Start of the range, in picoseconds (`RangeStartTime`).

long? RangeEndTimePs

End of the range, in picoseconds (`RangeEndTime`).

ReadOnlyDictionary<string, object?> AsOptions()

void Validate()

Validate the range — throws `ExportOptionException` if end precedes start.

11.8.7. ExportOptions

The export-option wire machinery: the value renderer, the option-to-wire serializer, and the range validator. (Export modes carry their own metadata on `ExportMode`.)

const long PicosecondsPerSecond = 1_000_000_000_000L

Picoseconds per second; matches the picosecond convention used elsewhere in the SDK.

static long PicosecondsFromTimeSpan(TimeSpan value)

Convert a `TimeSpan` to integer picoseconds (rounded).

static string? RenderOptionValue(object? value)

Render ONE typed option value to its exact wire string, or `null` to OMIT it.

The wire rules (the single source of truth): `null` → `null` (unset; never sent, server default applies) `bool` → `"true"` / `"false"` (.NET `bool.Parse` form) `TimeSpan` → integer picoseconds as a decimal string integral types (`long/int`) → decimal string `string` → verbatim (carriers pre-render any token strings)

11.8.8. Analyzer

void Export(string output, ExportMode mode, IExportOptions? options = null)

Run any export mode (the generic primary method, and an escape hatch for modes newer than this SDK).

Parameters:

- `output`: Output path on the analyzer machine.
- `mode`: The export mode (server `exportName`).
- `options`: Typed option carrier, or `null` for no options.

`output` is a path on the analyzer machine (like `StopAndSave`). `mode` is the server `exportName` — a named member of `ExportModes` / a product holder, or any raw string. `options` is a typed carrier (e.g. `BluetoothAudioOptions`) or `null`. A product mode used on the wrong analyzer raises `OperationException` at call time.

IReadOnlyList<(string OptionName, string OptionValue)> ExportDryRun(

Render the exact `(OptionName, OptionValue)` wire pairs that `Export` would send, without contacting the analyzer (offline).

Calls `Validate` on `options` first, then returns the rendered pairs. Makes no proxy call — useful for tests and for inspecting omit-null / bool / picosecond rendering before committing a long export. `output` and `mode` are accepted for signature symmetry with `Export`.

void ExportFilteredTraceTimeRange(

Export a filtered trace by time range (BASE; valid on any analyzer).

*Ps values are picoseconds (use `PicosecondsFromTimeSpan` to convert a `TimeSpan`); `maxSizeBytes` is a byte count. `null` omits the option (server defaults: `StartTime` 0, the rest = full trace).

void ExportFilteredTraceActiveOverview(

Export a filtered trace by the active overview (BASE; valid on any analyzer). Same options as the time-range form.

void ExportThroughput(

Export throughput over a time range (BASE; valid on any analyzer). `null` uses the server default (trace start/end).

11.9. Bluetooth exports

11.9.1. BluetoothExportModes

The Bluetooth export modes (valid only on a Bluetooth analyzer).

static ExportMode Audio

Bluetooth audio extraction.

static ExportMode MobilePhoneData

Bluetooth mobile phone data.

static ExportMode Channels

Bluetooth per-channel statistics.

static ExportMode Airtime

Airtime usage.

static IReadOnlyList<ExportMode> All

All Bluetooth export modes.

11.9.2. BluetoothPacketLossMode (enum)

How `bluetooth_audio` export fills gaps caused by packet loss (the `PacketLoss` option). Bluetooth analyzers only. Each member maps to an exact server token.

- `NegativeDc` — Fill gaps with a negative-DC value (easy to spot). Wire token `NegativeDC`.
- `Silence` — Fill gaps with silence, no smoothing. Wire token `Silence`.
- `SilenceSmooth` — Fill gaps with silence, smoothed to conceal the gap. Wire token `SilenceSmooth`.

11.9.3. BluetoothAudioOptions

Options for `Audio` (Bluetooth analyzers only).

bool? AddRawPayloadFiles

Also write raw payload files (`AddRawPayloadFiles`).

long? SynchroBufferMs

Synchronisation buffer, in milliseconds (`SynchroBufferMilliseconds`).

bool? ForceIsochronousBuffering

Force isochronous buffering (`ForceIsochronousBuffering`).

BluetoothPacketLossMode? PacketLoss

How to fill gaps caused by packet loss (`PacketLoss`).

IRReadOnlyDictionary<string, object?> AsOptions()**void Validate()**

No range constraints to check.

11.9.4. BluetoothChannelsOptions

Options for `Channels` (Bluetooth analyzers only).

long? RangeStartTimePs

Start of the range, in picoseconds (`RangeStartTime`).

long? RangeEndTimePs

End of the range, in picoseconds (`RangeEndTime`).

