



Ellisys Bluetooth Analyzer Remote Control User Guide

Version 2.3, November 26, 2025

Table of Contents

1. Introduction	1
2. Installing the Remote Control plugin	1
3. Running the provided samples	3
3.1. C# / .net sample	3
3.2. Python sample	3
3.3. Using a different language or platform	3
4. Overriding Connection Properties	3
4.1. By file	3
4.2. By command line	4
4.3. Priority order	4
5. Overview queries	4
5.1. Syntax	4
5.2. Examples	4
5.3. Behavior notes (important for automation)	5
6. Using with AI agents	5
6.1. Installing the agent skill	6
6.2. The MCP server	6
6.3. Choosing between the skill and the MCP server	7
7. API reference	8
7.1. Generic functions (AnalyzerRemoteControl.ice)	8
7.1.1. AppInfo GetAppInfo()	8
7.1.2. void ExitApp()	8
7.1.3. StringArray GetAvailableDataSources()	8
7.1.4. void SelectDataSource (string dataSourceUniqueId)	8
7.1.5. string GetSelectedDataSource()	8
7.1.6. bool IsRecording()	8
7.1.7. RecordingStatus GetRecordingStatus()	9
7.1.8. void StartRecording()	9
7.1.9. void StopRecordingAndSaveTraceFile (string filename, bool overwrite)	9
7.1.10. void AbortRecordingAndDiscardTraceFile()	9
7.1.11. string GetRecordingOptions (bool relevantOnly)	9
7.1.12. void ConfigureRecordingOptions (string options)	10
7.1.13. void InsertMessage (MessageSeverity severity, string message)	10
7.1.14. bool IsLoading()	11
7.1.15. void StartLoading (string filename)	11
7.1.16. TraceFileInfo GetTraceFileInfo()	11
7.1.17. void CloseTraceFile()	11
7.1.18. bool IsModified()	11

7.1.19. void SaveChanges()	12
7.1.20. void AddMarkerOnSelectedOverviewItem (AddMarker marker)	12
7.1.21. void AddMarkerAtTime (long timeInPicoseconds, AddMarker marker)	12
7.1.22. GetMarkerArray GetMarkers()	12
7.1.23. void Export (string outputFilename, string exportName, ExportOptionArray exportOptions)	12
7.1.24. void Export (string outputFilename, "filtered_trace_time_range", ExportOptionArray exportOptions)	12
7.1.25. void Export (string outputFilename, "filtered_trace_active_overview", ExportOptionArray exportOptions)	13
7.1.26. void Export (string outputFilename, "throughput", ExportOptionArray exportOptions)	13
7.1.27. void Export (string outputFilename, "bluetooth_audio", ExportOptionArray exportOptions)	13
7.1.28. void Export (string outputFilename, "bluetooth_mobile_phone_data", ExportOptionArray exportOptions)	14
7.1.29. void Export (string outputFilename, "bluetooth_channels", ExportOptionArray exportOptions) ..	14
7.1.30. void Export (string outputFilename, "airtime", ExportOptionArray exportOptions)	14
7.1.31. StringArray GetAvailableOverviews()	14
7.1.32. string GetSelectedOverview()	14
7.1.33. void SelectOverview (string overviewName)	14
7.1.34. StringArray GetAvailableProtocolLayers()	15
7.1.35. string GetSelectedProtocolLayer()	15
7.1.36. void SelectProtocolLayer (string protocolLayerName)	15
7.1.37. string GetOverviewQuery()	15
7.1.38. void SetOverviewQuery (string query)	15
7.1.39. int OverviewRootItem()	15
7.1.40. string GetOverviewItemDescription (int itemHandle)	16
7.1.41. StringArray GetOverviewItemsDescription (HandleArray itemHandles)	16
7.1.42. long GetOverviewItemTimeInPicoseconds (int itemHandle)	16
7.1.43. TimeArray GetOverviewItemsTimeInPicoseconds (HandleArray itemHandles)	16
7.1.44. ByteArray GetOverviewItemData (int itemHandle)	17
7.1.45. ByteArrays GetOverviewItemsData (HandleArray itemHandles)	17
7.1.46. string GetOverviewItemXmlReport (int itemHandle)	17
7.1.47. StringArray GetOverviewItemsXmlReport (HandleArray itemHandles)	17
7.1.48. string GetOverviewItemXmlReportFiltered (int itemHandle, StringArray fieldNameFilters)	18
7.1.49. StringArray GetOverviewItemsXmlReportFiltered (HandleArray itemHandles, StringArray fieldNameFilters)	18
7.1.50. int GetOverviewItemChildCount (int itemHandle)	18
7.1.51. int GetOverviewItemChild (int itemHandle, int childIndex)	18
7.1.52. HandleArray GetOverviewItemChildren (int itemHandle, int childIndex, int childCount)	19
7.1.53. HandleArray SearchOverviewItems (int itemHandle, int childIndex, int childCount, int	

