


AICONTEXTBUILDER

# AICB - Desktop Application

Describes AICB V0.5.505.1

As of 2026-10-26

Dadera · www.dadera.de · aicb@dadera.de

# Contents


1 Startup, window and navigation

- 1.1 Starting AICB
- 1.2 The main window
- 1.3 Global error handling
- 1.4 Closing AICB
- 1.5 Drag and drop a solution
- 1.6 How this manual names the parts of the interface
- 1.7 The shell
- 1.8 The Start page
- 1.9 Global controls

2 Workspace, sessions and snapshots

- 2.1 The solution list
- 2.2 Overview
- 2.3 Profiles
- 2.4 Sessions
- 2.5 Snapshots
- 2.6 Working over time

3 The Context Builder: layout and solution tree

- 3.1 From opening a solution to the exported document
- 3.2 Solution tabs
- 3.3 Layout modes: the surface starts minimal
- 3.4 The workspace layout
- 3.5 The Solution Tree in detail
- 3.6 The selection chip bar
- 3.7 The loading overlay
- 3.8 Keyboard shortcuts of this surface

4 The Context Builder: configuration panels and code editor

- 4.1 The six configuration panels
- 4.2 `Prompt` : the five prompt fields
- 4.3 `Sections` : the tag schema per output section
- 4.4 `Graph Selection` : which graphs go into the document
- 4.5 `Selection Engine` : presets and expansion per detail level
- 4.6 `Test Description`
- 4.7 `Export Output` : the notation
- 4.8 The `Details` tab: a built-in code workspace
- 4.9 `Selection Engine` panel vs. symbol inspector

5 The Context Builder: document, runs and results

- 5.1 Finding your way
- 5.2 The run template and the run actions
- 5.3 The `MD Input` tab - the generated document
- 5.4 The `Reasoning` tab - what the model did
- 5.5 The `Active Run` tab - progress while a run is running
- 5.6 Token, cost and progress displays
- 5.7 From solution to context document, step by step
- 5.8 Run types: what is released

6 Insights in the desktop app

- 6.1 Where the Insights tab lives
- 6.2 Categories, principles and severities

6.3 Running the analysis
6.4 The health strip
6.5 The finding cards
6.6 Sending findings to your LLM
6.7 Dismissing, marking as intentional, and restoring
6.8 Queuing a finding as work
6.9 Where the state is stored
6.10 Empty and loading states
6.11 Defaults and limits

7 Runs and language models

7.1 What a run is
7.2 Run types
7.3 Sending a context document
7.4 What a prompt contains
7.5 Model profiles
7.6 Token counting and prices
7.7 Watching and reviewing a run
7.8 Step by step: connect your first model and send a context document

8 Settings

8.1 Where the settings pages live
8.2 Where a setting is stored and when it takes effect
8.3 The master/detail pattern
8.4 General
8.5 Storage
8.6 Model Profiles
8.7 Prompts
8.8 Test Descriptions
8.9 Layer Profiles
8.10 Exclude Namespaces
8.11 Quality Profiles
8.12 Test Profiles
8.13 MD Profile
8.14 Tag Schemata
8.15 Detail Presets
8.16 Other File Types
8.17 Expansion Strategies
8.18 Compression Rules
8.19 Pipeline Profiles
8.20 Constellations
8.21 About
8.22 Configuring AICB for a team

9 MCP Profiles, MCP Usage and Templates

9.1 MCP Profiles
9.2 MCP Usage
9.3 Templates
9.4 Layer Profiles, Exclude Namespaces and Test Profiles appear twice
9.5 Where settings are stored and when they take effect

10 Dialogs

10.1 How dialogs behave
10.2 Message dialogs
10.3 Input dialogs
10.4 Recovery and safety dialogs
10.5 Import and data dialogs

10.6 Editors
10.7 Information windows
10.8 Native file dialogs

11 Troubleshooting the desktop app

11.1 First start checklist
11.2 Where errors appear
11.3 The `Unexpected error` dialog
11.4 The log file `aicb.log`
11.5 Problems and their solutions

# 1 Startup, window and navigation

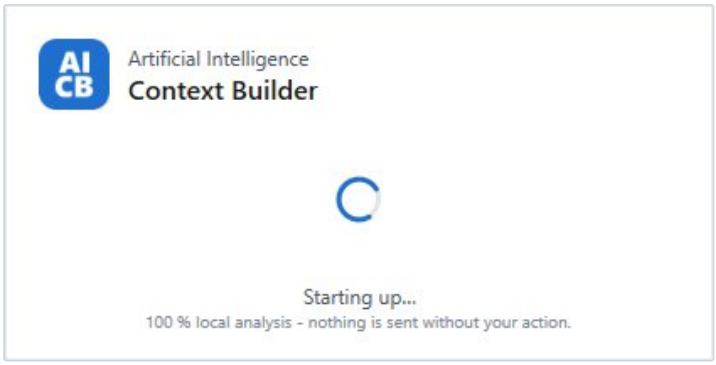
This chapter describes what happens when you start AICB, how the main window is built, how you move between its areas, and which keys are available everywhere. It also fixes the vocabulary this manual uses for the parts of the interface.

## 1.1 Starting AICB

### The splash window

As soon as you launch AICB, a small frameless window appears in the center of the screen while the application prepares itself. It shows the AICB logo, the product name in two tones, a turning ring and the line `Starting up…`, followed by the note `100 % local analysis - nothing is sent without your action.`

The splash runs on its own thread, so the ring keeps turning even while the main window is still being built. It closes as soon as the main window is shown. If startup fails, the splash closes too and AICB reports the problem.




The splash window while AICB starts

Note: The splash follows the Windows system theme, read directly from Windows. If you have pinned AICB to the opposite theme of your Windows installation, the splash still appears in the system theme for the few seconds it is up; the main window then switches to the theme you chose.

### The startup sequence

AICB works through the following steps before you see the main window. Each of them can end the start:

| Step | What happens |
|---|---|
| 1 | The splash window appears. |
| 2 | AICB looks for an MSBuild installation (Visual Studio, `vswhere.exe` or a .NET SDK). Without one, no C# solution can be loaded: the dialog `MSBuild not found` appears and the application exits. |
| 3 | The application settings file is read. If it cannot be read, AICB continues on defaults and reports this in the startup notice line. |
| 4 | The database schema is checked and, if necessary, migrated. A database written by a newer AICB version or a failed migration stops the start (see the table below). |
| 5 | Runs that were still marked as running when the previous session ended are offered for recovery. |
| 6 | The shell is initialized. On a large database this is the step that takes most of the startup time. |
| 7 | Your theme, font size and UI zoom are applied. |
| 8 | The main window appears, and the splash closes. |
| 9 | An automatic backup starts in the background if one is due. |

The startup time depends mainly on the size of your database. The splash makes the wait visible, but it does not shorten it.

Note: If automatic backup is enabled and the backup interval has elapsed, AICB writes the backup in the background once the main window is open, so it does not delay the start. A backup that fails does not advance the last-run time, so it runs again on the next start until the cause is fixed; the startup notice line reports it.

A dialog that appears during startup - a database newer than the build, a failed migration, unfinished runs, MSBuild not found, an unexpected startup error - is shown in front of the splash with the keyboard focus and its own taskbar button. The splash disappears while the dialog is open.

### Messages during startup

| Message | When it appears | What you can do |
|---|---|---|
| `MSBuild not found` | No MSBuild installation was found. | Install Visual Studio or the .NET SDK and start AICB again. AICB exits. |
| `Database is newer than this build` | The database was written by a newer AICB version than the one you started. AICB refuses to write to it so that an older version cannot silently save the outdated data shape back. The dialog names both schema versions. | Update AICB, or point the database path under Settings › Storage at a different file. AICB exits. |
| `Database migration failed` | The database schema could not be updated. | Make sure the database path under Settings › Storage is reachable and writable, then start AICB again. AICB exits. |
| `Unfinished runs from previous session` | Runs were still marked as running when the previous session ended - typically after a crash, a hard kill or a power loss. The dialog lists up to five of them and adds `... and N more` for the rest. | `Yes` marks them as cancelled, `No` pauses them for a later resume, `Cancel` leaves them unchanged so you can decide later. |
| `Crash recovery` (warning) | The recovery check itself failed. | AICB starts anyway; the unfinished runs remain marked as running and can be offered again on the next start. |
| `Application settings` in the startup notice line | The settings file could not be read. AICB then runs on defaults - including the database path, which may make your sessions, solutions and runs appear to be gone. | Fix or replace the settings file, then restart AICB. |
| `Automatic backup at startup` in the startup notice line | The backup archive is incomplete ( `Partial` ), could not be written at all ( `Failed` ), or was written but its completion could not be recorded. | Check the backup folder and its free space, then restart AICB. |

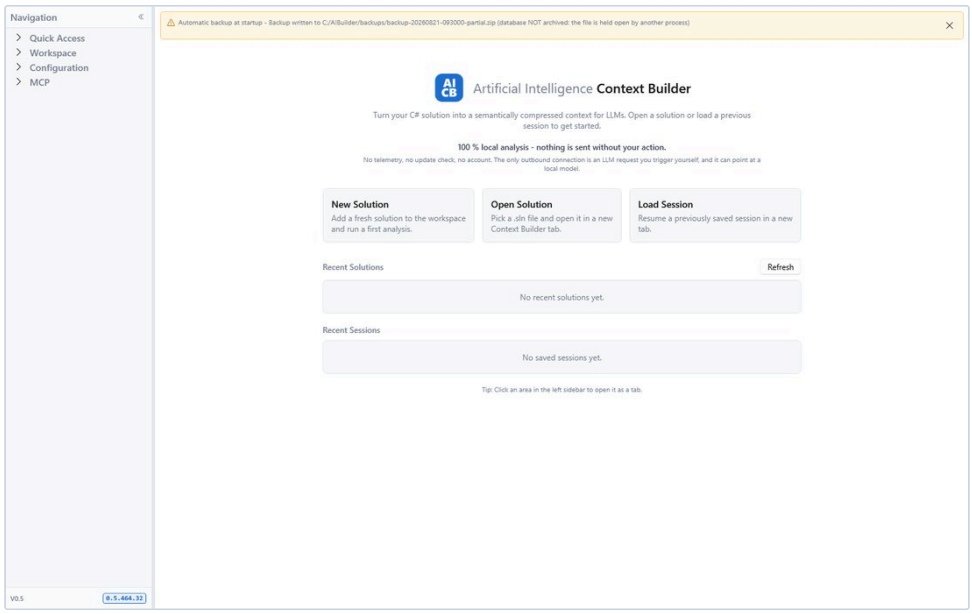
Note: The `On startup` option under Settings › General currently has no effect. Session restore is a planned feature; AICB always opens on the `Start` page with no tab open.

### The startup notice line

Some outcomes happen before any window exists, so AICB parks them and shows them in a tinted notice line at the very top of the right-hand area, above the `Start` page or the tab strip. The line appears only when there is something to say; on a normal start it takes no space at all. It reports:

| Source | Text prefix | When |
|---|---|---|
| Automatic backup | `Automatic backup at startup -` | The archive is incomplete or failed, or was written while its completion record could not be saved. A clean backup, and a start on which none was due, stay silent. |
| Application settings | `Application settings -` | The settings file could not be read. |

If both occur on the same start, the line shows both, one per line. The `x` button on the right closes the line. Nothing is saved: if the problem is still there on the next start, the line reports it again. The tooltip of the line explains what it covers.




The startup notice line reporting an incomplete automatic backup

## 1.2 The main window

### Frame, size and backdrop

AICB starts maximized, on a 900 × 500 restore size, centered on the screen. The window uses the Windows 11 Mica backdrop; on Windows 10 it falls back to a solid color.

Note: The window size, position and state are not remembered. A window you restore and resize during a session is maximized again on the next start.

### The title bar

The title bar shows the AICB icon and the product name in two tones: `Artificial Intelligence` in a muted tone, followed by `Context Builder` in semibold. The full name `Artificial Intelligence Context Builder` is the window title that the taskbar and Alt+Tab display.

The version number is deliberately not in the title bar. It sits in the footer of the navigation sidebar, at the bottom left: the major-line label `V0.5` on the left, and a pill with the exact four-part build version on the right. The version is read from the program file at runtime; if it cannot be determined, the pill shows `0.0.0.0`.

### Theme

AICB ships three theme values, set under Settings › General in the group `Application`:

| Value | Meaning |
|---|---|
| `Light` | Always uses the light palette. |
| `Dark` | Always uses the dark palette. |
| `System Default` | Follows the Windows app theme. This is the default. |

The theme switch is live: the Fluent controls, the window backdrop and the application's own color tokens all change without a restart, and both palettes are complete, so no area keeps the old colors. The change applies when you save the settings.

Note: An already-open dropdown popup may keep the colors it was opened with until you close and reopen it.

### Font size and UI zoom

`Font size` under Settings › General is a zoom over the whole interface - text, controls and spacing together - not just the base font. It has exactly three steps:

| Value | Factor |
|---|---|
| `Small` | 0.90 |
| `Medium` | 1.00 (default) |
| `Large` | 1.15 |

The checkbox `Scale with window size` (on by default) adds a gentle responsive component: the zoom follows the window width relative to 1920 px, limited to the range 0.92 to 1.12. The combined factor is finally limited to 0.80 to 1.35, so nothing becomes extreme. The title bar is deliberately not scaled; everything below it is.

The `Font family` dropdown sets the UI font. Only a plain family name is honored; a value that looks like a file path or a URI is rejected and the system UI font is used instead.

Keyboard zoom works on top of the saved size and applies for the current run of the application only:

- `Ctrl+Plus` or `Ctrl+Add` scales the window up one step.
- `Ctrl+Minus` or `Ctrl+Subtract` scales it down one step.
- `Ctrl+0` or `Ctrl+NumPad0` returns to the size saved under Settings › General, which is not necessarily 100 %.

The zoom uses the same three tiers as the setting ( `Small` , `Medium` , `Large` ). At the smallest or largest tier, further presses in that direction change nothing. Choosing a size in the settings discards a zoom that is currently in effect, so the setting always wins.

### Remembered layout sizes

AICB remembers the positions of its splitters, so you do not have to rebuild your layout on every start. There is one shared layout; positions saved by older versions are carried over once.

The sidebar is the value you will notice most in this chapter: its width, and whether it is collapsed, are both remembered, as is the width of the solutions list on the Snapshots sub-tab. Nine layout values are actively stored across the application: the shell-sidebar width and collapsed state, the Settings sub-sidebar width, the Context Builder tree-sidebar width, collapsed state and toolbar state, the Sessions detail-column width, the Snapshots solutions-column width and the Run Templates master-column width.

Note: A splitter position is written when you release the splitter, but it is only read when the view loads. A change therefore takes effect the next time that view is opened, not immediately.

## 1.3 Global error handling

If an error escapes a command, an event handler or a background task, AICB does not close. It logs the error and shows the dialog `Unexpected error` with the text:

> `An unexpected error occurred.`
>
> `The application is still running, but its state may be inconsistent. Save any open sessions and restart.`
>
> `Details: ...`

The reason AICB stays open is your work: several Context Builder tabs can hold unsaved sessions in memory, and terminating would discard them without asking. After such a message, save your sessions and restart the application.

Two rules keep the dialogs from taking over the screen:

- At most three error dialogs are shown per run of the application. After that, further errors are only logged.

- Each distinct error is shown only once. Two different errors that happen to carry the same message are still treated as different errors.

Errors that the runtime reports while it is already shutting down, and unobserved background-task exceptions, are logged but cannot be recovered.

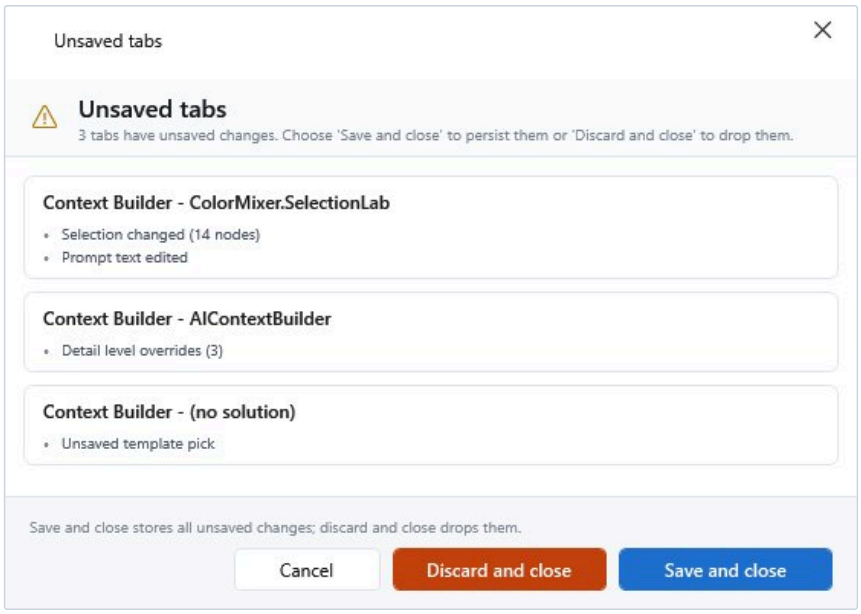
## 1.4 Closing AICB

When you close the main window, AICB checks every Context Builder sub-tab for unsaved changes:

1. If no sub-tab has unsaved changes, AICB closes without a prompt.
2. Otherwise the dialog `Unsaved tabs` appears with one card per affected tab, listing what would be lost. Its subtitle names the number of tabs, for example `2 tabs have unsaved changes. ...`.
3. Choose one of the three buttons:

| Button | Effect |
|---|---|
| `Save and close` | Saves all unsaved changes and closes. Existing sessions are updated silently; only a session without a saved record asks for its name. If you cancel one of those name dialogs, the application stays open. |
| `Discard and close` | Closes without saving. This cannot be undone. |
| `Cancel` | Cancels the close; you are back in the application with everything still there. |

The footer of the dialog repeats the rule: `Save and close stores all unsaved changes; discard and close drops them.`




The Unsaved tabs dialog when closing the application

Note: Closing the whole application always asks, regardless of the `Tab-close confirmation` setting under Settings › General. That setting only controls the confirmation for closing an individual tab.

If the check for unsaved changes itself fails, AICB cancels the close and shows the error dialog `Close Application`, so that no work is lost silently. Save your sessions manually and close again.

## 1.5 Drag and drop a solution

You can open a solution by dropping it onto the AICB window. The whole window is a drop target.

1. Drag a file onto the window. AICB accepts exactly one file with the extension `.sln`, `.slnx` or `.slnf` that exists on disk. Anything else shows no drop cursor.
2. Drop the file. AICB accepts the drop only while that `.sln`, `.slnx` or `.slnf` file still exists at the dropped path. The recovery flow for a moved file is available from `Open Solution` and Recent Solutions, not from drag and drop.

If the drop fails, the error dialog `Open Solution` appears with the message `Could not open solution:` followed by the technical detail.

## 1.6 How this manual names the parts of the interface

AICB's interface has several nested levels. This manual uses one word per level:

| Level | Term | What it is | Examples |
|---|---|---|---|
| 1 | window | the application window | the main window |
| 2 | tab | an entry in the top tab strip | `Start`, `Settings`, a solution tab |
| 3 | page | a whole surface with its own header, reached from the sidebar | `Layer Profiles`, `General`, `MCP Usage` |
| 4 | sub-tab | a tab inside a page or surface | `Facets` in the MCP profile editor, `Details` in the Semantic Context surface |
| 5 | panel | a delimited block inside a page, with its own heading | `Prompt`, `Sections`, `Graph Selection` |
| 6 | group | a group inside a panel, without a heading of its own class | the `Export content` group in the template editor |

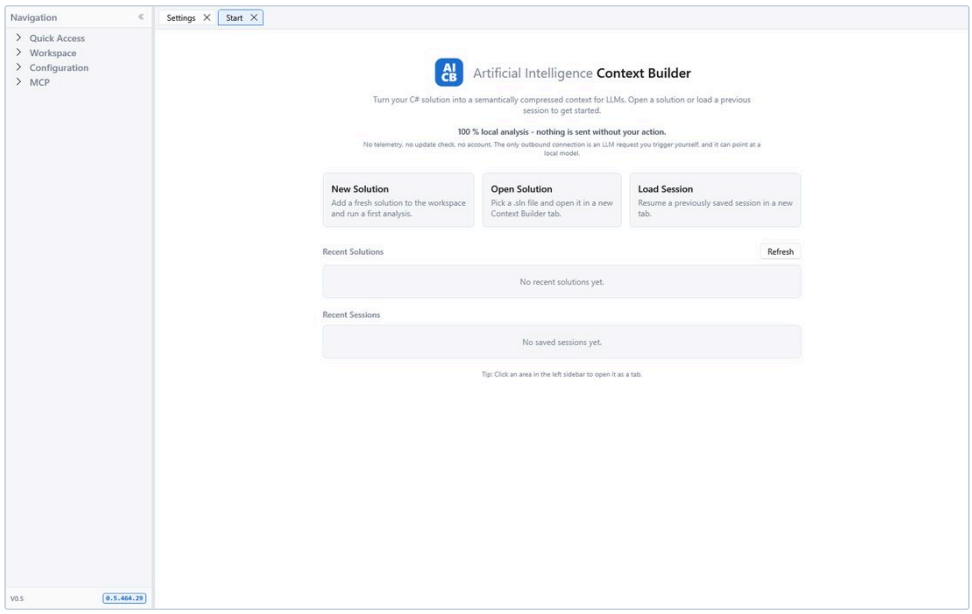
The product itself uses the word `area` for a sidebar entry: the tip at the bottom of the `Start` page reads `Tip: Click an area in the left sidebar to open it as a tab.` An area is a page in the table above, and the result of clicking it is a tab.

The large working surface of the Context Builder is called the **Semantic Context surface** in this manual. It is too large to be a page and is not a tab; the name is a documentation term, not a label on screen.

## 1.7 The shell

The shell is the frame around every page. It consists of three columns - the navigation sidebar on the left, a narrow splitter, and the content area on the right - plus the startup notice line above the content:

- The sidebar is 220 px wide by default, with a minimum of 160 px. It can be resized with the splitter.
- The splitter is 4 px wide.
- The content area has a minimum width of 320 px.
- The right-hand side has two rows: the startup notice line (see "Starting AICB"), and below it either the `Start` page - when no tab is open - or the tab strip with the tab content.




The main window with the navigation sidebar on the left and two open tabs

### Sidebar groups and entries

The sidebar is a tree with four groups and eight entries. Group rows are labels, not buttons; only the entries open a tab. Clicking an entry that is already open activates its tab instead of opening a second one.

| Group | Entry | What it opens |
|---|---|---|
| `Quick Access` | `Start` | The Start page: open actions, recent solutions and recent sessions. |
| `Workspace` | `Context Builder` | The Context Builder, where solutions are loaded and worked on. |
| `Workspace` | `Workspace` | The Workspace page, where all known solutions are managed. |
| `Workspace` | `Run Templates` | The run templates, the working material of the Workspace. |
| `Configuration` | `Templates` | The context templates. |
| `Configuration` | `Settings` | The application settings. |
| `MCP` | `MCP Profiles` | The MCP profiles. |
| `MCP` | `MCP Usage` | The MCP tool-call telemetry. |

The tooltip of the tree explains the click behavior: `Click an entry to open the matching area as a tab (or activate it if already open).`

The header of the sidebar carries the label `Navigation` and a button that collapses the sidebar. Its footer shows the version, as described under "The main window".

### Collapsing the sidebar

Press `Ctrl+B`, or use one of the two icon buttons, to collapse and expand the sidebar. Collapsed, the column shrinks to a 36 px rail that contains an expand button and the word `Menu`, turned by 90 degrees. The splitter is hidden while the rail is shown, so the rail cannot be dragged wider.

Expanding restores the width you last dragged (the default is 220 px, the minimum 160 px). Both the width and the collapsed state are remembered.

### The tab strip and the tab content

AICB has two tab levels:

- **Tabs** in the top strip. There is at most one tab per sidebar entry, managed by the shell.
- **Sub-tabs** inside a page. Only the Context Builder has a second level.

Every tab carries a close button on its right-hand side. The tooltip of a tab reads `Switch to this area's tab. The close button on the right closes it.`; the close button's tooltip is `Close tab (Ctrl+W)`.

Page content is long-lived: closing a top-level tab keeps its page view model, so reopening it preserves application state such as the current selection and expanded tree nodes. The view itself is recreated, however, so its scroll position is not restored. The Context Builder is the exception: closing its top-level tab disposes its solution sub-tabs and releases the loaded solutions.

Three pages refresh when you activate them, so what you see is current:

| Tab | What happens on activation |
|---|---|
| `Start` | The recent solutions and recent sessions lists are reloaded. |
| `Workspace` | The solutions list and the embedded sessions and snapshots are reloaded. |
| `MCP Usage` | The telemetry is reloaded, because the MCP server writes into the same database while the application runs. |

When you navigate from outside the tab strip - a sidebar entry or a keyboard shortcut - AICB moves the keyboard focus into the newly activated tab. Without that, a sequence like "click `Context Builder` in the sidebar, then press `Ctrl+S`" would not work, because the focus would still sit on the sidebar. Switching between tabs inside the strip does not move the focus.

### Sub-tabs in the Context Builder

The Context Builder keeps one sub-tab per loaded solution. Its strip sits at the top of the page and contains:

- the sub-tabs themselves, each with a close button (`Close tab (Ctrl+W)`),
- the `+` button, `New Solution tab (Ctrl+N)`, which adds an empty solution tab,
- the `Save Session` button, `Save the current tab as a Session (Ctrl+S)`.

A sub-tab is labeled `(no solution)` while it is empty and shows the solution's file name once a solution is loaded. Unsaved changes add a `*` to the label.

While a model request is running, the sub-tab strip is locked, so you cannot switch away and lose the run state.

### Opening, activating and closing tabs

To open a page:

1. Click the entry in the sidebar. If no tab for it exists, AICB opens one and activates it.
2. If a tab for that entry already exists, AICB activates it instead.

To close a tab, use its close button, or `Ctrl+W`. `Ctrl+W` works in three steps, from the inside out:

1. In the code editor it closes the open file.
2. In the Context Builder it closes the frontmost solution sub-tab. When no sub-tab is left, it closes the whole Context Builder tab.
3. Elsewhere it closes the frontmost top-level tab.

When you close the last open tab, the `Start` page appears and its recent lists are reloaded. When other tabs remain, the last tab in the strip becomes active.

### Unsaved changes

A Context Builder sub-tab is considered to have unsaved changes when you modify:

- the solution tree selection: include and exclude, detail level, and manual overrides;
- one of the five prompt fields;
- the run-template selection.

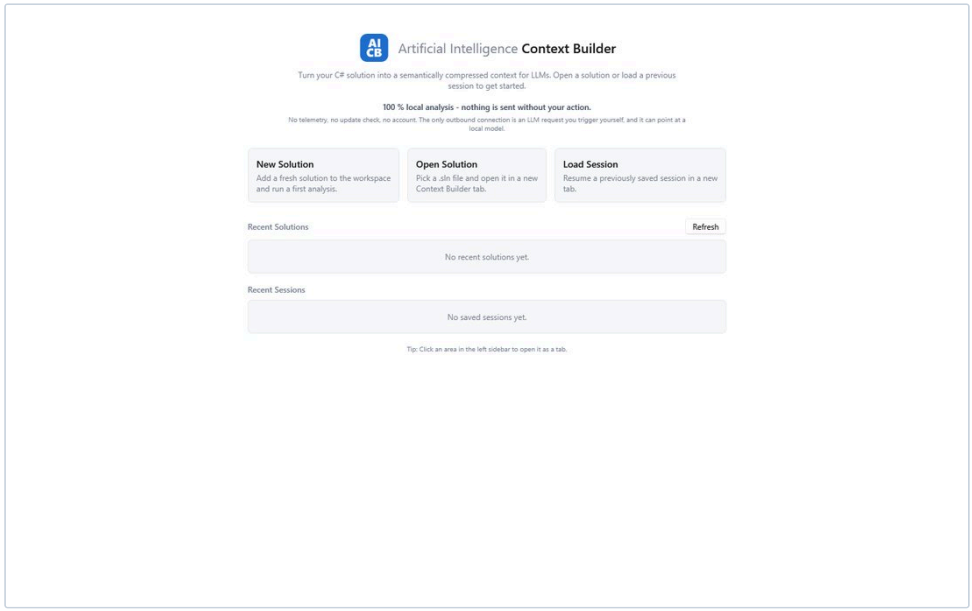
Per-slot editors in the Details panel are not tracked this way.

The two close paths use different messages. Closing one dirty solution sub-tab opens `Close Tab`, says `Tab "<label>" has unsaved changes.`, lists the changes and asks `Save before closing?`. Closing the whole Context Builder tab instead names the number of dirty sessions, for example `The tab "Context Builder" contains 2 sessions with unsaved changes.` In either case, choose `Yes` to save first, `No` to discard, or `Cancel` to keep the tab open. If you cancel one of the save dialogs that `Yes` opens, the tab stays open.

This confirmation is controlled by `Tab-close confirmation` under Settings › General (on by default). If you clear it, a dirty tab closes immediately and its unsaved changes are discarded.

## 1.8 The Start page

The `Start` page is what you see while no tab is open. The same page is also available at any time as a regular tab through `Quick Access` → `Start`.




The Start page on a fresh installation

From top to bottom it shows:

1. The hero: the AICB logo, the product name in two tones, and the sentence `Turn your C# solution into a semantically compressed context for LLMs. Open a solution or load a previous session to get started.` Below it stand the claims `100 % local analysis - nothing is sent without your action.` and `No outbound telemetry, no update check, no account. The desktop app contacts only an LLM endpoint you configure - for manual sends, connection tests or armed first-load proposals - and it can point at a local model.`
2. Three action cards:

| Card | Subtitle | What it does |
|---|---|---|
| `New Solution` | `Add a fresh solution to the workspace and run a first analysis.` | Opens the solution file picker and loads the selected file into a new Context Builder tab. |
| `Open Solution` | `Pick a .sln file and open it in a new Context Builder tab.` | Opens the same file picker and loads the selected file the same way. |
| `Load Session` | `Resume a previously saved session in a new tab.` | Opens the `Workspace` page, where you pick the session to resume. |

Note: `New Solution` and `Open Solution` currently open the same file picker and load the chosen solution identically. The conceptual difference between them is planned.

3. `Recent Solutions`, with a `Refresh` button on the right that reloads both recent lists. The list shows at most eight recently opened solutions in most-recently-used order. Each row has a colored dot, the solution name and the full `.sln` path. A warning icon appears when the file is missing on this machine; its tooltip reads `Solution file not found at saved path. Open to relocate.`, and opening the row starts the relocate flow. Double-click a row or use its `Open` button to load the solution in a new Context Builder tab. The empty state reads `No recent solutions yet.`
4. `Recent Sessions`: the same row layout, at most five entries, sorted by their last update. The subtitle of a row reads `<Solution> · <relative time>`, where the relative time is `just now`, `N min ago`, `N h ago`, `N d ago`, or a date in the form `yyyy-MM-dd` for anything older than seven days. If the solution behind a session is unknown, the row shows `(unknown solution)`. A warning icon marks a session whose solution file is missing. Double-click a row or use its `Open` button to resume the session. The empty state reads `No saved sessions yet.`
5. The footnote `Tip: Click an area in the left sidebar to open it as a tab.`

Both lists are refreshed when the page becomes visible - after you close the last tab or activate the `Start` tab - and after every open action.

#### The bundled sample solution

AICB ships a small sample solution named `ColorMixer.SelectionLab` with five projects (Contracts, Core, Application, Infrastructure and a console application). It is copied next to the program file and can be loaded for a trial run without a solution of your own.

To open it:

1. Open the `Context Builder` and go to the `Insights` tab. If no solution has been analyzed yet, the first-run card is shown.
2. Click `Open sample Solution`. AICB loads `ColorMixer.SelectionLab.sln` from the `SampleSolutions` folder next to the program file, read-only, and the insights producers run once on the first solution load after a fresh installation, so the tab is not empty.

The tooltip of the button names the file: `Loads the bundled ColorMixer.SelectionLab.sln (read-only sample) for a trial Insights run.` If the file is missing, the dialog `Sample not found` appears and points to `{exe}/SampleSolutions/`.

The sample is loaded in place from the application folder. Copy it to a writable folder first if you want to change it; a `copy into sandbox` command is planned.

### 1.9 Global controls

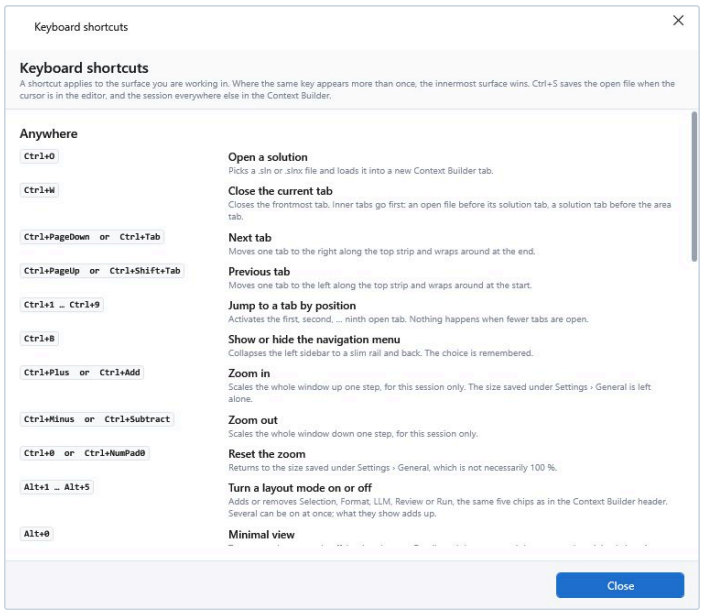
#### The F1 keyboard shortcut overview

AICB has no menu bar, so `F1` is the place to look up a key. The overview is a window titled `Keyboard shortcuts`, 760 × 660 px (minimum 520 × 360), centered over the main window, without a taskbar entry and without minimize or maximize buttons.

Under the title stands this explanation:

> `A shortcut applies to the surface you are working in. Where the same key appears more than once, the innermost surface wins. Ctrl+S saves the open file when the cursor is in the editor, and the session everywhere else in the Context Builder.`

Below it the shortcuts are grouped by scope, each as a monospace pill for the gesture, a bold action name and a muted description. The `Close` button at the bottom is the default button and the cancel button at once, so both Enter and Esc close the window.




The Keyboard shortcuts window (F1)

## Keyboard shortcuts reference

### Anywhere

| Gesture | Action | What it does |
|---|---|---|
| `Ctrl+O` | Open a solution | Picks a `.sln`, `.slnx` or `.slnf` file and loads it into a new Context Builder tab. |
| `Ctrl+W` | Close the current tab | Closes the frontmost tab. Inner tabs go first: an open file before its solution tab, a solution tab before the area tab. |
| `Ctrl+PageDown`, `Ctrl+Tab` | Next tab | Moves one tab to the right along the top strip and wraps around at the end. |
| `Ctrl+PageUp`, `Ctrl+Shift+Tab` | Previous tab | Moves one tab to the left along the top strip and wraps around at the start. |
| `Ctrl+1` … `Ctrl+9` | Jump to a tab by position | Activates the first, second, … ninth open tab. Nothing happens when fewer tabs are open. |
| `Ctrl+B` | Show or hide the navigation menu | Collapses the left sidebar to a slim rail and back. The choice is remembered. |
| `Ctrl+Plus`, `Ctrl+Add` | Zoom in | Scales the whole window up one step, for this session only. The size saved under Settings › General is left alone. |
| `Ctrl+Minus`, `Ctrl+Subtract` | Zoom out | Scales the whole window down one step, for this session only. |
| `Ctrl+0`, `Ctrl+NumPad0` | Reset the zoom | Returns to the size saved under Settings › General, which is not necessarily 100 %. |
| `Alt+1` … `Alt+5` | Turn a layout mode on or off | Adds or removes `Selection`, `Format`, `LLM`, `Review` or `Run`, the same five chips as in the Context Builder header. Several can be on at once; what they show adds up. |
| `Alt+0` | Minimal view | Turns every layout mode off, leaving the tree, Details and the generated document - the minimal view. Any chip or its Alt+digit brings the rest back. |
| `F1` | Keyboard shortcuts | Opens this overview. |

### Context Builder

| Gesture | Action | What it does |
|---|---|---|
| `Ctrl+S` | Save the session | Opens the pre-filled `Save Session` dialog for the active solution tab, including when the session already has a name. The key does nothing until a solution is loaded in the active sub-tab. |
| `Ctrl+N` | New solution tab | Adds an empty solution tab at the end of the strip and activates it. |
| `Ctrl+M` | Create the Markdown context | Builds the context document from the current tree selection, the same as the Create MD button. |
| `Ctrl+Enter` | Generate and send | Builds the context and sends it to the configured model in one step. |
| `Ctrl+Shift+C` | Copy the model response | Copies the response text. Plain `Ctrl+C` keeps its usual meaning inside text fields. |
| `F5` | Reload the solution | Re-reads the solution from disk and rebuilds the tree. |
| `Ctrl+F` | Filter the solution tree | Puts the cursor in the tree's search box and selects what is already there. |
| `Ctrl+Shift+A` | Analyze the solution | Runs all insight producers over the active solution, the same as the Analyze Solution button, from any sub-tab. |

**Code editor**

| Gesture | Action | What it does |
|---|---|---|
| `Ctrl+S` | Save the file | Writes the open source file back to disk, keeping its original encoding and line endings. |
| `Ctrl+W` | Close the file | Closes the open source file. Unsaved edits are confirmed first. |

**Settings and master data**

| Gesture | Action | What it does |
|---|---|---|
| `Ctrl+S` | Save the entry | Persists the editor on the right. Editing a built-in marks it as overridden. |
| `Ctrl+N` | New entry | Adds a custom entry to the list on the left. |
| `Ctrl+D` | Duplicate the entry | Creates a custom copy of the selected entry as a starting point. |
| `F5` | Reload | Re-reads the list from the database and discards unsaved editor changes. |

**Lists**

| Gesture | Action | What it does |
|---|---|---|
| `F5` | Refresh | Re-reads the list currently shown. |
| `Ctrl+F` | Filter the list | Puts the cursor in the surface's filter box, where it has one. |
| `Escape` | Clear the filter | Empties the filter box while the cursor is inside it. |

Note: `Ctrl+S` and `Ctrl+W` appear in more than one scope. The innermost surface wins, exactly as the overview's subtitle says. `Ctrl+Tab` is paired with `Ctrl+PageDown` and `Ctrl+Shift+Tab` with `Ctrl+PageUp` because some controls use `Ctrl+Tab` for focus navigation and would otherwise swallow it; the zoom keys are paired with their numeric-keypad counterparts for the same reason.

### Conventions used throughout the interface

A few display rules are the same everywhere, so you can read any list in AICB the same way:

- Counts are grammatically correct and use `-` when a number has not been measured yet, for example `1 session` rather than `1 sessions`.
- Enum values are shown as words: `System Default` rather than `SystemDefault`.

- In lists that use the path-elision converter, long paths are shortened to `beginning ... file name` and the full path is in the tooltip. Recent Solutions on the Welcome page is an exception: it displays the raw path, trims its end when space runs out and uses a fixed action tooltip rather than the full path.
- Every solution carries a stable color from a six-color palette, so the same solution is recognizable across all lists. The dot is never empty.
- Nodes in the Solution Tree carry a colored chip for their detail level: Compact (blue), Normal (green), Detailed (orange) and Source (red); a neutral gray marks no level.
- Times are shown relative to now: `just now`, `N min ago`, `N h ago`, `N d ago`, and a date ( `yyyy-MM-dd` ) for anything older than seven days.
- Numeric fields accept digits only; spaces and letters are blocked. While the field is focused the value is plain, and when it loses focus it is grouped with the thousands separator of your regional settings (up to two decimal places for decimal fields). A pasted value such as `4,000` is cleaned up automatically.

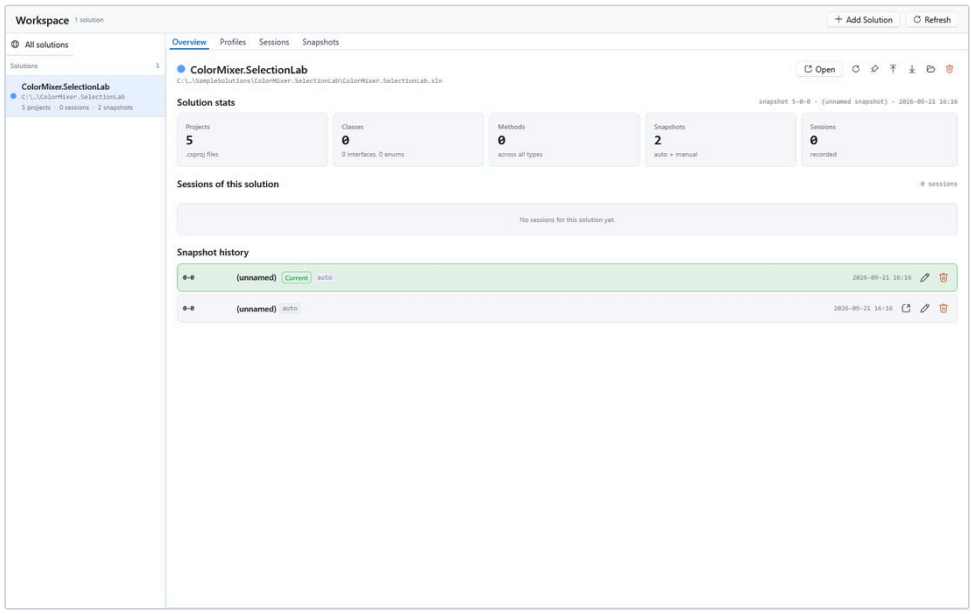
# 2 Workspace, sessions and snapshots

The Workspace page is where solutions, sessions and snapshots are managed. It is the page behind the sidebar entry `Workspace` in the `Workspace` group. A solution is the primary object here; sessions and snapshots are always tied to exactly one solution and are shown as detail regions of the page. Everything on this page is stored in the application database, so sessions and snapshots survive a restart even though open tabs do not.

Open the page from the sidebar: `Workspace` → `Workspace`. The header shows the title `Workspace`, the number of registered solutions (for example `3 solutions`), and two buttons:

| Control | Tooltip | What it does |
|---|---|---|
| `Add Solution` | `Pick a new .sln file and add it to the workspace. Triggers the first analysis.` | Opens the `Open solution` file dialog (filter `Solution files (*.sln;*.slnx;*.slnf)`) and registers the chosen solution. |
| `Refresh` | `Reload the Solutions list from the DB. (F5)` | Reloads the solution list from the database. |

The page is split into two columns: the solution list on the left (240 px wide) and the detail area on the right. The detail area has four tabs: `Overview`, `Profiles`, `Sessions` and `Snapshots`. `Overview` and `Profiles` are per solution and are hidden while the global view is active.



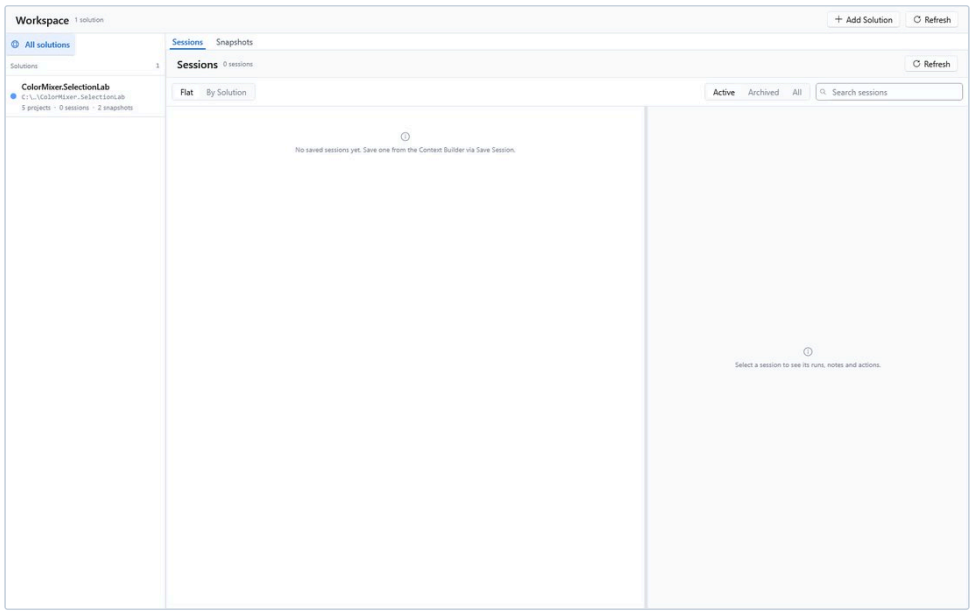

The Workspace page with the Overview of the selected solution

The page refreshes itself when you switch to it: the solution list and the embedded `Sessions` / `Snapshots` lists are reloaded in the background, so freshly saved entries appear without a manual refresh. If nothing has changed, the refresh is a no-op and your selection stays where it was.