IRReadOnlyDictionary<string, object?> AsOptions()**void Validate()**

Validate the range — throws `ExportOptionException` if end precedes start.

11.9.5. BluetoothAirtimeOptions

Options for `Airtime` (Bluetooth analyzers only).

long? RangeStartTimePs

Start of the range, in picoseconds (`RangeStartTime`).

long? RangeEndTimePs

End of the range, in picoseconds (`RangeEndTime`).

IRReadOnlyDictionary<string, object?> AsOptions()**void Validate()**

Validate the range — throws `ExportOptionException` if end precedes start.

11.10. Markers

11.10.1. MarkerColor (enum)

A marker colour. Values match the analyzer's `MarkerColor` Slice enum.

- `Yellow` — Yellow.
- `Blue` — Blue.
- `Red` — Red.
- `Green` — Green.
- `Orange` — Orange.
- `Purple` — Purple.

11.10.2. Marker

A marker read from the trace (the result of `Markers`). Handle-free; it is a snapshot, not a live reference to a marker on the analyzer.

MarkerColor Color

The marker's colour.

string Text

The marker's label text.

long TimePs

The marker's time, in picoseconds.

TimeSpan Time

The marker's time as a `TimeSpan` (100-nanosecond resolution).

double TimeSeconds

The marker's time in seconds.

11.10.3. Analyzer

`void AddMarkerAtTime(long timePs, string text, MarkerColor color = MarkerColor.Yellow)`

Adds a marker at a given time (→ `AddMarkerAtTime`).

Parameters:

- `timePs`: The marker time, in picoseconds.
- `text`: The marker's label text.
- `color`: The marker's colour. Defaults to `Yellow`.

void AddMarkerOnSelectedOverviewItem(string text, MarkerColor color = MarkerColor.Yellow)

Adds a marker on the currently selected overview item (→ `AddMarkerOnSelectedOverviewItem`).

Parameters:

- `text`: The marker's label text.
- `color`: The marker's colour. Defaults to `Yellow`.

IReadOnlyList<Marker> Markers()

Every marker in the trace, as handle-free `Marker` snapshots (→ `GetMarkers`).

Returns: The markers currently in the trace.

11.11. Session, trace files and info

11.11.1. MessageSeverity (enum)

Severity for `InsertMessage` (mirrors the `Slice` enum; 1-based).

- `Info` — An informational message.
- `Warning` — A warning message.
- `Error` — An error message.

11.11.2. AppInfo

Information about the remote application (from `GetAppInfo`).

string Id

The application identifier.

string Description

A human-readable description of the application.

string Version

The application version.

string FileExt

The trace file extension this application uses.

bool Interactive

Whether the application is running interactively (with a UI).

11.11.3. RecordingStatus

Status of the current recording (from `GetRecordingStatus`).

string DataSource

The data source being recorded.

int DurationSeconds

How long the recording has been running, in seconds.

long FileSize

The current trace size, in bytes.

TimeSpan Duration

`DurationSeconds` as a `TimeSpan` (exact).

11.11.4. RunningTask

A background task running in the analyzer (from `GetRunningTasks`).

string Name

The task name.

int ProgressPercent

The raw progress percentage (only meaningful when `ProgressApplicable` is `true`).

bool ProgressApplicable

Whether `ProgressPercent` is meaningful for this task.

int? Progress

Percent complete, or `null` when the server says progress isn't meaningful.

11.11.5. TraceFileInfo

Information about the loaded trace file (from `GetTraceFileInfo`).

string Path

The trace file path (on the analyzer machine).

long FileSize

The trace file size, in bytes.

long StartDateTimeUtcRaw

The raw Slice `long` start time (milliseconds since 2000-01-01 UTC); see `StartedAtUtc`.

string SoftwareVersion

The analyzer software version that produced the trace.

string DataSource

The data source that was recorded.

string UniqueId

A unique identifier for the trace.

string RecordingOptions

The opaque recording-options blob (the same string `ConfigureRecordingOptions` consumes).

DateTime? StartedAtUtc

The trace start time as a UTC `DateTime`, or `null` when the raw value is non-positive (unset).

`StartDateTimeUtcRaw` is milliseconds since 2000-01-01 UTC (the epoch the analyzer's own trace indexer uses); reinterpret the raw value yourself if you ever need a different one.

11.11.6. Analyzer**bool IsRecording**

`true` while a capture is in progress (`IsRecording`).

void StartRecording()

Starts a capture (equivalent to `Record > Start` in the GUI; `StartRecording`).

void StopAndSave(string filename, bool overwrite = false)

Stops the capture and saves the trace to `filename` (`StopRecordingAndSaveTraceFile`).

Parameters:

- `filename`: The destination path on the analyzer machine.
- `overwrite`: Set `true` to replace an existing file.