maxDepth, StringArray descriptionFilters, StringArray fieldNameFilters, StringArray fieldValueFilters)	19
7.1.54. void ReleaseAllOverviewItemHandles()	20
7.1.55. void SelectOverviewItem (int itemHandle)	20
7.1.56. int GetSelectedOverviewItem()	20
7.1.57. void GetLogicSignalsState (long timeInPicoseconds, out int logicSignalsState)	20
7.1.58. void FindLogicSignalsTransition (long fromTimeInPicoseconds, long toTimeInPicoseconds, int signalsMask, LogicSignalTransitionType transitionType, out int foundLogicSignalsState, out long foundTimeInPicoseconds)	21
7.1.59. RunningTaskArray GetRunningTasks()	21
7.1.60. void AbortRunningTask (string name)	21
7.1.61. ByteArray GetSettings()	21
7.1.62. void ConfigureSettings (ByteArray settings)	22
7.1.63. void CancelUserInteraction()	22
7.1.64. void InsertComment (string comment, string overviewName)	22
7.2. Bluetooth functions (BluetoothAnalyzerRemoteControl.ice)	22
7.2.1. void SplitTraceFileAndContinueRecording (string filename)	22
7.2.2. void AddLinkKey (long bdaddr1, long bdaddr2, LinkKeyByteArray linkKey)	22
7.2.3. void ConfigureDeviceFilter (DeviceFilterMode mode, DeviceAddressArray deviceAddrs)	23
7.2.4. void GetSpectrumRssi (long timeInPicoseconds, int rfChannelNumber, out double rssi, out long startTimeInPicoseconds, out long stopTimeInPicoseconds)	23
7.2.5. void GetSpectrumRssiRange (long fromTimeInPicoseconds, long toTimeInPicoseconds, int rfChannelNumber, out RssiArray rssi, out long startTimeInPicoseconds, out long stopTimeInPicoseconds)	23
7.2.6. ChannelsSummary GetChannelsSummary (long fromTimeInPicoseconds, long toTimeInPicoseconds)	24
7.2.7. void ExportAudio (string outputDirectory)	24
7.3. Wi-Fi functions (EthernetAnalyzerRemoteControl.ice)	24
7.3.1. void AddWifiKeyByApSsid (string apSsid, string key)	24
7.3.2. void AddWifiKeyByApMacAddr (long apMacAddr, string key)	24
7.3.3. void ConfigureDeviceFilter (DeviceFilterMode mode, DeviceAddressArray deviceAddrs)	25
7.4. WPAN / 802.15.4 functions (WpanAnalyzerRemoteControl.ice)	25
7.4.1. void AddThreadNetworkMasterKey (int panId, ByteArray key128, long sequenceCounter)	25
7.4.2. void RemoveThreadNetworkAllMasterKeys()	25
7.4.3. void RemoveThreadNetworkMasterKeys (int panId)	25
7.4.4. void RemoveThreadNetworkMasterKey (int panId, ByteArray key128)	25
7.4.5. void AddZigbeeApsKey (ByteArray destinationAddress64, ByteArray sourceAddress64, ByteArray key128)	26
7.4.6. void RemoveZigbeeApsAllKeys()	26
7.4.7. void RemoveZigbeeApsKeys (ByteArray destinationAddress64, ByteArray sourceAddress64)	26

7.4.8. void RemoveZigbeeApsKey (ByteArray destinationAddress64, ByteArray sourceAddress64, ByteArray key128)	26
7.4.9. void AddZigbeeNetworkKey (int panId, ByteArray key128)	26
7.4.10. void RemoveZigbeeNetworkAllKeys ()	26
7.4.11. void RemoveZigbeeNetworkKeys (int panId)	26
7.4.12. void RemoveZigbeeNetworkKey (int panId, ByteArray key128)	27
Revisions History	27
Copyright and Intellectual Property	27

1. Introduction

This Ellisys analysis software automation API plugin is based on a third-party networking library from ZeroC named ICE (<https://zeroc.com/products/ice>). In this context, the Ellisys analysis software is the server and the user script is the client. The Ellisys software runs on Windows natively — and on Linux and macOS through the Ellisys-provided runtime — and is accessible from any client on the network. The samples are provided in C#.NET and Python but the ZeroC tools can be used to create wrappers for various languages and platforms. Please refer to the section below for instructions to access the API from other languages and platforms supported by the ZeroC ICE library.

The automation API can be used for very diverse and advanced tasks such as:

- *Common:* Controlling the capture (start / stop capture), saving traces and loading traces.
- *Common:* Selecting the data source, to control multiple analyzers accessible from the same machine.
- *Common:* Accessing the Overviews for parsing the traffic and detecting protocol conditions (either during live capture or with a pre-captured trace).
- *Common:* Adding markers at specific time or on specific protocol items, to flag conditions to be reviewed later manually.
- *Bluetooth:* Injecting known link keys.
- *Bluetooth:* Parsing logic signals captured with the integrated logic analyzer, for example to detect an electrical condition on an external signal, then look at the protocol traffic at that specific time.
- *Bluetooth:* Parsing spectrum information, for example to determine if a retransmission is related to an interference.
- *Bluetooth:* Exporting captured audio traffic.