## 2.1 The solution list

At the top of the left column is the button `All solutions` (globe icon, tooltip `Show sessions and snapshots across all solutions (global view).`). It switches to the global view:

- No solution is selected.
- `Overview` and `Profiles` are hidden.
- `Sessions` and `Snapshots` show all solutions instead of one.
- The active state of the button is highlighted.




The global view (All solutions): sessions and snapshots across all solutions

Below it is a row `Solutions` with the total count. Each list row shows:

| Element | Meaning |
|---|---|
| Color dot (9 px) | The solution's color, used consistently in the session and snapshot lists. |
| Name (bold) | The solution's display name. |
| `Loaded` pill (green) | The solution is currently open in a Context Builder tab. |
| Directory path | The folder of the `.sln`, shortened in the middle so the file name stays visible; hover for the full path. |
| Mini statistics | `N projects · M sessions · K snapshots`. The project count comes from the latest snapshot's analysis; until it has been read (or if the solution was never analyzed) it reads `- projects`. |

The selected row gets a 3 px accent bar on its left edge. Double-click a row to open the solution live in a new Context Builder tab; the row tooltip says `Double-click to open this solution live (or select it and click Open).`

If no solution is registered yet, the column shows an empty state: `No solutions yet. Use Add Solution above to register a .sln file - it stays in this list with its sessions and snapshots.`

Selecting a solution also switches the embedded `Sessions` and `Snapshots` regions to that solution and jumps back to the `Overview` tab.

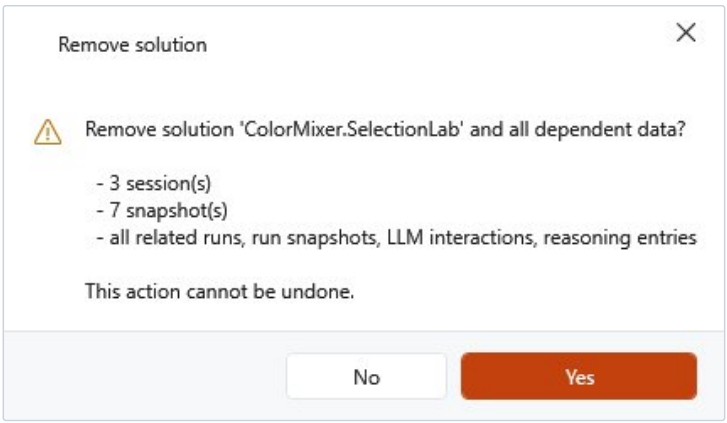
## 2.2 Overview

The `Overview` tab shows everything about the selected solution. Without a selection it shows `Select a solution on the left to view details.`

The header carries a 12 px color dot, the solution name and the solution path (shortened to about 80 characters, full path in the tooltip). On the right is one labeled primary action and six icon buttons whose labels live in their tooltips:

| Button | Tooltip | What it does |
|---|---|---|
| `Open` | `Open this solution live in a new Context Builder tab.` | Opens the solution and runs the analysis. |

| Button | Tooltip | What it does |
|---|---|---|
| Re-Analyze | `Re-Analyze: re-run the open-solution workflow on this solution (creates a fresh snapshot).` | Runs the open workflow again, which produces a fresh analysis and a new automatic snapshot. Afterwards an info dialog says `Re-Analyze started - a new snapshot will appear in the snapshot history.` If the `.sln` is missing, a warning says `The .sln file was not found at the given path.` |
| Pin Snapshot | `Pin Snapshot: promote the newest snapshot to a named manual snapshot (manual ones are kept longer).` | Promotes the newest snapshot to a named manual snapshot (see "Snapshots"). |
| Export Config | `Export Config: write the active layer profile + exclusion list to a git-tracked <Solution>.aicb.json sidecar next to the .sln. The headless MCP server / CLI auto-discover and apply it (no --db-path needed).` | Writes the solution configuration to the sidecar file. The tooltip summarizes the original layer/exclusion behavior; the file now carries additional axes whose consumers differ (see "Exporting and importing the solution configuration"). |
| Import Config | `Import Config: read a .aicb.json sidecar and apply it to this solution - creates + activates a layer profile and an exclusion list in the DB.` | Reads a sidecar file and applies it to this solution. The tooltip names the original layer/exclusion behavior; the import now covers the test axis too (see "Exporting and importing the solution configuration"). |
| Show in Explorer | `Show in Explorer: open the Solution folder in Windows Explorer.` | Opens Windows Explorer with the `.sln` selected. If the file is missing, a warning appears. |
| Remove (red) | `Remove: take the Solution out of the workspace - the files on disk are kept.` | Removes the solution and its dependent data after a confirmation that lists everything that is deleted with it. |




The confirmation before a solution is removed, listing the dependent data

### Solution statistics

Five cards show the state of the latest analysis:

| Card | Main value | Sub-line | Tooltip |
|---|---|---|---|
| `Projects` | Number of projects | `.csproj files` | `Number of .csproj files (projects) in the Solution. The Roslyn workspace counts every MSBuild project entry in the .sln file.` |
| `Classes` | Number of all types (classes + interfaces + enums) | `N interfaces, M enums` | `Number of class declarations in the Solution (all projects). Interfaces + enums are listed separately below the main value.` |
| `Methods` | Number of methods | `across all types` | `Number of method declarations across all types (class + interface + record + struct). Property getters/setters are counted too.` |

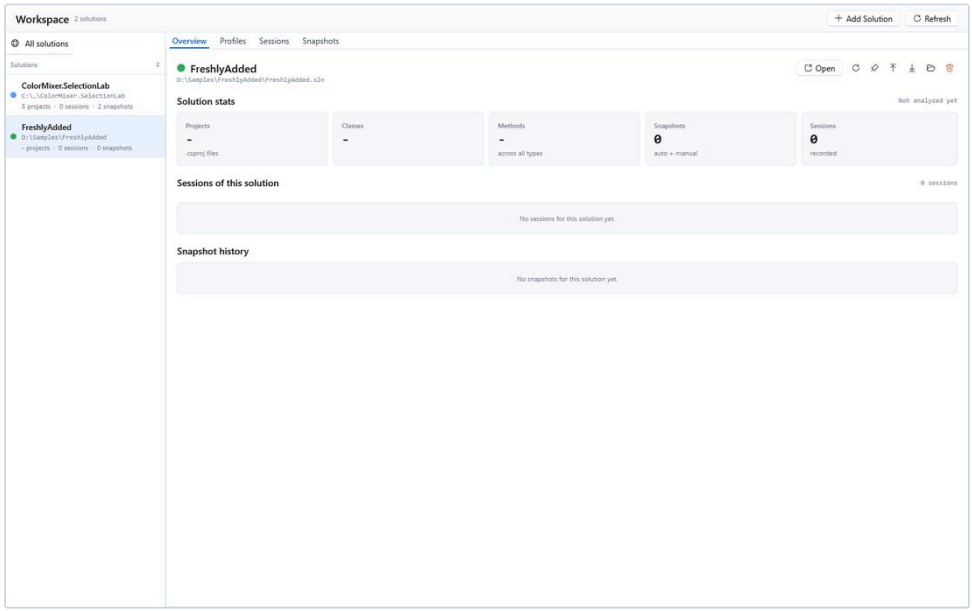
| Card | Main value | Sub-line | Tooltip |
|---|---|---|---|
| `Snapshots` | Number of snapshots | `auto + manual` | `Number of saved Solution Snapshots (automatic + manual). Snapshots are point-in-time captures for diff comparisons.` |
| `Sessions` | Number of sessions | `recorded` | `Number of saved user Sessions (Selection + tree state + optional LLM run history) for this Solution.` |

The four analysis-derived values ( `Projects` , `Classes` , `Methods` and the interfaces/enums split) come from the newest snapshot's analysis. If there is no snapshot yet, or it contains no analysis, the card shows an em dash ( `-` ) instead of a number - a `0` would wrongly say the solution contains no code. The `Snapshots` and `Sessions` counts come straight from the database and are always real numbers. The `N interfaces, M enums` line is hidden while nothing has been measured.

The section header `Solution stats` carries a status line on the right in the form `snapshot <projects>-<types>-<methods> · <name> - yyyy-MM-dd HH:mm` . Possible messages instead of that line are:

- `Not analyzed yet` - no snapshot, or no analysis stored in it.
- `Snapshot history could not be read` - the snapshot metadata could not be read.
- `Statistics could not be read` - the stored analysis could not be read.

The cards are filled in the background, so the numbers may appear a moment after the solution name.




A solution that has not been analyzed yet: the statistics cards show dashes

### Sessions of this solution

The section `Sessions of this solution` lists the solution's sessions, most recently used first. A card shows the solution color dot, the session name, the creation date ( `created yyyy-MM-dd` ) and the last activity ( `last activity just now` / `N min ago` / `N h ago` / `N d ago` / a date). Click a card to open the session in a new workspace tab (tooltip: `Open this Session in a new workspace tab.` ). Empty state: `No sessions for this solution yet.`

### Snapshot history

The section `Snapshot history` lists all snapshots of the solution, newest first. Each row shows:

| Element | Meaning |
|---|---|
| Short number | `<types>-<methods>` from the snapshot's quality stamp; for older snapshots without one, the first eight characters of the snapshot ID. |

| Element | Meaning |
|---|---|
| Name | The snapshot name, or `(unnamed)` for an automatic snapshot that was never named. |
| `Current` pill | Marks the newest snapshot. |
| Type pill | `auto` or `manual` (lower case). |
| Created | Local creation time as `yyyy-MM-dd HH:mm`. |
| `Load` | Opens the snapshot's captured analysis in a new Context Builder tab (read-only). Hidden on the current snapshot. |
| Rename | Prompts `Rename snapshot` / `New snapshot name:`. |
| Delete (red) | Asks `Delete snapshot '<name>'?` before deleting. |

Empty state: `No snapshots for this solution yet.`

### 2.3 Profiles

The `Profiles` tab holds three pickers side by side: `Layer Profile`, `Exclude Namespaces` and `Test Profile`. Each one chooses which entry of a library applies to the selected solution:

- The **layer profile** controls how namespaces are mapped to architecture layers.
- The **exclusion list** controls which namespaces the analyzer skips.
- The **test profile** defines which projects and attributes count as tests.

Note: this tab only *selects*. Creating, editing, deleting and importing library entries happens in `Settings` (see "Settings"); the picker texts point there as `Manage profiles in Settings > Layer Profiles.`, `Manage lists in Settings > Exclude Namespaces.` and `Manage profiles in Settings > Test Profiles.`

Without a selected solution all three pickers are inert and explain why: `Select a solution to choose its active profile.` - the exclusion picker says `Select a solution to choose its active list.`

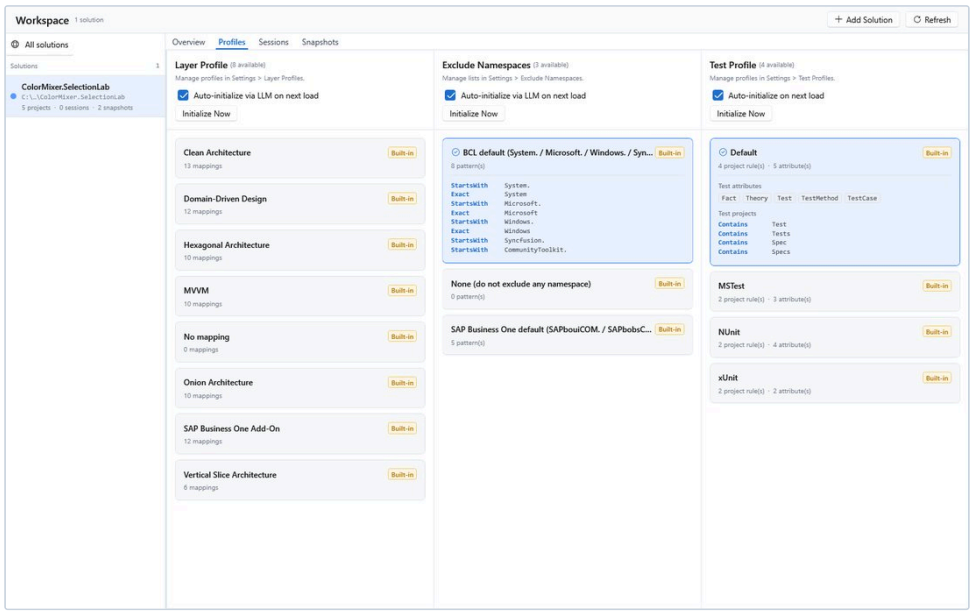
Each picker shows its title with the number of available entries in parentheses, for example `(7 available)`, and contains:

| Element | What it does |
|---|---|
| `Auto-initialize via LLM on next load` | Checkbox. A freshly registered solution starts with all three auto-init flags enabled. On its next load in the Context Builder, the app uses the default model profile to request one proposal for the still-unconfigured layer and exclusion axes, then asks whether to apply it. The request therefore precedes the confirmation. The test profile's checkbox is labeled `Auto-initialize on next load`, because its detection is local and needs no language model. Missing model settings, no detected tests or a declined proposal can leave an axis unconfigured. Existing choices are not overwritten. |
| `Initialize Now` | Restores this axis from the solution's `<Solution>.aicb.json` sidecar immediately - no LLM, no analysis. On success a dialog reports `Restored + activated for this solution (from the sidecar):` with the restored name and rule count, and the checkbox is cleared because the pending intent is fulfilled. If there is nothing to restore, a dialog explains that the axis is already configured or the sidecar is missing or does not cover the axis. |
| Card list | One card per library entry with the name, a `Built-in` pill for built-in entries and a checkmark on the selected card. The meta line under the name shows the size of the entry ( `N mappings`, `N pattern(s)`, or `N project rule(s) · N attribute(s)` ). The selected card additionally unfolds a preview of its rules: `Pattern > Layer` for layer profiles, `MatchType Pattern` for exclusion lists, and for test profiles the attribute chips under `Test attributes` plus the project rules under `Test projects`. |

Selecting a card persists the choice for that solution only. The resolution order is always: **per solution → global default (Settings) → built-in default**. If no solution has its own choice, the global default applies.

Selecting a different layer profile takes effect from the next analysis run; selecting a different exclusion list takes effect from the next run as well. The test profile is resolved freshly for every analysis run.

Auto-initialization writes the resulting per-solution choices to the local database only. It does not create the sidecar automatically; use `Export Config` after reviewing the three axes if the configuration should be versioned with the repository.




Profiles region: the layer profile, namespace exclusion and test profile pickers side by side

### Exporting and importing the solution configuration

`Export Config` writes the solution's active configuration into a git-tracked file named `<SolutionName>.aicb.json` next to the `.sln`. The location is not freely choosable, because the headless MCP server and the CLI look for exactly this file. The file contains:

- the active layer profile's rules,
- the active exclusion list's rules,
- the active test profile's project rules and test attribute names (only when it has any),
- suppressed findings (triage decisions),
- the auto-initialize opt-ins of the three axes,
- the analysis-scope setting `analyzePreferredTfmOnly` (multi-targeted projects), which has no control in the GUI and is carried over from the existing file,
- the design-time build platform `designTimeBuildPlatform`, which has no control in the GUI either and is carried over from the existing file.

If the solution has none of these, a warning appears instead of an empty file: `This solution has nothing to export: no active layer profile, exclusion list or test profile, and no suppressed findings. Pick a per-solution profile in the right-hand panels first.` If the file already exists, an overwrite confirmation appears first. The success dialog names the counts and reminds you to commit the file:

```
Exported solution config to:
<path>

Layer rules: N   Exclusions: M   Test rules: K   Suppressed findings: L

Commit the .aicb.json next to the .sln - the MCP server / CLI then apply it headless (no --db-path needed).
```

That sentence is the application's confirmation text. "Apply it" is not one uniform precedence rule: database-free headless analysis uses the sidecar's layer/exclusion fallback and analysis scope, while test detection resolves from the database or built-in default. Suppression-aware reading tools handle sidecar suppressions separately.

`Import Config` opens a file dialog ( `Import solution config (sidecar)` , filters `AICB sidecar (*.aicb.json)` , `JSON files (*.json)` , `All files (*.*)` ), reads the file and applies it to the selected solution: for each axis the file carries, it creates and activates a layer profile named `Imported layer profile (<Solution>)` , an exclusion list named `Imported exclusions (<Solution>)` and a test profile named `Imported test profile (<Solution>)` , and marks those axes as initialized; a confirmation lists what was imported per axis. A file that switches the exclusion off ( `"exclusionsDisabled": true` ) activates the built-in list `None (do not exclude any namespace)` instead of an imported one. A file without layer rules, exclusions and test project rules is rejected with `The file contains no layer rules, exclusions or test rules to import.` , and a file the sidecar reader refuses is not imported in part ( `The file was not imported: ...` ). GUI import covers the layer, exclusion and test axes. The sidecar's suppressions, auto-init flags, analysis-scope key and platform key remain in the file, but each has its own reading path; they are not all applied by one generic headless-profile resolver.

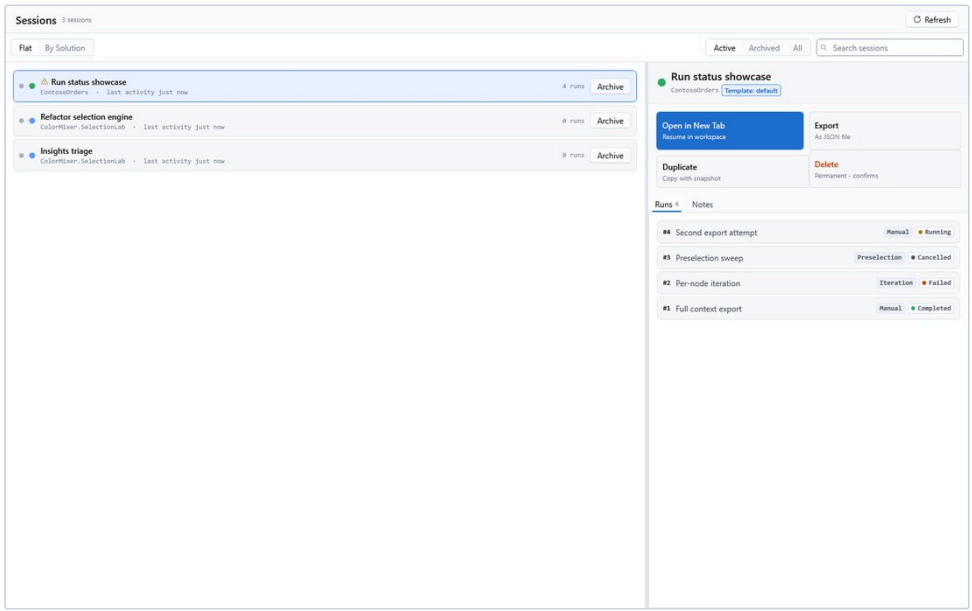
## 2.4 Sessions

A session is a named, saved working state of one solution in the Context Builder. It stores the tree selection, the per-node detail-level overrides, the prompt text, the active context template, an optional pinned snapshot, free-text notes and the per-picker overrides of the configuration panels - plus the history of the runs that were started from it. A session always belongs to exactly one solution.

The region is the embedded `Sessions` tab of the Workspace page. It is scoped to the selected solution, or unscoped while `All solutions` is active.

### Controls

The header shows `Sessions` and the total number of sessions in the database, not the number in the current solution scope or filtered result. `Refresh` ( `F5` ) reloads the list from the database.




The Sessions region: session list on the left, detail pane with actions and runs on the right

Two segmented switches and a search field control the list:

| Group | Values | Default | Tooltips |
|---|---|---|---|
| List shape | `Flat` · `By Solution` | `Flat` | `Show all Sessions as a flat list.` / `Group Sessions by Solution - one section header per Solution.` |
| Archive scope | `Active` · `Archived` · `All` | `Active` | `Show only non-archived Sessions.` / `Show only archived Sessions (old/completed work).` / `Show all Sessions (active + archived).` |

The search field carries the placeholder `Search sessions` . It filters live and case-insensitively over both the session name and the solution name. `Ctrl+F` focuses the field and selects its content, `Escape` clears it.

The filters are applied in this order: solution scope, then archive scope, then search text. The flat list is sorted by last activity, newest first. The grouped list sorts the groups alphabetically by solution name and sorts the sessions inside each group by last activity, newest first.

### The session list

Each row shows:

| Element | Meaning |
|---|---|
| Small dot (left) | Green when the session (or at least its solution) is currently open in a Context Builder tab; gray otherwise. |
| Color dot | The solution's color. |
| Warning icon | The saved solution file no longer exists at its path. Tooltip: `Solution file not found at saved path. Open to relocate.` |
| Name | The session name. |
| Second line | The solution name and the last activity. |
| `N runs` | Number of saved runs of this session; `- runs` when the count could not be read. |
| `Archive` / `Activate` | Toggles the archived flag (flat list only). Tooltip: `Archive / activate session`. |

Archiving is non-destructive: runs and snapshots stay, the session simply disappears from the default `Active` filter. The toggle saves immediately. If the row drops out of the active filter, the selection advances to the next visible session. If saving fails, the flag is reverted so list and database stay consistent.

Empty state: `No saved sessions yet. Save one from the Context Builder via Save Session.` Like the header counter, this state is based on all sessions in the database; a solution-scoped list can therefore be empty while this global empty state remains hidden.

Note: the header count and the empty state refer to all sessions in the database, not to the current solution scope or the filtered list. A scope or filter that hides every row therefore does not replace the list with the empty-state hint.

### The detail pane

Without a selection the pane says `Select a session to see its runs, notes and actions.`

With a selection, the header shows the color dot, the session name, the solution name and an accent pill `Template: <template id>`. Below it are four action cards in a 2×2 grid:

| Card | Subtitle | What it does |
|---|---|---|
| `Open in New Tab` (accent) | `Resume in workspace` | Opens the session in a new Context Builder tab and restores its state (see "Opening a session"). |
| `Export` | `As JSON file` | Opens a save dialog ( `JSON files (*.json)` ) and writes the complete session - including the resolved template snapshot, the tree selection, the prompt and the notes - as indented JSON. The suggested file name is derived from the session name with characters that are invalid in file names replaced by `_` ; an unnamed session suggests `session-<id>.json` . Note: there is no import counterpart; the export is meant for backup and for passing a session on. |
| `Duplicate` | `Copy with snapshot` | Creates a copy with a new ID. The name becomes `<Name> (Copy)` , and on a name collision `<Name> (Copy 2)` , `<Name> (Copy 3)` , and so on. The copy keeps the solution, the pinned snapshot, the active template, the resolved template snapshot, the tree selection, the prompt and the notes, and is selected afterwards. |
| `Delete` (red) | `Permanent - confirms` | Asks `Delete session "<name>"?` and then deletes the session. |

Below the cards are two sub-tabs:

- `Runs` - one card per run with `#<number>`, the run name (or `(unnamed run)`), a run-type pill and a status pill with a colored dot. Empty state: `No runs in this session yet. Open the session and start a run - it is recorded here when it finishes.`
- `Notes` - a multi-line text field (tooltip `Free-text notes about this Session. Use Save Notes below to persist changes.`) and the button `Save Notes` (tooltip `Save the notes text in the DB.`). Saving updates the session's last-activity timestamp and confirms with `Notes saved.`

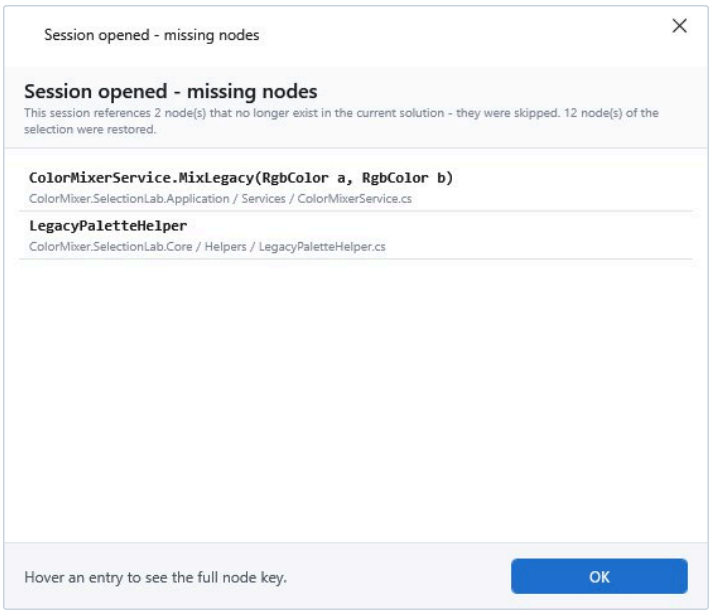
### Opening a session

`Open in New Tab` (or clicking a session card in the `Overview` region) runs the following steps:

1. If the session no longer exists, a warning says `Session no longer exists.` If its solution record is missing, it says `The associated solution could not be found.`
2. If the saved `.sln` path no longer exists, a relocate dialog appears so you can pick the new location. Cancelling ends the flow.
3. If the session is already open in a Context Builder sub-tab, the app simply switches to that tab.
4. Otherwise a new sub-tab is created. The session's pinned snapshot is used, but only while it still matches the code on disk: if the snapshot is still current, the tab is promoted to a full live load, is editable and shows no read-only banner. Only when the code has changed since the snapshot does the tab stay read-only, and then the banner is accurate. A session without a pinned snapshot (an auto-session created before the first send) always loads live.
5. The tree selection, the detail-level overrides, the active template, your prompt, the session ID and the overrides of the configuration pickers are restored. The tab is then marked as unmodified.
6. Nodes referenced by the session that no longer exist in the solution are collected and reported in a dedicated window titled `Session opened - missing nodes`:

```
This session references N node(s) that no longer exist in the current solution - they were skipped. M node(s) of the selection were restored.
```

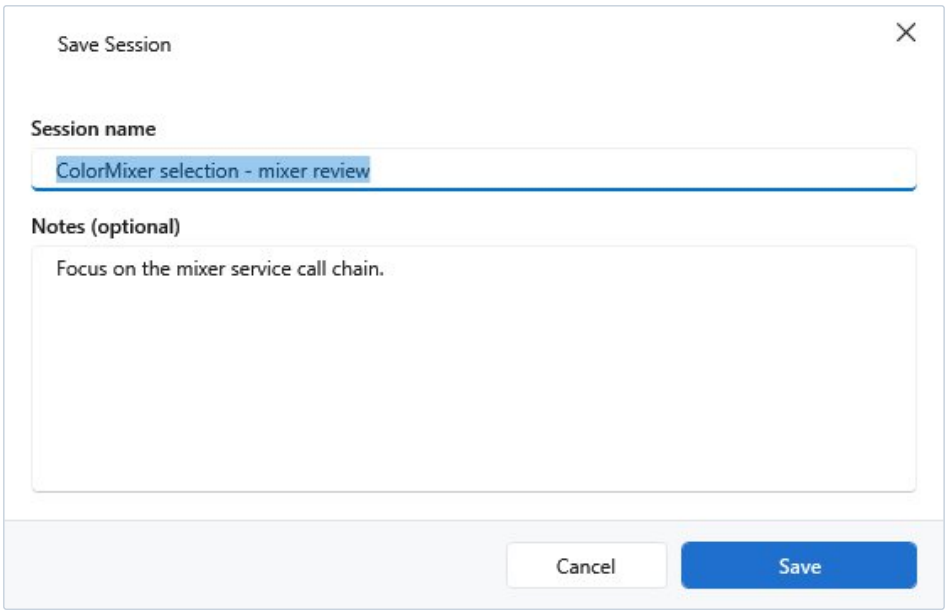
Each entry shows `Type.Method`, a path hint line (project / folder / file), and the full node key on hover (`Hover an entry to see the full node key.`). The orphaned nodes are removed from the session on the next save.




The window Session opened - missing nodes

### Saving and auto-saving

Sessions are created from the Context Builder with `Save Session` (tooltip `Save the current tab as a Session (Ctrl+S)`), which asks for a session name and optional notes. Every tab starts unmodified after a save.




The Save Session dialog

If `Auto-save enabled` is on in `Settings` → `General` → `Session & Analysis`, a timer saves all modified tabs every `Auto-save interval (minutes)` minutes (default: on, every 5 minutes). Changes to either setting take effect after the next application start. Two rules matter:

- Auto-save skips only tabs that have no session ID. A session created automatically as `Untitled ...` by the first send does have an ID and is included even if you never opened `Save Session`.
- The first time you send a prompt from a tab that has no session yet, the app creates one automatically. It is named `Untitled - <solution file name> - yyyy-MM-dd HH:mm`, pins the current snapshot, and from then on is a regular session you can rename in the `Sessions` region.

## 2.5 Snapshots

A snapshot is a point-in-time capture of a solution's analysis (the analyzed structure plus quality and debt metrics). Snapshots are used to compare two states and to reopen an older state in a read-only view. There are two kinds:

| Type | Created by |
|---|---|
| `auto` | Automatically on every fresh analysis, for example when you open or re-analyze a solution. Automatic snapshots are unnamed. |
| `manual` | Through `Create Manual Snapshot` in the `Snapshots` region, or by pinning the newest snapshot in the `Overview` region. Manual snapshots are named and are kept longer. |

Every snapshot is stamped with a quality aggregate (debt total and rating, code metrics) computed against the active quality profile - the same stamp used by the CLI and by automatic snapshots. The stamp supplies the short number shown in the `Overview` snapshot history.

The `Snapshots` region is the embedded `Snapshots` tab of the Workspace page. In the solution-scoped view its own solution list is hidden; in the global view (`All solutions`) it shows a solution list on the left with a splitter.

### Controls

The header shows `Snapshots`, the number of snapshots as `N in this solution`, and `Refresh` (`F5`, tooltip `Reload the Snapshot list from the DB. (F5)`). While a refresh or a diff is running, the refresh button is disabled so that holding `F5` cannot stack up overlapping reads.

The action bar contains:

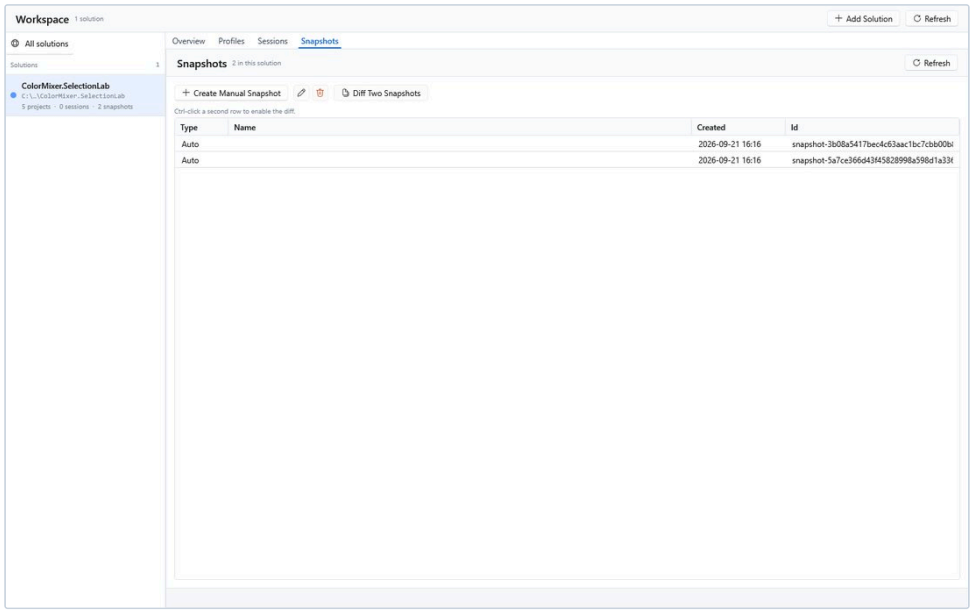
| Button | Tooltip | Enabled when |
|---|---|---|
| `Create Manual Snapshot` | `Save a new manual Snapshot of this Solution's current analysis data structure (name + optional notes).` | Always clickable; the precondition is checked after the click. |
| Rename (icon) | `Rename the selected Snapshot.` | A snapshot is selected. |
| Delete (icon, red) | `Permanently delete the selected Snapshot (cannot be undone).` | A snapshot is selected. |
| `Diff Two Snapshots` | `With two Snapshots selected (Ctrl-click) - show the Diff window with structural changes (Added/Removed/Changed per type).` | Two different snapshots are selected. |

Below the buttons a hint line reads `Ctrl-click a second row to enable the diff.`

The snapshot table is read-only and allows multiple selection. Its columns are `Type` (`Auto` or `Manual`), `Name`, `Created` (local time) and `Id` (the full snapshot ID). Rows are listed newest first. Select a single row to rename or delete it; Ctrl-click a second row to enable the diff.

Empty state: `No snapshots yet for the selected solution. Use 'Create Manual Snapshot' above, or enable auto-snapshots in Settings > General > Session & Analysis.` Note: every fresh analysis writes an automatic snapshot; the setting `Max snapshots per solution` under `Settings` → `General` → `Session & Analysis` controls how many automatic snapshots are kept, not whether they are written.

A status line at the bottom of the region reports the result of the last action.




The Snapshots region: the snapshot table with the action bar

### Creating a manual snapshot

A manual snapshot needs the live analysis, so the solution must be open in the Context Builder right now. The button is always clickable; if the selected solution is not the one that is open, a dialog explains: `Manual snapshots can only be created for the solution currently open in the Context Builder. Open the selected solution there first.` Without any selected solution the status line says `Select a solution first.`

1. Select the solution in the Workspace list and make sure it is open in the Context Builder.
2. Click `Create Manual Snapshot`.
3. Enter a name in the `Create Manual Snapshot` dialog (`Name for the snapshot:`). The button tooltip mentions a name plus optional notes, but the dialog itself asks only for the name; its input tooltip merely explains Enter and Escape.
4. The status line confirms `Manual snapshot "<name>" created.`

### Renaming and deleting

Select a row and use the rename icon: the prompt is `Rename Snapshot` / `New name:`; on success the status line says `Snapshot renamed.` The rename in the `Overview` snapshot history uses the prompt `Rename snapshot` / `New snapshot name:` instead.

Deleting asks for confirmation and warns about the consequence:

```
Delete snapshot "<name>"?
Runs that reference this snapshot keep their RunSnapshot reproducibility, but the original code state will no longer be available.
```

On success the status line says `Snapshot deleted.`

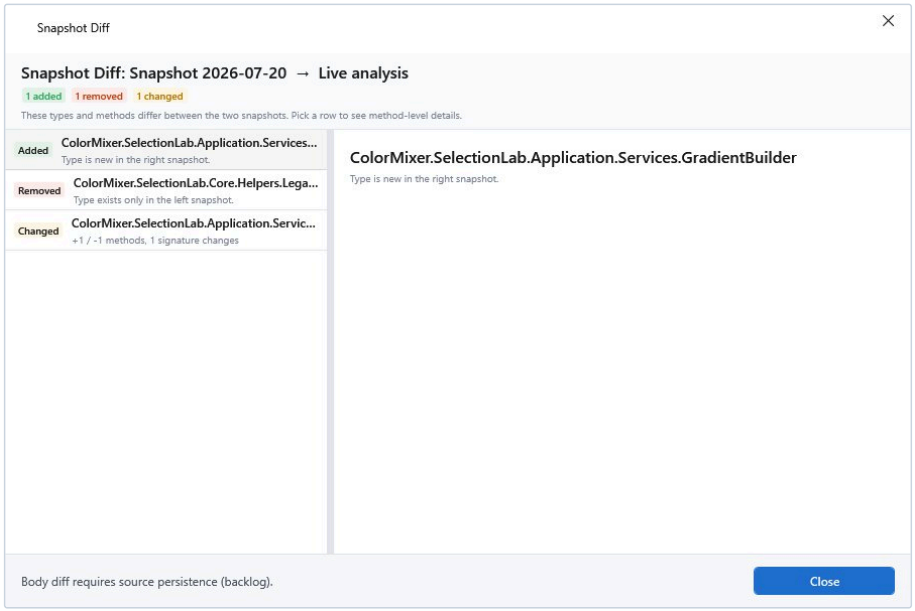
### Comparing two snapshots

1. Click the first snapshot row.
2. Ctrl-click the second row. The button `Diff Two Snapshots` becomes enabled.
3. Click `Diff Two Snapshots`. Both snapshots are loaded and compared in the background (a spinner overlays the region); then the diff window opens and the status line reports `Diff: N changes.`

If one of the two snapshots has no stored analysis, the diff is refused with `At least one of the selected snapshots has no complete solution analysis - diff not possible.`

The diff window is titled `Snapshot Diff: <left> → <right>`, where each side is labeled `<name or first 8 characters of the ID> (yyyy-MM-dd HH:mm)`. It shows three count badges - `N added`, `N removed`, `N changed` - and a list of the changed types. Selecting a row shows its method-level details in the detail pane: `Added methods` (prefixed `+`), `Removed methods` (prefixed `-`) and `Changed signatures` (previous signature as `-`, new signature as `+`). If both snapshots have the same class/method structure, the window says `Both snapshots have identical class/method structure.`

The comparison is structural: it covers types, methods and method signatures. A line-by-line body diff is not available (it would require storing the source text; it is planned but not released). The window's footer states this: `Body diff requires source persistence (backlog).`




Snapshot diff window showing added, removed and changed types

#### Opening a snapshot read-only

The `Load` button in the `Overview` snapshot history opens the snapshot's captured analysis in a new Context Builder tab. The tab is read-only; it shows the state that was captured, not the current code. The button is hidden on the current (newest) snapshot. Loading a snapshot is not the same as switching to it - see the note below.

Note: a snapshot is **not a restore point**. There is no way to write a snapshot back onto a solution, and none is planned. What you can do is open it read-only, compare snapshots, or reload the live code.

#### Retention of automatic snapshots

`Max snapshots per solution` (default 10) caps the number of automatic snapshots per solution. When the cap is exceeded, the oldest automatic snapshots are deleted. Manual snapshots are never deleted by this rule and do not count towards the cap. A snapshot that a completed run references is also kept even when it is beyond the cap, because the run's reproducibility depends on it; a snapshot referenced only by a saved session can be removed by the cap.

### 2.6 Working over time

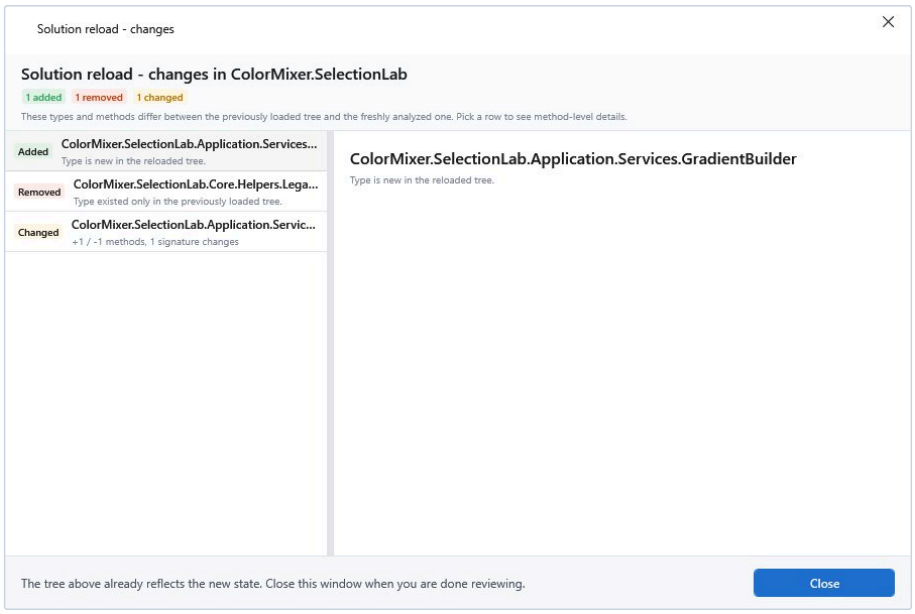
The following workflows cover the situations that change after the first successful run: the code changes, the session no longer matches the code, you want to compare two states, the solution or the database moves, or the app is updated.

#### When the code has changed: Reload solution

Use `Reload solution` in the Context Builder's persistent left Solution Tree sidebar, which remains visible on every top-level Context Builder tab (tooltip `Re-analyze the Solution from disk and show the structural delta against the previously loaded Tree as a diff window. (F5)` ; the same command is on `F5` while the Context Builder is active).

1. Click `Reload solution` . If no source file has been touched since the last load (the check covers the `.cs` , `.csproj` , `.xaml` , `.cshtml` and `.razor` files below the solution folder), a dialog asks `No code changes detected since the last reload (same file-set hash).\n\nRe-analyze anyway?` Answer `No` to skip the analysis; the app stays as it is.
2. The analysis runs again against the code on disk. A live reload is performed, so a new automatic snapshot is created and the tab leaves read-only snapshot mode.
3. Your tree selection and your detail-level overrides are collected before the reload and re-applied to the freshly built tree. Keys that no longer exist are dropped here; they show up in the diff instead.
4. If the structure changed, the diff window opens ( `Solution reload - changes in <solution>` ); if nothing changed, an info dialog says `No structural changes detected. The reloaded tree is identical to the previously loaded one.`

The reload diff window compares the previously loaded tree with the freshly analyzed one. It has the same layout as the snapshot diff, with `Added` / `Removed` / `Changed` entries and method-level details. Its footer reminds you that the tree already shows the new state: `The tree above already reflects the new state. Close this window when you are done reviewing.`




The diff window after Reload solution

### Reopening a session against changed code

If a session's pinned snapshot no longer matches the code on disk, the reopened tab is read-only and shows a banner explaining why. The wording depends on the cause:

| Banner text | Cause |
|---|---|
| `Read-only - the code changed since this session was saved. Use 'Reload solution' to re-analyze the live code.` | The source changed after the snapshot was taken. |
| `Read-only - this snapshot was saved by a different version of the tool, so it cannot be reused directly. This says nothing about your code. Use 'Reload solution' to re-analyze.` | The snapshot's payload format differs from the current app version. |
| `Read-only - this snapshot was produced by different analyzer code, so it cannot be reused directly. This says nothing about your code. Use 'Reload solution' to re-analyze.` | The snapshot was produced by a different analyzer build. |
| `Read-only - could not determine whether the code still matches this snapshot. Use 'Reload solution' to re-analyze the live code.` | The staleness check could not decide. |
| `Read-only - Original snapshot loaded. Use 'Reload solution' to re-analyze the live code.` | A snapshot was opened deliberately, for example through `Load`. |

In every case, `Reload solution` re-analyzes the live code and makes the tab editable again. The nodes the session can no longer restore are listed in the `Session opened - missing nodes` window described above.

### Comparing two states

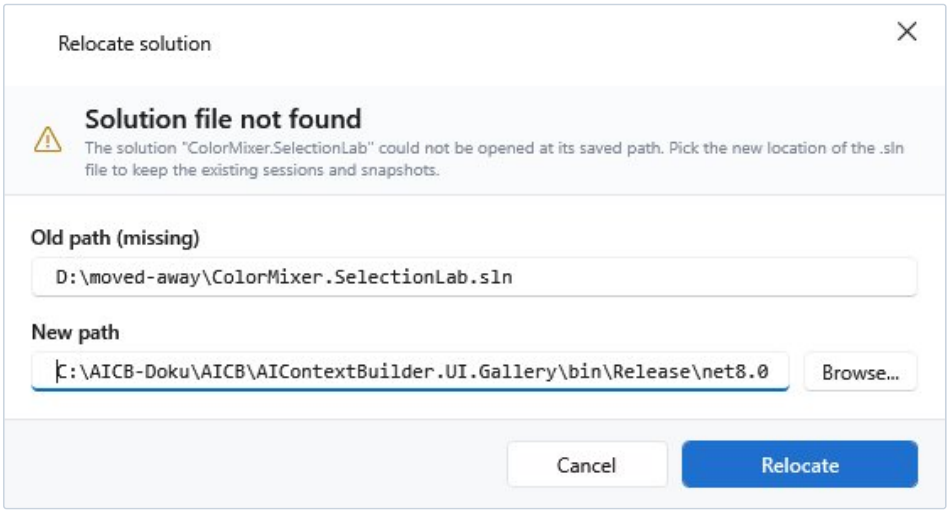
Two comparison paths exist and should not be confused:

| Path | Compares | Where |
|---|---|---|
| `Reload solution` | the previously loaded tree with the freshly analyzed one | Context Builder, automatically after the reload |
| `Diff Two Snapshots` | two stored snapshots with each other | Workspace, `Snapshots` region |

### When the solution has moved

Opening a solution whose `.sln` no longer exists at the saved path triggers the relocation flow. This applies to all three open paths: opening a solution directly, opening a session, and loading a snapshot with a pinned ID.

- If the path is unknown (for example a typo when opening a fresh file), a plain error appears: `Solution file not found:\n<path>`, with the title `Open solution`. No relocation is offered because there is nothing to relocate.
- If the path belongs to a registered solution, a relocation dialog appears. Pick the new `.sln` location and the app updates the stored path and retries. The pinned snapshot of a session is carried through the relocation, so the reopening attempt keeps its pin.
- If relocation itself fails, its message appears under the title `Relocate solution`.
- If the freshly picked file disappears between confirmation and loading, the app shows `Solution file still not accessible at:\n<path>` instead of asking again.