The path is interpreted on the analyzer machine, not the client running this code.

void AbortRecording()

Stops the capture and discards the trace — nothing is saved (`AbortRecordingAndDiscardTraceFile`).

IDisposable Recording(string saveTo, bool overwrite = false)

Starts a capture and returns a handle that stops and saves the trace to `saveTo` when disposed.

Parameters:

- `saveTo`: The destination path on the analyzer machine.
- `overwrite`: Set `true` to replace an existing file.

Returns: An `IDisposable` that stops and saves on dispose.

Intended for a `using` block: the trace is saved when the block exits (best effort even if the body throws — a capture is rarely worth losing because the code that triggered it failed).

bool IsLoading

`true` while a trace file is loading (`IsLoading`).

void StartLoading(string filename)

Begins loading `filename` and returns immediately (`StartLoading`).

Parameters:

- `filename`: The trace file path on the analyzer machine.

Loading is asynchronous: poll `IsLoading` (or use `WaitUntilLoaded` / `Load`) to know when it finishes. `filename` is a path on the analyzer machine.

void WaitUntilLoaded(TimeSpan? timeout = null, TimeSpan? pollInterval = null)

Blocks until `IsLoading` is `false`, polling against a deadline.

Parameters:

- `timeout`: Maximum time to wait. Defaults to two minutes. Pass `InfiniteTimeSpan` (or any negative value) to wait forever.
- `pollInterval`: How often to poll. Must be `> 0`. Defaults to 250 ms.

void Load(string filename, bool wait = true, TimeSpan? timeout = null, TimeSpan? pollInterval = null)

Loads a trace file (`StartLoading`); by default blocks until it finishes.

Parameters:

- `filename`: The trace file path on the analyzer machine.
- `wait`: When `true` (the default), block until loading finishes.
- `timeout`: Maximum time to wait when `wait` is `true`. Defaults to two minutes; pass a negative `TimeSpan` to wait forever.

- `pollInterval`: How often to poll. Must be > 0. Defaults to 250 ms.

The primary entry point: it calls `StartLoading` and then, unless `wait` is `false`, `WaitUntilLoaded`, so the next statement can use `TraceFileInfo`. The path is on the analyzer machine.

TraceFileInfo TraceFileInfo()

Returns information about the loaded trace file (`GetTraceFileInfo`).

Returns: The `TraceFileInfo` for the loaded trace.

void CloseTraceFile()

Closes the loaded trace file (`CloseTraceFile`).

bool IsModified

`true` if the trace has unsaved changes (`IsModified`).

void SaveChanges()

Saves changes to the loaded trace file (`SaveChanges`).

AppInfo AppInfo()

Returns information about the remote application (`GetAppInfo`).

Returns: The `AppInfo` for the remote application.

void ExitApp()

Asks the remote application to exit (`ExitApp`).

RecordingStatus RecordingStatus()

Returns the status of the current recording (`GetRecordingStatus`).

Returns: The current `RecordingStatus`.

Only valid while a recording is active; the analyzer rejects it otherwise (surfacing as `OperationException`). Gate it on `IsRecording`.

string GetRecordingOptions(bool relevantOnly = false)

Returns the recording options as an opaque string (`GetRecordingOptions`).

Parameters:

- `relevantOnly`: When `true`, return only the options relevant to the current setup.

Returns: The opaque recording-options blob.

void ConfigureRecordingOptions(string options)

Applies recording options from an opaque string (`ConfigureRecordingOptions`).

Parameters:

- `options`: The opaque recording-options blob (as returned by `GetRecordingOptions`).

ReadOnlyList<string> DataSources()

Returns the available data source unique IDs (`GetAvailableDataSources`).

Returns: The available data source unique IDs.

void SelectDataSource(string dataSourceUniqueId)

Selects the data source by its unique ID (`SelectDataSource`).

Parameters:

- `dataSourceUniqueId`: The unique ID of the data source to select.

string? SelectedDataSource()

Returns the selected data source unique ID, or `null` (`GetSelectedDataSource`).

Returns: The selected data source unique ID, or `null` if none is selected.

void InsertMessage(MessageSeverity severity, string message)

Inserts a message in the Message Log (`InsertMessage`).

Parameters:

- `severity`: The message severity.
- `message`: The message text.

void WaitUntilIdle(TimeSpan? timeout = null, TimeSpan? pollInterval = null, TimeSpan? settlePeriod = null)

Blocks until no background task is running (polls `GetRunningTasks`).

Parameters:

- `timeout`: Maximum time to wait. Defaults to ten minutes. Pass `InfiniteTimeSpan` (or any negative value) to wait forever.
- `pollInterval`: How often to poll. Must be > 0 . Defaults to 250 ms.
- `settlePeriod`: How long idleness must persist before it is trusted (the background task can take a moment to appear after the triggering call). Defaults to 1 s.