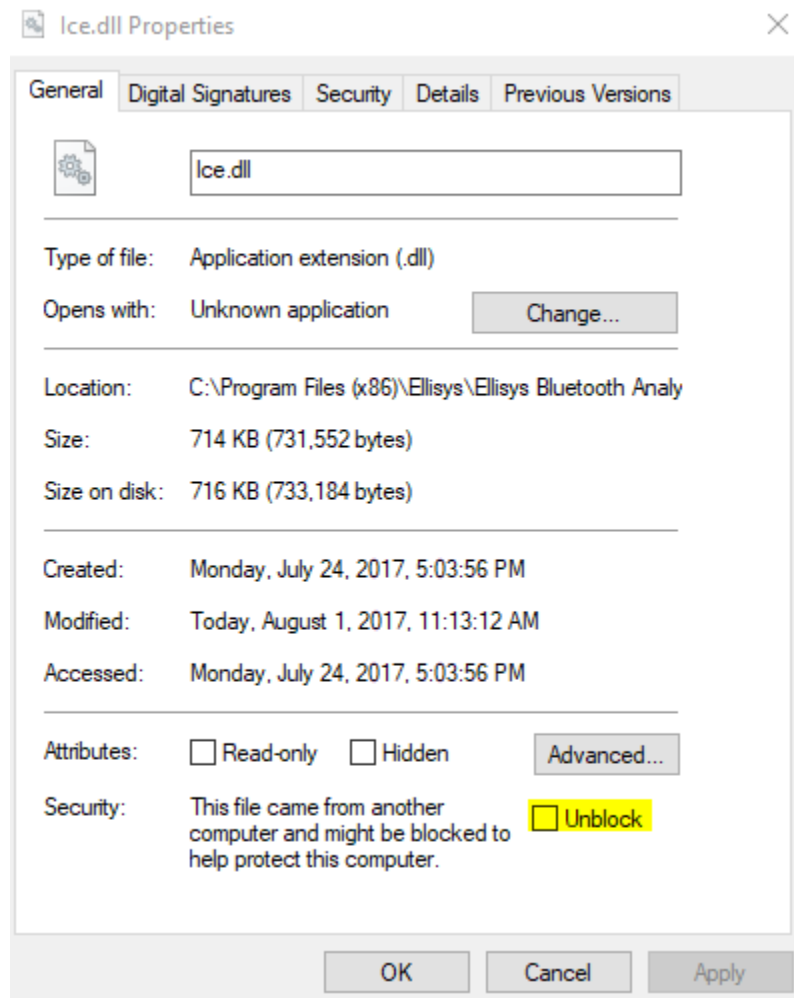
For **Python** or **C#** automation we recommend the higher-level SDKs — the [Python SDK Guide](#) and the [C# SDK Guide](#) — which wrap this API in idiomatic packages (`ellisys_analysis_automation / Ellisys.Analysis.Automation`) with bounded Overview traversal, typed exports, and product facets (including the Bluetooth operations above). This guide remains the language-neutral, operation-level reference that the SDK is built on.

2. Installing the Remote Control plugin

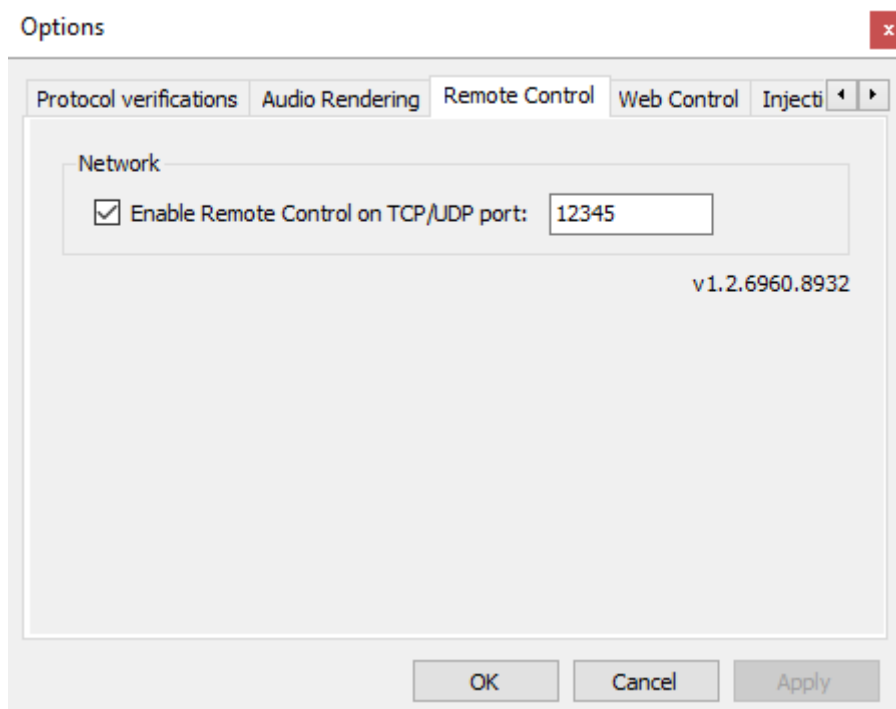
The remote control plugin is available on Ellisys website and is linked from the user manual of the analysis software. Please follow these steps to install the plugin:

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. Running the provided samples

3.1. 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 a key before to stop recording.
3. The provided sample can be used as starting point for custom control software.

3.2. 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 a key before to stop recording.
4. The provided sample can be used as starting point for custom control software.

3.3. 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>

4. Overriding Connection Properties

4.1. 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>

4.2. 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`.

4.3. Priority order

The priority is given as follow:

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

5. 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(...)`).

5.1. 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.

5.2. 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>	Foo is 1, 3, 7, 8, 9 or 10
<code>Foo >= 0x0F << 2, 4+5 && (Bar = 1 Status != "OK")</code>	Foo is at least 9, and either Bar is 1 or Status is not OK
<code>Item = "AVDTP*" && Status = "OK"</code>	the Item starts with <code>AVDTP</code> and the Status is OK
<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>

5.3. 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.

6. 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).

6.1. 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.

6.2. 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).

6.3. 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.

7. API reference

7.1. Generic functions (AnalyzerRemoteControl.ice)

Generic remote-control operations common to every Ellisys analyzer.

7.1.1. AppInfo GetAppInfo()

Returns information about the remote application, such as version, file extension, etc.

Returns: Application info.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.2. void ExitApp()

Exits the remote application.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.3. StringArray GetAvailableDataSources()

Returns the available analyzers accessible by the analysis software.

Equivalent to list of the Record menu > Select analyzer dialog.

Returns: Array of available data sources.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.4. void SelectDataSource(string dataSourceUniqueId)

Select the active data source for capturing data. This is typically a string returned from the `GetAvailableDataSources()` function.

Equivalent to selecting an analyzer from the Record menu > Select analyzer dialog.