The Relocate solution dialog

### When the app version changes

The database carries a schema version, and the direction decides what happens:

- **Newer app, older database** - the migration runs automatically at startup, without asking. This affects the shared database immediately: parallel MCP hosts and a second GUI instance see the migrated schema from that moment on. Older binaries then report drift.
- **Older app, newer database** - the GUI does not start. It shows a dialog explaining that the database was written by a newer version, names both schema versions, and exits rather than writing to it. The two ways out are updating the app or pointing `Storage` at a different database file.

### When the database moves

`Settings` → `Storage` offers several ways to change the active database: edit `BasePath` and save, use `Switch Database...` for an existing file, create a new database, or pick a recent database from the list. All of them run the same checks:

1. While a background run is active, the change is refused with `Cannot {action} while N background task(s) are active.`
2. The target database is checked for compatibility with this app build. An incompatible file is rejected with `Not saved - the database under the new BasePath is not compatible.`
3. A confirmation dialog `Change data directory?` names the new path, whether the file already exists and how many migrations will be applied. It states the important point: the current database file stays untouched, and its content does **not** move - settings, sessions and solutions are read from the new file from now on. Cancelling reports `Not saved - the data-directory change was cancelled.`

The application database file is named `aicb.acb`; the app accepts no other extension for it. The explicit database paths of CLI and MCP calls are not affected by this rule.

### Moving usage numbers between machines

The `MCP Usage` region can move its report from one machine to another:

1. On machine A, click `Export`. It writes the same JSON that the `usage_report` MCP tool returns.
2. On machine B, click `Import` and pick the exported file. Tooltip: `Import an exported usage report as a named snapshot beside the live data. Imported sets are never merged into the live numbers.`
3. The imported report then sits in the `Snapshots` region of that page, next to the live numbers, as a named set. Empty state: `No imported reports yet. Import an exported usage-report JSON to keep it here as a named set.`
4. The `Tools`, `Errors` and `Clients` regions always show the live database; the import does not change them. Tooltip: `Tools, Errors and Clients always show the live database. Imported snapshots sit beside it in the Snapshots region.`

See "MCP Profiles, MCP Usage and Templates" for the full description of that page.

### What survives a restart

Sessions and snapshots are stored in the shared application database and are still there after a restart. Open tabs are not restored: the app always starts on its start page, and the last used solutions and sessions are offered there. Saving a session is therefore the way to keep work across restarts.

# 3 The Context Builder: layout and solution tree

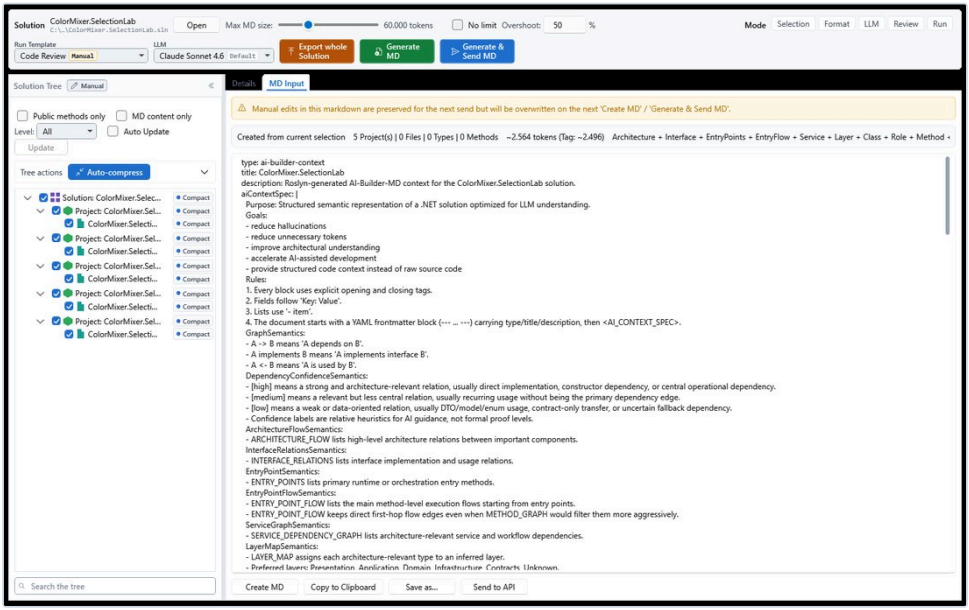
The Context Builder is the workspace where a C# solution becomes a curated context document. You open a solution, decide in the solution tree which parts of it belong in the document, optionally refine how the document is shaped, and then render it - either for copying into an external LLM tool or for sending directly to a configured model. Everything on this surface serves that one chain; the layout is built around it, not around a feature list.

## 3.1 From opening a solution to the exported document

The typical path through the workspace:

1. **Open a solution.** Use the `Open` button in the header of the Context Builder, or press `Ctrl+O` anywhere in the application. The file picker accepts `.sln`, `.slnx` and `.slnf` files. The solution is analyzed, and a modal loading overlay with a progress bar and a `Cancel` button covers the tree while it is being built.
2. **Select in the solution tree.** Every row carries a three-state checkbox (`Include` / `Exclude` / `Inherit`). This is the workspace's multi-selection mechanism - the row selection itself is single-select and only drives the detail view.
3. **Optionally refine.** Set a detail level per node with the chip at the end of the row, or use the bulk actions in the tree toolbar (`Auto-compress`, `Apply expansion`, `Refresh tree`, `Reload solution`).
4. **Optionally configure.** The six panels on the `Main` tab (`Prompt`, `Sections`, `Graph Selection`, `Selection Engine`, `Test Description`, `Export Output`) each have a usable default; none of them is mandatory.
5. **Render.** The header offers three buttons, deliberately ordered from the widest to the narrowest scope: `Export whole Solution` (ignores the selection), `Generate MD` (builds the document from the current selection, no model call, `Ctrl+M`) and `Generate & Send MD` (builds it and sends it, `Ctrl+Enter`).
6. **Review the result.** The document lands in the `MD Input` tab. The text is editable, and the header above it summarizes what it was created from: the selection, a token estimate, the active graphs and the code mode.
7. **Hand it over.** Use `Copy to Clipboard` or `Send to API`. A model answer appears in the `LLM Response` tab; `Use as MD input` copies it back into `MD Input` for a follow-up round.

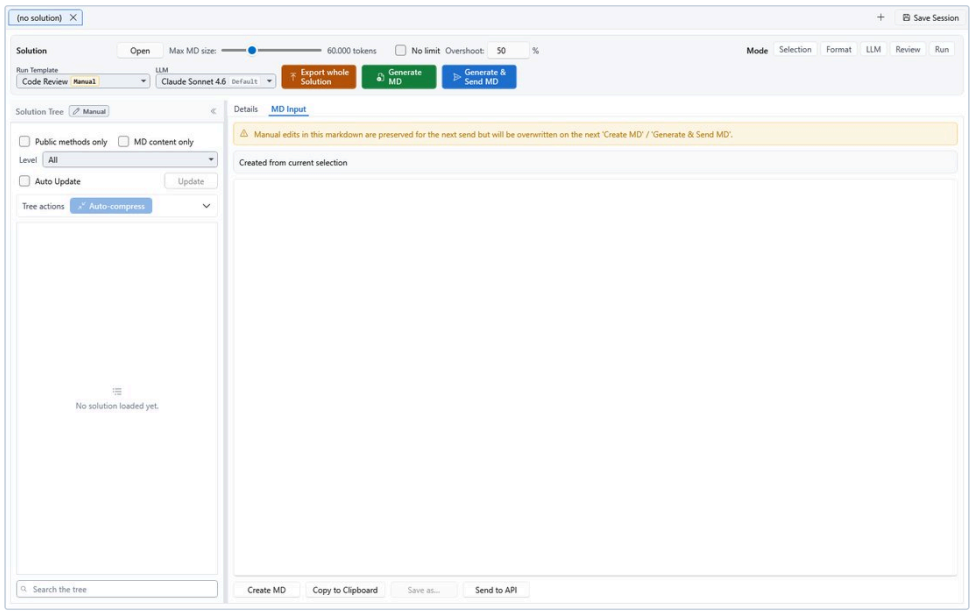
Only a handful of controls are strictly necessary for this: opening a solution, the tree rows, the export checkboxes, `Generate MD`, and the document with its copy button. Everything else is optional refinement.




The Context Builder with a loaded solution: the solution tree on the left, the context document on the right

## 3.2 Solution tabs

The Context Builder itself is a tab area at the top of the workspace, one tab per solution you have open.




The solution tabs of the Context Builder

- A fresh tab is labeled `(no solution)` until a solution is loaded. A loaded tab is labeled with the solution file name (for example `MyApp.sln`), and a `*` suffix marks unsaved changes.
- `+` (`Ctrl+N`) adds another empty solution tab.
- `Save Session` (`Ctrl+S`) saves the active tab as a session. The button stays disabled until a solution is loaded in the tab. It always opens the dialog; for an existing session, the current name and notes are pre-filled. Auto-save and the non-interactive save-before-close path can update an existing record silently.
- Each tab has a close button; `Ctrl+W` closes the frontmost tab.
- While a run is in progress, the tab strip is locked so you cannot switch away from the tab that is working. The lock is lifted as soon as the run finishes.
- Closing a tab with unsaved changes asks `Save before closing?` and offers `Yes` / `No` / `Cancel`, provided the `Tab-close confirmation` option under Settings › General is on.
- A background auto-save runs at the interval configured under Settings › General (`Auto-save enabled`, `Auto-save interval (minutes)`, default 5 minutes); setting changes take effect after the next application start. It skips only tabs with no session ID; an `Untitled ...` session created automatically by the first send is included. If one save fails, a generic warning says that one of the open sessions could not be saved and points you to `Save Session`; the session ID is written only to the log, and other tabs continue to auto-save.

When you save, two safeguards apply. If the current state is identical to the last saved state of the session, the app asks whether to overwrite anyway. And if the chosen name already exists for the same solution (case-insensitive), a dialog names the conflicting session and its last modification date and offers to pick another name. A successful save confirms with the session name and the number of selected nodes.

## 3.3 Layout modes: the surface starts minimal

The Context Builder starts with a deliberately small set of controls. Five chips under the label `Mode`, at the right edge of the header, add the areas you need. The chips are additive: selecting several shows the union of their areas, and selecting none leaves the minimal view.

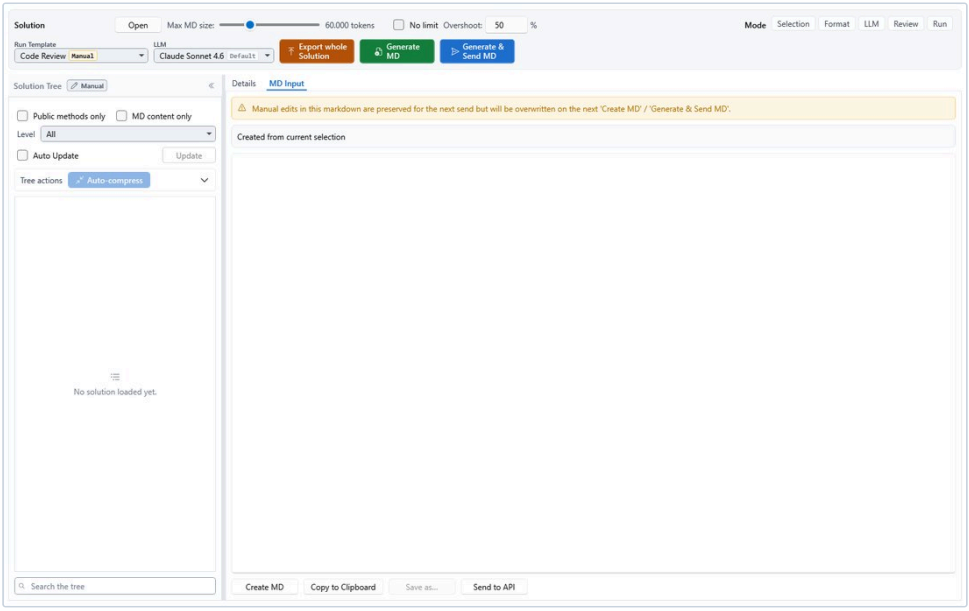
| Chip | What it adds | Fallback tab |
|---|---|---|
| `Selection` | The `Graph Selection` and `Selection Engine` panels inside `Main` | `Main` |
| `Format` | The `Sections` and `Export Output` panels inside `Main`, including the document notation | `Main` |
| `LLM` | The `Prompt` panel inside `Main` and the `LLM Response` tab | `Main` |
| `Review` | The `Test Description` panel inside `Main` and the `Insights` tab | `Insights` |

| Chip | What it adds | Fallback tab |
|---|---|---|
| `Run` | The `Active Run` and `Reasoning` tabs | `Active Run` |

The tooltips of the chips read:

- `Selection` - "Selection - what goes into the document. Adds the Graph Selection and Selection Engine panels. (Alt+1)"
- `Format` - "Format - how the document is shaped. Adds the Sections and Export Output panels, including the notation. (Alt+2)"
- `LLM` - "LLM - authoring the prompt and reading the answer. Adds the Prompt panel and the LLM Response tab. (Alt+3)"
- `Review` - "Review - walking through quality findings. Adds the Insights tab and the Test Description panel. (Alt+4)"
- `Run` - "Run - watching a run step through. Adds the Active Run and Reasoning tabs. (Alt+5)"

Two tabs are always visible and cannot be hidden by any mode: `Details` and `MD Input`. `Main` is derived - it appears exactly when the selected modes reveal at least one of its panels. That keeps the minimal view meaningful: a tree click has to land somewhere, and `MD Input` carries the document the application exists to produce.




The Context Builder in the minimal view, before a solution is loaded

The chip order follows the workflow (select, shape, send, review, run). The keyboard twins are `Alt+1` to `Alt+5`, which toggle one mode each, and `Alt+0`, which turns all modes off and returns to the minimal view. Both the chips and the keys write the same state, so they can never disagree.

A mode change only moves the active tab when the tab you are on has just been hidden; it then falls back to the mode's home tab, or to the first visible tab. If the change leaves your current tab visible, nothing moves.

Two properties of the mode selection are worth knowing:

- It applies to the whole application, not per solution tab, and it is remembered between sessions.
- A stored selection that this version does not recognize is ignored; the workspace opens in the minimal view instead of guessing.

Note: `Run` reveals the `Active Run` and `Reasoning` tabs, but their content comes from iteration and preselection runs. Those run types are not released yet (see the `Run Template` selector below), so with a released template these two tabs stay empty.

## 3.4 The workspace layout

The workspace consists of a header card spanning the full width, and below it a left sidebar with the solution tree, a splitter, and the tab area on the right.

### The global header

The header card is persistent across all sub-tabs ( `Main` , `Details` , `MD Input` , and so on) and holds four groups. It is responsive: while the window is wide, the solution identity, the token budget and the mode chips share the top row and the run-template group sits below it; when the window narrows, the groups stack and the mode chips get a row of their own.

**Solution.** The label `Solution` , the name of the loaded solution and a shortened path (the full path is in the tooltip), with the `Open` button on the right. `Open` loads a solution into this tab - if one is already loaded, it is replaced.

**Max MD size.** One home for the token budget that applies to every rendered document ( `Create MD` , `Generate & Send MD` , `Export whole Solution` , and the send-selected-lines path of the Insights tab).

| Control | Meaning |
|---|---|
| Slider | Target token count for every render. Range 8,000 (floor) to 200,000, snapping in steps of 1,000. Default 60,000. |
| Readout | The current value, for example `60,000 tokens` , or `No limit` . |
| `No limit` | Removes the cap and renders the entire solution - the tooltip says: "Render the entire solution with no size cap (former behavior - can be very large)." The slider is disabled while this is checked. |
| `Overshoot:` | The maximum headroom above `Max MD size` , as a whole number from 0 to 1000. The ceiling is the target plus this percentage; the renderer still aims at the target and does not deliberately spend the allowance. The default comes from the active pipeline profile (50 unless changed); edit the saved default under Settings › Pipeline Profiles. |
| Warning readout (right) | How far the last generated document actually exceeded the budget, measured against `Max MD size` . Empty when the render stayed within budget. |

The renderer converges on the target by lowering detail and pruning the least relevant content; it never cuts text in the middle of a block.

Note: when a tree was restored from a stored snapshot (read-only snapshot mode), `Create MD` , `Generate & Send MD` and the MD-content preview bypass the budget and render the stored analysis verbatim. `Export whole Solution` and the Insights path for sending selected lines still apply the slider budget and can trim that snapshot render.

**Run Template and LLM.** Two selectors side by side, each with its caption above it.

- `Run Template` lists every available template with its name and a colored run-type pill. The tooltip lists the active graphs in addition. Only templates of type `Manual` can be selected; the tooltip states: "All run-template types are listed, but only type "Manual" can be selected. Iteration and Preselection runs are wired end to end and simply not finished, so they are not released yet. Pipeline is a placeholder: it is accepted by the stored data and the repositories, but nothing executes it. Both will follow in a future version." An already-selected entry remains selectable regardless of its type, so you can always switch back to it.
- `LLM` selects the model-profile override for subsequent sends from this tab. The first entry shows the resolved default model with a `Default` badge and keeps the normal resolution chain. The choice remains active until you select another entry, close the tab or restart the application.

**The three run actions.** All three buttons are the width of the widest label and each has its own color.

| Button | Color | Shortcut | Effect |
|---|---|---|---|
| `Export whole Solution` | Warning | - | Renders the entire solution into the `MD Input` tab. It ignores the tree selection and needs no producer run; the size is capped by `Max MD size` . |
| `Generate MD` | Success | `Ctrl+M` | Builds the document from the current selection without calling an API, and switches to the `MD Input` tab. |
| `Generate & Send MD` | Accent | `Ctrl+Enter` | Builds the document and sends it directly to the active model (or the model chosen in the `LLM` override). |

`Export whole Solution` needs an active context template; if none is selected, a status message is shown and nothing is rendered. A second click while a whole-solution render is already running is ignored.

**Test display.** When a test description is set in the `Test Description` panel, the header shows `Test:` followed by its text. The whole pair is hidden while the description is empty.

`Recent runs:` appears only when there are runs to show. Each run is a pill with its number, its name (shortened with an ellipsis), its run type, a status with a colored dot, and a timestamp.

### The solution tree sidebar

The left sidebar holds the solution tree and stays in place across all sub-tabs, so the tree is always the central navigation point.

- The default width is 320 px, the minimum 220 px; drag the splitter to change it. The width is remembered.
- The sidebar can be collapsed to a 36 px rail with a reopen button and a rotated `Tree` label. The collapse button's tooltip is "Collapse the Solution Tree - a slim rail stays to reopen it.", the rail's button says "Expand the Solution Tree." The collapsed state is remembered too.
- The header strip of the sidebar carries the title `Solution Tree`, the selection chip bar (see "The selection chip bar") and the collapse button.
- Changing layout modes never touches the sidebar: your width and collapse choice survive every mode change.

### The tab area

The right-hand tab area hosts up to seven sub-tabs:

| Tab | Purpose |
|---|---|
| `Main` | The six configuration panels. Visible as soon as a mode reveals at least one of them. |
| `Details` | The source files of the classes and methods selected in the tree - a syntax-highlighted editor plus a metadata inspector. Always visible. |
| `Insights` | Code-quality insights for the loaded solution. The header reads `Insights (N)` while there are unseen insights, otherwise `Insights`. |
| `MD Input` | The generated markdown document - review or edit it, then send it. Always visible. |
| `LLM Response` | The model's answer to the last send. |
| `Reasoning` | Per-branch reasoning of the last run: output, thinking, tool calls, findings and evaluation. |
| `Active Run` | Live progress of a running iteration: steps, status and token counts. |

The six panels inside `Main` are shown one at a time, and only those a selected mode reveals. When the active panel is hidden by a mode change, the selection moves to the first visible panel.

| Panel | Purpose |
|---|---|
| `Prompt` | The five prompt fields (system role, instructions, goal, constraints, context). |
| `Sections` | A tag schema per output section for this run - a per-run override of the MD profile. |
| `Graph Selection` | Which dependency and relation graphs to include in the output. |
| `Selection Engine` | Per detail level: the detail preset and the expansion strategy for this run. |
| `Test Description` | The test-description fields used for evaluation runs. |
| `Export Output` | The inner notation of the generated document ( `Tag` or `YAML` ). |

Note: the `Test Description` labels mark two fields as required for evaluation runs, but nothing blocks `Generate MD` or `Generate & Send MD` while they are empty.

#### The whole-solution export overlay

While `Export whole Solution` renders, a dimming scrim covers the entire workspace with an indeterminate ring and the text `Rendering whole solution...`. The render runs off the UI thread, so the window stays responsive although the workspace is blocked. A second export started during a running one is ignored.

## 3.5 The Solution Tree in detail

#### Levels

The tree has seven node kinds: `Solution`, `Project`, `Folder`, `File`, `Type` and `Method`, plus an unknown fallback. Every kind has its own icon with its own color; hovering the icon shows the kind name. After a solution loads, the solution row and its project rows are expanded, deeper rows are collapsed.

#### Row layout and markers

Each row has five cells, left to right:

```
[checkbox] [icon] name ..................... [markers] [detail-level chip]
```

1. **Export checkbox**, 13 px, tooltip "Include in AI context". It has three visual states: - checked, filled in the accent color with a tick - the node is effectively included; - unchecked with a warning-colored fill - the node's children are a mix of included and excluded (the fill is visible while the box itself is unchecked); - unchecked, plain - the node is effectively excluded.
2. **Kind icon** of the node.
3. **Name** - the only elastic cell. Long names are shortened with an ellipsis; the full name is in the tooltip. A node that is explicitly excluded is grayed out.
4. **Marker cluster.** All three markers can appear together: - a small accent dot - "Detail differs from the parent"; - an `A` badge - "Comes into the export via Auto-Expansion (read-only)"; - an `M` badge - "Manual Override (Include/Exclude explicitly set by the user)".
5. **Detail-level chip**, right-aligned and always visible. It shows the *effective* level of the row and its color dot carries the level: Compact (blue), Normal (green), Detailed (orange), Source (red). Clicking it opens a popup with `Inherit` plus the four levels. The closed chip shows the resolved level; the open list shows whether the node currently inherits. `Source` can be disabled for a node type when the active MD profile does not allow it; the entry then explains "Source level is locked for this node type in the active MD profile."

#### Selection and multi-selection

- Every node is in one of three export states: `Inherit` (the default), `Include` or `Exclude`. A node inherits its effective state from its parent; an explicitly set state is marked with the `M` badge.
- Clicking a checkbox sets `Include` or `Exclude` for that node **and for every descendant that does not have an explicit state of its own**. Explicitly set descendants are preserved.
- On a freshly loaded solution everything inherits, and the inherited default is included - so the whole solution is in context until you change something.
- The row selection is separate from the export selection: clicking a row selects that one node (single selection) and drives the `Details` view. The export is decided by the checkboxes alone.
- A single click on a `Type` or `Method` node opens its source file in the `Details` tab and scrolls to the symbol. Whether the workspace also switches to the `Details` tab depends on the setting `Reveal Details tab on tree click` under Settings › General (on by default). Folder, project, solution and file nodes never switch.

#### Filters and search

The filters sit above the tree in three rows that keep their order at any sidebar width: the two visibility switches, the `Level` filter, and the update switch with the `Update` button on the right.

| Control | Effect |
| --- | --- |
| `Public methods only` | Hides non-public methods. Off shows all methods. |
| `MD content only` | Shows only the symbols that actually enter the generated document - the exact set after the `Max MD size` trimming, with the hierarchy kept. Off shows the full tree. The set is computed when you toggle the option and after `Create MD`; with `Auto Update` off, the tree applies it when you press `Update`. |
| `Level` | Shows only rows whose effective detail level matches the choice, plus their parent chain. `All` clears the filter. |
| `Auto Update` | Off by default. While off, filter and view changes do not recompute the tree immediately; they arm the `Update` button instead. |
| `Update` | Applies the pending tree update. It turns red while a deferred change is waiting. |

The search box sits at the bottom of the panel, below the tree, and reads as the tree's find bar. It matches a substring of the node name, case-insensitively, and is always live - even when `Auto Update` is off. Its tooltip: "Filter tree (substring match on node name) - Ctrl+F, Esc clears". `Ctrl+F` puts the cursor in the box and selects the existing text; `Esc` clears the box while the cursor is in it.

When a filter actually hides something, the ancestors of the matches are expanded automatically. A filter that every node satisfies does not expand the whole tree - otherwise a default level filter would unfold tens of thousands of rows for no benefit.

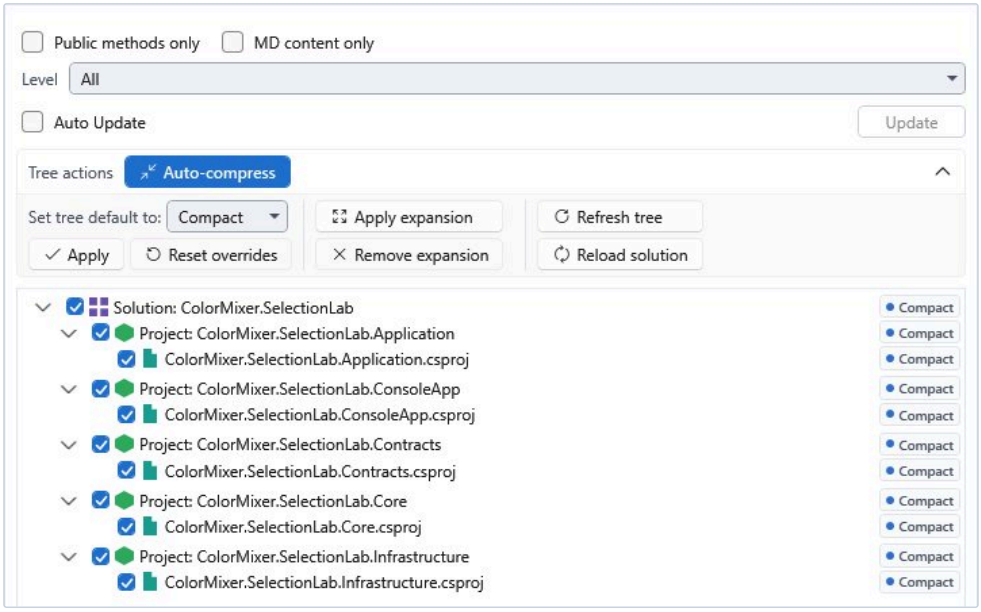
Note: expanding a very large tree fully (tens of thousands of rows) makes scrolling sluggish, because the view only realizes the rows it shows. The tree is virtualized, so the cost stays proportional to the visible part - but a fully expanded huge solution is a lot of visible part.

### The `Tree actions` bar

The action bar is a collapsible drawer, folded by default; its state is remembered. One button stays outside the fold so it is always one click away:

- `Auto-compress` (accent). It sets a detail level (Compact / Normal / Detailed) for every method, based on caller count and role. Manual overrides are preserved. Its tooltip names the active profile and points to where the profiles are edited: "Profiles are editable under Settings > Compression > Compression Rules > Section 'Auto-Compression heuristic'." Below the button, a status line reports the last run, for example how many methods were set to each level and how many existing overrides were preserved.

Inside the drawer are three groups:




The solution tree with the Tree actions drawer open

| Group | Controls | Effect |
|---|---|---|
| Detail level | `Set tree default to:` + `Apply` + `Reset overrides` | `Apply` sets the chosen level across the entire tree (not just the selection). Nodes can still override it afterwards with their own chip. `Reset overrides` clears all detail-level overrides; it is disabled while there are none, and its tooltip shows the current count. From 10 overrides on, a confirmation dialog asks first. |
| Expansion | `Apply expansion` + `Remove expansion` + status | `Apply expansion` marks, read-only, which additional nodes the active expansion strategy would pull into the export, starting from your checkbox selection. Your selection stays unchanged; the added nodes get the `A` marker. `Remove expansion` clears only those markers - neither your selection nor the export changes. The status line reports how many nodes were marked, or that the selection already matches the expansion. |
| Solution | `Refresh tree` + `Reload solution` | `Refresh tree` re-reads the `AllowSourceLevel` setting from the active MD profile - useful after a template change when the `Source` level does not update. `Reload solution` re-analyzes the solution from disk and shows the structural difference to the previously loaded tree as a diff window ( `F5` ). |

#### Read-only snapshot mode

If the tree was rebuilt from a stored analysis instead of a live one, a yellow banner appears above the tree: "Read-only snapshot mode - Auto-Compress + Tree mutations are disabled." It offers the link `Reload solution now` ("Re-analyze the Solution from disk and switch to live mode.") and, below it, the checkbox `Allow Reload + Apply in one click` . With that option enabled, `Auto-compress` becomes active even in the snapshot tab and triggers a reload first, then applies the compression.

#### Empty state

While no solution is loaded, the tree area shows a muted icon and the sentence `No solution loaded yet.`

### 3.6 The selection chip bar

The header strip of the tree sidebar shows where the current selection came from and how many detail overrides exist.

The left pill is a read-only indicator with three states:

| State | Text | Icon | Color |
|---|---|---|---|
| `Manual` | `Manual` | Pencil | neutral |
| `Insights` | `Insights` | Light bulb | accent |

| State | Text | Icon | Color |
|---|---|---|---|
| `InsightsModified` | `Insights · edited` | Light bulb | warning |

Its tooltip: "Where the current tree selection came from. Manual = you picked it; Insights = set by adopting an insight; Insights · edited = adopted, then changed by you." The third state exists because the indicator must stop claiming a selection is purely the insight's once you have edited it. This state is display-only and is never persisted.

The right pill shows `{n} overrides` and appears only when at least one detail-level override exists. Clicking it jumps to the first node with a detail-level override, selects it and expands its parent chain. Tooltip: "Click to jump to the first node with a detail-level override."

When the selection comes from an insight, a row above the tree names it as `from: <insight name>` and offers two actions:

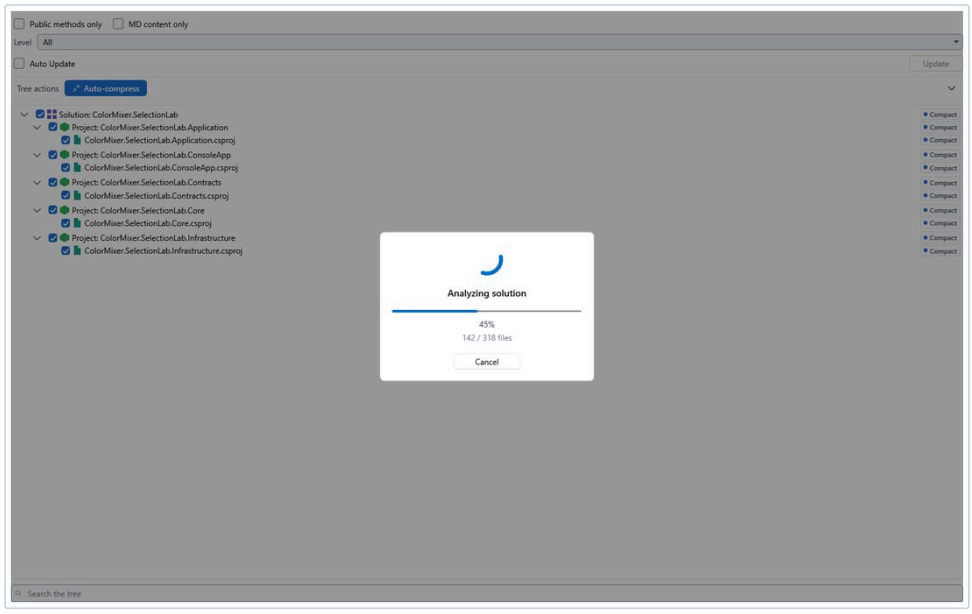
- `Manual` - "Detach this selection from the insight and edit it as a manual one. Your current checkboxes are kept unchanged."
- `Re-apply` - "Restore the original insight selection, discarding your manual changes."

## 3.7 The loading overlay

While a solution is being loaded, a modal overlay covers the tree sidebar: a dimming scrim that blocks interaction with the half-built tree, and a centered card with

- an indeterminate spinner,
- the current phase text,
- a progress bar that switches from indeterminate to determinate once per-file progress is known,
- a percentage that is shown only in the determinate state,
- a detail line such as `142 / 318 files`, which disappears when there is nothing to report,
- a `Cancel` button, enabled only while the load can still be cancelled ("Abort the running Solution analysis / load.").

The overlay appears only after a short delay (about 150 ms), so fast loads do not flash it. The phases include `Opening workspace`, `Analyzing code`, `Saving snapshot` and `Done`; restoring a stored snapshot reports `Restoring snapshot`. When you press `Cancel`, the phase reads `Cancelling...`, the load is aborted, and the tab returns to its empty state.




The loading overlay while a solution is analyzed

Note: this overlay belongs to the solution tree. The whole-solution export has its own overlay covering the entire workspace.

### 3.8 Keyboard shortcuts of this surface

The complete overview is available in the application via `F1`. The shortcuts that matter in the Context Builder:

| Shortcut | Action |
|---|---|
| `Ctrl+O` | Open a solution ( `.sln` , `.slnx` , `.slnf` ). |
| `Ctrl+N` | New solution tab. |
| `Ctrl+S` | Save the session of the active solution tab. Inside the code editor, `Ctrl+S` saves the open file instead. |
| `Ctrl+W` | Close the frontmost thing: the open file first, then the solution tab, then the workspace tab. |
| `Ctrl+M` | Build the context document from the current selection - the same as `Generate MD` . |
| `Ctrl+Enter` | Build the context document and send it - the same as `Generate & Send MD` . |
| `Ctrl+Shift+C` | Copy the model response. Plain `Ctrl+C` keeps its usual meaning inside text fields. |
| `Ctrl+Shift+A` | Run all insight producers over the active solution, from any sub-tab. |
| `F5` | Reload the solution from disk and rebuild the tree. |
| `Ctrl+F` | Put the cursor in the tree's search box and select what is already there. |
| `Esc` | Clear the tree search box while the cursor is in it. |
| `Alt+1` … `Alt+5` | Toggle a layout mode: Selection, Format, LLM, Review, Run. |
| `Alt+0` | Turn every layout mode off - the minimal view. |

Note: `Ctrl+S` and `Ctrl+W` are deliberately layered. The innermost surface that handles the key wins: with the cursor in the code editor, `Ctrl+S` saves the file and `Ctrl+W` closes the file; everywhere else in the Context Builder the same keys act on the session or the solution tab.

# 4 The Context Builder: configuration panels and code editor

The Context Builder is the workspace in which a loaded solution becomes a context document. It opens as a row of top-level tabs; this chapter covers two of them:

- the `Main` tab with its six configuration panels, which decide what the generated document contains and how it is written;
- the `Details` tab, a built-in code workspace with an editable source editor and a symbol inspector.

The run-template selector sits in the workspace header above the tabs, not inside `Main`, and it is the anchor of most panels: the prompt fields, the active MD profile, the detail presets and the expansion strategies are all resolved from the run template's context template. The selector lists every run type, but only `Manual` can be selected - `Iteration` and `Preselection` runs are **not yet released**, and `Pipeline` is a **planned placeholder** that the stored data and repositories accept, but that nothing executes yet. The tooltip of the selector also lists the graphs the current selection includes.

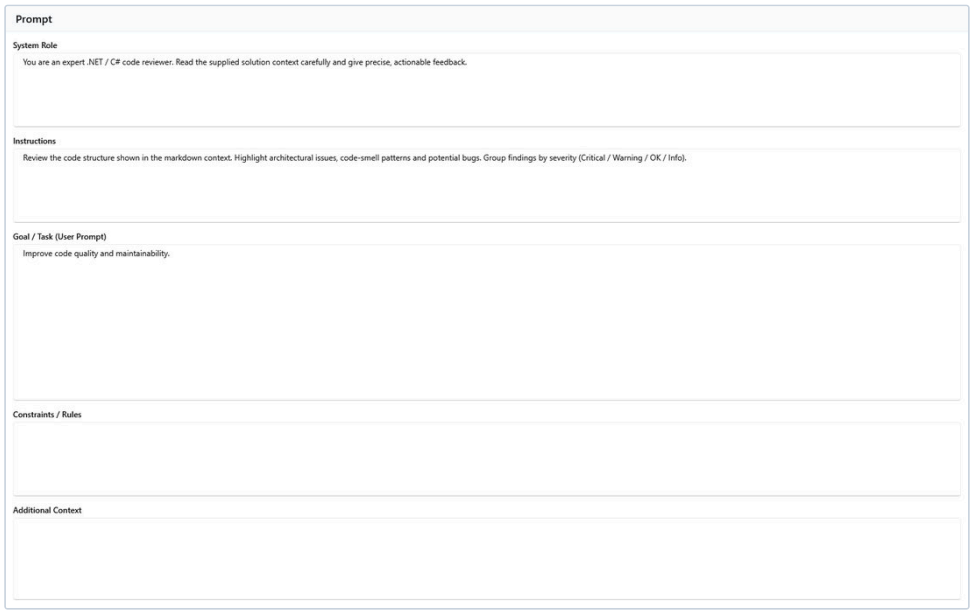
A distinction that matters throughout this chapter: everything on `Main` configures **the current run**. The panels never edit the libraries behind these settings - prompt records, MD profiles, detail presets and expansion strategies are maintained in `Settings`.

## 4.1 The six configuration panels

| Sub-tab | Configures |
|---|---|
| `Prompt` | The five prompt fields sent to the LLM |
| `Sections` | Which textual sections the document contains, and which tag schema renders each |
| `Graph Selection` | Which of the ten graphs the document contains, and the layout mode |
| `Selection Engine` | Which detail preset renders a node and how far the selection expands around it, per detail level |
| `Test Description` | The test scenario, expected behavior and acceptance criteria for evaluation |
| `Export Output` | The inner notation of the generated document |

## 4.2 `Prompt`: the five prompt fields

The `Prompt` panel holds five free-text fields, stacked vertically. All of them accept line breaks and wrap; each field scrolls internally, so long content is reachable without scrolling the panel. `Goal / Task (User Prompt)` is given twice the height of the other four because it is the primary field.




The Prompt panel with its five fields

| Label | Role | Where the initial value comes from |
|---|---|---|
| `System Role` | The LLM system message; sets the model's stance for the run | The active prompt record. If no prompt record resolves, the fallback `You are a senior .NET software architect. Stick to Solid principles` is used |
| `Instructions` | Standing instructions, prepended to the prompt | The active prompt record. Fallback: `- Be precise` / `- Use provided context only` |
| `Goal / Task (User Prompt)` | The actual task. `Adopt selection` writes a fix task here | The active prompt record when it defines one; otherwise empty. This field is stored with the session |
| `Constraints / Rules` | Boundary conditions, appended to the prompt | The active prompt record when it defines one; otherwise empty. Stored with the session |
| `Additional Context` | Free additional context, appended **after** the generated solution context | The active prompt record when it defines one; otherwise empty. Stored with the session |

The tooltips of the fields read:

- `System Role`: `The system role / persona - sent as the LLM system message; sets the model's stance for the run.`
- `Instructions`: `Standing instructions - how the model should approach the task; prepended to the prompt sent to the LLM.`
- `Goal / Task (User Prompt)`: `The main user prompt - the task the model should perform over the Solution context. 'Adopt selection' writes a fix task here.`
- `Constraints / Rules`: `Constraints and rules the model must follow - appended to the prompt to bound the answer.`
- `Additional Context`: `Extra free-form context appended after the generated Solution context (e.g. background the analyzer cannot see).`

**Per-area visibility.** Each of the five areas can be hidden. Which areas appear is owned by the prompt record: on the `Prompts` settings page there is one checkbox per area next to the field label, and the setting is re-read whenever a prompt is applied. Note the following:

- Hiding an area does **not** clear its text. The value stays in the field and still goes into the assembled request.
- A hidden area also costs no height, and it leaves the keyboard tab order.
- If no prompt record resolves at all (no template, no prompt reference, deleted prompt), all five areas are shown and the fallback texts apply.

**Switching the run template.** Picking a different run template applies that template's context template and reloads all five fields from its prompt record. The `Goal / Task (User Prompt)` field is then pre-filled from the template's default user prompt hint - but only while it is empty; your own typing is never overwritten.

`Adopt selection`**.** The `Adopt selection` action on an Insights card writes a prepared fix task into `Goal / Task (User Prompt)`, activates the insight's symbols in the Solution Tree, and switches to the `Main` tab. Nothing is sent automatically; you review the prompt and start the run yourself.

### 4.3 `Sections`: the tag schema per output section

The `Sections` panel is the configurator for **one run**. It never changes the MD profile library - that is what the `MD Profile` page under `Settings` is for.

`Active MD Profile` **card.** The card shows the resolved profile name, a `Built-in` pill when the profile ships with the application, and a source line. The resolution chain is:

1. the run template selects a context template;
2. the context template names an MD profile;
3. that MD profile is used.

If the context template names no profile, or the named profile can no longer be found, the globally marked default profile is used instead and the source line says so, for example `Default MD profile 'X' (template specifies none)`.

If nothing resolves at all, the panel shows an empty state and hides the slot list:

> No MD profile is resolved for the current run template. Pick a run template whose context template references an MD profile, or attach a profile in Settings -> MD Profiles.

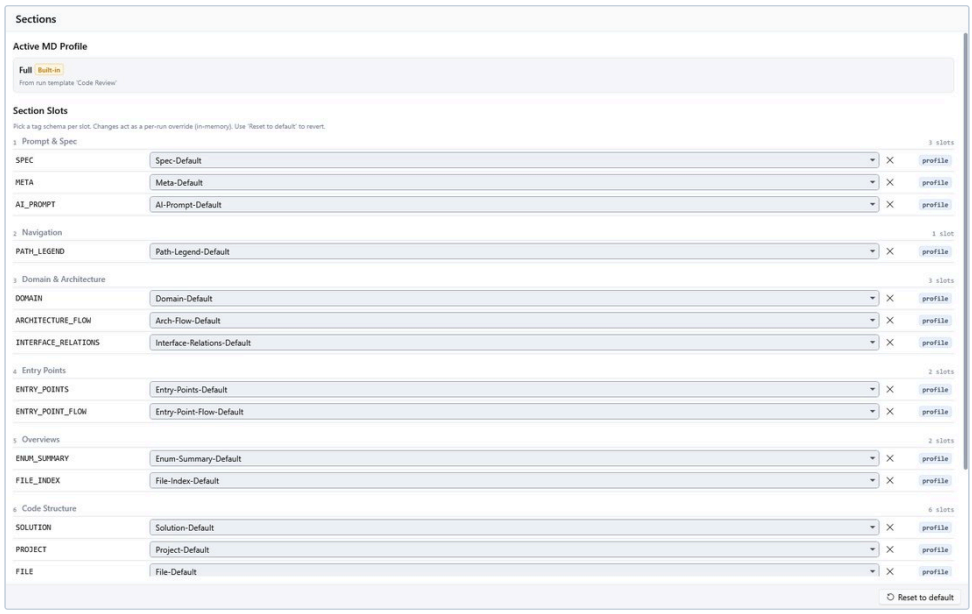
`Section Slots` . The list holds 17 slots in six groups, in render order:

| Group | Slots |
|---|---|
| `Prompt & Spec` | `SPEC` , `META` , `AI_PROMPT` |
| `Navigation` | `PATH_LEGEND` |
| `Domain & Architecture` | `DOMAIN` , `ARCHITECTURE_FLOW` , `INTERFACE_RELATIONS` |
| `Entry Points` | `ENTRY_POINTS` , `ENTRY_POINT_FLOW` |
| `Overviews` | `ENUM_SUMMARY` , `FILE_INDEX` |
| `Code Structure` | `SOLUTION` , `PROJECT` , `FILE` , `CLASS` , `INTERFACE` , `ENUM` |

Each group header carries its number and a slot counter ( `1 slot` / `N slots` ). Each slot row offers:

- the slot label, shown in monospace uppercase;
- a dropdown with the tag schemas whose schema type matches the slot. Selecting nothing disables the section for this run - the tooltip says `Empty selection disables the section for this run.` ;
- a clear button ( `Clear slot (section disabled for this run)` ), which empties the dropdown;
- a source badge with one of three values: `default` = built-in, `profile` = taken from the active MD profile, `override` = per-run override.

A summary card counts the state across all 17 slots: `{n} slots populated | {n} empty | {n} run overrides` .




The Sections panel: the active MD profile and the section slots

`Reset to default` (bottom right of the panel) discards all per-run slot overrides and reverts to the active MD profile; the tooltip reads `Discards all per-run slot overrides and reverts to the active MD profile.` and the status line confirms `Reverted to MD profile defaults.`

The status line reports active overrides as `1 slot override active for this run.` or `{n} slot overrides active for this run.` , and says `No MD profile resolved - pick a run template first.` in the empty state.

**Persistence.** Your overrides live in memory for the current run and are written with the session (see "Workspace, sessions and snapshots"); reopening a session restores them. Switching the run template re-resolves the active MD profile but keeps your overrides until you reset them.