Filter changes (overview queries, device filters) re-process the trace in the background, reported as a running

task (e.g. "Filtering data") while `IsLoading` stays `false` — so this, not `WaitUntilLoaded`, is the wait that matches them.

`IReadOnlyList<RunningTask> RunningTasks()`

Returns the background tasks currently running (`GetRunningTasks`).

Returns: The currently running `RunningTask`'s.

`void AbortRunningTask(string name)`

Aborts the running task named `name` (`AbortRunningTask`).

Parameters:

- `name`: The name of the task to abort.

`byte[] GetSettings()`

Returns the application settings as an opaque byte blob (`GetSettings`).

Returns: The opaque settings blob.

`void ConfigureSettings(byte[] settings)`

Applies an opaque settings byte blob (`ConfigureSettings`).

Parameters:

- `settings`: The opaque settings blob (as returned by `GetSettings`).

`void CancelUserInteraction()`

Cancels a pending user interaction on the analyzer (`CancelUserInteraction`).

11.12. Logic signals

11.12.1. `LogicSignalTransitionType` (enum)

Which edge to search for (mirrors the `Slice LogicSignalTransitionType` enum).

- `Any` — Match either a rising or a falling edge.
- `RisingEdge` — Match a low-to-high transition.
- `FallingEdge` — Match a high-to-low transition.

11.12.2. `LogicSignalTransition`

A found logic-signal transition (the result of `FindLogicSignalTransition`).

Logic signals are the analyzer's digital I/O lines. `State` is a bitmask where bit `N` is the level of logic signal `N` at the moment of the transition.

int State

Bitmask of all signal levels at the transition (bit N is the level of signal N).

long TimePs

Time of the transition, in picoseconds.

TimeSpan Time

The transition time as a `TimeSpan` (100-nanosecond resolution).

11.12.3. Analyzer**int LogicSignalsState(long timePs)**

Reads the logic-signal state at a given time, as a bitmask where bit N is the level of logic signal N.

Parameters:

- `timePs`: The time to sample, in picoseconds.

Returns: The signal-state bitmask at `timePs`.

LogicSignalTransition FindLogicSignalTransition(

Searches for the next logic-signal transition within a time range.

Parameters:

- `fromTimePs`: Start of the search range, in picoseconds (inclusive).
- `toTimePs`: End of the search range, in picoseconds.
- `signalsMask`: Bitmask selecting which signals to watch (bit N selects logic signal N).
- `transitionType`: Which edge to match. Defaults to `Any`.

Returns: The matching `LogicSignalTransition`.

11.13. Errors**11.13.1. RemoteControlException**

Base type for every error raised by the SDK. Catch this to catch them all.

RemoteControlException(string message, Exception? innerException = null)

Create a remote-control error with a message and optional inner exception.

11.13.2. ConnectionFailedException

The analyzer could not be reached, or the endpoint is not an Ellisys analyzer.

ConnectionFailedException(string message, Exception? innerException = null)

Create a connection failure.

11.13.3. OperationException

The analyzer rejected the operation (a server-side failure, e.g. a wrong-product call).

OperationException(string message, Exception? innerException = null)

Create an operation failure.

11.13.4. LoadTimeoutException

A trace did not finish loading within the allotted time.

LoadTimeoutException(string message, Exception? innerException = null)

Create a load-timeout failure.

11.13.5. NotAvailableException

A requested product facet or feature is not available on this analyzer.

NotAvailableException(string message, Exception? innerException = null)

Create a not-available failure.

Revisions History

Date	Rev	Changes
June 9, 2026	1.0	Initial C# SDK guide.
June 10, 2026	1.1	Added the Using with AI agents and Overview queries sections.

Copyright and Intellectual Property

Copyright Disclaimer

Copyright © Ellisys 2026. All Rights Reserved.

No part of this document or any of its contents may be reproduced, copied, modified or adapted, without the prior written consent of Ellisys.

Intellectual Property Disclaimer

This document is provided to you "as is" with no warranties whatsoever, including any warranty of merchantability, non-infringement, or fitness for any particular purpose. Ellisys disclaims all liability, including liability for infringement of any proprietary rights, relating to use or implementation of information in this document. The provision of this document to you does not provide you with any license, express or implied, by

estoppel or otherwise, to any intellectual property rights.

Open Source Disclaimer

This Ellisys analysis software automation API plugin is based on a third-party networking library from ZeroC named ICE (<https://zeroc.com/products/ice>). This library is licensed under the GNU GPL v2 or later. This plugin is also licensed under the GPL, and its source code can be requested at gpl@ellisys.com.

This plugin is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.