Parameters:

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

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.5. string GetSelectedDataSource()

Returns the currently selected analyzers, if any.

Returns: Analyzer currently selected.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.6. bool IsRecording()

Indicates if a recording is currently running.

Returns: true if the analyzer is currently capturing data following the call of the StartRecording() method, false otherwise

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.7. RecordingStatus GetRecordingStatus()

Returns information on the running recording, such as the analyzer, the duration and trace size.

Returns: Status of the recording

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.8. void StartRecording()

Starts the analyzer capture with the current settings. The settings can be either configured manually in the GUI software or configured with the ConfigureSettings() method.

Equivalent to Record > Start recording in the GUI software.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.9. void StopRecordingAndSaveTraceFile(string filename, bool overwrite)

Stops the analyzer capture and save the resulting file to the specified filename.

Equivalent to Record > Stop recording followed by File > Save in the GUI software.

Parameters:

- `filename`: path to the file to be saved, based on the remote computer
- `overwrite`: destination filename will be replaced if already exists

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.10. void AbortRecordingAndDiscardTraceFile()

Aborts the ongoing capture and discard any captured data.

Equivalent to Record > Stop recording followed by File > New in the GUI software.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.11. string GetRecordingOptions(bool relevantOnly)

Retrieve the current recording options in JSON format.

Equivalent to Record > Recording Options > Save in the GUI software.

Parameters:

- `relevantOnly`: indicates if the function shall return only the fields which are set with non-default values, or if default values shall be returned as well.

Returns: JSON formatted-string containing the recording options

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.12. void `ConfigureRecordingOptions(string options)`

Configure the recording options of the data source. The provided settings can be saved from the Recording Options dialog of the GUI software or can be created manually.

The format of the options string is a modified JSON format with the following customizations to the standard JSON format:

- The root object braces can be omitted. For example, "{ field:value }" can optionally be expressed as "field:value" for brevity.
- The name double quotes can be omitted. For example, "field":"value" can optionally be expressed as field:"value" for brevity.
- Encapsulated parent objects having a single child object can use the '.' hierarchical notation. For example, "a:{ b:value }" can optionally be expressed as "a.b:value" for brevity.

The provided string can contain only the fields to be updated.

Equivalent to Record > Recording Options > Load in the GUI software.

Parameters:

- `options`: recording options in extended JSON format

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.13. void `InsertMessage(MessageSeverity severity, string message)`

Insert a message in the Message Log overview at the current time. This function can only be called during recording. The message can either be raw text, or hierarchically formatted text in JSON or XML.

The JSON format is as follows:

```
{
  fields:[
    { field:"field1", value:[
      { field:"field2", value:"value2" },
      { field:"field3", value:"value3" }
    ] }
  ]
}
```

The XML format is as follows:

```
<fields>
  <field name="field1">
    <field name="field2">value2</field>
    <field name="field3">value3</field>
  </field>
</fields>
```

</fields>

Parameters:

- **severity:** indicates if this message is an information, a warning or an error
- **message:** the message that will be inserted, either in raw text, JSON or XML

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.14. bool IsLoading()

Indicates if the software is loading a trace.

Returns: true if the file is loading, false otherwise.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.15. void StartLoading(string filename)

Starts loading the specified filename.

Equivalent to File > Open in the GUI software.

Parameters:

- **filename:** path to the file to be loaded, based on the remote computer

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.16. TraceFileInfo GetTraceFileInfo()

Returns information on the loaded trace file, such as the size, start date, analyzer used and software version.

Returns: Information on the loaded trace file

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.17. void CloseTraceFile()

Close the loaded trace file. Changes will be lost if any, unless saved explicitly with `SaveChanges()`.

Equivalent to File > New in the GUI software.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.18. bool IsModified()

Indicates if the open file has changes. The changes can be saved with `SaveChanges()`.

Returns: true if the open file has changes, false otherwise

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.19. void SaveChanges()

Save changes in the currently open file.

Equivalent to File > Save in the GUI software.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.20. void AddMarkerOnSelectedOverviewItem(AddMarker marker)

Adds a marker on the currently selected item in the Overview.

Equivalent to main toolbar > Markers > Mark selected item in the GUI software.

Parameters:

- `marker`: information on the marker to be added

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.21. void AddMarkerAtTime(long timeInPicoseconds, AddMarker marker)

Adds a marker at the specified time.

Equivalent to Instant Timing > Right Click Menu > Add new marker here in the GUI software.

Parameters:

- `timeInPicoseconds`: the time in picoseconds of the marker to be added
- `marker`: information on the marker to be added

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.22. GetMarkerArray GetMarkers()

Get all markers of the trace file.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.23. void Export(string outputFilename, string exportName, ExportOptionArray exportOptions)

Export the specified data to the output filename with the specified options. The options are optional and are specific to the export mode. The available export modes are described in the following entries.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.24. void Export(string outputFilename, "filtered_trace_time_range", ExportOptionArray exportOptions)

Equivalent export to "Filtered trace based on time range" in the GUI.

Export options:

- `StartTime`: relative start time, in picoseconds. Optional, default is zero.

- **MaxSize:** maximum exported trace size, in bytes. Optional, default is full trace.
- **MaxDuration:** maximum exported trace duration, in picoseconds. Optional, default is full trace.
- **MaxItems:** maximum number of items in the export trace. Optional, default is full trace.

7.1.25. void Export(string outputFilename, "filtered_trace_active_overview", ExportOptionArray exportOptions)

Equivalent export to "Filtered trace based on active overview" in the GUI.