A clarification card at the bottom of the panel draws the line between this panel and the next one:

> **Sections vs. Graph Selection:** This panel exposes 17 non-graph section slots for direct configuration. The MD profile itself has 21 non-graph slots: the panel does not expose the four level-resolved slots `CLASS`, `INTERFACE`, `ENUM` and `METHOD`. The six graph slots (`METHOD_GRAPH`, `METHOD_USED_BY_GRAPH`, `SERVICE_DEPENDENCY_GRAPH`, `LAYER_MAP`, `CLASS_DEPENDENCY_GRAPH`, `ROLE_GRAPH`) live in the Graph Selection sub-tab, for 27 MD-profile slots in total.

Note: four entries appear in both panels. `Architecture Flow`, `Interface Relations`, `Entry Points` and `Entry Point Flow` are section slots here and also graphs in `Graph Selection`.

### 4.4 `Graph Selection`: which graphs go into the document

The panel header carries the title `Graph Selection` and, on the right, the `Layout (compression)` checkbox. It is a per-tab override of the app-global setting `Settings → General → Default graph layout`; its initial value comes from the active run template, with the app setting as fallback. It controls the compact/adaptive presentation for **all** graphs and for some non-graph sections (for example `Entry Points`).

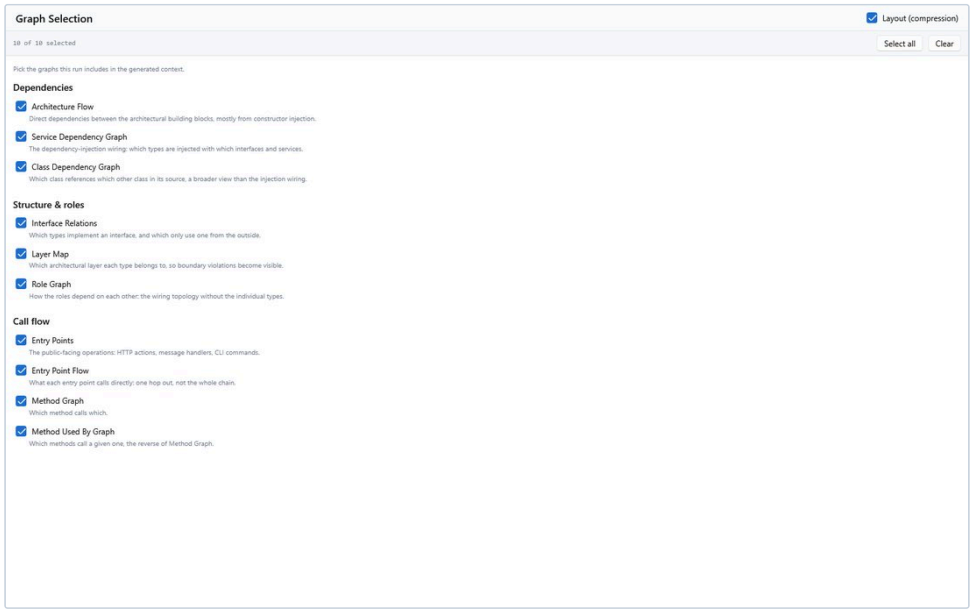
The toolbar below shows the running count `{n} of {m} selected` on the left and two buttons on the right:

| Button | Effect | Tooltip |
|---|---|---|
| `Select all` | Checks every graph | `Check every graph.` |
| `Clear` | Unchecks every graph | `Uncheck every graph.` |

The body opens with the help text `Pick the graphs this run includes in the generated context.` and then lists ten graphs in three groups. Every entry is a checkbox plus an explaining sentence, because `Method Graph` and `Method Used By Graph` cannot be told apart from their names alone:

| Group | Graph | What it shows |
|---|---|---|
| `Dependencies` | `Architecture Flow` | Direct dependencies between the architectural building blocks, mostly from constructor injection |
| `Dependencies` | `Service Dependency Graph` | The dependency-injection wiring: which types are injected with which interfaces and services |
| `Dependencies` | `Class Dependency Graph` | Which class references which other class in its source, a broader view than the injection wiring |
| `Structure & roles` | `Interface Relations` | Which types implement an interface, and which only use one from the outside |
| `Structure & roles` | `Layer Map` | Which architectural layer each type belongs to, so boundary violations become visible |
| `Structure & roles` | `Role Graph` | How the roles depend on each other: the wiring topology without the individual types |
| `Call flow` | `Entry Points` | The public-facing operations: HTTP actions, message handlers, CLI commands |
| `Call flow` | `Entry Point Flow` | What each entry point calls directly: one hop out, not the whole chain |
| `Call flow` | `Method Graph` | Which method calls which |
| `Call flow` | `Method Used By Graph` | Which methods call a given one, the reverse of `Method Graph` |

**Start state.** A graph that has a schema type starts checked when the corresponding slot is filled in the active MD profile. The four graphs without a schema type - `Architecture Flow`, `Interface Relations`, `Entry Points` and `Entry Point Flow` - always start checked, regardless of the profile. Because the start state follows the active MD profile, two solutions can show this panel pre-filled differently. Switching the run template rebuilds the list and resets it to this start state.




The Graph Selection panel with the ten graphs in three groups

Note: three output options that affect the same export are not on this panel. `Include line numbers`, `Include quality metrics (QUALITY_HOTSPOTS + complexity/coupling attributes)` and `Include config files that may contain secrets (appsettings, *.config, launchSettings)` are part of the context template and are edited under `Configuration → Templates`.

## 4.5 `Selection Engine`: presets and expansion per detail level

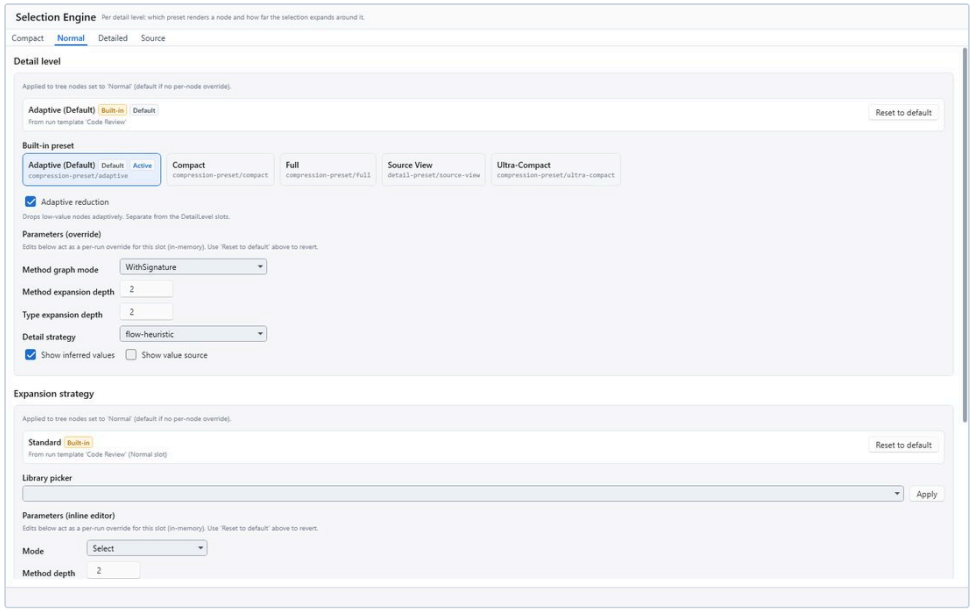
The `Selection Engine` panel configures two axes per detail level: which detail preset renders a node, and how far the selection expands around it. Its subtitle states it in one line: `Per detail level: which preset renders a node and how far the selection expands around it.`

The panel has four sub-tabs - `Compact`, `Normal`, `Detailed` and `Source` - and opens on `Normal`. Each sub-tab contains two cards.

**Card 1:** `Detail level`

- A slot description explains when the level applies:
- `Compact`: `Applied to tree nodes set to 'Compact' - minimal output, token-efficient.`
- `Normal`: `Applied to tree nodes set to 'Normal' (default if no per-node override).`
- `Detailed`: `Applied to tree nodes set to 'Detailed' - maximum detail without source code.`
- `Source`: `Applied to tree nodes set to 'Source' - detailed plus source-code block.`
- The active preset is shown with its name and, when applicable, a `Built-in` and/or `Default` pill, plus a source line such as `From run template 'X'`, `Default preset`, `Run override (built-in picker)` or `Custom override (this run)`.
- `Reset to default` discards the run override for this slot and reverts to the run-template or default preset.
- `Built-in preset` offers the shipped presets as clickable cards, each marked `Default` and/or `Active` where applicable. A click activates that preset for this slot as a per-run override. The built-ins offered here are `Full`, `Adaptive (Default)`, `Compact`, `Source View` and `Ultra-Compact`.
- `Adaptive reduction` is a separate axis: `Adaptive reduction is a separate axis - switches this slot to an adaptive built-in.` The help text below it says `Drops low-value nodes adaptively. Separate from the DetailLevel slots.` Switching it on selects an adaptive built-in; switching it off selects `Full`.
- `Parameters (override)` holds six fields. Every edit here is a per-run override held in memory; use `Reset to default` to revert it.

| Field | Type | Meaning |
|---|---|---|
| `Method graph mode` | Dropdown: `Full`, `SemanticOnly`, `WithSignature`, `WithBody` | How the METHOD_GRAPH renderer displays call relationships |
| `Method expansion depth` | Integer | Maximum traversal depth for methods; `0` = top level only |
| `Type expansion depth` | Integer | Maximum traversal depth for types (inheritance/composition); `0` = the type itself |
| `Detail strategy` | Dropdown: `flow-heuristic`, `aggressive`, `none` | Render strategy for member bodies. The tooltip warns: `Has a strong impact on output Tokens.` |
| `Show inferred values` | Checkbox | Shows default values inferred from the code (constructor defaults, auto-properties). The checkbox starts from the active preset; the shipped default preset `Adaptive (Default)` has it on |
| `Show value source` | Checkbox | Annotates values with their source (for example literal / config / DI) |




The Selection Engine panel with the detail-level card and its parameters expanded

### Card 2: `Expansion strategy`

- A slot description per level, for example `Compact`: `Applied to tree nodes set to 'Compact' - narrow expansion, low token footprint.`
- The active strategy with a `Built-in` pill and its source line.
- `Reset to default` discards the run override for this slot and reverts to the run-template or built-in default strategy.
- `Library picker`: a dropdown over the strategy library plus an `Apply` button. `Apply` loads the selected strategy as a per-run override into the editor below. The list contains the shipped strategies and any custom strategies you created under `Settings → Expansion Strategies`.
- `Parameters (inline editor)` holds 16 fields in three blocks: the basic grid (`Mode`, `Method depth`, `Type depth`, `Max nodes total`), `Include flags` (8) and `Exclude filters` (4). All edits are per-run overrides held in memory.

| Block | Fields |
|---|---|
| Basic | `Mode` (`Reachable`, `Select`, `Frontier`), `Method depth`, `Type depth`, `Max nodes total` |

| Block | Fields |
| --- | --- |
| `Include flags` (8) | `Callers`, `Callees`, `Used-by members`, `Inheritance hierarchy`, `Interface implementations`, `Created types`, `Used types`, `Injected dependencies` |
| `Exclude filters` (4) | `Framework types` (`System.*` / `Microsoft.*`), `External assemblies`, `Test classes`, `Generated code` |

The tooltips explain the individual fields:

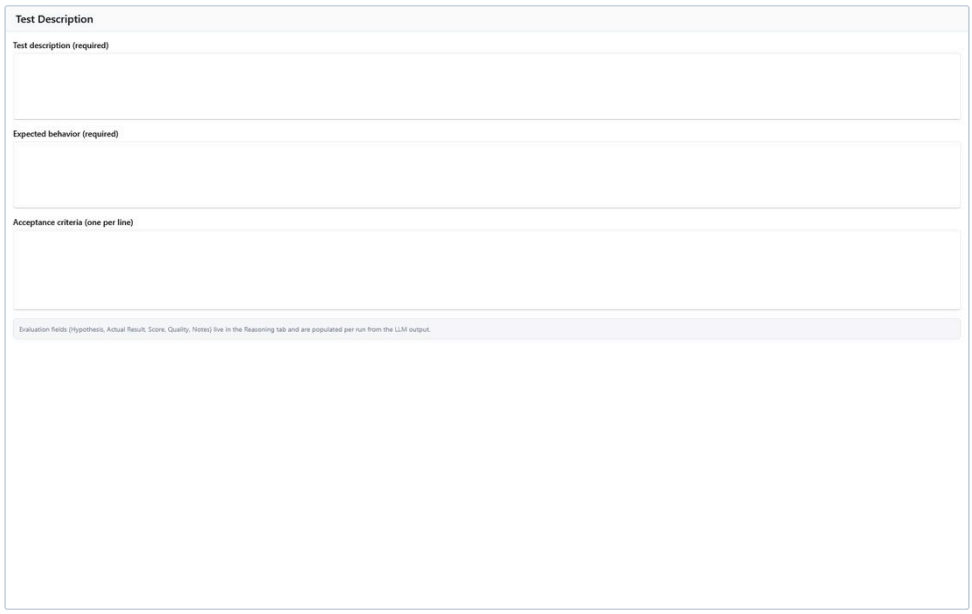
- `Mode`: `ExpansionStrategy.Mode - controls the walker mode when expanding Nodes.`
- `Method depth`: `Max BFS depth for method walks. Ignored when Mode=Select; capped at 1 effectively when Mode=Frontier.`
- `Type depth`: `Max BFS depth for type walks (Dependencies / UsedTypes / Injected).`
- `Max nodes total`: `ExpansionStrategy.MaxNodesTotal - global cap on the number of expanded Nodes per Run. Blank = no cap (can lead to long render times).`
- `Callers`: walk callers of selected methods (reverse direction), used-by
- `Callees`: walk methods called by the selected methods (forward direction)
- `Used-by members`: include members that reference the selected type or method as a member reference
- `Inheritance hierarchy`: walk base classes and derived classes of selected types
- `Interface implementations`: walk interfaces this type implements plus classes implementing the selected interface
- `Test classes`: skip test fixtures detected via `[TestClass]` / `[Fact]` / `[Theory]` attributes or `*Tests` / `*Test` type names
- `Generated code`: skip `[GeneratedCode]`-annotated types and `*.g.cs` / `*.designer.cs` source files

**Status line.** The panel has one status line for all eight slot editors, and it names the slot that last reported, for example `Normal detail: Parameter override active - Reset to revert.` or `Normal expansion: Strategy override active - Reset to revert.` Activating a built-in preset answers `Active detail preset set to '{Name}' for this slot.`; applying a strategy from the library answers `Active expansion strategy set to '{Name}' for this slot.`

**Which level counts for the export?** The `Normal` slot supplies the global fallback: a node the per-node walk does not cover is rendered with the effective preset of the `Normal` slot, and it falls back to the effective expansion strategy of the `Normal` slot. The budget estimate also reads `Max nodes total` from the `Normal` expansion slot. The per-node map, however, is built from all four slots, so a node set to `Detailed` is rendered with the `Detailed` pair, and a node set to `Source` with the `Source` pair.

### 4.6 `Test Description`

Three free-text fields. The first two have a minimum height of 100 px, the acceptance criteria field 120 px and a monospace font - one criterion per line.




The Test Description panel

| Label | Tooltip |
|---|---|
| `Test description (required)` | `The test scenario being evaluated - what this run is meant to verify. Required for evaluation runs.` |
| `Expected behavior (required)` | `The expected / correct behavior - the LLM output is scored against this during evaluation.` |
| `Acceptance criteria (one per line)` | `Acceptance criteria, one per line - the concrete checks the answer must satisfy.` |

While the test description is not empty, the Context Builder header shows it next to `Test:`; when it is empty, the header hides the pair entirely.

A footnote in the panel explains where the evaluation data lives:

> Evaluation fields (Hypothesis, Actual Result, Score, Quality, Notes) live in the Reasoning tab and are populated per run from the LLM output.

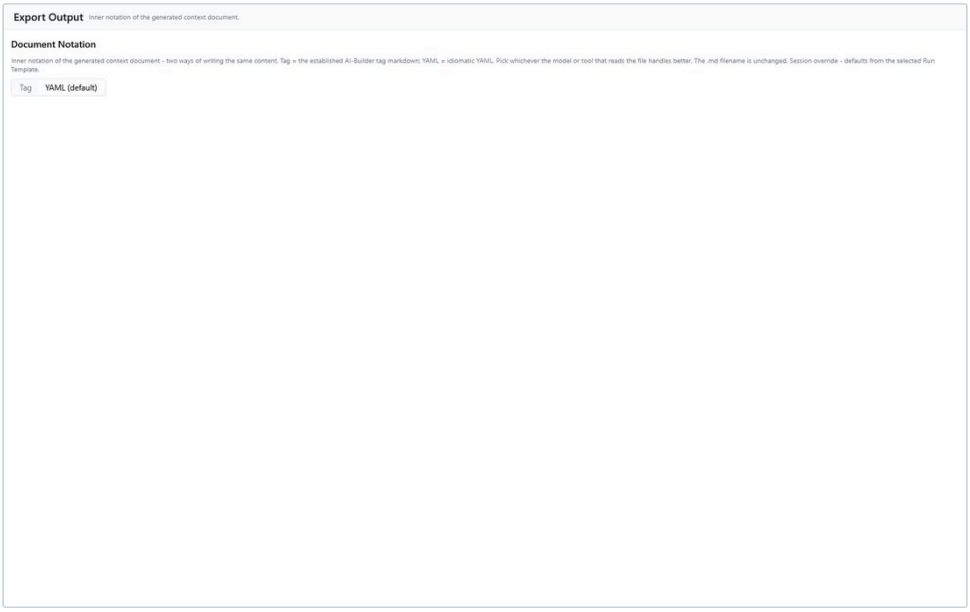
Note: the two `(required)` labels are a hint, not a condition. The fields are not validated, and leaving them empty blocks neither saving a session nor starting a run. The runs that fill the evaluation fields (`Iteration`, `Preselection`) are not yet released.

### 4.7 `Export Output`: the notation

The `Export Output` panel makes exactly one decision: the inner notation of the generated context document. The subtitle states the scope: `Inner notation of the generated context document.`

| Option | Tooltip |
|---|---|
| `Tag` | `Use the established AI-Builder tag notation inside the document.` |
| `YAML (default)` | `Use YAML notation inside the document (default). Lossless - only the notation differs.` |

The explanation text below the `Document Notation` heading reads:




The Export Output panel

> Inner notation of the generated context document - two ways of writing the same content. Tag = the established AI-Builder tag markdown; YAML = idiomatic YAML. Pick whichever the model or tool that reads the file handles better. The .md filename is unchanged. Session override - defaults from the selected Run Template.

Note: the tooltip of the `Export Output` tab also mentions the export file format and included sections. Those controls are not part of this panel - the document sections are chosen in the `Sections` panel.

## 4.8 The `Details` tab: a built-in code workspace

The `Details` tab is the built-in code workspace, and it is the one place in the GUI where the application changes your source code: a file that loads cleanly from disk becomes editable, and `Save` or `Ctrl+S` write it back to disk, preserving its encoding and line endings. The tab is always available, independent of the layout mode.

### Opening and closing files

- Click a class or method in the Solution Tree. Its source file opens in the `Details` tab and the editor scrolls to that symbol.
- There is one tab per `.cs` file, de-duplicated by absolute path: opening a second class that lives in an already open file re-activates the existing tab instead of opening a second one.
- The tab header is the class name that opened the file. A trailing `*` marks unsaved changes.
- Every open file keeps its own editor state (cursor and scroll position survive a tab switch) and its own symbol inspector.
- Close a file with the `×` on its tab or with `Ctrl+W`.
- The empty state reads: `Click a class or method in the Solution Tree to open its source file here.`
- Whether a click in the tree also switches to the `Details` tab depends on `Reveal Details tab on tree click` under `Settings → General`. When it is off, the tab still updates on click, but you open it yourself.

### The editor

The editor is editable, shows line numbers, does not wrap long lines, uses a monospace font at the body size (12 px) and colors C# syntax. Freshly typed characters are re-colored after a short pause - the full re-coloring pass is debounced by 250 ms so typing stays smooth.

The safeguard that makes a writing editor bearable: the editor is set to read-only **first** and becomes editable **only** when the file loads cleanly from disk. A file that is missing or cannot be read receives a comment placeholder - `// File not available on disk: ...` or `// Could not open file: ...` - and stays read-only, so the placeholder can never be saved over the real file.

### The toolbar

The toolbar shows the file path on the left (elided; the full path is in its tooltip), the status line next to it, then the three action buttons, and the inspector toggle on the far right.

| Button | Effect | Enabled when |
|---|---|---|
| `Check` | Compiles the **buffer** in memory against the live solution - no file is written. Results appear in the panel `Problems in this buffer` | a diagnostics service is available |
| `Diff` | Shows the pending changes (buffer vs. the file on disk) in the read-only panel `Changes vs. saved file` | the file is editable and changed |
| `Save` (accent, `Ctrl+S`) | Writes the buffer back to the real file, preserving its encoding and line endings, then re-analyzes the solution | a save delegate is available, and the file is editable and changed |

`Ctrl+S` while the cursor is in the editor saves the **file** - it shadows the Context Builder's save-session shortcut.

### What happens when you save

1. Without a writer or without a detected encoding, the status reads `Save unavailable - file not editable.` and nothing happens.
2. The content to be written is captured **before** the write, because the buffer can still change during the asynchronous write.
3. **External-change guard:** if the file changed on disk since you opened it, a dialog asks: `"{Title}" changed on disk since you opened it.` / `Overwrite the external changes with your edits?` (title `Save File`). Anything other than `Yes` sets the status `Save cancelled - the file changed on disk.` and writes nothing. `Yes` means last writer wins - the external changes are gone.
4. If the write fails, the status becomes `Save failed: {message}` and the save is aborted.
5. On success, the written content becomes the new clean baseline and the status runs through `Saved.`, `Saved. Re-analyzing...` and finally `Saved and re-analyzed.` - or `Saved (re-analyze skipped).` if the re-analysis fails. The save itself has already succeeded in that case.

### `Check` and the diagnostics panel

The status messages of a check are:

| Status | Meaning |
|---|---|
| `Checking...` | The check is running |
| `No problems.` | The buffer produced no diagnostics |
| `N errors, M warnings.` | The non-zero counts, with correct singular/plural forms; for example `1 error, 3 warnings.` or `2 errors.` |
| `Check unavailable - no live solution for this file (e.g. a read-only snapshot).` | No live solution is available |
| `Check failed: {message}` | The check raised an exception |

The `Problems in this buffer` panel lists one row per problem: the severity (errors in red, warnings in the warning color), the diagnostic ID, `Ln {n}` and the message. Click a row to jump to that line in the editor. Close the panel with its `×`.

### The diff panel

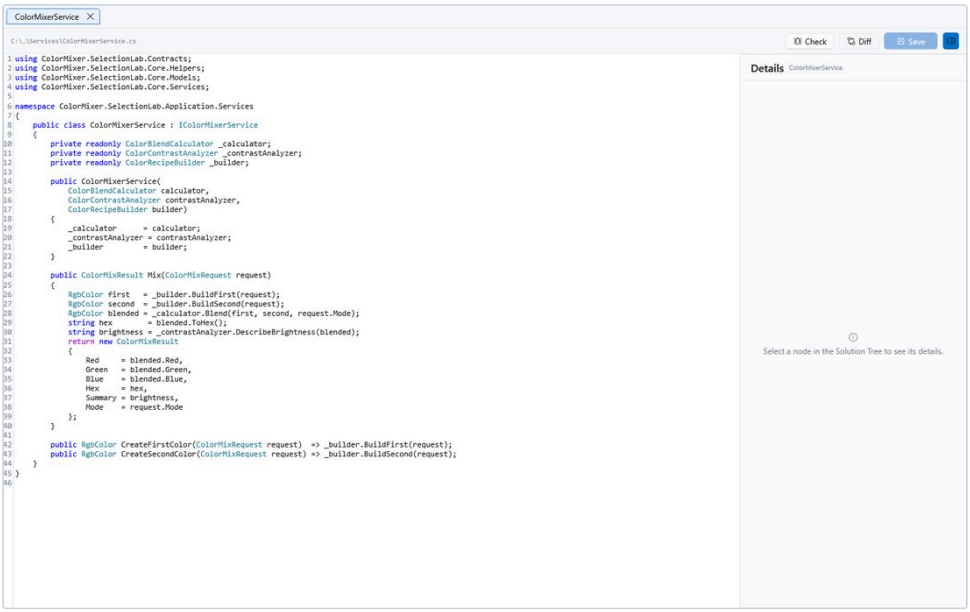
`Diff` shows your edits against the file on disk in the read-only panel `Changes vs. saved file`; added and removed lines are colored. If the buffer differs only in whitespace or newlines, the panel says: `(No line-level changes; only whitespace or newline differences.)` Close it with its `×`.

### The symbol inspector

The inspector sits on the right of the editor. Its toggle is in the toolbar, it is **on by default**, and the column has a fixed width of 340 px (there is no resize handle). Its tooltip reads: `Show or hide the symbol inspector (metadata, dependencies, used-by, metrics) on the right.`

### Unsaved changes when closing

- Closing a single tab with unsaved edits asks `"{Title}" has unsaved edits.` / `Discard them and close the file?` (title `Close File`). The tab closes only on `Yes`.
- Loading a different solution closes **all** open file tabs. If any of them have unsaved edits, one combined prompt appears first. It names up to five files and adds `... and {n} more` beyond that: `"{n} open file(s) have unsaved edits:"` … `The loaded solution changed. Discard these edits and close the files?` (title `Discard Unsaved Files`). Declining keeps the tabs open. Note that the solution context has already changed at that point, so the choice is "lose the edits" or "keep them and save them", not "undo the load" - a kept tab still saves to its own absolute path.




The code editor with syntax highlighting and the Save button

## 4.9 `Selection Engine` panel vs. symbol inspector

The two share the word "detail", but they have nothing to do with each other:

| | `Selection Engine` **panel** | **Symbol inspector** |
|---|---|---|
| Visible as | Main panel with the header `Selection Engine` | Right-hand column of the code editor, 340 px |
| Role | **Configuration**: which preset and which expansion strategy apply per detail level | **Display**: what the currently selected node is |
| Writes | Per-run overrides (in memory, saved with the session) | Nothing - a pure projection |

The inspector shows the selected symbol in blocks, and each block hides itself when it is empty:

- the title line `Details` plus the symbol name;
- a kind pill and, when available, an accessibility pill;
- the signature in monospace;
- the XML doc summary;
- `Namespace:`, `Declared in:` and `Base:`;
- `Interfaces`;
- `AI tags`;
- `Metrics`;

- `Overview` (structural facts for solution, project, folder and file nodes);
- `Dependencies` ;
- `Used by` , with a caveat text when the fan-in list can be incomplete: `Name-based fan-in (inverted forward dependencies); same-named types across namespaces may merge.`

The empty state reads: `Select a node in the Solution Tree to see its details.`

Because every open file tab carries its own inspector, the details shown follow the file tab you switch to. Note that the inspector deliberately contains no source-code block - the editor beside it already shows the code.

# 5 The Context Builder: document, runs and results

The Context Builder is the workspace in which a loaded solution becomes a context document. It is where you decide what goes into the document, how the document is shaped and where you send it to a model - and where you read what the model answered. Each solution you open gets its own tab; the tab header shows the solution file name and appends `*` while there are unsaved changes.

The workspace has three fixed parts:

- the **global header** above everything: solution identity with `Open`, the `Max MD size` budget, the mode chips, and the run-template group with the run actions;
- the **Solution Tree** in the left sidebar - the scope selector for everything you render. It can be collapsed to a slim rail; its width is remembered;
- the **tab area** on the right: the configuration panels, the document, the model response and the run views.

## 5.1 Finding your way

Which tabs and panels are visible depends on the five mode chips in the header ( `Selection` , `Format` , `LLM` , `Review` , `Run` ); they are described in "The Context Builder: layout and solution tree". `Details` and `MD Input` are always visible. `Reasoning` and `Active Run` , described below, need the `Run` chip.

## 5.2 The run template and the run actions

The run-template group of the header carries two selectors and three actions.

- `Run Template` - selects the run template for this tab. Picking one switches the active context template, reloads the prompt fields from it and fills `Goal / Task (User Prompt)` from the template - but only while that field is still empty; text you have typed is never overwritten. The current view shows no confirmation message for this switch.
- `LLM` - selects the model-profile override for subsequent sends from this tab. The first entry carries a `Default` badge and stands for the standard resolution order (model pinned by the run template → profile marked as default → first profile). The choice remains active until you select another entry, close the tab or restart the application.

| Action | What it does | Shortcut |
|---|---|---|
| `Export whole Solution` | Renders the entire solution into `MD Input` , ignoring the tree selection. | - |
| `Generate MD` | Renders the current tree selection into `MD Input` - the same action as `Create MD` on the `MD Input` tab. | Ctrl+M |
| `Generate & Send MD` | Renders the current selection and sends it immediately. | Ctrl+Enter |

Below the actions, `Recent runs:` shows up to three chips for the most recent runs of the current session (run number, name, run type, status, time). The row is hidden while there are none.

## 5.3 The `MD Input` tab - the generated document

This is where every render lands. Its tooltip reads `The generated markdown context - review or edit it, then send it to the LLM.`

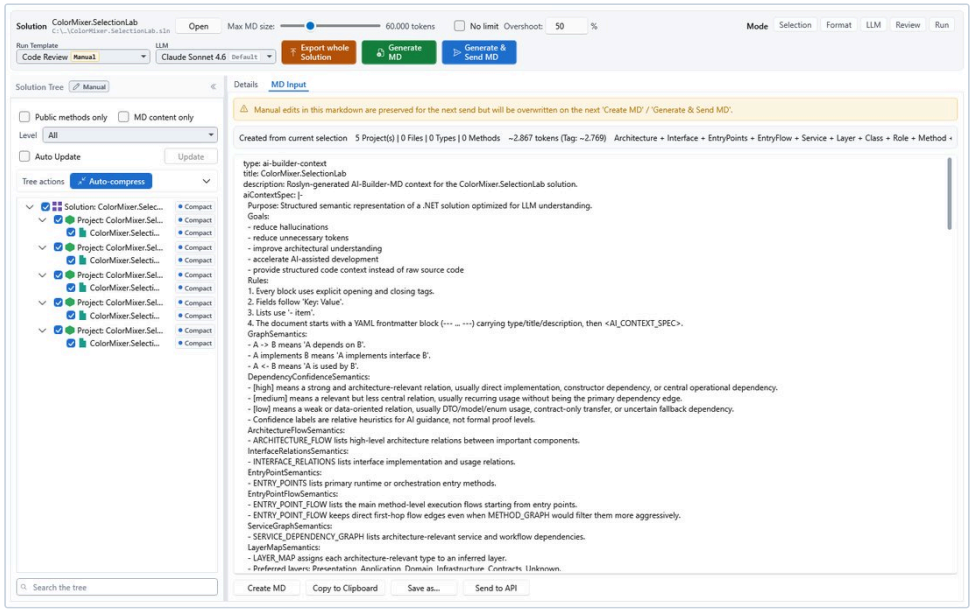
A permanently visible warning banner sits above the document: `Manual edits in this markdown are preserved for the next send but will be overwritten on the next 'Create MD' / 'Generate & Send MD'.`

Below the banner a card reports what the document contains:

| Field | Content |
|---|---|
| `Created from current selection` | fixed label |

| Field | Content |
|---|---|
| selection summary | `N Project(s) \| N Files \| N Types \| N Methods` |
| token estimate | `~12,400 tokens` - measured with the tokenizer of the resolved model profile (fallback: the default tokenizer). When the notation rewrite produced a different size than the tag baseline, the baseline is added: `~12,400 tokens (Tag: ~10,900)` |
| graphs | the graph sections actually present in the document, as short labels joined with `+` (for example `Architecture + Service + Layer`); `-` when the document contains none |
| code mode | `Enabled` or `None` - whether any node contributed source code |

The document itself is an editable text box with a permanently visible scrollbar. Your edits are taken over when the box loses focus, not on every keystroke, so a large whole-solution document stays responsive.




The MD Input tab with a generated context document

### The buttons under the document

| Button | Action |
|---|---|
| `Create MD` | Regenerates the document from the current tree selection and the panel settings - no API call. Tooltip: `Generates the MD output from the current Tree selection + Template settings - no API call. The text can then be edited and 'Send to API' pressed. (Ctrl+M)` |
| `Copy to Clipboard` | `Copies the current MD input text to the clipboard (e.g. for manual pasting into an external LLM tool).` |
| `Save as…` | `Writes the current MD input text to a .md file you pick. Disabled while there is nothing to save.` |
| `Send to API` | `Sends the current MD input to the active LLM (or the Override-selected one). The response appears in the 'LLM Response' tab.` |
| `Cancel` | Visible while a request is running: `Aborts the running LLM request - the server may still finish computing, but the response is discarded.` |
| `Pause` | Visible while a request is running: `Pauses the iterative Run after the current step (Iteration Run Templates only).` |
| `Resume` | Visible while a run is paused: `Resumes a paused iterative Run.` |

To the right of the buttons the status line reports the progress and the result of the last action.

`Save as…` opens a save dialog titled `Save context document` with the filter `Markdown files (*.md)|*.md|All files (*.*)|*.*`. The suggested name is derived from the loaded solution - `<SolutionName>-context.md` - so a folder of saved documents does not end up as a pile of files all called `context.md`. On success the status line reads `Context document saved to <path>`; on failure it reads `Save failed: <message>`.

### The `LLM Response` tab

The response to the last send. The text box is editable, and the buttons below it are:

- `Copy to Clipboard` - `Copies the LLM output to the clipboard. (Ctrl+Shift+C)`.
- `Use as MD input` - `Copies the LLM output into the markdown input (tab 'MD Input') so a follow-up run can iterate on the previous answer.` The view switches to `MD Input` and the status line confirms with `LLM output copied to MD input.` The button is disabled while the response is empty.

### What happens when you press `Send to API`

A send runs in phases, and each phase has a visible outcome in the status line.

**Preflight.** The send is validated before anything is transmitted. A failed preflight writes `Send failed:` plus the reason:

| Situation | Message |
|---|---|
| the document is empty | `no markdown to send. Run Create MD first.` |
| no model profile exists | `no model profiles defined. Settings > Model Profiles.` |
| an API profile has no key | `API key for model '<name>' is missing. Settings > Model Profiles.` |
| no executor for the run type | `RunType '<type>' is not supported. Only Manual, Iteration and Preselection runs can be started.` |
| an iterative run without a loaded solution | `no solution loaded.` |
| an iterative run without selected nodes | `no <granularity>-nodes selected in the tree. Mark some nodes first.` |

**Budget confirmation.** For `Iteration` and `Preselection` runs (not yet released, see "Run types: what is released") a cost estimate runs before the send and a dialog can warn that the configured node cap would be exceeded. If you decline that dialog, nothing happens - no request is sent and no status message is written.

**Auto-session.** When a solution is loaded, every run is stored with a session. If the tab has no session yet, one named `Untitled` is created automatically so the run is persisted; the status line appends `· Auto-session 'Untitled ...' created - rename it via the Sessions tab.` You can rename it later on the `Sessions` tab (see "Workspace, sessions and snapshots"). Without a loaded solution, the run can still execute, but no auto-session or snapshot is created and the status reports `(not persisted - open a solution first)`.

**Execution with streaming.** The status line shows `Sending to <model> ...`. For a manual run the answer streams live into `LLM Response`, and the view switches there with the first chunk.

**Completion.** On success the status line reads `Done. <model> · <ok>/<total> nodes · in <n> / out <m> tokens`, followed by the auto-session note when one was created. A manual run leaves you on `LLM Response`. On failure it reads `Send failed: <reason>`.

**Post-processing.** The answer is then examined for thinking blocks, tool calls, findings and an evaluation; the `Reasoning` tab is reloaded for the finished run and the `Recent runs:` row is refreshed.

While a request is running, `Cancel` and `Pause` are available. `Cancel` writes `Cancelling ...`, `Pause` writes `Pausing at next step boundary ...`. `Resume` continues a paused iterative run and reports `Resumed run completed. <ok>/<total> nodes.` (or `Paused again at step boundary.` / `Resume failed: <reason>`).

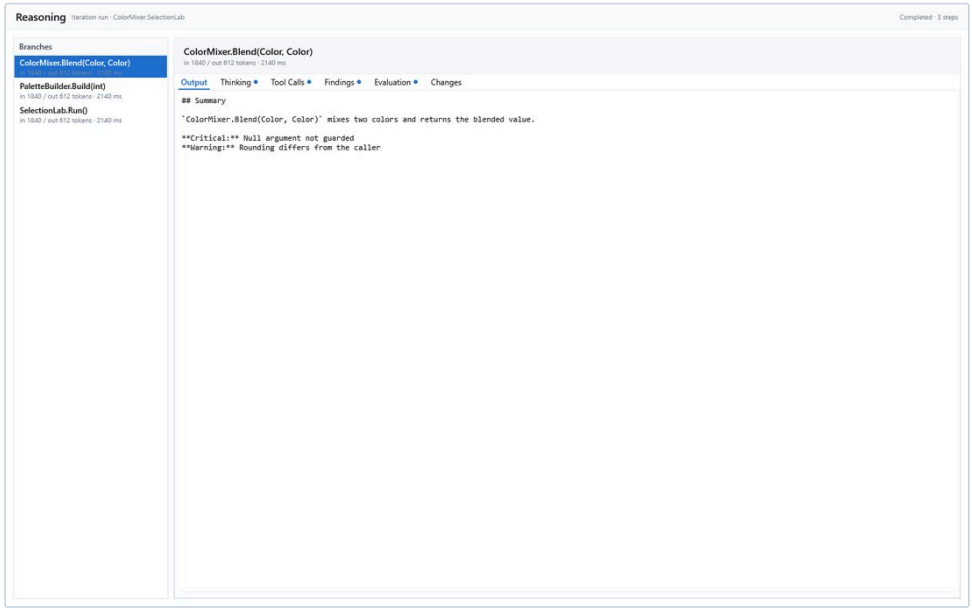
Note: A `Manual` run has no pre-flight cost estimate - it starts as soon as you press the button. The confirmation dialog exists only for `Iteration` and `Preselection`.

## 5.4 The `Reasoning` tab - what the model did

The `Reasoning` tab shows a finished run branch by branch. It requires the `Run` mode chip.

The header shows the run's name and its status. On the left, `Branches` lists the steps the run was split into; each entry shows the branch name and, as a subtitle, its token usage and latency ( `in <n> / out <m> tokens · <ms> ms` ) or the error message if the branch failed. Empty state: `No run analyzed yet. Branches appear after a run completes.`

For the selected branch there are six sub-tabs. A small dot in a tab header marks that the selected branch has content for that sub-tab.




The Reasoning tab: branches on the left, the model output of the selected branch on the right

| Sub-tab | Content | Empty state |
|---|---|---|
| `Output` | the raw model output, read-only | - |
| `Thinking` | the model's thinking blocks | `No thinking blocks in this step. Thinking is requested through the system prompt, and the model answers with <thinking>...</thinking> tags - a model that does not use them produces no blocks here.` |
| `Tool Calls` | the tool-use calls parsed from the answer | `No tool calls in this step. Tool calls are read from <tool_use>... </tool_use> tags in the model's answer.` |
| `Findings` | findings parsed from the answer, filterable by severity | `No findings to show. Either this step produced none, or the severity filters above are hiding them.` |
| `Evaluation` | `Hypothesis` , `Actual result` , `Score` , `Quality` and `Notes` | `No evaluation in this run. The parser looks for Hypothesis / Result / Score / Quality sections in the LLM output.` |
| `Changes` | files the answer proposed a new state for, diffed against the working tree | `No diff to show. A diff appears when the answer states the full content of a file it changed and that file is found in the loaded solution - which is what the Code Change prompt asks for.` |

**Findings here are not the Insights.** The `Findings` sub-tab shows what the model itself reported in its answer, sorted into the four severity categories `Critical` , `Warning` , `OK` and `Info` (the answer marks them as `**Critical:**` , `**Warning:**` , `**OK:**` and `**Info:**` ). All four filters are on when the tab opens; clear a filter to hide that severity. The separate `Insights` tab (mode chip `Review` ) is about your code, not about the answer - it holds the application's own code-quality heuristics for the loaded solution. Two different sources, two different tabs.

#### `Changes` - the proposed file states

Above the file list a summary line states how many files the answer proposed and how many were resolved against the working tree: `<n> file(s) proposed, <m> resolved against the working tree.` If the answer did not use the expected block form, the summary says so instead: `This answer states no file content in the applicable form. The Code Change prompt is the one that asks for it.`

A proposed path that cannot be shown as a diff is named together with its reason:

| State | Displayed text |
|---|---|
| no solution loaded | `No solution loaded - the path cannot be resolved.` |
| resolved | the absolute path |
| ambiguous | `Ambiguous - <n> files of this solution match this path.` |
| found, unreadable | `Found but unreadable: <path>` |
| too large | `Too large to show as a diff: <path>` |
| not found | `No file of this solution matches this path.` |

Selecting a file shows a unified diff between the file on disk and the state the answer proposed. When the two are identical, the diff pane says `(No line-level changes; the proposed content matches the file on disk.)` Malformed blocks are reported in warning color above the list - for example an unterminated block (usually the answer hit its output limit mid-file), a block without a path, empty content, a path named twice, a stray closing marker, or content wrapped in a Markdown fence (stripped, so the file is still usable).

Note: The `Changes` sub-tab is a view only. There is no button that applies a proposed change; to edit a file and write it back, use the `Details` tab.

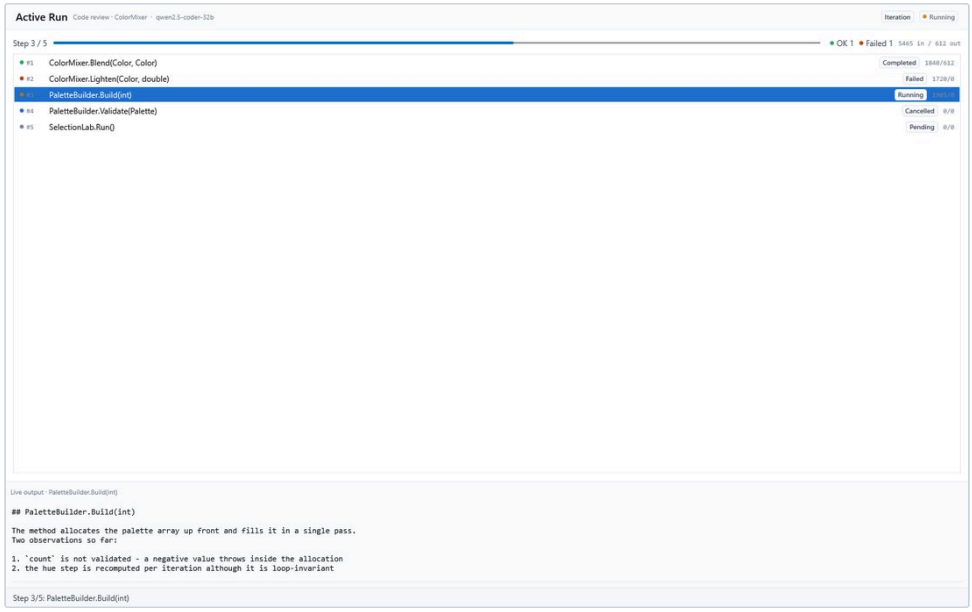
### 5.5 The `Active Run` tab - progress while a run is running

The `Active Run` tab requires the `Run` mode chip. Its header carries the title `Active Run`, the run name and the model, a run-type pill and a status pill that switches between `Running` and `Complete`.

| Display | Meaning |
|---|---|
| `Step <n> / <m>` with a progress bar | the current step of the run |
| `OK <n>` | successfully completed steps |
| `Failed <n>` | failed steps |
| `<n> in / <m> out` | input and output tokens of the whole run |

The step list shows one row per step: its number ( `#1` ), its name, a status pill ( `Pending` , `Running` , `Completed` , `Failed` , `Cancelled` , `Skipped` ) and the step's own token count as `<in>/<out>` (tooltip `Input / output tokens of this step.` ). The status is also color-coded on the dot in front of each row.

Select a step to read what it streamed: a pane under the list titled `Live output · <step>` shows the model output that step produced, with the tooltip `Model output streamed by this step. Select text to copy.` The first output of a run opens this pane automatically; after that your own selection decides.




The Active Run tab during a multi-step run

Empty state: `No run started yet. Steps appear here as soon as a run is running.`

Note: Only `Manual` runs are released, and a manual run is a single request - it appears here as one step named after the run. The multi-step view is what an iterative run will show once it is released.

## 5.6 Token, cost and progress displays

| Where | Display | Updated by |
|---|---|---|
| Header | `Max MD size` (target tokens) and `Overshoot %` | your input |
| Header | `+X% over budget` - how far the last document actually exceeded the target; empty when within budget | the last render |
| `MD Input` header | token estimate, selection summary, active graphs, code mode | the last render |
| `MD Input` status line | `Done. <model> · <ok>/<total> nodes · in <n> / out <m> tokens` | the end of a send |
| `Active Run` | step counter, OK/Failed, tokens in total and per step | live progress |
| Load overlay | a percentage plus a detail line such as `142 / 318 files` | solution analysis progress |
| Export overlay | an indeterminate ring plus `Rendering whole solution...` | a whole-solution render |
| `Sections` panel | `<n> slots populated \| <n> empty \| <n> run overrides` | slot changes |
| `Graph Selection` panel | `<n> of <m> selected` | checkbox changes |
| Tree sidebar | selection provenance and `<n> overrides` | your tree edits |

Note: Token counts are always available. A pre-flight dialog also shows `Estimated cost: $X.XX USD` when input and output prices have been entered on the selected model profile; without those prices, no cost is shown.