Export options:

- **StartTime:** relative start time, in picoseconds. Optional, default is zero.
- **MaxSize:** maximum exported trace size, in bytes. Optional, default is full trace.
- **MaxDuration:** maximum exported trace duration, in picoseconds. Optional, default is full trace.
- **MaxItems:** maximum number of items in the export trace. Optional, default is full trace.

7.1.26. void Export(string outputFilename, "throughput", ExportOptionArray exportOptions)

Equivalent export to "Throughput" in the GUI.

Export options:

- **RangeStartTime:** Integer value indicating the start of the range, when ExportRange is true, in picoseconds. Optional, default is trace start time.
- **RangeEndTime:** Integer value indicating the end of the range, when ExportRange is true, in picoseconds. Optional, default is trace end time.

7.1.27. void Export(string outputFilename, "bluetooth_audio", ExportOptionArray exportOptions)

Equivalent export to "Bluetooth Audio" in the GUI.

Export options:

- **AddRawPayloadFiles:** boolean value indicating if raw files shall be stored in addition to the decoded files. Optional, default is false.
- **SynchroBufferMilliseconds:** synchro buffer size, in milliseconds. Optional, default is 250.
- **ForceIsochronousBuffering:** boolean value indicating if buffering shall be used for audio over isochronous transports such as SCO/eSCO, LE ISOC, HCI ISOC, etc. If false, audio gaps are inserted immediately in the audio stream. If true, gaps are accumulated up to SynchroBufferMilliseconds. Optional, default is false.
- **PacketLoss:** string value indicating how to handle audio gaps due to packet loss. Optional, default is NegativeDC. The following values are available:
 - **NegativeDC:** fill gaps with negative DC (-1) value, to make the gaps easily identifiable.
 - **Silence:** fill gaps with silence (0) value, but without smooth transition at the start and end of the gap.
 - **SilenceSmooth:** fill gaps with silence (0) value, and with smooth transition at the start and end of the gap, to conceal the gap.

7.1.28. void Export(string outputFilename, "bluetooth_mobile_phone_data", ExportOptionArray exportOptions)

Equivalent export to "Bluetooth Mobile Phone Data" in the GUI.

7.1.29. void Export(string outputFilename, "bluetooth_channels", ExportOptionArray exportOptions)

Equivalent export to "Channels statistics" in the GUI.

Export options:

- `RangeStartTime`: Integer value indicating the start of the range, when `ExportRange` is true, in picoseconds. Optional, default is trace start time.
- `RangeEndTime`: Integer value indicating the end of the range, when `ExportRange` is true, in picoseconds. Optional, default is trace end time.

7.1.30. void Export(string outputFilename, "airtime", ExportOptionArray exportOptions)

Equivalent export to "Airtime" in the GUI.

Export options:

- `RangeStartTime`: Integer value indicating the start of the range, when `ExportRange` is true, in picoseconds. Optional, default is trace start time.
- `RangeEndTime`: Integer value indicating the end of the range, when `ExportRange` is true, in picoseconds. Optional, default is trace end time.

7.1.31. StringArray GetAvailableOverviews()

Get the list of available overviews.

Returns: the list of available overviews

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.32. string GetSelectedOverview()

Retrieve the active overview.

Returns: the name of the active overview

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.33. void SelectOverview(string overviewName)

This method makes a given overview active. All the methods for navigating the overview tree structure will work in the active overview so it is required to first ensure the overview is active. For example `SelectOverview("USB 3.0")` will make the USB 3.0 Overview active.

Parameters:

- `overviewName`: Name of the overview to select

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.34. `StringArray GetAvailableProtocolLayers()`

Get the list of available protocol layers.

Returns: the list of available protocol layers

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.35. `string GetSelectedProtocolLayer()`

Retrieve the current protocol layer in the active overview.

Returns: the selected protocol layer

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.36. `void SelectProtocolLayer(string protocolLayerName)`

This method will switch the protocol layer in the active overview. It is equivalent to clicking the protocol layers buttons in the GUI.

Parameters:

- `protocolLayerName`: Name of the protocol layer to select, as displayed in the popup on the button in the GUI

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.37. `string GetOverviewQuery()`

Retrieve the current query in the active overview.

Returns: the current query if any, empty string otherwise

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.38. `void SetOverviewQuery(string query)`

Applies the specified query string to the active overview. This enables filtering the overview root items with the specified conditions.

Equivalent to Search > Filter in the GUI software.

Parameters:

- `query`: the query to apply

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.39. `int OverviewRootItem()`

The root item is a conceptual item that contains all the items displayed in the overview. The overview content is

a tree structure where each item can have sub-items, called children. Fully traversing such a tree structure can be implemented as a recursive iteration. The root item is not a visible item and is just used for the purpose of iterating the tree structure.

Returns: handle to the active overview's root item

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.40. `string GetOverviewItemDescription(int itemHandle)`

This method returns the string of the Item column for a given item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: string of the Item column for a given item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.41. `StringArray GetOverviewItemsDescription(HandleArray itemHandles)`

This method returns the strings of the Item column for the specified item handles.

Parameters:

- `itemHandles`: handles of the items

Returns: strings of the Item column for the specified item handles

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.42. `long GetOverviewItemTimeInPicoseconds(int itemHandle)`

This method returns the time in picoseconds for a given item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: time in picoseconds for a given item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.43. `TimeArray GetOverviewItemsTimeInPicoseconds(HandleArray itemHandles)`

This method returns the times in picoseconds for the specified items handles.

Parameters:

- `itemHandles`: handles of the items

Returns: time in picoseconds for the specified items handles

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.44. `ByteArray GetOverviewItemData(int itemHandle)`

This method returns the data displayed in the Raw data view for the specified item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: data displayed in the Raw data view for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.45. `ByteArrays GetOverviewItemsData(HandleArray itemHandles)`

This method returns the data displayed in the Raw data view for the specified items handles.