## 5.7 From solution to context document, step by step

The following walkthrough follows the five decision points that measurably change the result: scope, detail and sections, prompt, budget, and the way the document leaves the application.

### 1. Choose the scope

Check the nodes that belong in the document in the Solution Tree and, where needed, set a detail level with the chip at the end of a row. `Auto-compress` in the tree toolbar sets a sensible level for every method in one click; `MD content only` shows exactly what will end up in the document. Details: "The Context Builder: layout and solution tree".

### 2. Shape detail, sections and graphs

With the `Selection` and `Format` chips on, the `Main` tab offers the `Selection Engine` (detail preset and expansion strategy per detail level), `Sections` (a tag schema per text section, or none), `Graph Selection` (which of the ten graphs to include) and `Export Output` ( `Tag` or `YAML` notation). All of them have usable defaults from the run template. Selection Engine and Sections overrides are stored with the session; Graph Selection and the Tag/YAML choice are reloaded from the run template when the session is opened or the template changes and are not session-persisted. Details: "The Context Builder: configuration panels and code editor".

### 3. Write the prompt

With the `LLM` chip on, the `Prompt` panel holds the five fields that are sent with the document: `System Role` and `Instructions` come from the context template's prompt record, `Goal / Task (User Prompt)` , `Constraints / Rules` and `Additional Context` are yours and are stored with the session. The goal is the one field you normally fill in.

### 4. Set the budget

`Max MD size` in the header caps the document (8,000 to 200,000 tokens, default 60,000 from the active pipeline profile); `No limit` switches the cap off, and `Overshoot:` sets the tolerated headroom above the target. The renderer converges on the target by lowering detail and dropping the least relevant content, never by cutting a block in half; `+X% over budget` reports what the last render actually used.

### 5. Render the document

| Action | Scope | Shortcut |
|---|---|---|
| `Create MD` (on `MD Input` ) | the current tree selection plus all panel settings | Ctrl+M |
| `Generate MD` (in the header) | the same render, from the same selection | Ctrl+M |
| `Export whole Solution` (in the header) | the entire solution, ignoring the tree selection and needing no analysis run | - |
| `Generate & Send MD` (in the header) | renders the current selection and sends it immediately | Ctrl+Enter |

All of them write their result to the same place: the document in `MD Input` , and the view switches there.

Details of `Export whole Solution` :

- The heavy render runs off the UI thread. A dimmed overlay with a spinner and the text `Rendering whole solution...` blocks the workspace until it is done, and the window stays responsive.
- The solution is already analyzed, so this does not analyze it again.
- A second click while a render is running is ignored.
- The button has no disabled state: a missing solution or a missing context template is reported in the status line instead - `No solution loaded - open a solution first.` or `No active context template - cannot render the whole solution.` A render error is reported as `Export failed: <message>` .
- It does not change your tree selection.

### 6. Get the document out

You have three ways out of the application:

- `Save as…` on the `MD Input` tab writes the document to the `.md` file you choose.
- `Copy to Clipboard` copies it for pasting into another tool.
- `Send to API` (or `Generate & Send MD` ) sends it to the configured model and shows the answer in `LLM Response` .

### 7. Save your work

The tree state is expensive to reconstruct, so the workspace tracks unsaved changes and marks the tab header with `*` while there are any. Three situations ask you to save: the explicit save command, closing a tab, and closing the window.

- `Save Session` in the tab bar (Ctrl+S) saves the current tab as a session. The dialog is pre-filled with a name of the form `<SolutionName> - <date> <time>` and the notes of an existing session. If the state is unchanged since the last save, a dialog asks whether to overwrite anyway. If the chosen name already exists for this solution, a dialog offers to save anyway or to go back and pick another name. On success a message confirms `Session "<name>" saved.` (or `updated.`) together with the number of selected nodes and the active template.
- **Closing a tab** (Ctrl+W, or the close button on the tab) with unsaved changes asks `Save before closing?`. `Yes` saves all unsaved sessions in the tab - if one save fails or is cancelled, the tab stays open. `No` discards the changes. `Cancel` aborts the close. The question can be switched off with the `Tab-close confirmation` setting under Settings → General; clearing that setting makes such a tab close straight away and discard its unsaved edits.
- **Closing the application** always asks, whatever the `Tab-close confirmation` setting says, because one unasked confirmation would discard every unsaved session at once.

**Auto-save** persists your work in the background. It is on by default and saves every 5 minutes; `Auto-save enabled` and `Auto-save interval (minutes)` under Settings → General configure the timer (values below one minute are raised to one), but changes take effect only after the next application start. Auto-save skips only tabs without a session ID; the `Untitled ...` session created by a first send is included even if it was never named through the dialog. If an auto-save fails, a warning appears once per session: `Auto-save could not save one of your open sessions. Your changes remain unsaved, and other tabs will continue to auto-save. Use Save Session to retry and see any detailed error.`

**Two access points and one lock.** The `+` button in the tab bar opens another solution tab (`New Solution tab (Ctrl+N)`), and `Save Session` sits next to it. The host carries three shortcuts: Ctrl+S saves the session, Ctrl+N opens a new tab, Ctrl+W closes the current tab. While a run is active in any tab, the whole tab bar is disabled so you cannot switch tabs and lose the run state - clicks on it simply do nothing until the run has finished.

## 5.8 Run types: what is released

| Run type | Status |
|---|---|
| `Manual` | Released. One request over the current document. This is the type you select in the workspace. |
| `Iteration` | Not yet released. One request per selected node; the executors are wired, but the type cannot be selected yet. |
| `Preselection` | Not yet released, same as `Iteration`. |
| `Pipeline` | Planned. It is accepted by the stored data, but nothing executes it. |

The `Run Template` selector lists every template, but only templates of type `Manual` can be chosen; the other entries are shown disabled with a tooltip explaining that they are not released yet. The entry that is already selected stays selectable regardless of its type, so you can always return to it.

Because only `Manual` runs are released, the `Active Run` tab normally shows a single step. Manual runs do not use the cost-threshold confirmation, but a pre-flight dialog can still appear when the selected model profile defines a context window and the request would exceed it.

# 6 Insights in the desktop app

The Insights tab is the app's own quality-review surface. It runs a set of heuristics (the *producers*) over the C# solution currently loaded in the Context Builder and reports what they find as individual findings: long methods, fat interfaces, missing async conventions, unused types, design smells, architecture problems, security-sensitive calls, and more. Each finding tells you what was detected, where it was detected, how expensive a fix is estimated to be, and - for many rules - a recommended approach. The complete rule catalog is described in "Insights: the code quality catalog" in the General manual.

Machine-written files that are committed to the repository - `*.Designer.cs` and `*.g.cs` outside the build output - are read, so the references they hold count, but the producers report no findings in them. The copies a build writes under `obj` or `bin` stay out of the analysis.

Use the tab before a code review, after a larger refactoring, or whenever you want to hand a concrete, pre-selected slice of a problem to your LLM.

## 6.1 Where the Insights tab lives

Insights is not an entry of the main navigation; it is a sub-tab of the Context Builder and appears when the `Review` mode is active:

- `Review` is one of the mode chips in the Context Builder header. Its tooltip reads `Review - walking through quality findings. Adds the Insights tab and the Test Description panel. (Alt+4)`.
- Modes combine (their areas are added together), so you can keep `Review` active alongside other modes.
- The tab sits between `Details` and `MD Input` in the Context Builder's tab strip.
- The tab header reads `Insights`, or `Insights (N)` while N findings are loaded. The number falls as you dismiss findings and comes back when you restore them.
- An info icon next to the page title summarizes the surface: heuristic hints for the currently loaded solution, pick a category on the left, read the findings on the right, and click `Analyze Solution` to re-evaluate.

The heuristics are not part of the solution-load pipeline itself - the tab triggers them (see "Running the analysis").

## 6.2 Categories, principles and severities

Findings are classified on three independent axes: category, SOLID principle and severity. All three are visible and filterable on the tab.

### Categories

The left column is a single-select category navigator titled `Categories`. `All` is always the first row and is selected by default; it shows the findings of every category, sorted worst-first.

Six categories are always listed, even at count 0: `Code Quality`, `Security`, `Async`, `Design`, `Architecture` and `Other`. A `Compression` row appears only if a finding maps to it - the shipped heuristics do not produce one.

| Category | What typically lands there |
|---|---|
| `Code Quality` | Complexity hotspots, long methods, dead private methods, fat interfaces, large classes, anti-patterns, deprecated APIs, LINQ in a loop, reflection usage, multi-concern hotspots, findings from the latest LLM run |
| `Security` | Methods that call security-sensitive external APIs, such as insecure deserializers, process execution or dynamic assembly loading |
| `Async` | Missing `Async` suffix on task-returning methods, async methods without a cancellation token |
| `Design` | Unused types, public mutable fields, methods with many parameters, layer violations, event-subscription concentration, unresolved XAML bindings |
| `Architecture` | Circular namespace dependencies, breaking changes to the public contract |
| `Other` | The catch-all row for findings that do not map to a named category (for example build/UI hints); the shipped heuristics do not currently produce one |

| Category | What typically lands there |
|---|---|
| `Compression` | Only shown if a finding maps to it; not produced by the shipped heuristics |

The number next to each category is a live filtered count: it reflects the active principle and severity filters, so it changes while the category selection stays. The category itself is not part of that count.

Each row has the tooltip `Show only this category's insights in the pane on the right.`

### Principles (SOLID)

The first filter row is labeled `Principle`. It contains an `All` chip plus one chip per principle: `S`, `O`, `L`, `I`, `D` and `·` (Other).

Each finding carries one or more SOLID badges in its card header, derived from the rule that produced it. The badge is neutral (a letter in a pill); hover it to see the principle name and explanation.

| Chip | Principle | Meaning (badge tooltip) |
|---|---|---|
| `S` | Single Responsibility | A type or method should have exactly one responsibility - a single reason to change. Long methods, large classes, many parameters, or mixed side effects bundle several concerns. |
| `O` | Open/Closed | Code should be open for extension but closed for modification. Exposed, mutable state breaks invariants and forces modification instead of extension. |
| `L` | Liskov Substitution | Subtypes must be usable in place of their base type without breaking its contract. A divergent member of a type family (pattern drift) violates substitutability. |
| `I` | Interface Segregation | Clients should not depend on methods they do not use. A fat interface forces unnecessary coupling - split it into smaller, role-based contracts. |
| `D` | Dependency Inversion | Modules should depend on abstractions, not on concrete implementations - and inner layers never on outer ones. Layer violations and circular dependencies break this direction. |
| `·` | Other | Convention/maintainability hint without a clear SOLID mapping (e.g. async convention, dead code, unused type, correctness smell, build hint). |

How the row filters:

- It is multi-select: several principles can be active at once.
- No active chip means "all principles". The `All` chip (tooltip `Clear all SOLID filters.`) clears the row.
- A finding without a clear SOLID mapping matches only the `·` chip.
- Otherwise one matching principle is enough for the finding to be shown.

### Severities

The second filter row is labeled `Severity`. It contains an `All` chip plus `Critical`, `Warning`, `Info` and `Ok`, each with a colored dot and a count (tooltip `Toggle this severity (multiple can be active).`).

Every finding has exactly one severity. Internally the levels are ranked Critical (10) > Warning (3) > Info (0.5) > Ok (0) - deliberately not the order in which the levels are declared, so that sorting is worst-first.

The row is multi-select; no active chip means "all severities". The `All` chip (tooltip `Clear the severity filter.`) clears the row.

### How the three axes combine

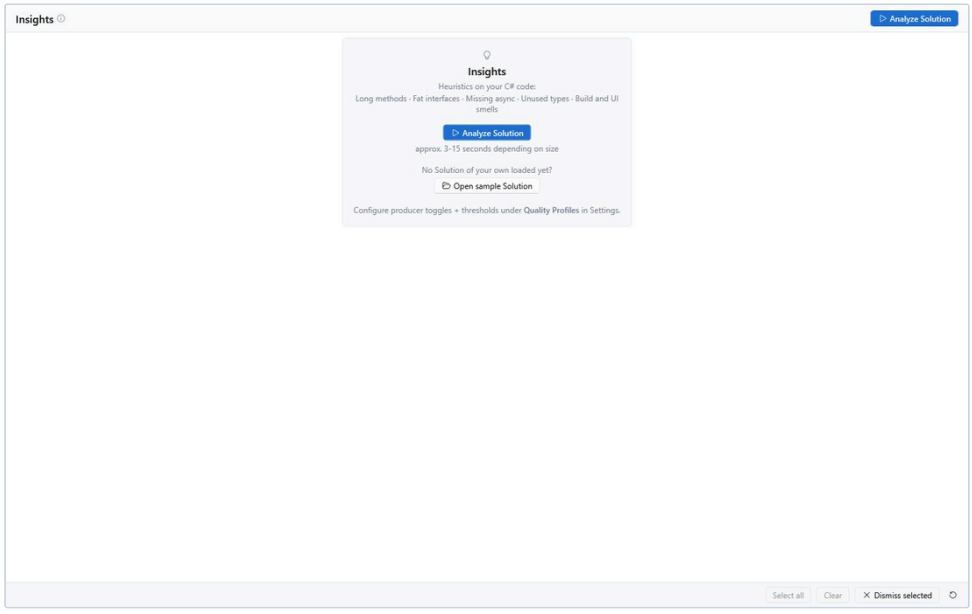
- Category, principle and severity are combined with AND: a card must pass all active filters.
- A chip whose count is 0 is disabled - an empty bucket cannot be selected.
- Both filter rows are hidden until the tab has run at least once; the first-run card is shown instead.
- Sorting is fixed and cannot be changed: severity (worst first), then category, then title. Within the right pane the findings are grouped by severity; each group header shows a colored dot, the level name and the number of cards.
- The tab has no free-text search. Use the category, principle and severity filters.

## 6.3 Running the analysis

1. Make sure a solution is loaded in the Context Builder.
2. Click `Analyze Solution` in the page header, or press `Ctrl+Shift+A`. The shortcut works from any Context Builder sub-tab. The button tooltip reads `Runs all producers over the active Solution. (Ctrl+Shift+A)`.
3. While the run is in flight the whole tab is dimmed by a translucent scrim and shows a progress ring. The run happens off the UI thread, so the rest of the app stays usable.
4. Depending on solution size the run takes roughly 3-15 seconds. The first-run card names the same range: `approx. 3-15 seconds depending on size`.
5. The results replace the previous list, grouped by severity. The tab header now shows the count.

After the run the page header also names the quality profile it used, `Profile: <Name>`, left of `Analyze Solution`. Its tooltip reads `Quality profile used by the last analysis. The active template's profile wins over the one chosen in Settings.` This matters when a template pins a profile: the built-in `Onboarding` and `Documentation` templates pin a profile that switches the producers off, and an empty result then says nothing about the solution. A result restored from an earlier run shows no profile, because which profile that run used is not stored; the label appears once the background recomputation (see "Automatic runs") or your next run replaces the restored result.

A run evaluates the 24 analyzer rules in the shipped configuration. The active Quality Profile can switch individual exposed rules off, so the effective set depends on your profile. The rule catalog and the profile options are described in "Insights: the code quality catalog" in the General manual; the profile itself is edited under `Settings` → `Quality Profiles`. The first-run card points there as well: `Configure producer toggles + thresholds under Quality Profiles in Settings.` LLM findings are promoted from a model run rather than evaluated by this analysis run.




The first-run card of the Insights tab

Re-running does not undo your triage: dismissals and intentional markings are applied to the new result again.

### Automatic runs

- **Setting.** `Auto-trigger insights on solution load` (`Settings` → `General` → `Session & Analysis`) is off by default. Its tooltip: `When on, analyzer insights run automatically after every solution load. When off, use the 'Analyze Solution' button in the Insights tab to run them manually.`
- **First-run exception.** On the first solution load after a fresh installation, the heuristics run once even when the setting is off, so that a new user does not sit in front of an empty tab. This happens once per installation.
- **Background recomputation.** In the normal case the tab shows the last stored result for the solution (if there is one) and recomputes the heuristics in the background. When that background run is ready, the panel adopts the fresh result by itself, so what you read matches the current code - which is also what `Adopt selection` needs to resolve its symbols. A `Ready - N`

`insights precomputed.` banner with a `Show` button belongs to this state; because the panel adopts the result as soon as it is ready, the banner is normally not seen.
- A background run that finishes after you switched to another solution is discarded.

Note: if you trigger the analysis with no solution loaded, the run cannot produce findings; the tab still leaves the first-run view, and the right pane then shows the empty hint described below.

Note: if a single heuristic fails during a run, its findings are simply missing from the result. The tab does not show a separate error message for it. Re-run the analysis if a category looks unexpectedly empty.

### 6.4 The health strip

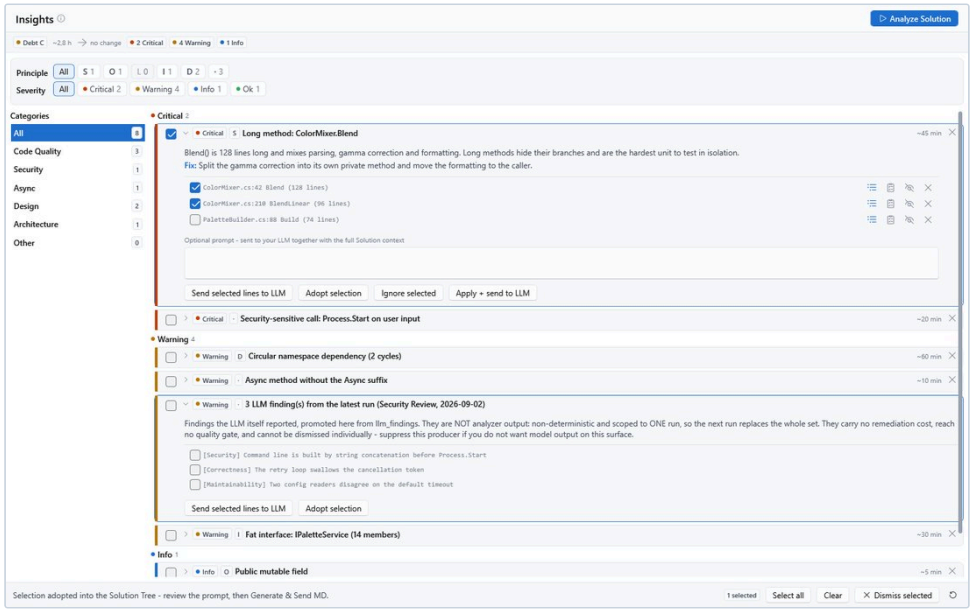
As soon as at least one finding is loaded, a summary strip appears above the filters. It is computed over all loaded findings and is **not** affected by the filters.

| Element | Meaning | Tooltip |
|---|---|---|
| `Debt` pill with a colored dot | Coarse technical-debt grade from `A` (best) to `E` (worst), estimated from the summed remediation minutes of all loaded findings. The dot is green for A/B, amber for C, red for D/E | `Coarse technical-debt grade (A best ... E worst), estimated from summed remediation minutes.` |
| `~... min` / `~... h` / `~... d` | Estimated total remediation effort, order of magnitude (never exact). Under one hour it is shown in minutes, under one working day in hours, above that in working days (one working day = 480 minutes) | `Estimated total remediation effort (order-of-magnitude).` |
| Trend chip | Change in the debt of the two most recent snapshots that carry a debt measurement: an upward amber arrow for worse, a downward green arrow for better, a flat muted arrow for unchanged. The label is signed, for example `+45 min` or `-1.5 h`, or `no change`. The chip is hidden until at least two such snapshots exist | `Technical debt decreased since your previous analysis (lower is better).` / `Technical debt increased since your previous analysis.` / `Technical debt is unchanged since your previous analysis.` |
| `N Critical` | Number of critical findings | `Critical-severity insights` |
| `N Warning` | Number of warning findings | `Warning-severity insights` |
| `N Info` | Number of info findings | `Info-severity insights` |

Each severity pill is hidden when its count is 0. The strip as a whole is hidden while no findings are loaded.

The debt grade is a rough order-of-magnitude estimate; it sums the per-finding remediation estimates. The grade thresholds are: `A` below 30 minutes, `B` from 30 minutes, `C` from 2 hours, `D` from one working day, `E` from three working days.

## 6.5 The finding cards




Insights: finding list with categories, severities and suggested fixes

The right pane lists the findings of the selected category as accordion cards, grouped by severity. Cards are collapsed by default.

Each card has:

- a 4-pixel severity ribbon on the left edge: red for Critical, amber for Warning, blue for Info, green for Ok;
- a checkbox that selects the card for the batch action `Dismiss selected` (tooltip `Select for a batch action (Dismiss selected).`);
- a one-line header with a chevron, the severity pill (colored dot + level name), the SOLID badges, the finding title and the estimated remediation cost (for example `~15 min`, hidden when the rule gives no estimate). The title is truncated in the row; hover it to see the full title;
- a dismiss icon at the top right (tooltip `Dismiss - marks the finding as seen; it will not come back`), hidden for findings that cannot be dismissed.

Clicking the header expands the card; an expanded card gets an accent-colored border. The header tooltip is `Expand or collapse this insight - shows its description, affected code, and fix actions.`

An expanded card contains:

- the description of the finding;
- a `Fix:` line with the recommended approach, shown only when the rule provides one;
- the list of affected items (see below);
- an optional prompt box (see "The optional prompt box");
- an action row with `Send selected lines to LLM`, `Adopt selection` and `Ignore selected`.

### Affected items

Each affected item is one monospaced line with its own checkbox (tooltip `Select this line for 'Send selected lines to LLM'.`).

- Every affected item is listed.
- The list is height-capped (about 260 pixels) and scrolls; long lists are virtualized, so they stay responsive.
- Note: other consumers of the same analysis (for example the MCP tools) may cap their output. The app itself shows the full list.

A line can carry a lifecycle badge and up to five icon buttons. Only the buttons that apply to the line are shown:

| Icon | Action | Visible when | Tooltip |
|---|---|---|---|
| Tree/list icon | Reveal the symbol in the Solution Tree | the line resolves to a symbol | `Reveal this symbol in the Solution Tree.` |
| Clipboard | Queue as work | the line resolves to a symbol and nothing is queued on it yet | `Queue as work - record this line as something to fix, with a run template of your choice. Nothing is started; the line stays visible and reports 'In progress'.` |
| Checkmark in a circle | Mark the queued work as done | work is queued on the line | `Mark the queued work as done. It records your claim - nothing re-scans to confirm it - and the line stays visible so a later run can contradict it.` |
| Crossed-out eye | Mark this finding as intentional (suppress) | the line resolves to a symbol | `Mark this finding as intentional - it stops being reported here and to the MCP. The quality gate (--fail-on / solution_metrics) still counts it. Export the solution config to share the decision via the .aicb.json sidecar.` |
| Dismiss (×) | Dismiss just this line | the line resolves to a symbol and belongs to a dismissable finding | `Dismiss just this line - it stops being reported here and to the MCP until you restore dismissed findings. Use 'Intentional here' instead when the hit is by design and the team should see the decision.` |

`Reveal` selects and expands the matching node in the Solution Tree (the persistent sidebar) and deliberately does not switch the active tab. A symbol that cannot be found in the loaded tree is ignored silently.

Marker and relation lines that carry no symbol cannot be revealed, queued, suppressed or dismissed on their own - there is nothing stable to key them on.

### Card actions

| Button | What it does | Visible when |
|---|---|---|
| `Send selected lines to LLM` | Sends the checked affected items, together with the full solution context, to the active LLM (see below) | the finding has affected items |
| `Adopt selection` | Replaces the tree selection with the checked symbols (plus their relevant neighbors) and writes a fix task into the prompt; nothing is sent | the finding has affected items |
| `Ignore selected` | Dismisses the whole finding - the same as the dismiss icon at the top right. Tooltip: `Marks this insight as seen (dismisses it) - same as the dismiss button top-right.` | the finding can be dismissed |

A secondary action button can exist next to these ( `Runs the secondary action + optionally sends the prompt text plus the full Solution context to the active LLM.` ). It is only shown when the finding defines a secondary action; the shipped heuristics do not, so it does not appear in normal use.

## 6.6 Sending findings to your LLM

### Send selected lines to LLM

1. Expand a finding card.
2. Check the affected items you want to work on.
3. Click `Send selected lines to LLM`.

The app renders the entire solution into the `MD Input` tab, writes a prompt into the prompt's goal field, and sends it to the active LLM with the full solution context. The prompt is built from the checked lines:

```
Please review the following items from the analysis:
- <checked line 1>
- <checked line 2>
```

If no line is checked, nothing is sent and the status line reads `No detail lines selected.`

#### Adopt selection

`Adopt selection` prepares the work instead of sending it:

1. Expand a finding card and check the affected items you want to fix.
2. Click `Adopt selection`.
3. The app replaces the tree selection with the checked symbols, pulls in their relevant neighbors, writes a fix task into the prompt's goal field, and switches to the `Main` tab. The tree selection is now marked as insight-driven.

Nothing is sent. Review the prepared prompt and use `Generate & Send MD` when you are ready. The written task text looks like this:

```
Resolve the following <category> finding in this C# solution.

Finding: <title>
<description>
Recommended approach: <suggestion>

Affected items (their code - and their relevant neighbors - are selected in the context):
- <checked line 1>
...

Implement the fix directly in the code. Preserve the existing public behavior,
keep the build and tests green, and briefly summarize what you changed and why.
```

The status line reports `Adopted the insight's symbols + their context as the selection (checked = generated context); prompt prepared.` or, when no tree symbol matched, `Prompt prepared; no matching tree symbols to select.`

#### The unreleased optional prompt path

The card view contains support for an optional prompt box and a secondary action such as `Apply + send to LLM`, but only the unregistered auto-compression producer supplies those fields. No finding in the shipped application displays this box or secondary button. Use `Send selected lines to LLM` or `Adopt selection` instead.

### 6.7 Dismissing, marking as intentional, and restoring

The tab distinguishes two kinds of triage:

- **Dismiss** means "I have seen this". It is a local decision for the current solution.
- **Mark as intentional** (suppress) means "this hit is by design". It is a decision you can share with your team.

#### Dismiss

You can dismiss at three levels:

- a single card, via the dismiss icon at its top right or the `Ignore selected` button;
- a single affected item, via the dismiss icon on the line;
- a batch, via `Dismiss selected` in the action bar over all checked cards.

A dismissed finding stops being reported in the panel and to the MCP tools until you restore it. Dismissals are stored per solution, so they survive a restart and the next load of the same solution.

The action bar shows a `N selected` pill while cards are checked. `Select all` checks every card currently shown on the right (disabled while the pane is empty); `Clear` unchecks all. `Dismiss selected` is enabled while at least one checked card can actually be dismissed - it stays enabled for a mixed selection and reports the rest on the status line: `N finding(s) cannot be dismissed - they are re-reported on every run.` Cards that cannot be dismissed stay checked on purpose, because the dismissal is exactly what did not happen to them.

Dismissing is a triage decision, not a fix: it does not change your code. `Restore` brings the findings back.

#### Mark as intentional (suppress)

The crossed-out-eye icon on an affected item records a permanent suppression for that item's symbol and rule. The item disappears from the panel and from the MCP tools, but the quality gate ( `--fail-on` / `solution_metrics` ) still counts it.

Suppressions are stored in the app's database for the current solution. To share the decision with your team, export the solution configuration - the `.aicb.json` file next to your solution - from the Solutions workspace. Suppressions declared in that committed file also apply, and they survive `Restore` (see below).

The button appears only on items that resolve to a symbol; marker and relation lines cannot be marked.

#### Restore

The counter-clockwise arrow button in the action bar restores everything you hid. Its tooltip: `Reset what you hid - dismissed Insights and findings you marked intentional become visible again, and all sections are re-evaluated immediately. Suppressions declared in the committed .aicb.json sidecar stay in force.`

`Restore` has no confirmation step. It clears the dismissals and the database-side intentional markings for the current solution and immediately re-runs all heuristics. Two things are deliberately not affected:

- Suppressions declared in the solution's `.aicb.json` file stay in force - they are a committed team decision, not a local one.
- Queued work items are not cleared. `In progress` and `Done` badges stay as they are.

#### Findings that cannot be dismissed

Some findings are persistent: the dismiss affordances are hidden for them and they are reported on every run. This happens in two cases:

- The active Quality Profile has the option `Insights are dismissable (persist hidden per solution)` switched off ( `Settings` → `Quality Profiles` , section `Behaviour` ). The help text next to it reads: `Default ON. When enabled, a dismissed insight stays dismissed for that solution. When OFF, insights reappear after dismissing until the heuristic itself stops reporting them.`
- Findings promoted from the latest LLM run are always persistent. Their detail lines carry no symbol, so the panel offers neither per-line `Mark as intentional` nor dismiss controls for them. To keep model output off this surface, suppress the producer as a whole, for example through the committed `.aicb.json` sidecar.

#### If a write fails

Note: if a triage write to the local database fails, the panel still updates, but the change may not survive the next refresh. If a state does not stick, run `Analyze Solution` again and retry.

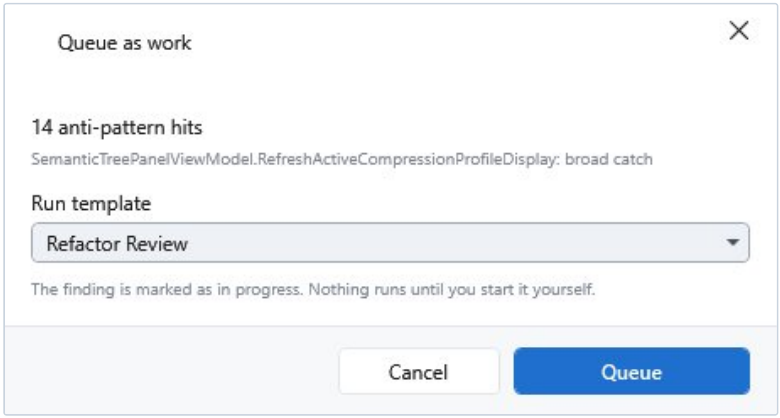
### 6.8 Queuing a finding as work

You can record an affected item as something to fix, together with the run template you want to handle it with. This is a note to yourself - **nothing is started**.

1. Expand a finding card.
2. Click the clipboard icon on an affected item. If no run templates are stored, no dialog opens and the status line reports `No run templates are available to queue against.`

3. In the dialog `Queue as work` you see the finding title, the text of the affected item, a `Run template` picker and the note `The finding is marked as in progress. Nothing runs until you start it yourself.` The picker lists all stored run templates, without filtering by run type. It is pre-filled with the run template configured for the active quality profile, if there is one; otherwise the first stored template.
4. Click `Queue` to record the item, or `Cancel` to close without queueing (Escape also cancels). `Queue` is disabled while no template is selected.
5. The item stays visible and now carries the badge `In progress`. Hover it to see the tooltip: `Queued as work to do, with the run template "...".`
6. When you are done, click the checkmark icon on the line. The badge changes to `Done`, with the tooltip `Marked as done. It is your claim, not a re-scan - the line stays visible so a later run can contradict it.`

The line stays visible in both states on purpose: a line that vanished because somebody started work on it would hide the state it is supposed to report, and a later analysis can contradict your "Done" claim.




The Queue as work dialog

The action reports itself on the status line:

| Situation | Status line |
|---|---|
| Item queued | `Queued as work to do.` |
| An item is already queued on this line | `Work is already queued for this line.` |
| Queueing failed | `Could not queue the work item.` |
| Item closed | `Marked as done.` |
| No queued item on this line | `There was no queued work on this line.` |
| Closing failed | `Could not close the work item.` |

## 6.9 Where the state is stored

- **Findings** are stored per solution, not per session, despite the setting name. `Persist insights with session` (`Settings` → `General` → `Session & Analysis`, on by default) stores the last analysis result with the solution and restores it on the next load, so you see the last state immediately without re-analyzing. When the setting is off, nothing is stored and the tab recomputes on every load.
- **Dismissals** are stored per solution and survive a restart.
- **Intentional markings** are stored in the local database for the solution and can be shared through the `.aicb.json` solution configuration file.
- **Queued work items** are stored per solution and survive a restart.

All writes are best-effort: a failure never blocks the panel, and a failed write may mean the state is not there after the next refresh.

## 6.10 Empty and loading states

| State | What you see |
|---|---|
| An analysis is running | A translucent scrim over the whole tab with a progress ring |
| The tab has never run (first start) | The filter rows and the body are hidden. In the middle sits a card with a lightbulb icon, the title `Insights`, the line `Heuristics on your C# code:`, the examples `Long methods · Fat interfaces · Missing async · Unused types · Build and UI smells`, the accent button `Analyze Solution`, the hint `approx. 3-15 seconds depending on size`, the question `No Solution of your own loaded yet?` with the button `Open sample Solution`, and the pointer `Configure producer toggles + thresholds under Quality Profiles in Settings.` |
| The selected category has nothing to show | A lightbulb icon and the hint `No insights to show here - pick a category on the left, or relax the principle/severity filters.` |
| No findings at all | The health strip is hidden, all category rows show 0, and the right pane shows the hint above |
| A background computation is ready | The green banner `Ready - N insights precomputed.` with a `Show` button. Because the panel adopts the result as soon as it is ready, the banner is normally not seen |

### The sample solution

`Open sample Solution` loads the bundled `ColorMixer.SelectionLab.sln` - a read-only sample - for a trial run (tooltip: `Loads the bundled ColorMixer.SelectionLab.sln (read-only sample) for a trial Insights run.`). The app opens it in the Context Builder; the usual load behavior then applies (first-run analysis or background recomputation). This button is the entry to the sample solution; if the file is missing, an info dialog titled `Sample not found` explains that it normally lives in the `SampleSolutions` folder next to the application.

## 6.11 Defaults and limits

| Behavior | Default / limit |
|---|---|
| Automatic analysis on solution load | Off; one exception: the first solution load after a fresh installation runs the analysis once |
| Background recomputation after a solution load | Always runs for the loaded solution and replaces the shown state when ready |
| Principle filter | No chip active (all principles) |
| Severity filter | No chip active (all severities) |
| Selected category | `All` |
| Sort order | Fixed: severity (worst first), then category, then title |
| Cards | Collapsed by default; no limit on the number of cards |
| Affected items per finding | No limit; the list is height-capped at about 260 pixels and scrolls |
| Debt trend | Hidden until at least two snapshots with a debt measurement exist |
| Chips at count 0 | Disabled |
| Free-text search | Not available |

Note: the `Max MD size` budget in the Context Builder header also caps `Send selected lines to LLM`; it is described in "The Context Builder: layout and solution tree".

# 7 Runs and language models

AICB never contacts a language model on its own. Every request belongs to a **run** that you start in the Context Builder: the context document you generated, together with your prompt fields, is sent to the model profile you selected, and the answer is displayed and recorded. This chapter describes the run types, the pre-flight checks, how a prompt is assembled, how model profiles are configured and tested, and how to watch and review a run.

## 7.1 What a run is

A run is one executed LLM request, or a chain of such requests, against the context you assembled. Which kind of chain it is follows from the **run type** stored in the selected run template:

- A **Manual** run sends exactly one request.
- An **Iteration** run sends one request per node of your tree selection.
- A **Preselection** run sends a small selection request first and the main request afterwards.

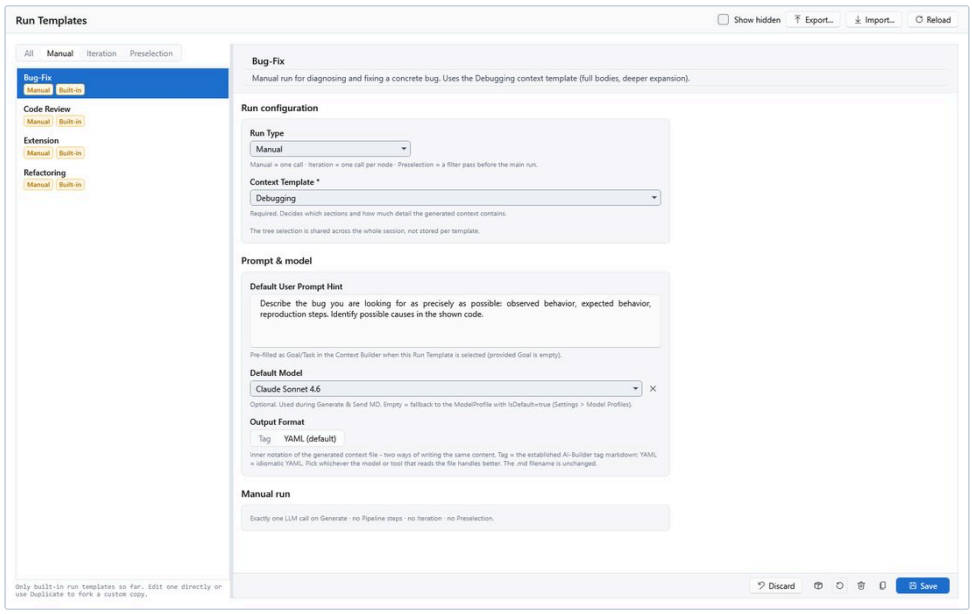
A run is recorded in the database when it has a session and a snapshot to attach to. In the Context Builder this happens automatically: the first send creates a session for you if necessary (see "Sending a context document" below). If no solution is loaded, a Manual run still runs, but it is not persisted.

## 7.2 Run types

The run type is a property of the run template. The Context Builder's run-template picker in the workspace header lists every template, but in the current release only templates of type `Manual` can be selected. Iteration and Preselection runs are implemented end to end but are not released yet; the `Pipeline` type is a placeholder that stored data and repositories already accept, but nothing executes it.

Note: **Iteration** and **Preselection** are **not yet released** and cannot be started from the interface. The description below documents their intended behavior so you know what the configuration fields mean; it is not an invitation to use them. **Pipeline** is **planned**.

| Run type | Requests per run | Status |
|---|---|---|
| `Manual` | one | Released |
| `Iteration` | one per selected tree node | Not yet released |
| `Preselection` | two or three (selection, optional refinement, main run) | Not yet released |
| `Pipeline` | multi-step with optional loops and splits | Planned |




The Run Templates page with a template of type Manual

### Manual

A Manual run is the simplest case: the generated context document plus your prompt fields go to the model in a single call, and the answer comes back as Markdown. Use it for anything that fits into one request - "explain this architecture", "summarize this", "find the bug in this excerpt".

A Manual run has no step boundary at which it could be interrupted cleanly, so **Pause aborts it** (see "Pause, resume and cancel"). It cannot be resumed.

### Iteration (not yet released)

An Iteration run walks your tree selection node by node and sends one request per node; the individual answers are concatenated into one Markdown document. It is meant for uniform work in many places, for example "write an XML doc comment for every method". The advantage over Manual is that every node gets the model's full attention; the price is one request per node.

| Setting | UI label | Values | Meaning |
|---|---|---|---|
| Granularity | `Per method` / `Per class` | `Method` (default), `Class` | One request per method node, or one request per type (all its methods in a single call). |
| On error | `Continue on error` / `Abort on first error` | continue (default), abort | With `Continue on error`, a failed node is recorded and the run continues; with `Abort on first error`, the first failure ends the whole run. |

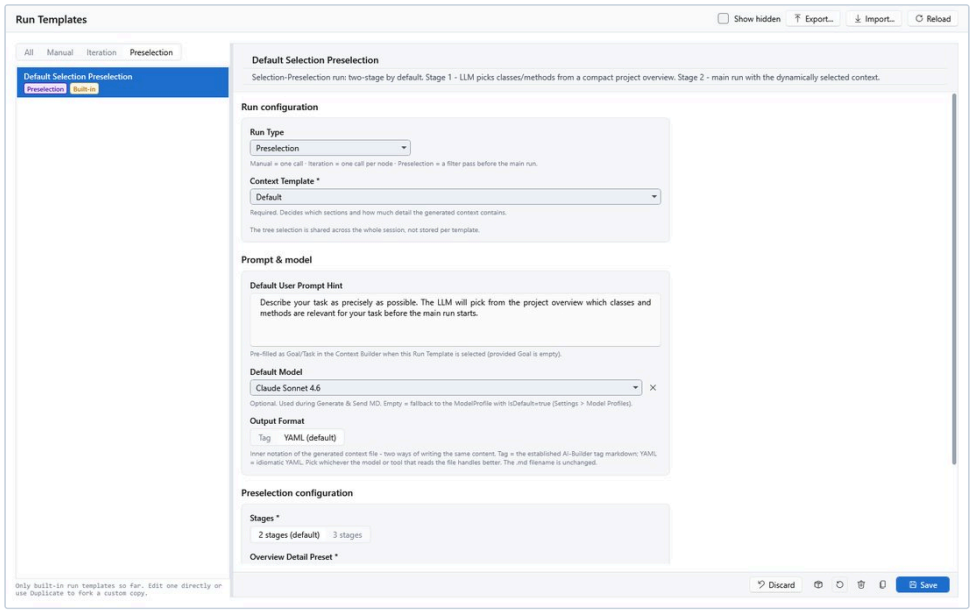
The node list is built from the current tree selection and de-duplicated by node key. The order always follows the solution tree.

For every node the app appends a short block to the request that names the step, the node key and, if known, the containing type, and asks the model to focus on that single node. Each step is stored individually, including its node key and display name, so a failed step is visible in the run history.

The result document contains one third-level heading per node - `Method: <name>` or `Class: <name>` - followed by either the answer or `**Error:** <message>`, separated by a horizontal rule. If the tree selection yields no nodes, the run stops immediately with the message `No nodes to iterate. Check tree selection and granularity.`

A run in which individual nodes failed is recorded with the status `Completed`; only the live progress message distinguishes it with `Completed with N failed node(s).` (there is no separate "completed with errors" status).

### Preselection (not yet released)




A run template of type Preselection in the editor (not yet released)

A Preselection run is a two- or three-stage flow. In stage 1 the model receives a compact overview of the solution and answers with a JSON selection of the classes and methods it considers relevant for your task. The main run then works on the filtered code block only. It is meant for solutions that are too large for the context window, or for models that are distracted by too much unrelated content.

| Setting | UI label | Values | Meaning |
|---|---|---|---|
| Stages | `2 stages (default)` / `3 stages` | `Two` (default), `Three` | Stage 1 selects; stage 2 is the main run. With three stages, stage 2 refines the selection before the main run. |
| Overview Detail Preset | `Overview Detail Preset *` | a detail preset | Detail preset for the compact stage-1 overview. The recommended value is the built-in `Preselection Overview`. |
| Classification Prompt | `Classification Prompt *` | a prompt with a selection-format block | Prompt that tells the model how to return its picks. Only prompts with a non-empty selection-format block are listed. The recommended value is the built-in `Selection Classifier`. |

The three fields marked with `*` are required. If one is missing, the run stops with a plain message, for example `Preselection: OverviewDetailId is required.` or `Preselection: Classification prompt '<id>' not found.`

The app is deliberately lenient when parsing the model's selection: it first tries to parse the raw answer as JSON, then the content of a ```` ```json ```` code block, and finally everything between the first `{` and the last `}`. If all attempts fail, the main run receives the unfiltered context document instead - a broken selection never leaves the main run with an empty context.

Pausing is checked between the stages, not within them. Resuming a preselection run is not supported: it restarts from stage 1 and notes that in the result document.

### Pipeline (planned)

`Pipeline` is reserved for future multi-step runs with optional loops and splits. It is accepted by the stored data, but there is no executor and the run-template editor does not offer it. Nothing in the interface can start it.

## 7.3 Sending a context document

Two commands in the Context Builder send a run. Both are disabled while a request is pending.

| Command | Where | What it does |
|---|---|---|
| `Send to API` | `MD Input` tab, button row | Sends the Markdown currently in the MD Input. |
| `Generate & Send MD` | workspace header, and Ctrl+Enter | Calls `Create MD` first and then sends the freshly generated document. |

In the workspace header you also choose the **run template** (the run type and the context template it uses) and, in the `LLM` list next to it, an optional **model override** for subsequent sends from this tab. The first entry of the `LLM` list is not a profile but the resolved default model; it carries a `Default` badge and keeps the standard resolution chain. An override remains active for every following send until you choose another entry, close the tab or restart the app.

If no override is selected, the model is resolved in this order:

1. the `Default Model` of the selected run template, if set,
2. the model profile marked as default ( `Apply as Active` in the Model Profiles panel),
3. the first profile in the list.

Note: If a run template points to a model profile that no longer exists, the app falls back to the default profile. The `LLM` list shows this in the label of the default entry, so you can see that the template's pin is dangling.