Parameters:

- `itemHandles`: handles of the items

Returns: data displayed in the Raw data view for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.46. `string GetOverviewItemXmlReport(int itemHandle)`

This method returns the content of the Details view as XML for the specified item handle. This XML can then be decoded by using any standard XML parser.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: content of the Details view as XML for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.47. `StringArray GetOverviewItemsXmlReport(HandleArray itemHandles)`

This method returns the content of the Details view as XML for the specified items handles. This XML can then be decoded by using any standard XML parser.

Parameters:

- `itemHandles`: handles of the items

Returns: content of the Details view as XML for the specified items handles

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.48. string GetOverviewItemXmlReportFiltered(int itemHandle, StringArray fieldNameFilters)

This method returns the content of the Details view as XML for the specified item handle as above, but only the fields matching the fieldNameFilters will be returned.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`
- `fieldNameFilters`: list of inclusion filters. The filters can be specific field names, and accept wildcards such as `*` and `?`, or a regular expression, in which case the string shall start by `regex:` followed by the regular expression. More information about the supported regex format: [.NET regular expression reference](#)

Returns: content of the Details view as XML for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.49. StringArray GetOverviewItemsXmlReportFiltered(HandleArray itemHandles, StringArray fieldNameFilters)

This method returns the content of the Details view as XML for the specified items handles as above, but only the fields matching the fieldNameFilters will be returned.

Parameters:

- `itemHandles`: handle of the items
- `fieldNameFilters`: list of inclusion filters. The filters can be specific field names, and accept wildcards such as `*` and `?`, or a regular expression, in which case the string shall start by `regex:` followed by the regular expression. More information about the supported regex format: [.NET regular expression reference](#)

Returns: content of the Details view as XML for the specified items handles

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.50. int GetOverviewItemChildCount(int itemHandle)

This method returns the child count for the specified item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: child count for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.51. int GetOverviewItemChild(int itemHandle, int childIndex)

This method returns the child item handle corresponding to a given index for the specified item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`

Returns: child item handle corresponding to a given index for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.52. `HandleArray GetOverviewItemChildren(int itemHandle, int childIndex, int childCount)`

This method returns the quantity of requested children items handles starting from the specified index for the specified item handle.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`
- `childIndex`: index of the first child to be returned
- `childCount`: count of children to be returned

Returns: child item handle corresponding to a given index for the specified item handle

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.53. `HandleArray SearchOverviewItems(int itemHandle, int childIndex, int childCount, int maxDepth, StringArray descriptionFilters, StringArray fieldNameFilters, StringArray fieldValueFilters)`

This method searches the specified filters in the children items of the specified parent item, going through the specified max depth. The value and field name filters can be used as follows:

- If `fieldNameFilters` is null, then the item will be kept if any field value matches any value filter.
- If `fieldNameFilters` is not null and `fieldValueFilters` is null, then the item will be kept if any field name matches any name filter.
- If both `fieldNameFilters` and `fieldValueFilters` are not null, then both arrays must be of the same size. If a field matches a field name filter, the field's value will be matched against the corresponding index in the value filters. So in other terms, `fieldNameFilters[0]` is related to `fieldValueFilters[0]`, `fieldNameFilters[1]` is related to `fieldValueFilters[1]`, etc. The item will be retained if any field matches any filter.

Parameters:

- `itemHandle`: handle of the item, either from `OverviewRootItem()` or `GetOverviewItemChild()`
- `childIndex`: index of the first child of the item to search from
- `childCount`: quantity of children of the item to search into
- `maxDepth`: maximum depth to go through the children. For example, a value of 1 will return only the direct children of the item.
- `descriptionFilters`: inclusion filters matching the item description. The filters accepts wildcards such as * and ?, or a regular expression, in which case the string shall start by regex: followed by the regular expression. Can be null if no filtering is required.
- `fieldNameFilters`: inclusion filters matching a fields names in the report. The filters accepts wildcards such as * and ?, or a regular expression, in which case the string shall start by regex: followed by the regular expression. Can be null if no filtering is required.

- `fieldValueFilters`: inclusion filters matching fields values in the report. The filters accepts wildcards such as * and ?, or a regular expression, in which case the string shall start by regex: followed by the regular expression. Can be null if no filtering is required. More information about the supported regex format: [.NET regular expression reference](#)

Returns: children items handles for the specified parent item handle, through the specified max depth, and matching the specified filters

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.54. void ReleaseAllOverviewItemHandles()

This method releases all item handles allocated by `GetOverviewItemChild()` or `GetOverviewItemChildren()`. It is useful to release allocated handles after allocating a large quantity of handles to avoid excessive memory consumption in the analysis application.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.55. void SelectOverviewItem(int itemHandle)

Highlights the specified item in the active overview.

Parameters:

- `itemHandle`: Item to be highlighted

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.56. int GetSelectedOverviewItem()

Returns the selected overview item handle.

Returns: Item handle highlighted in the active overview

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.57. void GetLogicSignalsState(long timeInPicoseconds, out int logicSignalsState)

This method returns the state of all logic signals at the specified time. The returned logic signals state contain the state of all signals. Bit 0 corresponds to logic signal 0, bit 1 to logic signal 1, etc.

Parameters:

- `timeInPicoseconds`: time at which the logic signals state shall be retrieved
- `logicSignalsState`: state of all logic signals at the specified time

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.58. void FindLogicSignalsTransition(long fromTimeInPicoseconds, long toTimeInPicoseconds, int signalsMask, LogicSignalTransitionType transitionType, out int foundLogicSignalsState, out long foundTimeInPicoseconds)

This method searches a certain transition in the specified time range, for the specified signals. The desired signals should be set to 1 in the signalsMask to be included. For example if signals 0, 2, and 3 should be searched, the corresponding binary mask shall be 1101 (bits 0, 2 and 3 set), which is 0xD in hexadecimal.

Parameters:

- fromTimeInPicoseconds: time from which to search for the logic transition
- toTimeInPicoseconds: time to which to search for the logic transition
- signalsMask: mask indicating which signals to search
- transitionType: specifies if the searched transition is rising edge, falling edge, or any edge
- foundLogicSignalsState: state of all logic signals at the specified time
- foundTimeInPicoseconds: time at which the condition is found

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.59. RunningTaskArray GetRunningTasks()

Returns the currently running tasks of the software, such as recording, saving, filtering, loading, etc.

Equivalent to the Tasks view.

Returns: Array of running tasks.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.60. void AbortRunningTask(string name)

Aborts the specified task if running.

Equivalent to canceling a task in the Tasks view.

Parameters:

- name: Name of the task to be aborted

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.61. ByteArray GetSettings()

Retrieve the current settings.

Equivalent to File > Import and Export settings > Export settings in the GUI software.

Returns: Settings in binary form

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.62. void ConfigureSettings(ByteArray settings)

Configure the software with the specified settings. The provided settings can be saved directly from the GUI software in order to create a few different configurations, or can be created dynamically.

Equivalent to File > Import and Export settings > Import settings in the GUI software.

Parameters:

- `settings`: content of a file created from File > Import and Export settings > Export settings

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.63. void CancelUserInteraction()

Cancels any message that might block the UI waiting for an user input.

Throws: `OperationFailed` — if the operation cannot be completed.

7.1.64. void InsertComment(string comment, string overviewName)

Insert a comment item in the selected overview at the current time. This function can only be called during recording. **Obsolete:** This method is deprecated, use `InsertMessage` instead.

Parameters:

- `comment`: text of the comment
- `overviewName`: name of the overview where the comment shall be inserted

Throws: `OperationFailed` — if the operation cannot be completed.

7.2. Bluetooth functions (BluetoothAnalyzerRemoteControl.ice)

Bluetooth-specific remote-control operations, available while capturing Bluetooth.

7.2.1. void SplitTraceFileAndContinueRecording(string filename)

This method will save the ongoing trace to the specific file path and will continue recording. It is equivalent to clicking the "Save & Continue" button in the GUI.

Parameters:

- `filename`: path to the file to be saved, based on the remote computer

Throws: `OperationFailed` — if the operation cannot be completed.

7.2.2. void AddLinkKey(long bdaddr1, long bdaddr2, LinkKeyByteArray linkKey)

This method will add the specified link key to the devices database for the specified connection. The views will not reload after the call of this function. If a loaded trace is affected by this new link key, it shall be reloaded manually.

Parameters:

- `bdaddr1`: BDADDR of the first device
- `bdaddr2`: BDADDR of the second device
- `linkKey`: 16 bytes composing the link key

Throws: `OperationFailed` — if the operation cannot be completed.

7.2.3. `void ConfigureDeviceFilter(DeviceFilterMode mode, DeviceAddressArray deviceAddrs)`

This method sets the device filter. It is equivalent to specifying a device filter from the GUI.

Parameters:

- `mode`: specifies if the filter is keep all, exclude background, keep only or keep involving
- `deviceAddrs`: list of devices to keep. No device shall be specified in case of keep all and exclude background.

Throws: `OperationFailed` — if the operation cannot be completed.

7.2.4. `void GetSpectrumRssi(long timeInPicoseconds, int rfChannelNumber, out double rssi, out long startTimeInPicoseconds, out long stopTimeInPicoseconds)`

This method returns the RSSI level in the captured spectrum at the specified time and on the specified channel. The RSSI value is in dBm.

Parameters:

- `timeInPicoseconds`: Time at which the spectrum RSSI shall be retrieved
- `rfChannelNumber`: RF channel number, from 0 to 78
- `rssi`: Signal strength in dBm
- `startTimeInPicoseconds`: Start time in picoseconds of the RSSI sample
- `stopTimeInPicoseconds`: Stop time in picoseconds of the RSSI sample

Throws: `OperationFailed` — if the operation cannot be completed.

7.2.5. `void GetSpectrumRssiRange(long fromTimeInPicoseconds, long toTimeInPicoseconds, int rfChannelNumber, out RssiArray rssi, out long startTimeInPicoseconds, out long stopTimeInPicoseconds)`

This method returns the RSSI level in the captured spectrum in the specified time range and on the specified channel.

Parameters:

- `fromTimeInPicoseconds`: time from which to get RSSI samples
- `toTimeInPicoseconds`: time to which to get RSSI samples
- `rfChannelNumber`: RF channel number, from 0 to 78
- `rssi`: Array of signal strength samples in dBm

- `startTimeInPicoseconds`: Start time in picoseconds of the first RSSI sample
- `stopTimeInPicoseconds`: Stop time in picoseconds of last RSSI sample

Throws: `OperationFailed` — if the operation cannot be completed.

7.2.6. ChannelsSummary GetChannelsSummary(long fromTimeInPicoseconds, long toTimeInPicoseconds)

This method returns the Bluetooth channels summary as displayed in the Instant Channels view for the specified time range.

Parameters:

- `fromTimeInPicoseconds`: time from which to get channels summary
- `toTimeInPicoseconds`: time to which to get channels summary

Returns: Bluetooth channels summary as displayed in the Instant Channels view

7.2.7. void ExportAudio(string outputDirectory)

This method exports all audio available in the trace to the specified directory. It is equivalent to the File menu > Export > Bluetooth Audio function in the GUI. See the [Export\(\)](#) function for more options on the audio export parameters.