### Pre-flight checks

Before anything is sent, the app runs five checks in a fixed order. Each failure is shown as a status message with the prefix `Send failed:` , and nothing is sent or stored.

| # | Check | Message |
|---|---|---|
| 1 | The MD Input is not empty | `no markdown to send. Run Create MD first.` |
| 2 | A model profile can be resolved | `no model profile available.` or `no model profiles defined. Settings > Model Profiles.` |
| 3 | An API key is set for a profile with `Kind` = `Api` | `API key for model '{Name}' is missing. Settings > Model Profiles.` |
| 4 | An executor exists for the run type | `RunType '{Type}' is not supported. Only Manual, Iteration and Preselection runs can be started.` |
| 5 | For Iteration only: a solution is loaded and at least one node is materialized | `no solution loaded.` or `no Method-nodes selected in the tree. Mark some nodes first.` (with `Class` for the class granularity) |

### The budget confirmation

Before the request is sent, the app estimates the token consumption and, when a trigger fires, asks for confirmation.

There are two independent triggers; either one is enough to show the dialog:

1. **Token threshold.** The estimated total number of tokens (input plus worst-case output) is greater than or equal to the configured threshold. The default is 50,000 tokens; you change it in Settings > General, section `Run Confirmation`, field `Confirm threshold (tokens)` . The value is read fresh on every send, so a change takes effect immediately, without a restart.
2. **Context window overflow.** The input of one single request plus its maximum response would exceed the context window of the model profile. This is a correctness check and applies to all run types, independently of the threshold.

The comparison for the threshold is "greater than or equal". A threshold of **0 therefore means: ask for every run**. There is no value that means "never ask"; to switch the confirmation off in practice, enter a number that no run will reach.

Note: The help text below the field currently says that `0` sends without asking. In practice `0` asks on every run. Use a very high value if you do not want to be asked.

The dialog has the title `Pre-flight budget check` and offers OK and Cancel. Its content looks like this:

```
Iteration over <N> nodes will send approximately:
  Input:  ~<N> tokens
  Output: ~<N> tokens (worst case)
  Total:  ~<N> tokens
Estimated cost: $<x.xx> USD

Model: <model name>

Note: Walker will stop at <N> nodes (ExpansionStrategy.MaxNodesTotal cap). Actual cost may be lower.

WARNING - Context window: one request (~<N> input + <N> response, ~<N> tokens)
exceeds the model's context window (<N>). The request may be rejected or truncated -
lower the token budget or pick a model with a larger window.

Continue?
```

- The **cost line** appears only when at least one of the two price fields of the model profile is filled in.
- The **cap note** appears only when the walker is expected to stop early because of the `ExpansionStrategy.MaxNodesTotal` limit of the active expansion strategy.
- The **context-window warning** appears only when the overflow trigger fired.

If you cancel the dialog, the send path ends silently: no request is sent, no status message appears, and no run is recorded.

In the current release, only Manual runs can be started. A Manual run has no cost pre-flight - it is the fast single shot - so the threshold does not come into play today. What can still appear for a Manual send is the context-window warning, and only when the selected model profile has a context window set: a single send with a large document can exceed the window of a small model.

### How the estimate is calculated

The estimate is produced per node (for a Preselection run, per stage, counted as one node-equivalent per stage):

```
input  = tokens(System prompt) + tokens(Context document) + tokens(User prompt) + 50
output = Max tokens (worst case: the model uses its maximum output length)
```

The fixed 50 tokens cover the step instruction and the node key that the app appends for an iteration step. Tokens are counted with a tokenizer chosen by the model name (see "Token counting and prices"); the numbers are an estimate, not a bill.

The cost estimate is calculated only when prices are configured:

```
cost = (total input tokens / 1,000,000) × input price
     + (total output tokens / 1,000,000) × output price
```

A missing price field counts as 0. If neither field is set, no cost is shown at all.

The context window comes from the `Context window` field of the model profile. If it is empty, no overflow check takes place.

### Pause, resume and cancel

While a request is pending, the `MD Input` button row shows `Cancel` and `Pause`; after a paused iterative run it also shows `Resume`.

| Command | Behavior |
|---|---|
| `Cancel` | Aborts the running request. The server may still finish computing, but the response is discarded. The run is recorded as `Cancelled`. |
| `Pause` | Pauses an iterative run after the current step. The run is stored as `Paused` and can be resumed. |

| Command | Behavior |
|---|---|
| `Resume` | Continues a paused iterative run from the next step. |

Notes:

- `Pause` applies to iterative runs only. A Manual run has no step boundary, so pressing `Pause` while a Manual run is pending aborts it and records it as cancelled; it cannot be resumed.
- `Resume` appears only for a run that was paused in the current app session. It is not offered for a run that was still paused when you closed the app.
- Resuming rebuilds the request from the current interface state: the system prompt, context document and user prompt are taken from your current fields, the tree selection and detail overrides from the current tree, and the iteration nodes are materialized again. Only the model profile and run template come from the stored run.
- The result document of a resumed run is rebuilt from the already-stored answers, so the earlier steps are still visible. The next step number is derived from the highest stored step.

#### While a run is active

- The `Active Run` tab shows the run name and model, a `Running` pill, a step counter with progress bar, the `OK` and `Failed` counters, the accumulated input/output tokens, and the step list. Selecting a step shows the text the model has streamed so far.
- The `Send to API` command is disabled; `Cancel` and `Pause` are visible.
- The app registers the run as an active activity. While it is active, the database cannot be switched: the action is refused with `Cannot {action} while {N} background task(s) are active.`
- On completion, the status line reports the outcome, for example `Done. <model> · 1/1 nodes · in 1234 / out 567 tokens`.

#### If something goes wrong

The app turns provider errors into a structured result instead of an exception, and always writes an end state for the run:

| Situation | What you see |
|---|---|
| You cancel | The run is recorded as `Cancelled`; the response text is discarded. |
| HTTP error from the provider | A message with the provider name and the status code, for example `OpenAI HTTP 401: ...`, `Anthropic HTTP 429: ...` or `Local HTTP 503: ...`. |
| Network error | `<Provider> network error: <details>` |
| Client timeout | `<Provider> timeout: HttpClient default timeout exceeded.` |
| Error in the middle of a stream | The text received so far is kept; the run is reported as failed with the partial content. |
| Failure in one step of an iterative run | With `Continue on error`, the step is marked failed and the run continues; with `Abort on first error`, the run ends. |

Rate limits and transient server errors are retried automatically (see "Automatic retries" below). If the app was closed while a run was still active, the next start offers a dialog `Unfinished runs from previous session`: `Yes` marks them as cancelled, `No` marks them as paused, `Cancel` leaves them unchanged.

### 7.4 What a prompt contains

Internally a request is assembled from four blocks:

| Block | Comes from |
|---|---|
| System prompt | the `System Role` and `Instructions` fields on the Prompt tab, plus an optional hint about the active layer profile |
| Code block | the context document in the `MD Input` tab |

| Block | Comes from |
|---|---|
| User prompt | the `Goal/Task`, `Constraints` and `Additional Context` fields on the Prompt tab |
| Response format | the step instruction appended by an iterative run; empty for a Manual run |

The provider receives two strings:

```
system prompt = System prompt
user message  = Code block + "\n\n" + User prompt + "\n\n" + Response format
```

Each part is included only when it is not empty. Constraints and additional context are preceded by the headers `Constraints:` and `Additional context:` respectively.

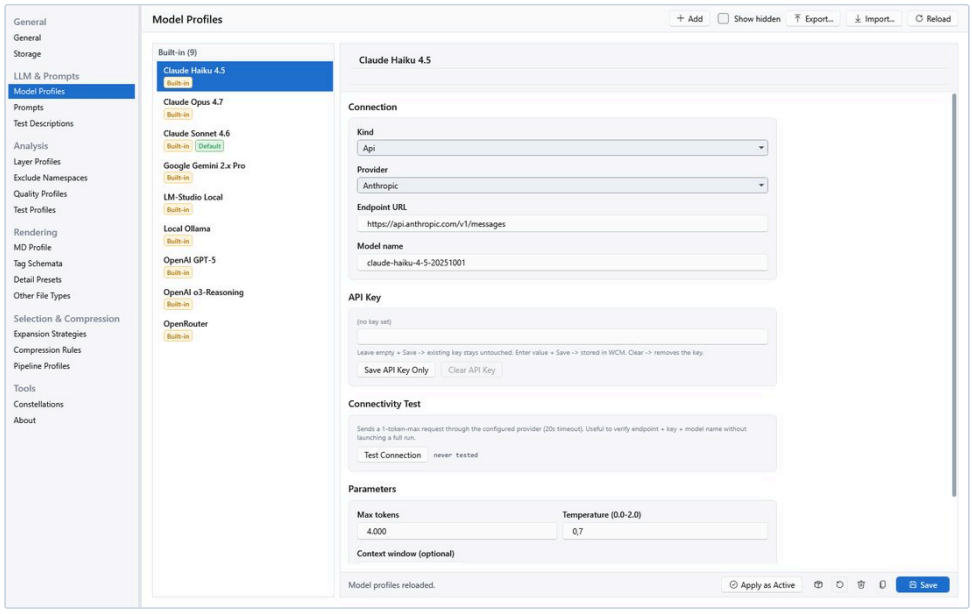
Before sending, every block is normalized: line endings are converted to LF, trailing spaces and tabs at the end of a line are removed, and trailing blank lines at the end of the block are removed. Blank lines inside the text are preserved. This keeps the system-prompt and code-block prefix byte-identical across calls, which is what provider-side prompt caching relies on.

If the active layer profile carries namespace rules, the app appends a sentence to the system prompt naming the profile and asking the model to flag cross-layer dependency violations as `**Critical:**` (strict policy) or `**Warning:**` (advisory policy). Profiles without rules produce no such hint.

## 7.5 Model profiles

A model profile describes one LLM endpoint: where it is, which model to call, how to authenticate, and the generation limits. Model profiles are master data; run templates point to them, and the `LLM` list in the Context Builder can override the choice for all subsequent sends from that tab until the selection changes or the tab closes.

Open Settings > `Model Profiles`. The panel is a master-detail editor: the list on the left, the editor on the right.




Model Profiles with connection details and the Test Connection button

The page header offers `Add` (create a custom profile), `Show hidden` (also list built-ins that were deleted), `Export...`, `Import...` and `Reload` (F5). The action bar at the bottom offers `Apply as Active`, `Export as Built-In`, `Restore Built-In`, `Delete`, `Duplicate` and `Save`.

- `Apply as Active` marks the profile as the default model - the one used when a run template does not pin a profile.
- `Duplicate` creates a copy named `<name> (Copy)`. The API key is deliberately not copied; enter it again in the copy.

- `Delete` removes a custom profile permanently. A built-in is only hidden and can be brought back with `Show hidden` and `Restore Built-In`.
- `Save` persists your edits. Editing a built-in marks it as `Overridden`; `Restore Built-In` resets it to its shipped values.

### Profile fields

| Field | Meaning | Default / range |
|---|---|---|
| Name | Display name, shown in the pickers. | - |
| Description | Optional short text shown under the name. | empty |
| `Kind` | `Api` for a remote cloud endpoint, `Local` for a local OpenAI-compatible server. Decides which fields are required and how the connection test runs. | `Api` |
| `Provider` | `Anthropic`, `OpenAI` or `Custom`. Selects the client implementation (request format, authentication header, streaming protocol). | `Anthropic` |
| `Endpoint URL` | HTTP endpoint of the API. | empty; for `Local` / `Custom` it is required |
| `Model name` | Provider-specific model identifier sent in the request. | empty; required |
| API key | Secret sent as `x-api-key` or `Bearer` header. Stored in the Windows Credential Manager. | empty |
| `Max tokens` | Maximum number of output tokens per request. Whole numbers only; minimum 1. Provider hard limits apply. | 8000 |
| `Temperature (0.0-2.0)` | Sampling temperature, up to two decimals. `0.0` = deterministic, `1.0` = balanced, `2.0` = creative. | 0.7; values are clamped to 0.0-2.0 |
| `Context window (optional)` | Total window (input plus output) the model can handle. Used by the pre-flight check. Whole numbers only. Leave empty to skip the check. | empty |
| `Pricing per 1M tokens (optional)` - `Input` / `Output` | Your price in USD per one million tokens, up to two decimals. Used for the cost estimate and the recorded cost. | empty |

For a profile with `Kind` = `Local`, a context window is expected, because it is the value the pre-flight check uses to protect a small local model from an oversized request.

Note: The `Provider` field has no effect on a `Local` profile. Every profile with `Kind` = `Local` is sent through the OpenAI-compatible client, whatever `Provider` says. The shipped local profiles show the default value `Anthropic` there; you can leave it as it is.

### Built-in profiles

Nine profiles are shipped. `Claude Sonnet 4.6` is the default.

| Profile | Kind | Provider | Endpoint | Model name | Context window | Max tokens | Temp. |
|---|---|---|---|---|---|---|---|
| Claude Opus 4.7 | Api | Anthropic | `https://api.anthropic.com/v1/messages` | `claude-opus-4-7` | 1,000,000 | 8000 | 0.7 |
| Claude Sonnet 4.6 (default) | Api | Anthropic | `https://api.anthropic.com/v1/messages` | `claude-sonnet-4-6` | 200,000 | 8000 | 0.7 |
| Claude Haiku 4.5 | Api | Anthropic | `https://api.anthropic.com/v1/messages` | `claude-haiku-4-5-20251001` | 200,000 | 4000 | 0.7 |

| Profile | Kind | Provider | Endpoint | Model name | Context window | Max tokens | Temp. |
|---|---|---|---|---|---|---|---|
| OpenAI GPT-5 | Api | OpenAI | `https://api.openai.com/v1/chat/completions` | `gpt-5` | 400,000 | 8000 | 0.7 |
| OpenAI o3-Reasoning | Api | OpenAI | `https://api.openai.com/v1/chat/completions` | `o3` | 200,000 | 8000 | 0.7 |
| Google Gemini 2.x Pro | Api | Custom | `https://generativelanguage.googleapis.com/v1beta` | `gemini-2.0-pro` | 1,000,000 | 8000 | 0.7 |
| Local Ollama | Local | - | `http://localhost:11434/v1` | empty | 32,768 | 4000 | 0.2 |
| LM-Studio Local | Local | - | `http://localhost:1234/v1` | empty | 32,768 | 4000 | 0.2 |
| OpenRouter | Api | OpenAI | `https://openrouter.ai/api/v1` | empty | 200,000 | 8000 | 0.7 |

Four of these are **templates** that you adapt rather than ready-to-run entries:

- `Local Ollama` and `LM-Studio Local`: enter the name of the model your local server has loaded, for example `qwen2.5-coder-32b`. Without a model name the call is refused with `Local: ModelName is required.`
- `Google Gemini 2.x Pro`: set the endpoint and model name to match your Google API access. The profile uses the generic OpenAI-compatible client, so the endpoint must be the OpenAI-compatible base path of your provider.
- `OpenRouter`: enter the model slug, for example `anthropic/claude-opus-4.7` or `openai/gpt-5`.

Note: No built-in profile carries prices. The cost line in the pre-flight dialog and the recorded cost per request therefore stay empty until you enter your own prices. Prices change faster than releases, so the app does not ship with them.

### Supported providers

Three clients sit behind the provider selection:

| `Provider` | Client | Default endpoint | Authentication |
|---|---|---|---|
| `Anthropic` | Anthropic Messages API | `https://api.anthropic.com/v1/messages` | header `x-api-key` plus `anthropic-version: 2023-06-01` |
| `OpenAI` | OpenAI Chat Completions API | `https://api.openai.com/v1/chat/completions` | header `Authorization: Bearer ...` |
| `Custom` | generic OpenAI-compatible client | none - the endpoint is required | `Bearer` only when a key is set |

The `Custom` client speaks the OpenAI chat-completions protocol and serves local or self-hosted servers such as Ollama, llama.cpp's llama-server, LM Studio, vLLM, KoboldCpp and text-generation-webui in OpenAI mode. Cloud providers that expose an OpenAI-compatible API (OpenRouter, Together, Groq, Mistral and others) are configured as `Provider` = `OpenAI` with their own endpoint.

The endpoint is normalized before the call:

- a bare host such as `http://localhost:11434` gets `/v1/chat/completions` appended,
- a versioned base such as `/v1`, `/v2` or `/v1beta` gets `/chat/completions` appended (the version segment is lowercased),
- a full path ending in `/chat/completions` is used unchanged.

`Anthropic` and `OpenAI` use their default endpoint when the `Endpoint URL` field is empty. For `Custom` there is no default: an empty endpoint is refused with `Local: Endpoint is required (e.g. http://localhost:11434/v1).`

### API keys and the Windows Credential Manager

API keys are stored in the **Windows Credential Manager**, not in the database. Each profile has its own entry, named `AIContextBuilder.ApiKey:<profile id>`.

- The key is never written to the database and never included in JSON exports of the configuration.
- The key is never shown again after you save it. When you switch profiles, the input field is cleared and the status line above it tells you only whether a key is stored: `✓ Key saved in Windows Credential Manager.` or `(no key set)`.
- Leaving the field empty and saving keeps the existing key untouched. Entering a value and saving stores it. `Clear API Key` removes the stored key; the profile itself stays.
- For profiles with `Kind` = `Api`, `Save API Key Only` stores just the key without saving your other editor changes.
- The app does not encrypt the key itself; it hands it to the Windows Credential Manager, which stores it protected by the operating system and bound to your Windows user account.
- Only the desktop app reads and writes this store. Headless hosts (the command-line and the MCP server) have no access to it, so a configuration imported there contains no key.

You can also manage the entry in the Windows Credential Manager yourself (Control Panel > User Accounts > Credential Manager > Windows Credentials); the app reads the entry with the name above.

The key is sent only over HTTPS, or over HTTP to a loopback address (localhost). All three clients refuse to send a key over plain HTTP to a non-local host with a message such as `Anthropic: API key not sent over plain HTTP to a non-localhost host. Use an HTTPS endpoint.` Plain HTTP to `localhost` remains allowed, which is what local Ollama and LM Studio installations need.

Apart from calls to an LLM endpoint you configured, the app does not send anything over the network: there is no telemetry upload, no update check and no crash reporting. The configured endpoint may be Anthropic, OpenAI, Google Gemini, OpenRouter, a custom OpenAI-compatible service, or a local Ollama/LM Studio address. A profile template contains no credential, and nothing contacts its endpoint until a run, connection test or armed first-load layer/exclusion proposal uses it.

### Testing a connection

The `Connectivity Test` section of the editor has a `Test Connection` button. It sends a real request to the endpoint, with the credentials and the model name currently in the editor - including edits you have not saved yet. That lets you probe a new endpoint and key combination before saving the profile.

- The probe uses a fixed system prompt, the user text `ping`, a maximum of 16 output tokens and temperature 0.0.
- It has a 20-second time limit.
- On success the status line shows `Connection OK.` and the result is stored with the profile, so it survives a restart. Next to the button you see the last result with its local date and time, for example `2026-09-21 14:03 - ok: Connection OK.`
- The button never throws: you always get a result. Its messages are `Connection OK.`, `Timeout after 20s.`, `Cancelled.`, `Provider returned no response.`, or the provider's own error message. A failure is shown as `Connection failed: <message>`.

The test result is reset to `not yet tested for this Kind` when you change `Kind`, because a previous test ran against a different client path.

### Automatic retries

Rate limits and transient failures are retried automatically with an exponential backoff: 1 second, then 2, 4 and so on, capped by the maximum backoff. Retried are the HTTP statuses 429, 500, 502, 503 and 504, and network errors. If the provider sends a `Retry-After` header, its value is used instead (also capped by the maximum backoff). A user cancel is never retried, and an abort in the middle of a stream is not repeated either. An unknown provider is reported immediately as a configuration error instead of being retried.

The three parameters are configured in Settings > General, section `LLM Retry`:

| Field | Meaning | Default |
|---|---|---|
| `Max attempts` | Total attempts per call; `1` means no retry. | 3 (one initial attempt plus two retries) |

| Field | Meaning | Default |
|---|---|---|
| `Base backoff (seconds)` | First retry delay; doubled for each further attempt. | 1 |
| `Max backoff (seconds)` | Upper bound for the backoff, including a provider `Retry-After` value. | 30 |

Note: Changes to these three values take effect after an app restart, because the retry policy is resolved once at startup.

## 7.6 Token counting and prices

Token counts are produced locally with Tiktoken encodings, without any network or file access at runtime:

- `o200k_base` is used for `gpt-4o` (and its variants) and for the OpenAI reasoning family ( `o1` , `o3` , `o4` ).
- `cl100k_base` is used for everything else, including Claude models and local models.

A vendor prefix in the model name is ignored, so `openai/o3-mini` is treated like `o3-mini` . Model families that the tokenizer does not know - `gpt-4.1` , `gpt-4.5` and `gpt-5` - fall back to `cl100k_base` ; their counts can therefore deviate from the provider's own count by a few percent.

Prices are entered per **one million tokens**, separately for input and output. They are used in two places:

| When | Where | Result |
|---|---|---|
| Before the run | the pre-flight dialog | an estimated cost, with the output counted at the maximum ( `Max tokens` ) |
| After the call | the stored run history | the `cost_usd` of each recorded request, computed from the token counts the provider reported |

The numbers you see before a run are therefore your own prices multiplied by locally counted tokens, while the cost recorded afterwards uses the provider's token counts. If no price is configured, the recorded cost is unknown rather than zero, and the cost line is omitted.

## 7.7 Watching and reviewing a run

### Active Run

The `Active Run` tab is the live view. Its header shows the run name, the model and the run type, plus a `Running` pill or a `Complete` pill. The progress row shows `Step <n> / <total>` , a progress bar, the counters `OK <n>` and `Failed <n>` and the accumulated tokens ( `<in> in / <out> out` ). The step list shows every step with its state ( `Pending` , `Running` , `Completed` , `Failed` , `Cancelled` , `Skipped` ), its number and its input/output tokens. Selecting a step opens the live output pane below, which shows the text that step has streamed.

A Manual run appears as a single step. Iterative runs report one step per node. In the current Preselection implementation, the step list also contains one synthetic step; its phases are reflected only by the `Step n / m` progress counter.

### LLM Response

The `LLM Response` tab contains the answer of the last send. For a Manual run the app switches to this tab as soon as the first streaming chunk arrives, so you can watch the answer being written. For other run types the aggregated document of all steps lands here.

- `Copy to Clipboard` (also Ctrl+Shift+C) copies the answer.
- `Use as MD input` copies the answer into the MD Input tab, so a follow-up run can work on the model's own answer. The button is disabled while there is no answer.

### Reasoning

The `Reasoning` tab shows the last run as branches: one branch per request, listed on the left. A Manual run has a single branch named `Manual run`; an iterative run names its branches after the node keys. Preselection branch labels follow the step number rather than a fixed semantic role: the standard two-phase run produces `Stage 1 - Selection` and `Stage 2 - Refinement`; a three-phase run additionally produces `Stage 3 - Main run`.

For the selected branch you get these sub-tabs:

| Sub-tab | Content |
|---|---|
| `Output` | The raw answer of that request. |
| `Thinking` | Thinking blocks extracted from the answer. |
| `Tool Calls` | Tool-use blocks extracted from the answer. |
| `Findings` | Findings extracted from the answer, filterable by `Critical`, `Warning`, `OK` and `Info`. |
| `Evaluation` | The evaluation block of the run, if the answer contained one. |
| `Changes` | Files for which the answer proposed a new state, diffed against the files on disk. |

A dot in a sub-tab header means that this sub-tab has content.

After every run (and after a resume) the app parses the stored answers and fills these views. The parser looks for text markers in the answer:

| What | Marker |
|---|---|
| Thinking block | `<thinking>...</thinking>` or `<reasoning>...</reasoning>` |
| Tool use | `<tool_use name="...">...</tool_use>` |
| Findings | a line starting with `**Critical:**`, `**Warning:**`, `**OK:**` or `**Info:**`; the content runs to the next such line |
| Evaluation | `**Hypothesis:**`, `**Result:**` (or `**ActualResult:**`), `**Score:**`, `**Quality:**`, `**Notes:**` |

All markers are matched case-insensitively, and a finding may be preceded by a list character (`-` or `*`). An evaluation is only stored when the answer contains a hypothesis or a result; the first such block in a run is used. The `Score` value is shown exactly as the model wrote it - the app does not normalize it or restrict it to a range, so a model that writes `85` is shown as `85`.

Note: These are text conventions that the model must follow because your prompt asks it to. The app reads only the text of the answer; provider-specific reasoning fields that some APIs return separately are not evaluated. If the prompt does not ask for these markers, the `Thinking`, `Tool Calls`, `Findings` and `Evaluation` sub-tabs stay empty, and each of them explains that in its empty state. The markers are not removed from the `Output` sub-tab - the extracted content is shown in addition to the raw text.

The `Changes` sub-tab is a view only: nothing is written to disk, and there is no accept or reject action. A diff appears when the answer states the complete new content of a file and that file is found in the loaded solution. If the answer's block is truncated or does not name a file, the app lists that as a form violation instead of showing a diff that looks complete.

### Run history

Two places show what has already run:

- **Sessions tab** - select a session and open its `Runs` sub-tab. It lists every run of that session with its number (`#1`, `#2`, ...), its name, a run-type pill and a status pill. A session without runs shows the hint that runs appear here when they finish.
- **Context Builder** - the `Recent runs:` row in the header shows up to three chips for the current session with the run number, name, run type, status and time.

The possible run states are `Running`, `Completed`, `Failed`, `Cancelled` and `Paused`. A run whose individual steps failed is stored as `Completed`.

**What a run stores**

When a run is persisted, the database receives the run itself (type, name, number, status, start and end time), a snapshot of the tree selection and detail overrides at send time, and one record per request with:

- the complete answer of the model,
- input and output tokens,
- the stop reason and the model name that was used,
- the error message, if any,
- the latency in milliseconds,
- the cost, when prices are configured,
- for iterative runs, the node key and display name of the step.

No API key is stored. The first line of your user prompt (the goal) is stored with the run snapshot. Because the complete model answers are kept, the database file is a document containing model output - keep that in mind when you share or back up the file.

## 7.8 Step by step: connect your first model and send a context document

1. Open **Settings > Model Profiles**. You see the nine built-in profiles; `Claude Sonnet 4.6` is marked as the default.
2. Select the profile of the provider you have an account with, for example `Claude Sonnet 4.6` or `OpenAI GPT-5`. To use a local server, select `Local Ollama` or `LM-Studio Local`.
3. Check the `Connection` section: the `Endpoint URL` and the `Model name` are pre-filled for the cloud profiles. For a local profile, enter the model name your server has loaded, for example `qwen2.5-coder-32b`.
4. Enter your API key in the `API Key` section. The field is empty even if a key is already stored; a status line above it tells you whether one is present.
5. Click `Test Connection` and wait for the result next to the button. `Connection OK.` means endpoint, key and model name work together. If the test fails, the message names the reason; correct the endpoint, the model name or the key and test again.
6. Click `Save`. If this profile should be used whenever a run template does not pin a model, click `Apply as Active` as well.
7. Switch to the **Context Builder**. Open your solution if it is not already loaded, and mark the nodes you want to send in the Solution Tree.
8. In the workspace header, choose the run template. In the current release this must be a template of type `Manual`. In the `LLM` list next to it, leave the default entry selected, or pick the profile you just configured for this send.
9. Fill in the Prompt tab: at least `Goal/Task`; `System Role`, `Instructions`, `Constraints` and `Additional Context` are optional.
10. Click `Create MD` (Ctrl+M) to generate the context document, and review it in the `MD Input` tab. You can edit the text before sending.
11. Click `Send to API` (or `Generate & Send MD`, Ctrl+Enter, to generate and send in one step). If the model profile has a context window and the request would exceed it, confirm the pre-flight dialog.
12. The app switches to the `LLM Response` tab and streams the answer. If a run had to be created first, the status line tells you that an `Untitled` session was created; rename it later in the Sessions tab if you want.
13. Review the answer, then use `Copy to Clipboard` or `Use as MD input` to continue working with it.
14. Open the `Reasoning` tab to see the request as a branch with its output, findings and any evaluation.
15. Find the recorded run afterwards in the Sessions tab: select the session and open its `Runs` sub-tab.

# 8 Settings

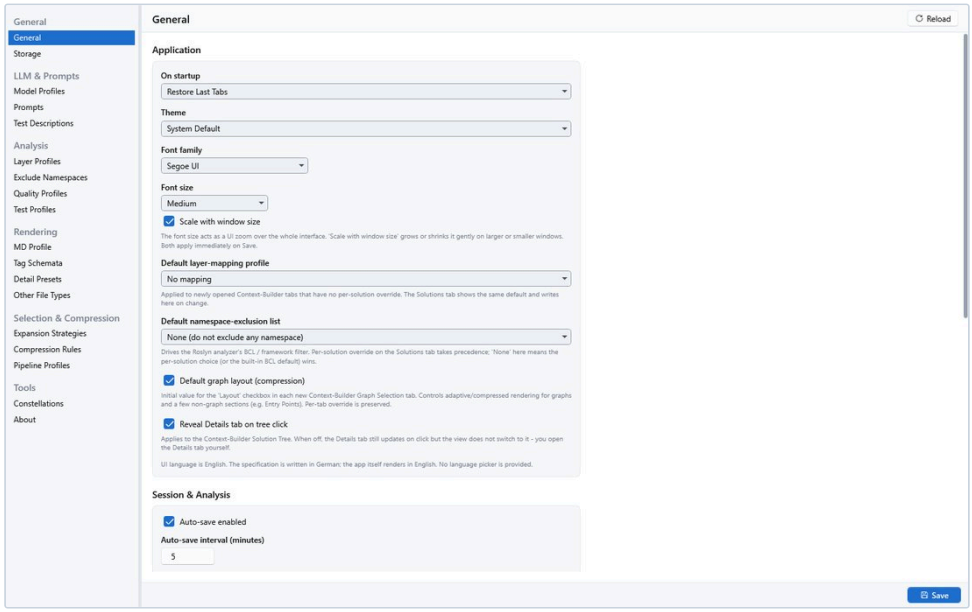
The Settings tab is the central place for everything that applies beyond one solution: appearance, storage, LLM connections, the reusable master data (prompts, profiles, schemata, templates), and the export bundles. Most pages share the same master/detail layout, so once you have used one of them you can use all of them.

A few things are worth knowing before you start:

- **Nothing has to be configured to use the core workflow.** Every axis ships with a built-in default that is already active: the context template, the compression rules, the pipeline profile, the quality profile, the MCP profile and the test profile all point at their built-in default entry. You can open a solution and create a context document without touching a single setting.
- **The one thing the LLM workflow needs is a model profile with an API key.** If it is missing, the app tells you exactly where to add it: `API key for model '{Name}' is missing. Settings > Model Profiles.`
- **The app's user interface is English.** A language picker is not provided; the General page states this explicitly.

## 8.1 Where the settings pages live

Open **Configuration** → **Settings** in the sidebar. The tab has its own sub-sidebar on the left (200 pixels wide, adjustable by dragging the splitter, minimum 160 pixels) and the content of the selected page on the right. The sub-sidebar is organized into six groups; the group captions are headings, not clickable entries.




Settings with its sub-navigation on the left, here the General page

| Group | Page | Sidebar tooltip |
|---|---|---|
| **General** | `General` | "App-wide preferences (theme, layer-profile, graph-layout, auto-save)." |
| | `Storage` | "Database file location, backup settings, app-settings JSON path." |
| **LLM & Prompts** | `Model Profiles` | "LLM-Provider configurations (Anthropic / OpenAI / LocalLLM / Mock)." |
| | `Prompts` | "Reusable user-prompt presets." |
| | `Test Descriptions` | "Reusable acceptance-test descriptions for RunTemplates." |
| **Analysis** | `Layer Profiles` | "Namespace-pattern to Architecture-Layer mappings." |
| | `Exclude Namespaces` | "Glob patterns to skip in analysis (e.g. System.*)." |

| Group | Page | Sidebar tooltip |
|---|---|---|
| | `Quality Profiles` | "Code-quality producer toggles, metric thresholds (LOC + cyclomatic complexity) + debt/gate behaviour." |
| | `Test Profiles` | "Pin per-solution test detection: which projects are test projects + which method attributes mark a test case. Default replicates the built-in heuristic." |
| **Rendering** | `MD Profile` | "Markdown render-profiles (per Section-Slot)." |
| | `Tag Schemata` | "Per-tag rendering rules (Class / Method / Interface / Enum)." |
| | `Detail Presets` | "Per-section Detail-Level mappings (Compact/Normal/Detailed/Source)." |
| | `Other File Types` | "Read-only reference: recognized file types + the 4 detail stages. The compression stage is chosen per node in the Solution Tree." |
| **Selection & Compression** | `Expansion Strategies` | "Tree-walk strategies (Method-Depth + Type-Depth + Flags)." |
| | `Compression Rules` | "Auto-Compression heuristics (Caller-Count + Force-Flags)." |
| | `Pipeline Profiles` | "Token-Budget Trimming-Pipeline configurations." |
| **Tools** | `Constellations` | "Bundle Master-Entities + AppSettings as portable JSON." |
| | `About` | "App version, components, license, support." |

Behavior of the Settings tab:

- **The last page you used is remembered.** The next time you open Settings, that page is selected again. If the name of the page no longer exists, the selection quietly falls back to the first entry ( `General` ).
- **Four keyboard shortcuts work on every page.** They are routed to the page that is currently active: `Ctrl+S` = Save, `Ctrl+N` = new entry, `Ctrl+D` = duplicate, `F5` = reload. On a page that has no such action (the two form pages `General` and `Storage` have no Add or Duplicate), the key does nothing - that is intended.
- **Tooltips can be switched off globally.** The switches are on the `General` page in the `UX` section and take effect immediately, in the Settings tab and its sub-pages.
- The group captions cannot be clicked or focused; only the 18 page entries can.

Three further master-data pages are **not** part of the Settings tab. They are separate entries in the sidebar:

| Path | Tab | What it is |
|---|---|---|
| **Configuration → Templates** | `Templates` | The context templates: prompt, MD profile, detail presets per level, export content and profile overrides. |
| **MCP → MCP Profiles** | `MCP Profiles` | The MCP working environments: tool set, instructions, output notation and token budget. |
| **MCP → MCP Usage** | `MCP Usage` | The recorded MCP tool-call telemetry, with notes, export/import and a controlled clear. |

`MCP Usage` reloads its data every time the tab is activated, because the MCP server keeps writing into the same database while the app is running. Without that, a re-opened tab would still show the numbers from the first time you opened it.

## 8.2 Where a setting is stored and when it takes effect

Settings are kept in three places, and the split explains which ones can travel with a bundle and which cannot:

| What | Where |
|---|---|
| Storage settings (base path, database path, backup) and the recent-solution/recent-database lists | `%APPDATA%\AIContextBuilder\app-settings.json` |

| What | Where |
|---|---|
| Everything else from the general settings, all active/default profile ids, and all master data (prompts, profiles, schemata, templates, MCP profiles, ...) | The SQLite database |
| API keys of model profiles | Windows Credential Manager |

Note: the `app-settings.json` file always lives at the default location `%APPDATA%\AIContextBuilder\app-settings.json`. Changing `Base path` on the Storage page does **not** move it - the Storage page therefore shows you the file's actual location in a read-only field. This is deliberate: the file must be readable before the database is open, so it cannot depend on a path that is stored inside it.

When a change takes effect:

| Effect | Examples |
|---|---|
| **Immediately on Save** | Font family, font size (UI zoom), theme for the standard controls, tooltips switch and delay |
| **On the next start of the app** | `Base path`, the LLM retry values, the theme in individual custom panels |
| **On the next start of the MCP server** | MCP tool set, MCP instructions, automatic re-analysis mode |
| **Live on every MCP call** | The context-template assignment of an MCP profile |
| **The next time a tab is opened** | Default layer profile, default namespace-exclusion list, default graph layout |

Note: the app and the standalone MCP server use the same database. A schema migration therefore applies to all running instances at the next start of each process - the desktop app and any MCP host included.

## 8.3 The master/detail pattern

Most settings pages are built from the same pattern: a list of entries on the left (the master), an editor for the selected entry on the right (the detail), and a fixed action bar at the bottom. The actions and their meaning are identical on every such page; only which of them are enabled and which badge a page uses differ.

### Page layout

From top to bottom, every master/detail page has the same three parts:

1. **The page header** - the page title on the left, the page-wide actions on the right.
2. **The two-column body** - the entry list on the left (280 pixels), the editor on the right.
3. **The action bar** - the entry-related actions on the right, the status line on the left.

The status line is the main feedback channel of these pages. Validation errors appear there, not in a dialog.

### Header actions

| Control | Label | Tooltip | Effect |
|---|---|---|---|
| Button | `Add` | Varies per page | Creates a new custom entry, saves it immediately and selects it. Status line: `New {Kind} created.` |
| CheckBox | `Show hidden` | "Show built-ins that were soft-deleted via Delete." | Shows built-ins that were soft-deleted with `Delete`. Default: off. Toggling reloads the list immediately. |
| Button | `Export...` | "Export master entities to JSON file" | Writes **all** entries of the list (built-ins *and* your own) to a JSON file. The suggested file name is `{EntityTypeIdentifier}-export-{yyyyMMdd-HHmm}.json`; characters that are not allowed in file names are replaced by `-`. |
| Button | `Import...` | "Import master entities from JSON file" | Reads a JSON file back in (see "Exporting and importing master data as JSON"). |

| Control | Label | Tooltip | Effect |
|---|---|---|---|
| Button | `Reload` | "Discard unsaved edits and reload all entries from the database. (F5)" | Discards unsaved edits. The selection is kept as long as the entry still exists. |

### Entry actions

| Control | Label | Tooltip | Effect |
|---|---|---|---|
| Button | `Save` | "Save: persist editor changes to the database. Editing a built-in sets the OVERRIDDEN flag. (Ctrl+S)" | Writes the editor values to the database (see "Saving an entry"). |
| Icon button | (Duplicate) | "Duplicate: create a custom copy of the selected entry as a starting point for tweaks. (Ctrl+D)" | Creates a copy with a new ID and the name suffix `(Copy)`. The copy is never a built-in, and it never inherits the active/default badge. |
| Icon button | (Delete) | "Delete: built-ins are soft-hidden (recoverable via Restore); custom entries are removed permanently." | See "Deleting an entry". |
| Icon button | (Restore) | "Restore Built-In: reset this built-in to its code defaults and clear the override/hidden flag." | See "Restoring a built-in". |
| Icon button | (Export as Built-In) | "Export as Built-In: show the C# snippet for this entity to paste into the BuiltIn*.cs source file - promotes the custom value to a permanent Built-In after build + restart." | See "Exporting an entry as built-in code". |
| Button | `Apply as Active` | Varies per page | Makes the selected entry the globally active (or default) one. Only visible on pages that have such a concept. Status line: `'{Name}' is now the active {Kind}.` or `'{Name}' is now the default {Kind}.` |

### The list and its badges

The list groups its entries by origin: built-ins first, then your own entries, each group sorted alphabetically by name. The group headers read `Built-in ({count})` and `Custom ({count})`. Long names wrap instead of being cut off - the list does not scroll horizontally.

A row can carry several badges:

| Badge | Meaning |
|---|---|
| `Built-in` | The entry ships with the app. |
| `Active` | This is the globally active entry of the page. |
| `Default` | This is the default entry of the page. |
| `Overridden` | A built-in that you saved and thereby overrode. |
| `Hidden` | A soft-deleted built-in (only visible when `Show hidden` is on). |
| `IN USE · {n}` | A reference counter: how often a custom entry is used elsewhere. |

Two pages deliberately build their own list instead: `Tag Schemata` (it has a filter box above the list and two-level group headers) and `Templates` (no grouping, its own hint block).

### Saving an entry

Saving always follows the same order, which is why a rejected input never arrives "half way":

1. The entry is fetched from the store.
2. The editor fields are written into it.
3. The entry is validated. If validation fails, the save is aborted, the message appears in the status line, and the rejected input is discarded - it does not stay alive in the background.
4. If the entry is a built-in, it is marked as `Overridden` *before* it is written. The list badge and the stored data stay in sync.
5. Then the write happens. If the store rejects the write (for example an invalid source path in `Tag Schemata`), the status line shows `Save failed: {message}` instead of crashing.
6. The list is reloaded, the selection is restored, and the status line reads `{Kind} '{Name}' saved.`

Built-ins are **not** read-only. The editor is active as soon as an entry is selected; saving a built-in turns it into an `Overridden` built-in, and `Restore` brings the code defaults back. "In use" states (a template that an open session references, a run template while a run is active) are advisory hint banners, not locks.

#### Deleting an entry

The `Delete` button does two different things depending on what is selected:

- **A built-in** is only *hidden* (soft-deleted). It disappears from the list and can be brought back with `Show hidden` plus `Restore`. Status line: `{Kind} '{Name}' hidden (built-in). Use 'Show hidden' + Restore to bring back.`
- **Your own entry** is removed permanently. Status line: `{Kind} '{Name}' deleted.`

Note: this asymmetry is intentional. A built-in that was deleted for real would simply be re-created from the code seed on the next read, so the delete would look as if it did nothing.

#### Restoring a built-in

`Restore Built-In` is only available when the selected entry is a built-in and carries the `Overridden` or `Hidden` badge. It clears the override flags and writes the code defaults back. Status line: `{Kind} '{Name}' restored to built-in defaults.`

#### Exporting an entry as built-in code

`Export as Built-In` does **not** change anything by itself. It opens a dialog titled `Export as Built-In Code` with the generated C# snippet for the selected entry. The dialog contains: Note: this function is intended for the product's own development; the snippet only has an effect in AICB's source code, not in your installation.

- a banner: "The exported code snippet must be manually inserted into the BuiltIn*.cs source file and committed to git. The application does not modify source files automatically.",
- an extra banner when the entry is a built-in that you already overrode: exporting creates a new candidate built-in, and merging it into the original source file is your responsibility,
- an extra banner for layer profiles and exclusion lists: review the snippet before committing, because it may contain solution-specific paths or namespaces,
- the read-only snippet, the target file name and an insertion hint,
- the buttons `Close` and `Copy to Clipboard`.

The workflow is deliberately manual: copy the snippet, paste it into the named source file, commit, and rebuild. After the rebuild and a restart, the entry is a real built-in and ships with the app.

#### Exporting and importing master data as JSON

`Export...` writes every entry of the current list - built-ins and your own - to a JSON file whose header names the entity type (for example `"Prompt"`, `"DetailPreset"` or `"MdProfile"`). The status line confirms: `Exported {n} item(s) to {file}.`

`Import...` checks that header against the page you are on. A Prompts file therefore cannot be loaded into the Detail Presets page by accident. The import runs in this order:

1. A preview is computed: new entries, conflicts, unchanged entries, and skipped built-ins.
2. If the header or the file itself is unusable, the import is aborted with `Import failed: {first error}`.
3. If there are conflicts (same ID, different data), a dialog appears with `Skip conflicts` (keep the existing entries) and `Replace conflicts`. Cancelling gives `Import cancelled.`

4. **Built-ins in the file are always skipped.**
5. The status line reads `Imported: {a} new, {b} replaced, {c} skipped.` - extended by `({n} error(s) - see log)` when individual entries could not be imported.

### Which pages have an active or default entry

Fifteen pages share the master/detail pattern. All of them offer `Add`, `Duplicate`, `Delete`, `Save`, `Reload`, `Import...` and `Export...`, and all of them support overriding built-ins and exporting a built-in snippet. The difference is the active/default concept:

| Pages | Badge | Meaning of `Apply as Active` |
|---|---|---|
| Compression Rules · Pipeline Profiles · Quality Profiles · Test Profiles · MCP Profiles · Templates | `Active` | Exactly one entry is globally active; the list shows an `Active` badge on it. |
| Detail Presets · Expansion Strategies · MD Profile · Model Profiles · Tag Schemata | `Default` | Exactly one entry is the default (for `Tag Schemata`: one per schema type). |
| Layer Profiles · Exclude Namespaces · Prompts · Test Descriptions | (none) | The `Apply as Active` button is not shown. |

For `Layer Profiles` and `Exclude Namespaces` the global default is set elsewhere, on the `General` page (`Default layer-mapping profile` and `Default namespace-exclusion list`).

### Hint banners

Some pages show an informational banner above the fields. It is display-only and never blocks editing. You will see it, for example:

- on `Layer Profiles` and `Exclude Namespaces` as a reminder that saving a built-in marks it as overridden,
- on `Tag Schemata` in the `Layout` tab, marking the layout fields as reserved for a future renderer,
- on `Constellations`, describing exactly what an import writes and what it skips.

## 8.4 General

**Where:** Settings → `General` group → `General` (the first entry, and the page that opens on the first start).

The header has a `Reload` button ("Reload all settings from disk, discarding unsaved edits in this panel."). The footer has the accent button `Save` ("Save all changes on this panel. Some values take effect immediately, others on restart.") and the status line.

### Application

| Field | Type | Default | Effect |
|---|---|---|---|
| `On startup` | Dropdown | `Restore Last Tabs` | Intended startup behavior: `Restore Last Tabs`, `Restore Last Session`, `Sessions Overview` or `Empty Session`. **Note:** the value is stored with the other settings, but the app does not act on it yet - it currently always opens on the welcome screen. The setting is prepared for a planned session-restore feature. |
| `Theme` | Dropdown | `System Default` | `Light`, `Dark` or `System Default` (follows the operating system). Standard controls switch immediately; individual custom panels apply the theme fully after a restart. |
| `Font family` | Dropdown | `Segoe UI` | Font family for the whole interface. Choices: `Segoe UI`, `Segoe UI Variable`, `Calibri`, `Verdana`, `Tahoma`, `Consolas`, `Cascadia Mono`. Applies immediately on Save. |
| `Font size` | Dropdown | `Medium` | `Small`, `Medium` or `Large`. Acts as a **UI zoom over the whole interface** - text, controls and spacing - not just the base font. Applies immediately on Save. |
| `Scale with window size` | Checkbox | on | The UI zoom additionally scales gently with the window size (clamped). |

| Field | Type | Default | Effect |
|---|---|---|---|
| `Default layer-mapping profile` | Dropdown | `No mapping` (role heuristic only) | Applied as the fallback for Context Builder tabs whose solution has no override. The Solutions tab resolves this default for display, but changing its picker stores a per-solution override; only this General setting changes the application-wide default. |
| `Default namespace-exclusion list` | Dropdown | `None` | Drives the framework filter of the code analysis. A per-solution override on the Solutions tab takes precedence; `None` here means the per-solution choice (or the built-in BCL default) decides. |
| `Default graph layout (compression)` | Checkbox | on | Initial value for the `Layout` checkbox in each new Context Builder graph-selection tab. Controls adaptive/compressed rendering for graphs and a few non-graph sections (for example entry points). A per-tab override is preserved. |
| `Reveal Details tab on tree click` | Checkbox | on | A single click on a class or method in the Context Builder's solution tree switches to the Details tab so the code is shown right away. Folder, project and solution nodes never switch. When off, the Details tab still updates on click, but you open it yourself. |

The interface is in English; there is no language picker.

Note on the font size: `Ctrl+Plus` and `Ctrl+Minus` zoom the current session without changing this setting; `Ctrl+0` returns to the size stored here.

### Session & Analysis

| Field | Type | Default | Effect |
|---|---|---|---|
| `Auto-save enabled` | Checkbox | on | Periodically persists tabs that have a session ID. This includes an `Untitled ...` session created automatically by the first send; only a tab with no session is skipped. Changes to this setting take effect after the next application start. |
| `Auto-save interval (minutes)` | Number | `5` | How often auto-save runs. Only effective when auto-save is enabled. Values below 1 are raised to 1. Changes take effect after the next application start. |
| `Max snapshots per solution` | Number | `10` | How many automatic snapshots are kept per solution. One snapshot is written on every fresh analysis; older ones are deleted when the limit is exceeded. Snapshots you saved yourself are never deleted and do not count towards the limit. Values below 1 are raised to 1. |
| `Tab-close confirmation` | Checkbox | on | Asks before closing a tab with unsaved edits. When off, such a tab closes immediately and its unsaved edits are discarded. Closing the whole application always asks, regardless of this setting. |
| `Record solution-load performance timings` | Checkbox | on | Appends solution-load phase timings to `%APPDATA%\AIContextBuilder\load-perf.log`. This is separate from the rotating warning/error support log. |
| `Persist insights with session` | Checkbox | on | Analyzer insights are stored per solution and restored on the next load (no re-analysis needed). When off, insights stay transient - recomputed each time and never written to the database. |
| `Auto-trigger insights on solution load` | Checkbox | off | When on, the analyzer insights run automatically after every solution load. When off, use the `Analyze Solution` button in the Insights tab. |

### UX

| Field | Type | Default | Effect |
|---|---|---|---|
| `Tooltips enabled` | Checkbox | on | When off, all tooltips in the Settings sub-tabs are suppressed. |
| `Tooltip show delay (ms)` | Number | `400` | Delay before a tooltip appears on hover. 400 ms is the Windows standard; lower values feel snappier but can flicker. Values are limited to 0-5000 (0 = immediate). |

### Run Confirmation

| Field | Type | Default | Effect |
|---|---|---|---|
| `Confirm threshold (tokens)` | Number | `50000` | Before sending a run whose estimated prompt reaches this token count, the app asks you to confirm. The comparison is `>=`, so a value of `0` makes the app ask before **every** run - there is no value that switches the pre-flight off by itself; to do that, set a value no run will reach. |

Note: the cost pre-flight applies to `Iteration` and `Preselection` runs. A `Manual` run is sent without a cost estimate by design; only the context-window check below still applies to it, and only when the model profile has a `Context window` value.

The confirmation dialog itself names the number of nodes, the estimated input, output and total tokens, the estimated cost (when pricing is configured), the model name and, where relevant, a note that the expansion will stop at the configured node cap or that a single request exceeds the model's context window.

### LLM Retry

| Field | Type | Default | Effect |
|---|---|---|---|
| `Max attempts` | Number | `3` | Total attempts per LLM call. `1` means **no retry**. `3` means one initial call plus two retries. Applies to rate-limit (429) and transient 5xx/network errors. Values below 1 are raised to 1. |
| `Base backoff (seconds)` | Decimal | `1.0` | First retry delay; doubled on each further attempt (1 s → 2 s → 4 s ...). A `Retry-After` header from the provider overrides it. |
| `Max backoff (seconds)` | Decimal | `30.0` | Upper bound for the backoff; also caps a provider's `Retry-After` value. |

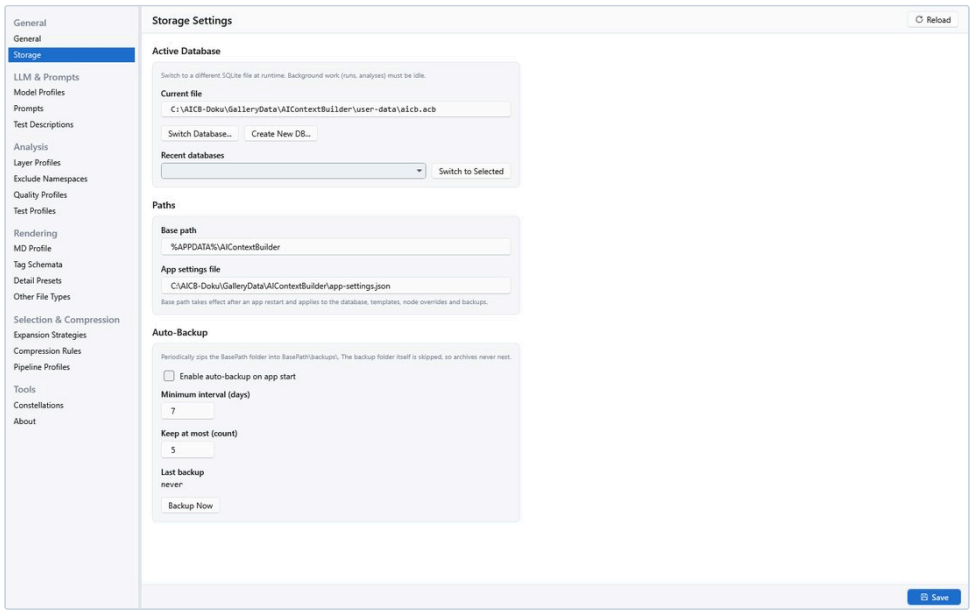
Below the section: "Auto-retry for LLM calls on rate-limit and transient errors. Changes take effect after an app restart."

### Empty numeric fields

An emptied or unreadable numeric field on the `General` and `Storage` pages falls back to the **default** of that setting - not to its minimum. The difference matters where the minimum has a special meaning: `Max attempts` = `1` means "no retry", and `Confirm threshold (tokens)` = `0` means "always ask". Clearing such a field restores the default (`3` and `50000` respectively).

## 8.5 Storage

**Where:** Settings → `General` group → `Storage`.




Settings > Storage: active database, paths and auto-backup

This page controls where the app keeps its data, which SQLite file is active, and the automatic backup.

### Active Database

Introductory text: "Switch to a different SQLite file at runtime. Background work (runs, analyses) must be idle."

| Field | Type | Notes |
|---|---|---|
| `Current file` | Read-only text | Full path of the currently active database. |
| `Switch Database...` | Button | Pick an existing SQLite file and make it active. Requires all background work (runs, analyses) to be idle. |
| `Create New DB...` | Button | Create a fresh, empty SQLite database at a chosen location and switch to it. Use `Switch Database...` for an existing file. |
| `Recent databases` | Dropdown | Recently used database paths, excluding the one that is currently active. |
| `Switch to Selected` | Button | Switches to the path selected in the dropdown. |

A separate status line next to the two switch buttons reports the outcome ( `Switched to {path}.` , `Created and switched to {path}.` , `Switch cancelled.` , or an error message).

Two rules apply here:

- **The app's database file must end in `.acb` .** The default is `aicb.acb` . Any other extension is rejected both when switching and when creating - the file dialog only offers `AICB database (*.acb)` . Note: if the app no longer starts because of a wrong database path, correct it in `%APPDATA%\AIContextBuilder\app-settings.json` (the `DatabasePath` entry). A `dbPath` that you pass explicitly to the command-line or MCP tools is not affected by this rule.
- **Switching to an existing database checks its schema.** If migrations are pending, a dialog shows the current and latest schema version plus the pending migrations and asks you to confirm before the switch and the migrations are applied. A target that is not compatible with this app build is rejected.
- While a background task is active (a run or an analysis), the switch and create buttons refuse with `Cannot {action} while {N} background task(s) are active.` - the number tells you how many tasks are in the way.

### Paths

| Field | Type | Default | Notes |
|---|---|---|---|
| `Base path` | Text | `%APPDATA%\AIContextBuilder` | Root folder for app data. **Takes effect only after an app restart** and applies to the database, templates, node overrides and backups. Must not be empty and must be an absolute path; otherwise Save reports `BasePath must not be empty.` or `BasePath must be an absolute path (e.g. C:\AIBuilder).` |
| `App settings file` | Read-only text | `%APPDATA%\AIContextBuilder\app-settings.json` | The JSON file that stores the storage paths and the recent-path lists. This field is a display of the file's **actual** location: it does not move when you change `Base path` . |

Derived from the base path: the database ( `{BasePath}\user-data\aicb.acb` ), `{BasePath}\templates.json` and `{BasePath}\node-overrides.json` .

Note: saving a new `Base path` can change which database file is active, because the default database path is derived from the base path. In that case the app asks first ("Change data directory?"), naming the new database file, whether it already exists and how many migrations will be applied. The old database file is left untouched, and its content does not move - settings, sessions and solutions are read from the new file from then on.

### Auto-Backup

Introductory text: "Periodically writes a ZIP into BasePath\backups\: a consistent copy of the active database, taken while the app keeps working, plus the other files under BasePath. Other database files there (retired copies) are left out, and so is the backup folder itself."

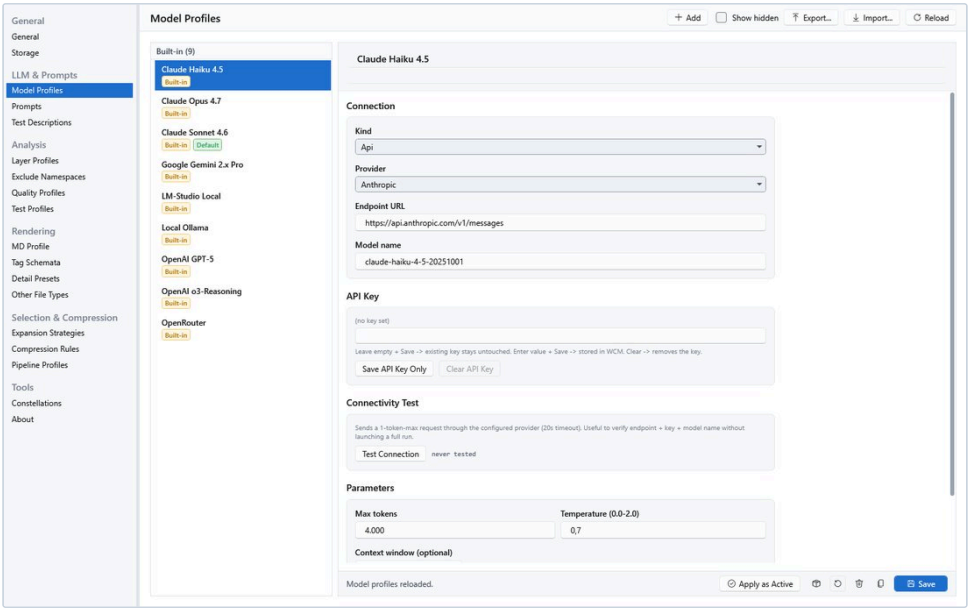
| Field | Type | Default | Notes |
|---|---|---|---|
| `Enable auto-backup on app start` | Checkbox | on for a new installation | When on, the app backs up in the background once the main window is open, if the minimum interval has elapsed. An existing installation keeps the value it had. |
| `Minimum interval (days)` | Number | `7` | Auto-backup only runs if at least this many days have passed since the last backup. Values below 1 are raised to 1. |
| `Keep at most (count)` | Number | `5` | Maximum number of backup archives that are kept. Older archives are pruned - except that the newest complete archive is never deleted. Values below 1 are raised to 1. |
| `Last backup` | Read-only text | `never` | Timestamp of the last successful backup. |
| `Backup Now` | Button | - | Runs a backup immediately, regardless of the configured interval, in the background; the button is disabled and the status line says `Backing up ...` while it runs. It can be used while an analysis or a run is active. |

The archive contains the files under the base path (the `backups` folder and other database files are skipped) and a consistent snapshot of the active database, even when that file lives outside the base path. The snapshot is taken with the backup function of SQLite itself while the app and any MCP server on the same database keep working, so the `-wal` and `-shm` siblings are not needed. While it is written, the archive is called `...zip.inprogress`; it gets its final name only when it is complete. A file that could not be read is listed in the result and the archive is renamed with the suffix `-partial`.

Note: the automatic backup runs in the background **after** the main window appears. If a run fails - for example because of an unreadable sub-folder - the failure is reported in the startup notice bar, and the run is retried at every start until the cause is fixed. If the automatic backup is not wanted, switch it off here. While a backup runs, switching or creating a database and changing the base path wait for it.

## 8.6 Model Profiles

**What it is:** the LLM provider configurations. Exactly one entry is the `Default` profile, used by run templates that do not pin a specific profile.




Settings > Model Profiles

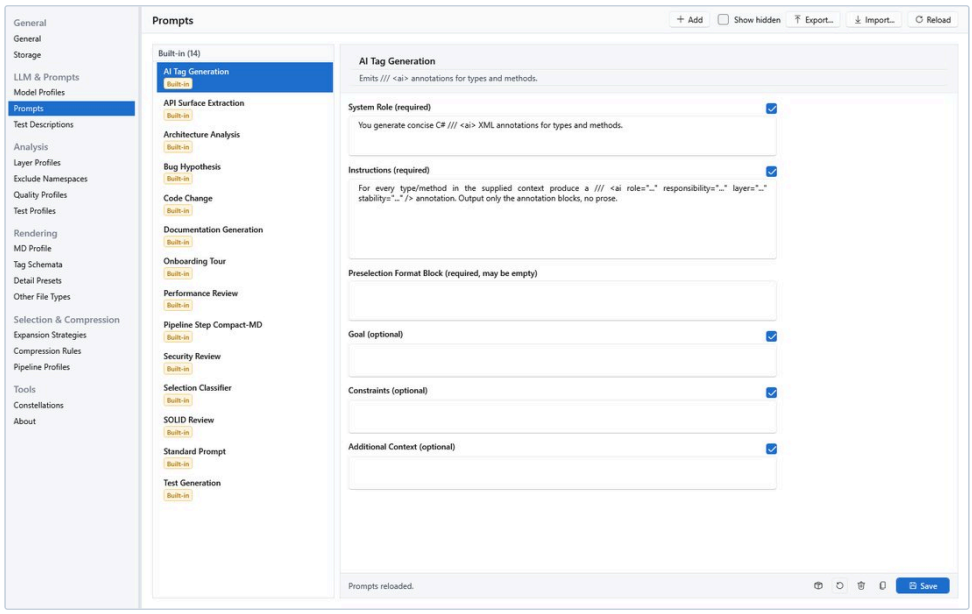
The editor has four sections:

- `Connection` - `Kind`, `Provider`, `Endpoint URL`, `Model name`.
- `Kind`: `Api` for a remote cloud endpoint, `Local` for a local server (for example llama-server, vLLM or Ollama).
- `Provider`: the request format and authentication scheme - `Anthropic`, `OpenAI` or `Custom`.
- `Endpoint URL`: for example `https://api.anthropic.com`, or `http://localhost:8080` for a local llama-server / `http://localhost:11434` for Ollama.
- `Model name`: the provider-specific model identifier that is sent in the request body (for example `claude-opus-4-5`, `gpt-4o`, `qwen2.5-coder-32b-instruct`).
- `API Key` - shown for `Api` and `Local`. The section reports whether a key is stored. Type a key and press `Save` (or `Save API Key Only`) to store it. The help text reads: "Leave empty + Save -> existing key stays untouched. Enter value + Save -> stored in WCM. Clear -> removes the key."
- `Save API Key Only` stores only the key without saving the other editor fields.
- `Clear API Key` removes the stored key; later runs fail until a new key is entered.
- **The key is not part of the profile.** It is kept in the Windows Credential Manager and is never written to the database, to a master-data export or to a constellation file.
- `Connectivity Test` - `Test Connection` sends a probe request with a maximum of 16 output tokens through the configured provider and uses a 20-second timeout. The small allowance lets reasoning models begin their reasoning prefix while verifying endpoint, key and model name without starting a full run. The result is shown next to the button and stored on the profile, so it survives a restart. The probe uses the values currently in the editor, so you can test before saving.
- `Parameters` - `Max tokens` (maximum output tokens per request; default `8000`, minimum 1; provider limits apply), `Temperature (0.0-2.0)` (default `0.7`; `0.0` is deterministic, `2.0` creative; values are clamped to 0.0-2.0), `Context window (optional)` (total input + output tokens the model can handle; used by the pre-flight check to warn when a single request would exceed it; leave empty to skip the check), and `Pricing per 1M tokens (optional)` with the sub-fields `Input` and `Output` in USD (used by the cost estimator in the run summary).

Note: duplicating a profile copies its settings but deliberately **not** the API key and **not** the default badge - you have to enter the key for the copy yourself.

## 8.7 Prompts

**What it is:** reusable prompt presets. There is no active/default entry; a template picks the prompt it uses.




Settings > Prompts

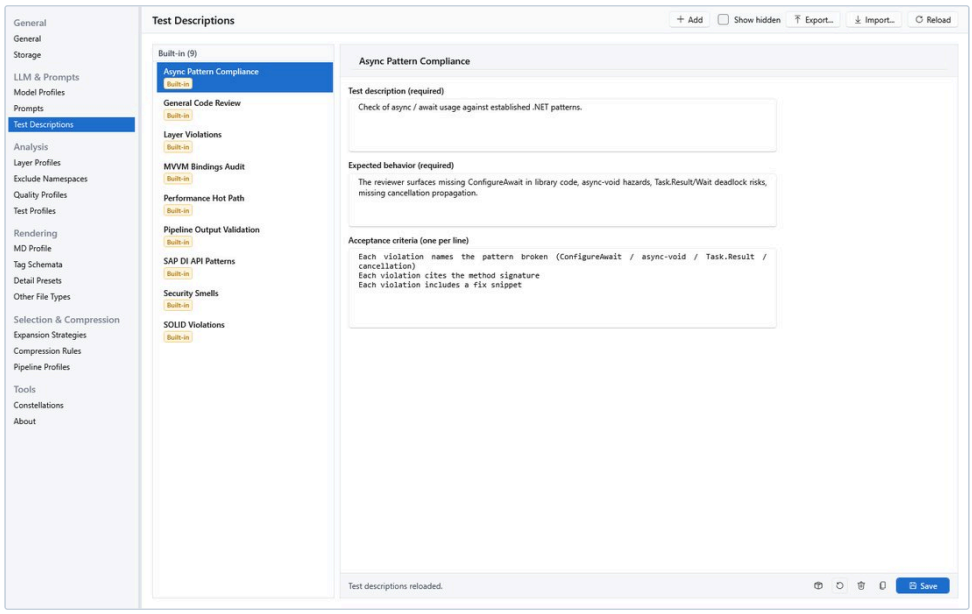
The editor fields:

| Field | Required | Notes |
|---|---|---|
| `System Role (required)` | yes | The system-role prompt sent to the LLM - persona, expertise, tone. |
| `Instructions (required)` | yes | The main task instructions: what to do with the provided context document. |
| `Preselection Format Block (required, may be empty)` | yes (may be empty) | Output-format directive for preselection runs (for example a JSON shape). May be empty for prompts that are not used for preselection. |
| `Goal (optional)` | no | A high-level goal, one or two sentences. Appended below the instructions when present. |
| `Constraints (optional)` | no | Guardrails, for example "do not invent APIs". Appended to the prompt when present. |
| `Additional Context (optional)` | no | Free-form context such as project conventions or domain hints. Appended last. |

Next to the labels of `System Role`, `Instructions`, `Goal`, `Constraints` and `Additional Context` there is a checkbox. It controls whether that area is **shown** on the Context Builder's `Prompt` tab while this prompt is active. Unchecking hides the area there, but the stored text is kept and still goes into the assembled prompt. All five checkboxes default to on. The `Preselection Format Block` has no such checkbox.

## 8.8 Test Descriptions

**What it is:** reusable acceptance-test descriptions. They can be attached to a template and are shipped into the context document, so the LLM knows the test contract. There is no active/default entry.



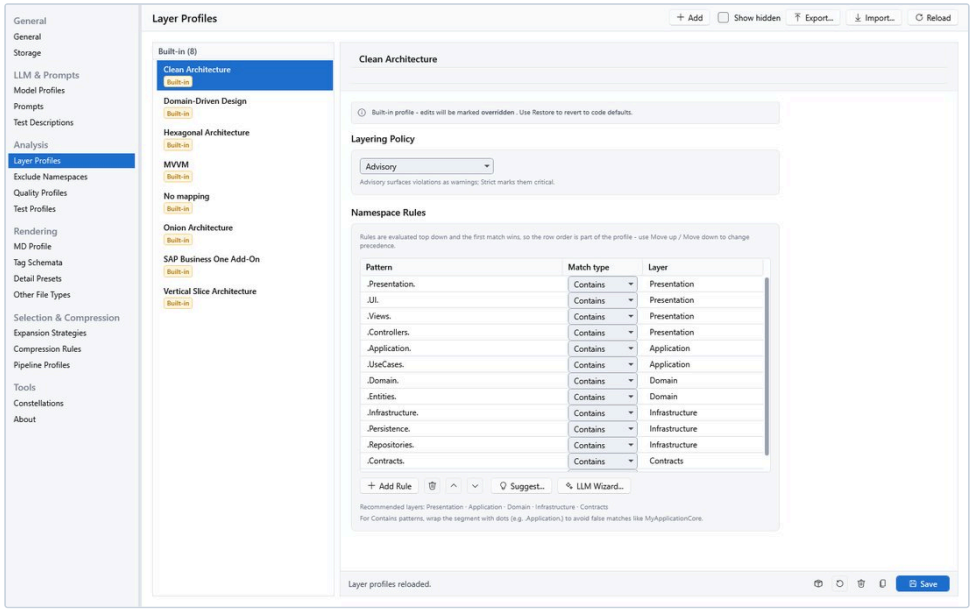

Settings > Test Descriptions

| Field | Required | Notes |
|---|---|---|
| `Test description (required)` | yes | Narrative description of what the test validates. The LLM uses it to understand the intent before generating or fixing code. |
| `Expected behavior (required)` | yes | What the system should do when the test runs successfully. Phrase it as observable behavior, not implementation details. |
| `Acceptance criteria (one per line)` | no | A checklist of pass conditions, one per line. The LLM treats them as a hard contract. |

The name in the header is a short identifier used in the picker when you attach the description to a template.

## 8.9 Layer Profiles

**What it is:** mappings from namespace patterns to architectural layers. There is no active/default entry here - the app-wide default is set on the `General` page, and a solution can pick its own profile in the Workspace.




Settings > Layer Profiles: namespace rules and layering policy

The editor has two sections:

- `Layering Policy` - `Advisory` (default): layer violations surface as warnings. `Strict`: layer violations are marked critical and act as a quality gate.
- `Namespace Rules` - a table with one row per rule:
- `Pattern`: the namespace pattern to match, for example `MyApp.Application` or `.Infrastructure.`.
- `Match type`: `Contains` (substring), `StartsWith` (prefix), `EndsWith` (suffix) or `Exact` (full namespace).
- `Layer`: the architectural layer assigned when the pattern matches, for example `Presentation`, `Application`, `Domain`, `Infrastructure` or `Contracts`.

**Rules are evaluated from top to bottom and the first match wins**, so the row order is part of the profile. Use the arrow buttons (`Move up`, `Move down`) to change precedence - the table cannot be sorted, precisely because sorting would misrepresent the profile.

Row actions:

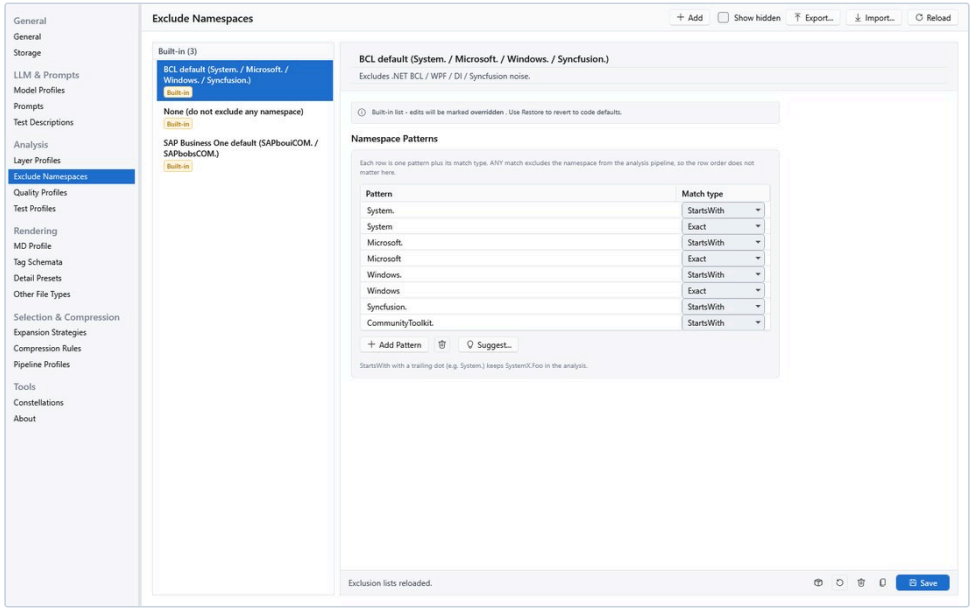
- `Add Rule` appends an empty row (default match type `Contains`).
- The delete icon removes the selected row.
- `Suggest...` lets you pick a `.sln` file, reads its declared namespaces, and lets you select the ones you want. Each pick is added as a new rule with the match type `StartsWith` and an empty layer - assign the layer per row afterwards.
- `LLM Wizard...` reads a solution's namespaces, asks you for a description of your architecture, and lets the **default model profile** propose layer rules (written into the grid, where you can fine-tune them) plus namespace exclusions (which you can optionally save as a new exclusion list). This needs a configured model profile; otherwise the wizard tells you to add one under `Settings > Model Profiles`.

Notes from the page:

- Recommended layers: `Presentation`, `Application`, `Domain`, `Infrastructure`, `Contracts`.
- For `Contains` patterns, wrap the segment with dots (for example `.Application.`) to avoid false matches such as `MyApplicationCore`.
- Saving requires a name and, for every rule, a pattern and a layer; otherwise the status line names the missing value (for example `Rule #2: Layer must not be empty.`).

## 8.10 Exclude Namespaces

**What it is:** lists of namespace patterns that the analysis skips, which reduces noise and token cost by filtering out framework or vendor code. There is no active/default entry here; the app-wide default is on the `General` page, and a solution can pick its own list.




Settings > Exclude Namespaces

The editor has one section, `Namespace Patterns`, with a table of `Pattern` and `Match type` rows. `Match type` offers `Contains`, `StartsWith`, `EndsWith` and `Exact`. **Any** matching rule excludes the namespace, so unlike layer rules the row order does not matter here.

Row actions: `Add Pattern` appends an empty row (default match type `StartsWith`), the delete icon removes the selected row, and `Suggest...` reads a solution's namespaces and adds the ones you pick as `StartsWith` exclusions.

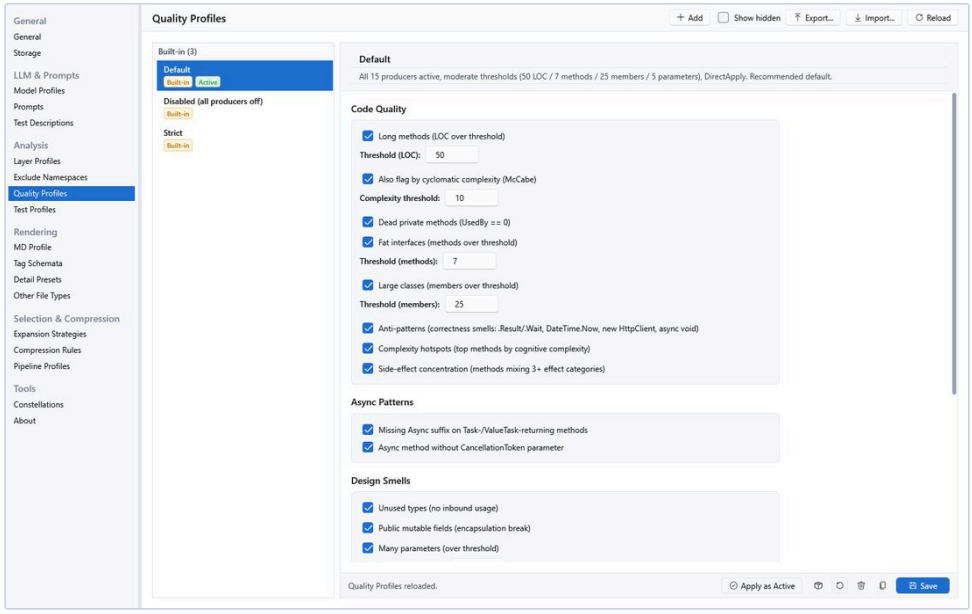
Rules:

- A name is required; the list may not contain more than 500 rules; a pattern must not be empty and must not exceed 500 characters. Violations appear in the status line.
- `StartsWith` with a trailing dot (for example `System.`) keeps a namespace such as `SystemX.Foo` in the analysis.

The built-in lists include `None (do not exclude any namespace)`, `BCL default (System. / Microsoft. / Windows. / Syncfusion.)` and `SAP Business One default (SAPbouiCOM. / SAPbobsCOM.)`.

## 8.11 Quality Profiles

**What it is:** the switches and thresholds of the code-quality checks, plus the behavior of the resulting insights. Exactly one profile is `Active`; a template can override it.




Settings > Quality Profiles: producer switches and thresholds

`Code Quality`

| Setting | Default | Notes |
|---|---|---|
| `Long methods (LOC over threshold)` | on | Emits an insight when a method exceeds the line threshold. |
| `Threshold (LOC):` | `50` | Above this line count a method counts as long. Typical values: 30-50. |
| `Also flag by cyclomatic complexity (McCabe)` | on | The long-method check additionally flags methods whose McCabe complexity exceeds the threshold below - even when they are short. |
| `Complexity threshold:` | `10` | Typical values: 10-15. Only effective when the checkbox above is on. |
| `Dead private methods (UsedBy == 0)` | on | Private methods with no detected inbound usage. Likely safe to delete. |
| `Fat interfaces (methods over threshold)` | on | Signals interface-segregation violations. |
| `Threshold (methods):` | `7` | Typical values: 8-15. |

| Setting | Default | Notes |
|---|---|---|
| `Large classes (members over threshold)` | on | Signals single-responsibility violations. |
| `Threshold (members):` | `25` | Typical values: 20-30. |
| `Anti-patterns (correctness smells: .Result/.Wait, DateTime.Now, new HttpClient, async void)` | on | Aggregates the correctness smells found by the analyzer. Note: these smell flags are available for live sessions only; they are not persisted in snapshots. |
| `Complexity hotspots (top methods by cognitive complexity)` | on | Ranks the methods with the highest cognitive complexity (deep nesting, hard-to-follow branching). Complements the long-method check without double-reporting. |
| `Side-effect concentration (methods mixing 3+ effect categories)` | on | Methods whose body mixes three or more distinct side-effect categories (I/O, state mutation, logging, ...). |

`Async Patterns`

| Setting | Default | Notes |
|---|---|---|
| `Missing Async suffix on Task-/ValueTask-returning methods` | on | Improves call-site readability. |
| `Async method without CancellationToken parameter` | on | Important for cooperative cancellation in long-running operations. |

`Design Smells`

| Setting | Default | Notes |
|---|---|---|
| `Unused types (no inbound usage)` | on | Likely safe to delete; mind reflection-only consumers. |
| `Public mutable fields (encapsulation break)` | on | Non-readonly public fields; prefer properties or readonly fields. |
| `Many parameters (over threshold)` | on | Signals refactoring to a parameter object or builder. |
| `Threshold (parameters):` | `5` | Typical values: 4-6. |
| `Layer violations (inner layer depends on outer layer)` | on | Clean/Onion rule: dependencies must point inward. Layer resolution uses the namespace conventions ( `.Domain.` , `.Application.` , `.Infrastructure.` , `.Contracts.` , `.Presentation.` ) plus a role heuristic. |
| `Circular namespace dependencies (dependency cycles)` | on | Cycles in the namespace dependency graph (A to B to A). Marked critical when a cycle spans multiple projects. |

`Behaviour`

| Setting | Default | Notes |
|---|---|---|
| `Insights are dismissable (persist hidden per solution)` | on | When on, a dismissed insight stays dismissed for that solution. When off, insights reappear after dismissing until the check itself stops reporting them. |
| `Send-Template-Id override (optional)` | empty | A run-template ID used by the `Apply + Send to LLM` action. Empty means the active tab's run template. If the ID is not found, the app silently falls back to the active tab's run template. |

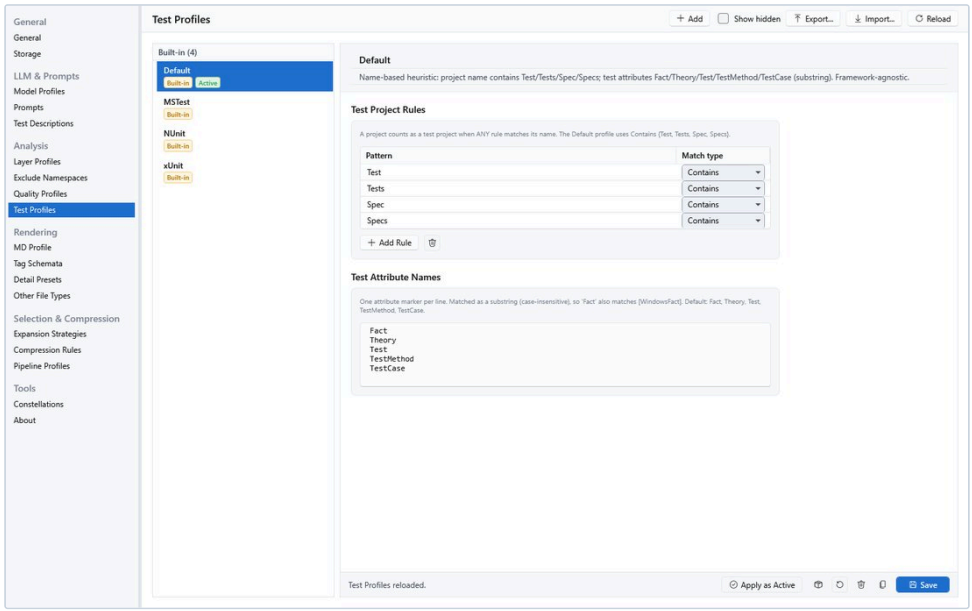
| Setting | Default | Notes |
|---|---|---|
| `Action mode` | `Direct Apply` | `Direct Apply` : clicking Apply runs the action immediately. `Confirm Dialog` : a confirmation dialog with context information is shown first. A profile chosen on the run template takes precedence over this setting. |

The editor exposes 14 producer switches. The profile model also stores `EnableUnresolvedXamlBinding` , but the current desktop panel has no switch for it; the separate McCabe checkbox controls the long-method rule rather than a fifteenth producer.

Note: an emptied threshold field falls back to its default ( `50` for LOC, `10` for complexity, `7` for fat interfaces, `25` for large classes, `5` for parameters).

## 8.12 Test Profiles

**What it is:** the test-detection configuration, pinnable per solution. It decides which projects count as test projects and which method attributes mark a test case. Exactly one profile is `Active` ; a solution can override it.




Settings > Test Profiles

| Section | Content |
|---|---|
| `Test Project Rules` | A table of `Pattern` and `Match type` rows ( `Contains` , `StartsWith` , `EndsWith` , `Exact` ). A project counts as a test project when **any** rule matches its name. |
| `Test Attribute Names` | One attribute marker per line, matched as a case-insensitive substring - so `Fact` also matches `[WindowsFact]` . |

Built-in profiles ship with the app:

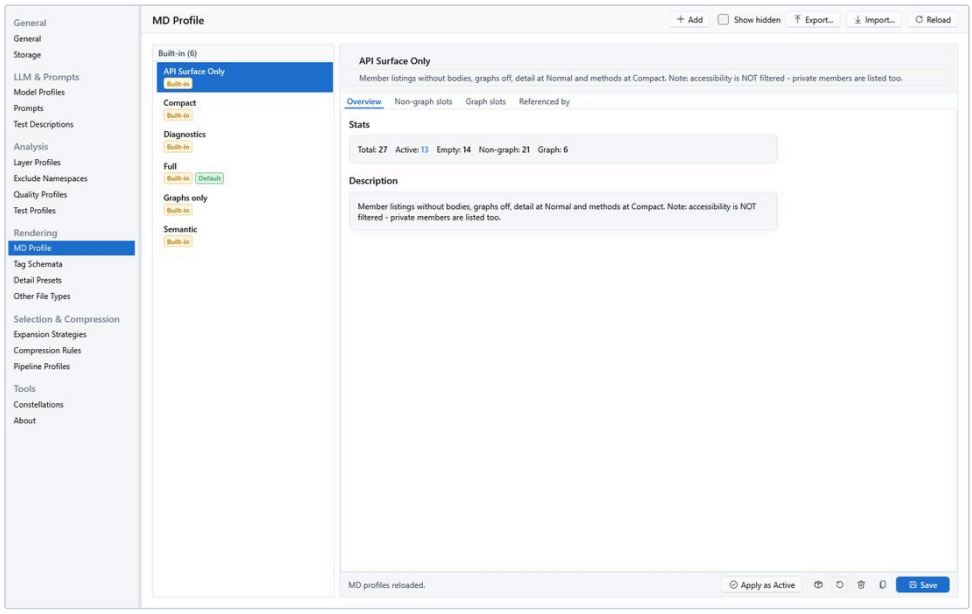
| Profile | Project name matches | Test attributes |
|---|---|---|
| `Default` | contains `Test` , `Tests` , `Spec` , `Specs` | `Fact` , `Theory` , `Test` , `TestMethod` , `TestCase` |
| `xUnit` | contains `Test` , `Tests` | `Fact` , `Theory` |
| `NUnit` | contains `Test` , `Tests` | `Test` , `TestCase` , `TestCaseSource` , `TestFixture` |
| `MSTest` | contains `Test` , `Tests` | `TestMethod` , `DataTestMethod` , `TestClass` |

The `Default` profile replicates the built-in heuristic exactly, so without an active custom profile the detection behavior is unchanged.

`Apply as Active` makes a profile the global default. The analyzer consumers use it unless a per-solution profile overrides it; the per-solution choice wins over the global one.

## 8.13 MD Profile

**What it is:** the mapping from section types to tag schemata, staged by detail level. Exactly one profile is the `Default`; a template picks the profile it uses.




Settings > MD Profile

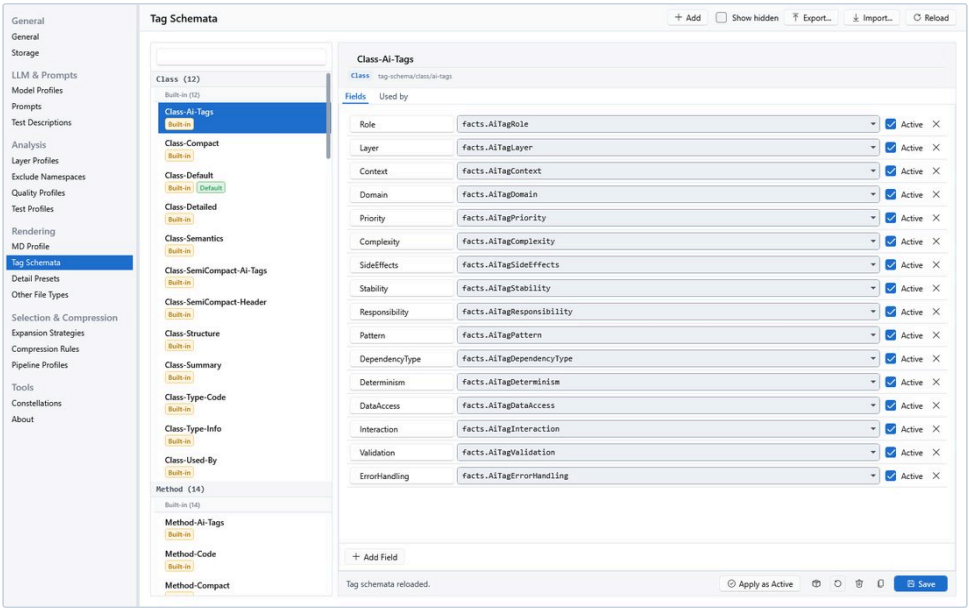
The editor has four tabs:

- `Overview` - a read-only summary: `Stats` with `Total`, `Active`, `Empty`, `Non-graph` and `Graph` slot counts, plus the profile's description.
- `Non-graph slots` - two groups:
- `Detail (Staged)` for the four code structure sections (class, method, interface, enum). Each row offers a schema for `Compact`, `Normal` and `Detailed`, a `Default level:` choice (`Compact`, `Normal`, `Detailed`, `Source`), and an `Allow Source` checkbox. `Source` as the default level requires `Allow Source` to be on. Clearing a sub-slot means that level falls through to the default level.
- `Section (Single)` - one schema per remaining section. **An empty slot skips that section in the generated output.**
- `Graph slots` - one schema per dependency-graph section (Mermaid / DOT). The help text notes that layout/compression is controlled app-globally (`Settings > General > 'Default graph layout'`) with a per-tab override in the Context Builder.
- `Referenced by` - the templates that currently reference this profile. If none do, the page says so.

`Apply as Active` marks the profile as the default. Note: a newly created MD profile starts as a copy of the current default profile, so it behaves like the default until you change it - it does not start empty.

## 8.14 Tag Schemata

**What it is:** the rendering rules per tag type (class, method, interface, enum, sections and graphs). Exactly one schema per schema type is the `Default`.




Settings > Tag Schemata

The list on the left is grouped first by schema type, then by built-in/custom, and has a filter box above it: type to filter by name or schema type; `Ctrl+F` focuses the box, `Esc` clears it.

The editor has three tabs:

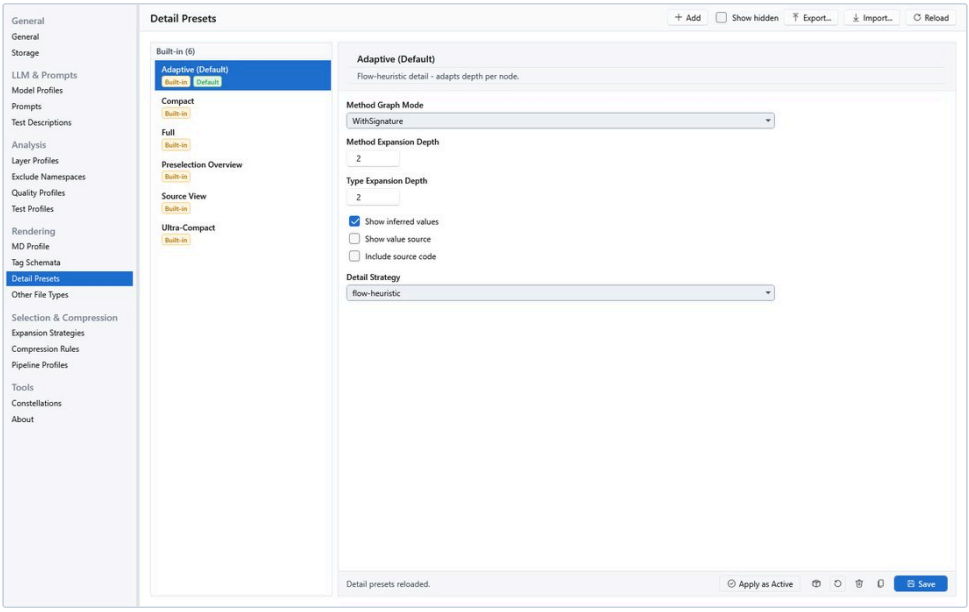
- `Fields` - the fields that are rendered for this schema. Each row has:
- a `Field name`,
- a `Source` path chosen from the list that is valid for this schema type (the accepted paths are deterministic facts and resolved values, for example `facts.TypeName` or `resolved.Layer`; the list offers only paths that produce output). A field whose stored path is no longer offered shows a warning, and the schema cannot be saved until the field is changed or removed,
- an `Active` checkbox: when unchecked, the field is kept in the schema but skipped when rendering,
- a remove button. `Add Field` appends a new row. Fields can only be added or removed on custom schemas.
- `Layout` - only visible for graph schemata. **Note: this tab is reserved for a future renderer.** The four read-only fields (`Node label`, `Edge style`, `Clustering`, `Depth limit (1-10)`) are stored with the schema but are currently not used by any renderer. App-global compression on/off is on the `General` page instead.
- `Used by` - the MD profiles that currently reference this schema. If none do, the page says so.