Parameters:

- `outputDirectory`: path to the directory to export audio files, based on the remote computer

7.3. Wi-Fi functions (EthernetAnalyzerRemoteControl.ice)

Wi-Fi capture remote-control operations, available on analyzers that capture Wi-Fi.

7.3.1. void AddWifiKeyByApSsid(string apSsid, string key)

Adds a Wi-Fi key for the access point identified by its SSID, so that traffic encrypted for that network can be decrypted.

Parameters:

- `apSsid`: SSID (network name) of the access point.
- `key`: the Wi-Fi key (passphrase or hexadecimal key) for that network.

Throws: `OperationFailed` — if the operation cannot be completed.

7.3.2. void AddWifiKeyByApMacAddr(long apMacAddr, string key)

Adds a Wi-Fi key for the access point identified by its MAC address (BSSID), so that traffic encrypted for that network can be decrypted.

Parameters:

- `apMacAddr`: MAC address (BSSID) of the access point.

- **key:** the Wi-Fi key (passphrase or hexadecimal key) for that network.

Throws: `OperationFailed` — if the operation cannot be completed.

7.3.3. `void ConfigureDeviceFilter(DeviceFilterMode mode, DeviceAddressArray deviceAddrs)`

This method sets the device filter. It is equivalent to specifying a device filter from the GUI.

Parameters:

- **mode:** specifies if the filter is keep all, exclude background, keep only or keep involving
- **deviceAddrs:** list of devices to keep. No device shall be specified in case of keep all and exclude background.

Throws: `OperationFailed` — if the operation cannot be completed.

7.4. WPAN / 802.15.4 functions (WpanAnalyzerRemoteControl.ice)

WPAN (IEEE 802.15.4) capture remote-control operations, including Thread and Zigbee key management.

7.4.1. `void AddThreadNetworkMasterKey(int panId, ByteArray key128, long sequenceCounter)`

Adds a Thread network master key used to decrypt traffic of the specified Thread network.

Parameters:

- **panId:** PAN identifier of the Thread network.
- **key128:** the 128-bit (16-byte) Thread master key.
- **sequenceCounter:** key sequence counter associated with the master key.

7.4.2. `void RemoveThreadNetworkAllMasterKeys()`

Removes all previously added Thread network master keys.

7.4.3. `void RemoveThreadNetworkMasterKeys(int panId)`

Removes all Thread network master keys previously added for the specified PAN.

Parameters:

- **panId:** PAN identifier whose master keys are removed.

7.4.4. `void RemoveThreadNetworkMasterKey(int panId, ByteArray key128)`

Removes a specific Thread network master key previously added for the specified PAN.

Parameters:

- **panId:** PAN identifier of the Thread network.
- **key128:** the 128-bit (16-byte) master key to remove.

7.4.5. void AddZigbeeApsKey(ByteArray destinationAddress64, ByteArray sourceAddress64, ByteArray key128)

Adds a Zigbee APS link key used to decrypt traffic between the two specified devices.

Parameters:

- `destinationAddress64`: 64-bit IEEE address of the destination device.
- `sourceAddress64`: 64-bit IEEE address of the source device.
- `key128`: the 128-bit (16-byte) APS link key.

7.4.6. void RemoveZigbeeApsAllKeys()

Removes all previously added Zigbee APS link keys.

7.4.7. void RemoveZigbeeApsKeys(ByteArray destinationAddress64, ByteArray sourceAddress64)

Removes all Zigbee APS link keys previously added for the specified device pair.

Parameters:

- `destinationAddress64`: 64-bit IEEE address of the destination device.
- `sourceAddress64`: 64-bit IEEE address of the source device.

7.4.8. void RemoveZigbeeApsKey(ByteArray destinationAddress64, ByteArray sourceAddress64, ByteArray key128)

Removes a specific Zigbee APS link key previously added for the specified device pair.

Parameters:

- `destinationAddress64`: 64-bit IEEE address of the destination device.
- `sourceAddress64`: 64-bit IEEE address of the source device.
- `key128`: the 128-bit (16-byte) APS link key to remove.

7.4.9. void AddZigbeeNetworkKey(int panId, ByteArray key128)

Adds a Zigbee network key used to decrypt traffic of the specified Zigbee network.

Parameters:

- `panId`: PAN identifier of the Zigbee network.
- `key128`: the 128-bit (16-byte) Zigbee network key.

7.4.10. void RemoveZigbeeNetworkAllKeys()

Removes all previously added Zigbee network keys.

7.4.11. void RemoveZigbeeNetworkKeys(int panId)

Removes all Zigbee network keys previously added for the specified PAN.

Parameters:

- `panId`: PAN identifier whose network keys are removed.

7.4.12. void RemoveZigbeeNetworkKey(int panId, ByteArray key128)

Removes a specific Zigbee network key previously added for the specified PAN.

Parameters:

- `panId`: PAN identifier of the Zigbee network.
- `key128`: the 128-bit (16-byte) network key to remove.

Revisions History

Date	Rev	Changes
June 21, 2007	1.0	Original version.
Aug. 16, 2019	2.0	New format.
Feb. 17, 2023	2.1	Added Export and CancelUserInteraction methods.
Aug. 12, 2023	2.2	Updated Export options for bluetooth_audio.
Nov. 26, 2025	2.3	Added Export modes (bluetooth_channels, throughput and airtime).
June 10, 2026	2.4	Added the Python/C# SDKs, the Using with AI agents and Overview queries sections.

Copyright and Intellectual Property

Copyright Disclaimer

Copyright © Ellisys 2025. 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@ellisis.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.