Under each schema the page says where it shows up in an export, or why it does not. A schema you create yourself reaches the export when an MD profile's `Class`, `Interface`, `Enum` or `Method` row, or the `Sections` panel of a tab, picks it: it becomes the first lines of every such block (for a method, its `METHOD_INFO`). A copy of a built-in block replaces just that block. The `Solution`, `Project` and `File` schemas render the header lines of their sections.

`Apply as Active` marks the schema as the default **for its schema type**. That default is used wherever a render resolves a section without an explicit schema - for example a slot whose schema was deleted, an empty level in a staged slot, or a render that runs without an MD profile. (An empty single slot in an MD profile suppresses its section instead - see "MD Profile".)

## 8.15 Detail Presets

**What it is:** presets that control graph mode, expansion depths and source-code inclusion. Exactly one preset is the `Default`; a template assigns one preset per detail level.

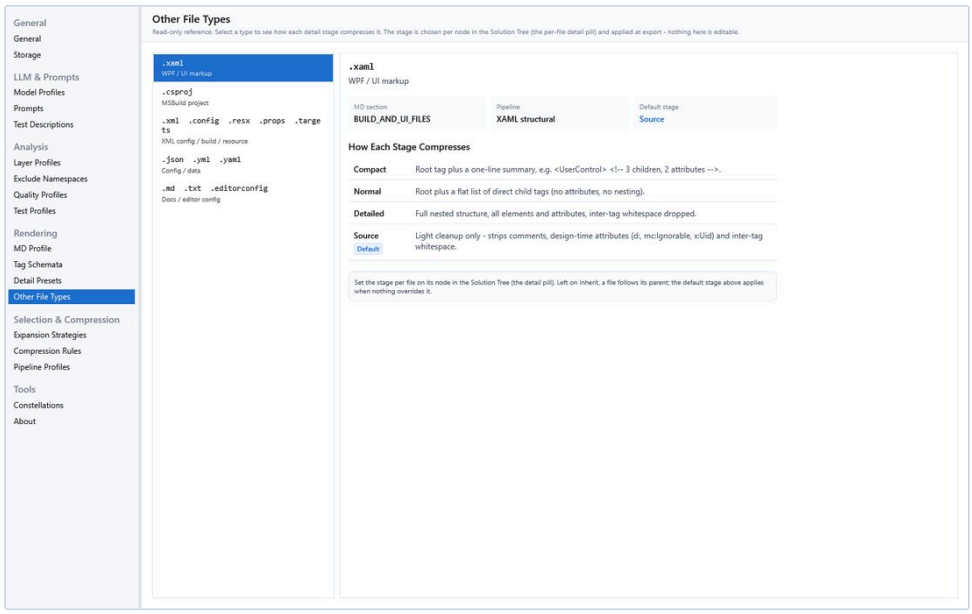



Settings > Detail Presets

| Field | Notes |
|---|---|
| `Method Graph Mode` | `Full`, `Semantic Only`, `With Signature` or `With Body`. |
| `Method Expansion Depth` | How many call-graph hops are expanded from the selected method. `1` = direct neighbors only; `2`-`3` = small clusters; higher values mean more context and more tokens. |
| `Type Expansion Depth` | How many type-dependency hops are expanded (uses / used-by). |
| `Show inferred values` | Includes inferred or resolved values in the output (for example const expressions and default parameter values). |
| `Show value source` | Annotates inferred values with their source for traceability. |
| `Include source code` | The renderer appends a source-code block for code nodes that resolve to this preset - typically the preset used for a template's source level. |
| `Detail Strategy` | One of `flow-heuristic` (= compact, adaptive), `aggressive` (= compact, no adaptive) or `none` (= full, no adaptive). |

## 8.16 Other File Types

**This page is a read-only reference and has no actions.** It lists the recognized non-code file types; selecting one shows which section of the context document it lands in, which compression pipeline it runs through, and what each of the four detail stages does for it.




Settings > Other File Types

The stage itself is **not** configured here. It is chosen per file node in the solution tree (the detail pill on the node) and applied at export, exactly like code nodes. A file left on `Inherit` follows its parent; the default stage applies when nothing overrides it.

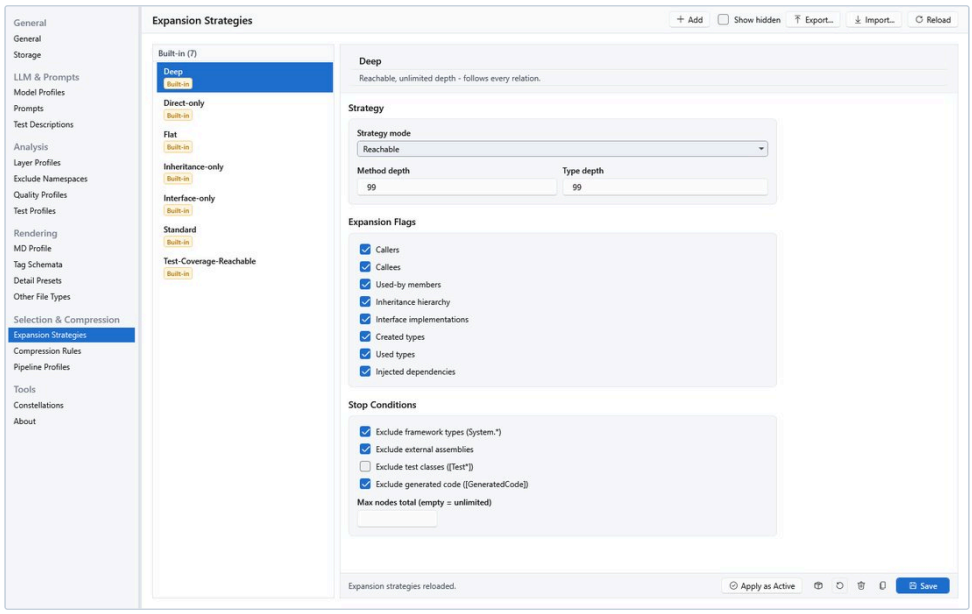
| Extensions | Category | Section | Pipeline | Default stage |
|---|---|---|---|---|
| `.xaml` | WPF / UI markup | `BUILD_AND_UI_FILES` | XAML structural | `Detailed` |
| `.csproj` | MSBuild project | `BUILD_AND_UI_FILES` | XML textual | `Normal` |
| `.xml` `.config` `.resx` `.props` `.targets` | XML config / build / resource | `AUXILIARY_FILES` | XML textual | `Source` |
| `.json` `.yml` `.yaml` | Config / data | `AUXILIARY_FILES` | Plain text | `Source` |
| `.md` `.txt` `.editorconfig` | Docs / editor config | `AUXILIARY_FILES` | Plain text | `Source` |

What the stages do, per pipeline:

- **XAML structural** ( `.xaml` ): `Compact` - root tag plus a one-line summary (for example `<UserControl> <!-- 3 children, 2 attributes -->` ); `Normal` - root plus a flat list of direct child tags (no attributes, no nesting); `Detailed` - the full nested structure with all elements and attributes, inter-tag whitespace dropped; `Source` - the raw file content.
- **XML textual** ( `.csproj` , `.xml` , `.config` , `.resx` , `.props` , `.targets` ): `Compact` - drops comments, empty groups and non-package item groups; keeps the project root, `TargetFramework` and package references (plain XML gets the comment/whitespace strip); `Normal` - strips XML comments, drops empty groups, collapses inter-tag whitespace and blank lines; `Detailed` - keeps comments and empty groups and only trims inter-tag whitespace and blank lines; `Source` - raw, uncompressed.
- **Plain text** ( `.json` , `.yml` , `.yaml` , `.md` , `.txt` , `.editorconfig` ): `Compact` - the `Normal` result, then capped at 60 lines with a visible truncation marker (the only stage that drops content - from the tail); `Normal` - trailing whitespace trimmed and runs of blank lines collapsed to one; `Detailed` - trailing whitespace trimmed per line, structure and blank lines kept; `Source` - raw, uncompressed.

## 8.17 Expansion Strategies

**What it is:** strategies that define how deep and how wide the analysis walks the call/type graph from the selected entry points. Higher depth means richer context but more tokens. Exactly one strategy is the `Default` .




Settings > Expansion Strategies

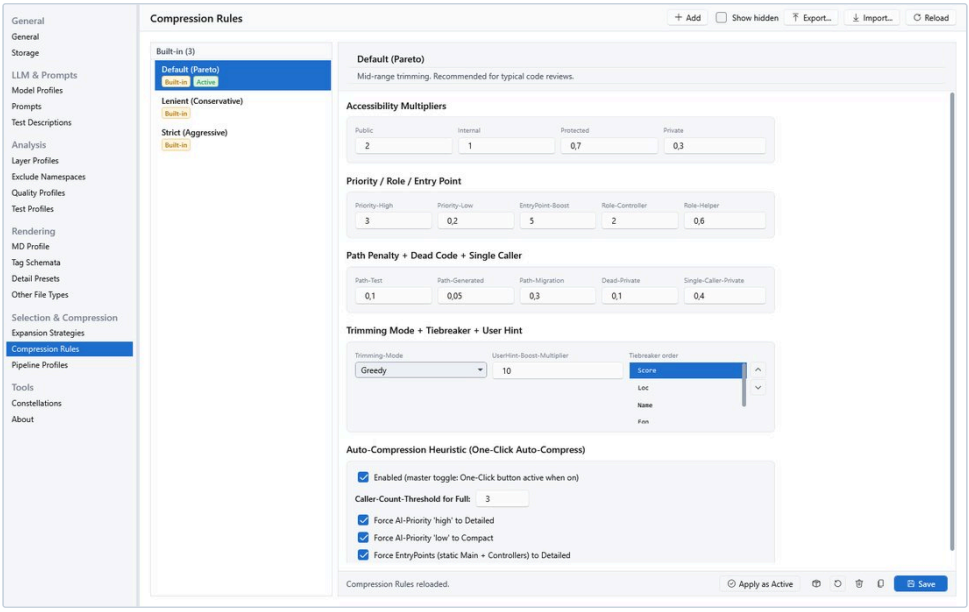
| Section | Field | Notes |
|---|---|---|
| `Strategy` | `Strategy mode` | `Reachable` - all reachable nodes from the selection root, bounded by depth limits and stop conditions. `Select` - only the explicitly selected methods/types, not transitively. `Frontier` - the direct neighbors of the selection without further expansion. |
| | `Method depth` | How many levels of called methods to include from the entry point. |
| | `Type depth` | How many levels of referenced types (parameters, fields, inheritance) to follow from each method. |
| `Expansion Flags` | `Callers` | Include methods that call the selected symbol (incoming call edges). |
| | `Callees` | Include methods called by the selected symbol (outgoing call edges). |
| | `Used-by members` | Include members that reference the selected symbol (reverse usage). |
| | `Inheritance hierarchy` | Include base and derived types along the inheritance chain. |
| | `Interface implementations` | Include the concrete types that implement a selected interface. |
| | `Created types` | Include types instantiated inside the selected method. |
| | `Used types` | Include types referenced as parameters, fields, locals or return values. |
| | `Injected dependencies` | Include constructor-injected dependencies of the selected type. |
| `Stop Conditions` | `Exclude framework types (System.*)` | Stop the walk at framework types. |
| | `Exclude external assemblies` | Stop the walk at types from external / NuGet assemblies - only solution code expands. |
| | `Exclude test classes ([Test*])` | Do not expand into test classes. |
| | `Exclude generated code ([GeneratedCode])` | Do not expand into generated code. |
| | `Max nodes total (empty = unlimited)` | Hard cap on the number of nodes the expansion may emit - a safety net against runaway graph walks. |

`Interface implementations` follows an interface to the types that implement it, within the strategy's type depth and stop conditions. For a generic interface ( `IHandler<T>` ) the implementers it can reach are the implementers of every closing ( `IHandler<Saved>` , `IHandler<Deleted>` ), whether a type implements it directly, through a base class or through another interface.

`Apply as Active` sets the selected strategy as the app-global default, used by tabs when no run- or template-specific strategy is set.

## 8.18 Compression Rules

**What it is:** the scoring rules behind compression and the one-click auto-compress. Exactly one profile is `Active` . The built-in profiles are `Default` (balanced), `Strict` (aggressive trimming) and `Lenient` (conservative trimming).




Settings > Compression Rules

`Accessibility Multipliers` - score multipliers; higher means more likely to be kept detailed.

| Field | Default | Field | Default |
|---|---|---|---|
| `Public` | `2.0` | `Internal` | `1.0` |
| `Protected` | `0.7` | `Private` | `0.3` |

`Priority / Role / Entry Point`

| Field | Default | Notes |
|---|---|---|
| `Priority-High` | `3.0` | For nodes flagged with high AI priority - keeps them detailed. |
| `Priority-Low` | `0.2` | Demotes them toward compact. |
| `EntryPoint-Boost` | `5.0` | For entry-point methods (static `Main` , controller actions). |
| `Role-Controller` | `2.0` | For methods classified as controller (orchestration/dispatch role). |
| `Role-Helper` | `0.6` | For helper methods; typically below 1.0 to allow aggressive compression. |

`Path Penalty + Dead Code + Single Caller`

| Field | Default | Notes |
|---|---|---|
| `Path-Test` | `0.1` | Penalty for symbols in test paths - test code is compressed more aggressively. |
| `Path-Generated` | `0.05` | Penalty for generated code paths ( `obj/` , `.Designer.cs` , `.g.cs` ). |

| Field | Default | Notes |
|---|---|---|
| `Path-Migration` | `0.3` | Penalty for migration/scaffolding paths. |
| `Dead-Private` | `0.1` | Multiplier for private methods with no callers. |
| `Single-Caller-Private` | `0.4` | Multiplier for private methods called from exactly one place. |

`Trimming Mode + Tiebreaker + User Hint`

| Field | Notes |
|---|---|
| `Trimming-Mode` | `Greedy` (methods are reduced one at a time until the budget is met) or `Sweep` (all methods are reduced in waves). Default: `Greedy`. |
| `UserHint-Boost-Multiplier` | Score boost for nodes mentioned in the goal text / user hint. Default `10`. |
| `Tiebreaker order` | The order in which ties are broken. The default order is `Score`, `Loc`, `Name`, `Fqn`. Use the arrow buttons to reorder; `Score` must stay first, so the buttons keep it in place. |

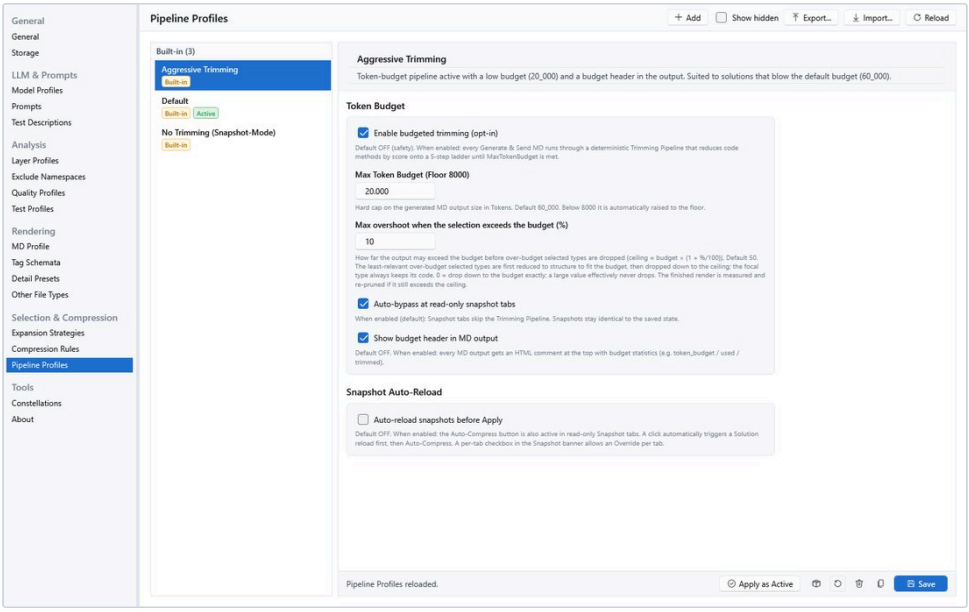
`Auto-Compression Heuristic (One-Click Auto-Compress)`

| Field | Default | Notes |
|---|---|---|
| `Enabled (master toggle: One-Click button active when on)` | on | When off, the tree toolbar's `Auto-compress` is disabled. The corresponding Insights `Apply` path belongs to an unregistered producer and is not visible in the shipped app. |
| `Caller-Count-Threshold for Full:` | `3` | A public method with fewer than this many callers stays compact, otherwise it is rendered at a higher level. Typical choices: `3` (balanced), `8` (aggressive), `1` (conservative). |
| `Force AI-Priority 'high' to Detailed` | on | High-priority nodes are always rendered detailed, regardless of caller count. |
| `Force AI-Priority 'low' to Compact` | on | Low-priority nodes are always rendered compact, even with many callers. |
| `Force EntryPoints (static Main + Controllers) to Detailed` | on | Detected entry points are always detailed - useful for code-tour exports. |

`Apply as Active` sets the profile as the active one for node scoring and budget trimming.

## 8.19 Pipeline Profiles

**What it is:** the token-budget trimming pipeline plus the snapshot auto-reload behavior. Exactly one profile is `Active`; a template can override it.




Settings > Pipeline Profiles

`Token Budget`

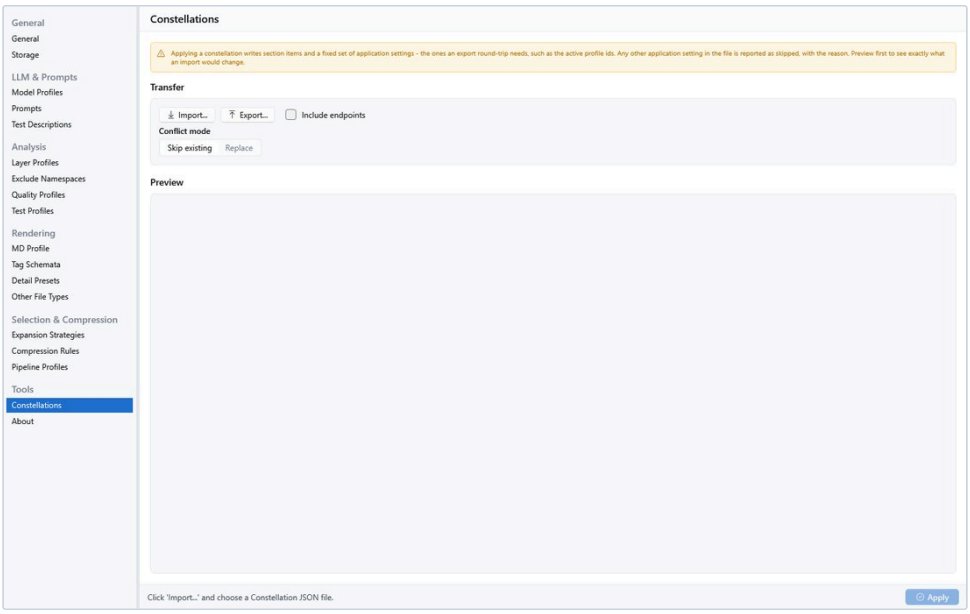
| Field | Default | Notes |
|---|---|---|
| `Enable budgeted trimming (opt-in)` | off | Master toggle. When on, every context document runs through a deterministic trimming pipeline before the LLM call, reducing code methods by score until the budget is met. |
| `Max Token Budget (Floor 8000)` | `60000` | Hard cap for the generated document size in tokens. Values below 8000 are raised to the floor. |
| `Max overshoot when the selection exceeds the budget (%)` | `50` | How far the output may exceed the budget before over-budget selected types are dropped. The ceiling is `budget × (1 + %/100)`. `0` means a hard cap (drop down to the budget exactly); a large value effectively never drops a selected type. Clamped to 0-1000. The least-relevant selected types are first reduced to structure, then dropped down to the ceiling; the focal type always keeps its code. The finished render is measured and pruned again if it still exceeds the ceiling. |
| `Auto-bypass at read-only snapshot tabs` | on | Snapshot tabs skip the trimming pipeline so they stay identical to the saved state. |
| `Show budget header in MD output` | off | Prefixes the output with an HTML comment containing budget statistics (token budget, estimated size after trimming, trimmed methods and, under an enforced budget, the types dropped or reduced to structure). |

`Snapshot Auto-Reload`

| Field | Default | Notes |
|---|---|---|
| `Auto-reload snapshots before Apply` | off | When on, the auto-compress button is also active in read-only snapshot tabs. A click automatically triggers a solution reload first, then auto-compress. A per-tab checkbox in the snapshot banner lets you override it for one tab. |

## 8.20 Constellations

**What it is:** a constellation bundles master entities plus selected application settings into **one portable JSON file**, and reads such a file back in. It is the way to move a complete configuration from one machine to another.




Settings > Constellations

The header tooltip: "A constellation bundles your configuration - prompts, templates, model profiles and selected application settings - into one portable JSON file for sharing or backup."

The page shows an informational banner: "Applying a constellation writes section items and a fixed set of application settings - the ones an export round-trip needs, such as the active profile ids. Any other application setting in the file is reported as skipped, with the reason. Preview first to see exactly what an import would change."

`Transfer`

| Control | Notes |
|---|---|
| `Import...` | Choose a constellation JSON file. Header validation and preview run automatically. |
| `Export...` | Writes the current settings plus your custom master entities as a constellation JSON file. Built-ins are deliberately **not** exported - they come from the app. |
| `Include endpoints` | Default **off**. When off, model profile endpoint URLs are redacted from the exported file, so it is safe to share. When on, endpoints are included. API keys are never exported - they live in the Windows Credential Manager. |
| `Conflict mode` | `Skip existing` (default): existing entries with the same ID stay unchanged, new entries are saved. `Replace`: existing entries with the same ID are overwritten. |

Below the buttons, the last file used by an import or export is shown ( `Last file: ...`, hover shows the full path). The `Preview` pane fills the rest of the page and shows exactly what applying the file would change.

The workflow is: `Import...` and choose a file → read the preview → `Apply`. The status line starts with "Click 'Import...' and choose a Constellation JSON file."

The preview names the bundle (name, description, minimum app version, export date), its origin and a trust label, then lists each section with its counts of new, conflicting, unchanged and skipped built-in entries, plus the application settings keys the file carries. A community-sourced file is labeled "Low trust - third-party source, inspect first!".

After `Apply`, the page reports `Applied: {n} Items.`, `Skipped: {n} Items.` and `AppSettings keys: {n} applied.`; when there are problems, a `Notes:` list is appended and an error dialog appears.

Supported entity types: `Prompt`, `MdProfile`, `TagSchema`, `DetailPreset`, `TestDescription`, `ModelProfile`, `ContextTemplate`, `ExpansionStrategy`, `RunTemplate`, `LayerMappingProfile`, `NamespaceExclusionList`, `CompressionRules`, `PipelineProfile`, `QualityProfile`, `McpProfile` and `TestProfile`. An unknown type is reported with that list.

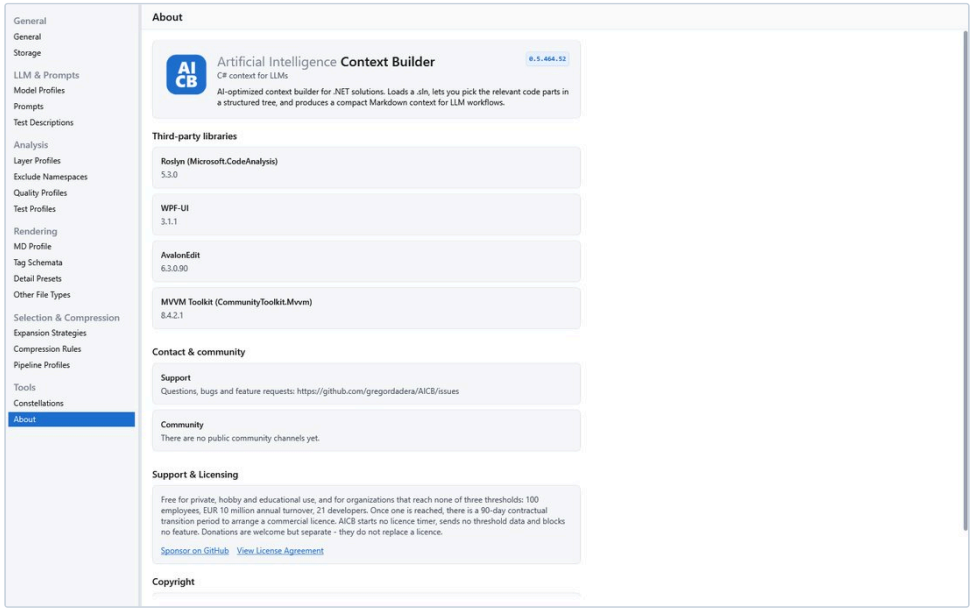
Two more rules are worth knowing:

- **Application settings are applied through a fixed whitelist.** A key outside the list is rejected, and the message names the allowed keys. The list covers the settings a round-trip needs - the active profile ids, the general settings, the default profile choices and the plain preferences. Settings that are tied to one machine (storage paths, the recent lists, splitter positions) are deliberately never imported.
- **Built-ins are always skipped on import.** They come from the app, not from the file. An import is fault-tolerant by design: an unusable application-settings section does not abort the entity import.

Note: if a model profile was exported with `Include endpoints` off, the endpoint is missing from the file. On re-import on the same machine it is restored from the existing profile with the same ID; on a different machine it stays empty - exactly the intended "share without endpoints" behavior.

## 8.21 About

**What it shows:** version, the bundled third-party libraries, contact information and the license.




Settings > About

- **The hero card** shows the product name, the current version, the tagline `C# context for LLMs` and a short description.
- `Third-party libraries` lists the key libraries this build ships with, each with the version of the assembly actually loaded at runtime (never maintained by hand): Roslyn ( `Microsoft.CodeAnalysis` ), WPF-UI, AvalonEdit and MVVM Toolkit ( `CommunityToolkit.Mvvm` ).
- `Contact & community` lists `Support` ("Questions, bugs and feature requests: https://github.com/gregordadera/aicb-roslyn-mcp/issues") and `Community` ("There are no public community channels yet.").
- `Support & Licensing` carries the license summary: "Free for private, hobby and educational use, and for organizations that reach none of three thresholds: 100 employees, EUR 10 million annual turnover, 21 developers. Once one is reached, there is a 90-day contractual transition period to arrange a commercial licence. AICB starts no licence timer, sends no threshold data and blocks no feature. Donations are welcome but separate - they do not replace a licence."
- `Sponsor on GitHub` opens the project's GitHub Sponsors page in your browser.
- `View License Agreement` opens the bundled full `EULA.md` from the distribution root in your default editor. A development build without that file falls back to the `LICENSE.txt` summary next to the executable.
- `Copyright` shows the copyright notice and product-name / third-party trademark attributions.

## 8.22 Configuring AICB for a team

This section is a practical order of work for setting up AICB on a team, from "nothing configured" to "shared configuration".

**Step 1 - Nothing to configure for the core workflow.** All analysis and rendering axes have built-in defaults that are already active. Open a solution and create a context document; you do not have to visit Settings at all.

**Step 2 - One model profile per machine.** For the LLM workflow, each user needs a model profile with an API key. The key is stored in the Windows Credential Manager and is tied to the Windows account, so it never travels in an export and every team member enters their own. If a key is missing, the app says where to add it: `API key for model '{Name}' is missing. Settings > Model Profiles.` The built-in model profiles can serve as starting points; use `Duplicate` to create your own entry and `Test Connection` to verify endpoint, key and model name before the first real run.

**Step 3 - Set the three per-solution axes where they belong.** The layer profile, the namespace-exclusion list and the test profile can each be chosen **per solution** in the Workspace's Solutions tab. The choice wins over the app-wide default from the `General` page; the resolution order is always per-solution → app-wide default → built-in default. Use the app-wide defaults for the common case and the per-solution override only for solutions that differ.

If you work headless or want the configuration in version control, write the solution configuration as a git-tracked sidecar file named `<SolutionName>.aicb.json` next to the `.sln` file. It carries rules rather than database IDs, so it is portable. The desktop app can restore its profile content; database-free headless analysis uses the layer/exclusion fallback and analysis scope, while test detection and suppressions follow their separately documented paths. Of the related MCP tools, `solution_config_status` only reads status and `init_solution_config` only gathers proposal material; `apply_solution_config` is the operation that writes the configuration database and sidecar.

Two sidecar settings have no wizard and are written by hand. The first is the analysis scope:

```
{ "analyzePreferredTfmOnly": true }
```

A multi-targeted project ( `<TargetFrameworks>net8.0;netstandard2.0</...>` ) is otherwise loaded once per target framework, so every source file is analyzed several times. With this key set, only the newest target framework's instance of each project is analyzed. The default is off, and on a solution without multi-targeting the setting does nothing. Note that it narrows the analysis: a fan-in edge whose only source is a non-preferred instance is lost, so "who uses this" queries can report a smaller blast radius.

The second is the platform at which every project is built when the solution is loaded, as `dotnet build -p:Platform=x64` would:

```
{ "designTimeBuildPlatform": "x64" }
```

Write the platform as the solution's configuration manager names it, for example `x64` , `ARM64` or `Any CPU` . Without the key, a solution that defines `Any CPU` is loaded with no platform set, so every project uses its own default. A solution that defines no `Any CPU` platform is loaded at the platform a build of the solution uses by default, where one platform can stand for that build for every project, and otherwise with no platform set. Set the key when AICB does not apply a platform, or applies one you do not want. A value the solution file does not name, or a solution platform under which its projects are built at more than one platform, is refused, not applied, and the load behaves as if the key were absent. The key is read every time a solution is opened or reloaded, in the desktop app, the CLI and the MCP server alike; in the MCP server, `refresh_session` with `force: true` reloads, and the CLI re-analyzes with `--force` where it would otherwise reuse a saved snapshot. The desktop app has no control for it and does not show which platform a load used: the MCP tool `get_diagnostics` reports it in its `designTimeBuildPlatform` field (omitted when no platform applies and no value was refused), and `aicb analyze` names a platform taken from the key before the analysis and warns about a refused one.

**Step 4 - Share a complete configuration with a constellation.** To move prompts, templates, profiles, schemata and the accompanying application settings to another machine or another team member, use the `Constellations` page: `Export...` on the source machine, then `Import...` and `Apply` on the target. Leave `Include endpoints` off for anything you share outside your network - the file is then safe to pass around, and API keys are never in it. Choose `Skip existing` when the target already has entries you do not want to overwrite, or `Replace` when the file should win. Read the preview before applying; it lists every section and every application setting the file would change. The same file can be applied headless:

```
aicb import --file <constellation.json> --db-path <target-db> --mode SkipExisting --preview
```

**Step 5 - Know what does not travel.** Storage settings are machine-local by design: the base path, the database path and the recent lists are never part of a constellation. The splitter positions and the first-run marker are also excluded. So after moving a configuration, each user still sets their own `Base path` (a change that takes effect after an app restart) and enters their own API keys.

**Step 6 - Protect the shared state.** The desktop app and the MCP server write into the same database. If your team uses both, enable `Enable auto-backup on app start` on each machine and set an interval that matches your working rhythm - the archive also contains the active database file even when it lives outside the base path. Remember that the backup runs at startup before any window appears, so a very large base path delays the start; and if a run fails, it is retried at every start until the cause is fixed.

# 9 MCP Profiles, MCP Usage and Templates

This chapter covers the three pages that belong to the MCP feature and to context templates: **MCP Profiles** (the graphical face of the MCP server), **MCP Usage** (the telemetry the server records about itself) and **Templates** (the context-template library). The MCP manual explains the server side of these concepts - what a profile, a facet, a bundle or a tool does at runtime; this chapter describes what you see and set in the app.

## 9.1 MCP Profiles

**Where:** main menu → `MCP` → `MCP Profiles`.

An MCP profile bundles a complete MCP working environment: the task facets (per facet a tool menu and a context template), the guidance blocks, the opt-in tool bundles, the output notation, the token budget, and how the server keeps its analysis in step with your edits. Exactly one profile is active and drives the MCP server; the active one carries the `Active` pill in the list.

The page follows the master/detail layout used throughout the app: the list on the left, the editor on the right. Header tooltip:

> `An MCP profile bundles the task facets (tool menus + context templates), the guidance blocks, the opt-in tool bundles, the output notation and the token budget of one MCP working environment.`

The list groups entries into `Built-in` and `Custom` (built-ins first, alphabetical within each group) and shows these pills:

| Pill | Meaning |
|---|---|
| `Built-in` | Shipped with the product. |
| `Active` | The profile that currently drives the MCP server. |
| `Overridden` | A built-in you edited and saved; the built-in seed leaves the row alone from then on. |
| `Hidden` | A built-in you deleted. Tick `Show hidden` in the header to see it again and restore it. |

Header fields:

| Field | Tooltip |
|---|---|
| `Name` | `The display name of this MCP profile, shown in the profile list and reported by list_mcp_profiles.` |
| Description | `A short description of this profile's purpose, shown under its name in the list.` |

Header actions: `Add` (creates a new custom profile), `Show hidden`, `Export...`, `Import...`, `Reload` (discards unsaved editor edits). Footer actions: `Apply as Active`, `Save`, `Duplicate`, `Delete`, `Restore Built-In`, `Export as Built-In`.

Keyboard shortcuts: `Ctrl+S` saves, `Ctrl+N` creates a new profile, `Ctrl+D` duplicates the selected one, `F5` reloads from the database.

### The four built-in profiles

| Name | What it is | Auto-refresh |
|---|---|---|
| `Default` | The everyday tool set across all nine task facets, with 18 long-tail tools withheld from the facet menus. Its shipped description says `24 core + 30 extras = 54 tools`. | `Reactive` |
| `Full Select` | The complete tool set: all nine facets enabled without any menu narrowing. Its shipped description says `24 core + 48 extras = 72 tools`. The infrastructure and DB-entity tools stay opt-in via `AICB_MCP_TOOLS`. | `Reactive` |
| `Refactoring Focus` | `Geared towards refactoring work: emphasizes structure, dependencies and change impact.` Enables the `Refactoring` facet - tool menu and working style. | `Reactive` |

| Name | What it is | Auto-refresh |
|---|---|---|
| `Debugging Focus` | `Geared towards debugging: emphasizes control flow, error handling and reproduction.` Enables the `Debugging` facet - tool menu and working style. | `Reactive` |

All four render every facet through the profile-wide `AI Optimized` context template in YAML notation and start with the auto-refresh mode `Reactive`.

The 18 tools that `Default` withholds from its menus stay registered. All nine facets are already enabled in `Default`; to restore one of the withheld tools, expand its facet and check the tool (marked with the `recommended` badge), or switch to `Full Select`.

Note: the tool counts in the two descriptions are literal text shipped with the profiles and age with the tool catalog. The reliable count for the profile you have selected is the `Effective tools/list` readout on the `Tools` tab; the reliable list of everything the server knows is the `list_skills` tool, which groups tools into `inPool` and `outOfPool`.

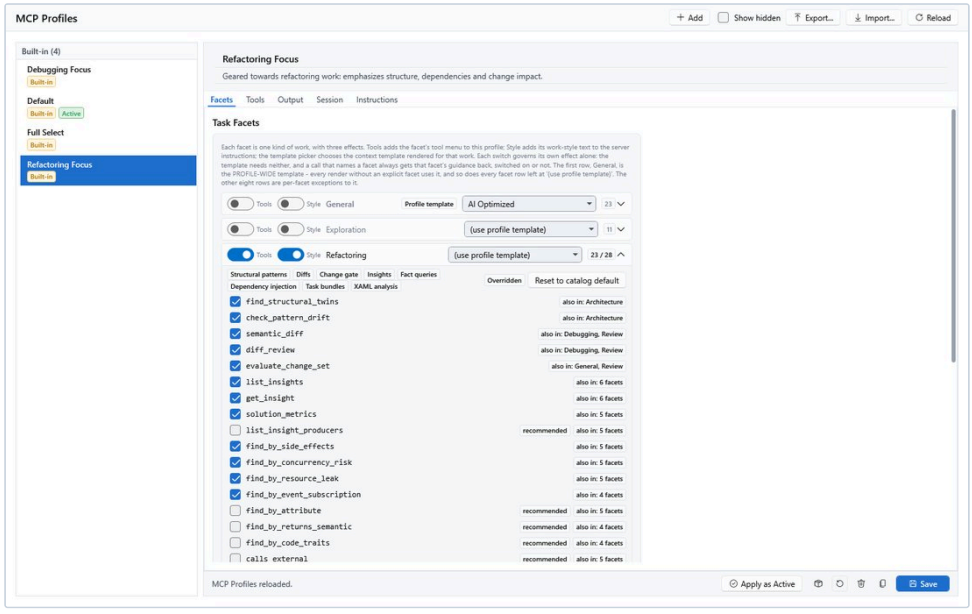
### The five editor tabs

| Tab | What it configures |
|---|---|
| `Facets` | The nine kinds of work: per facet its tool menu, its working style and its context template. |
| `Tools` | Everything outside the facet menus - the navigation core, the opt-in bundles - plus the effective tool list they add up to. |
| `Output` | Notation and token budget of this profile's MCP render. |
| `Session` | How the MCP server keeps its analysis current, and who pays for that. |
| `Instructions` | What the server sends the LLM on connect: guidance styles, free text, and a live preview of both. |

The editor remembers the tab you last had open; switching to another profile does not reset it.

### Facets tab - the nine kinds of work

Help text on the tab:




The Facets tab of the MCP profile editor

> `Each facet is one kind of work, with three effects. Tools adds the facet's tool menu to this profile; Style adds its work-style text to the server instructions; the template picker chooses the context template rendered for that work. Each switch governs its own effect alone: the template needs neither, and a call that names a facet always gets that facet's guidance back, switched on or not. The first row, General, is the PROFILE-WIDE template - every render without an explicit facet uses it, and so does every facet row left at '(use profile template)'. The other eight rows are per-facet exceptions to it.`

The nine facets, in catalog order:

| Facet | Covers |
|---|---|
| `General` | Balanced, task-neutral context work: navigate, read and verify. |
| `Exploration` | Understanding an unfamiliar codebase before changing it: start from the architecture and the public surface, then drill into symbols. |
| `Refactoring` | Structural clarity and dependency direction; minimal, behavior-preserving edits; coupling and cohesion smells. |
| `Debugging` | Tracing control and data flow to the fault, fixing the root cause, reasoning about reproduction. |
| `Review` | Reviewing a change end to end: blast radius, introduced findings, test coverage. |
| `Testing` | Focused, deterministic unit tests; finding gaps and pinning invariants and edge cases. |
| `Documentation` | Explaining intent over mechanics; dense and accurate summaries against the real API surface. |
| `Architecture` | Clean/Onion layering: inward dependencies, layer violations, cycles and coupling. |
| `Performance` | Hot paths and complexity hotspots; avoid unnecessary allocations; measure before optimizing. |

Each row has three independent controls:

| Control | Effect |
|---|---|
| `Tools` switch | Adds this facet's tool menu to the profile's `tools/list`. Unfold the row to fine-tune which of its tools are included. |
| `Style` switch | Adds this facet's work-style text to what the server sends on connect, so it applies to every call. |
| Template picker | Chooses the context template rendered for this facet (for example `export_markdown` with facet `"refactoring"`). |

The two switches are deliberately separate: a user who wants only the working style should not have to widen `tools/list` as well. And the `Style` switch only decides the *standing* instruction sent at connect - a call that names a facet always gets that facet's guidance back, switched on or not.

**The `General` row is special.** Its template picker is the **profile-wide** context template: every render without an explicit facet uses it, and the MCP server also resolves the QualityProfile and PipelineProfile from it. The row is shown in bold with a `Profile template` pill. Because of this, the two empty values in the template pickers are labeled differently:

| Row | Empty value | Falls back to |
|---|---|---|
| `General` | `(use active context template)` | The globally active context template. |
| Any other facet | `(use profile template)` | The General row's profile-wide template. |

The counter in the row header reads `checked / available` while the facet is switched on, so you can see whether the menu has been tuned away from the catalog default. While the facet is switched off, the header shows only the menu size - an off facet contributes none of its tools, and `14 / 14` would claim tools the server does not serve. The counter is muted whenever the facet contributes no tools (because it is off or no menu item is checked), regardless of the `Style` switch. Only the row name waits for the stricter condition that the facet contributes nothing at all: neither tools nor working style.

Unfolding a row reveals:

- Chips naming the tool groups that feed this facet's menu; the `General` row shows `Curated menu (no tool classes)` because its menu is a hand-picked list.
- The `Overridden` pill and the `Reset to catalog default` button once the selection deviates from the catalog default. The button checks every tool on the menu again.
- One checkbox per tool function, with the tool's description as its tooltip. Each row can additionally carry:
- `recommended` - this tool is on the facet's catalog menu but currently deselected.
- `not in the menu` - this tool is not on the catalog menu; the stored selection brought it along, and `Reset to catalog default` removes it.
- `also in: ...` - the other facets whose menu also carries the tool. Each facet keeps its own selection; the server exposes the union of the enabled facets' menus.

When a change takes effect - tooltip:

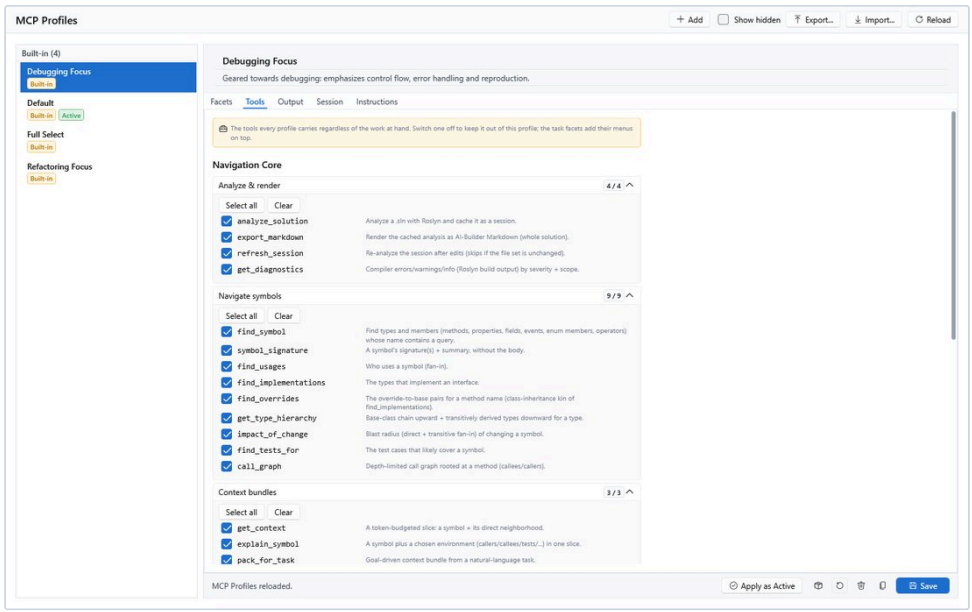
> `Tool set and instructions are applied at server start - a change needs a server restart. The template assignment applies live per call.`

**To trim a facet's menu step by step:**

1. Select the profile in the list.
2. Open the `Facets` tab.
3. Unfold the facet row and clear the checkboxes you do not want.
4. Click `Save` in the footer.
5. Restart the MCP server (reconnect your client).

### Tools tab - navigation core, bundles and the result

`Navigation Core` - the tools every profile carries regardless of the work at hand:




The Tools tab: navigation core, opt-in bundles and the effective tools/list

> `The tools every profile carries regardless of the work at hand. Switch one off to keep it out of this profile; the task facets add their menus on top.`

| Group | Tools |
|---|---|
| Analyze & render | `analyze_solution`, `export_markdown`, `refresh_session`, `get_diagnostics` |
| Navigate symbols | `find_symbol`, `symbol_signature`, `find_usages`, `find_implementations`, `find_overrides`, `get_type_hierarchy`, `impact_of_change`, `find_tests_for`, `call_graph` |
| Context bundles | `get_context`, `explain_symbol`, `pack_for_task` |
| Service & discovery | `server_info`, `usage_report`, `batch`, `measure`, `list_mcp_profiles`, `list_skills`, `docs`, `install_agent_hooks` |

Each group header shows how many of its tools this profile exposes and stays readable when folded (the counter turns to the warning color if something in the group is switched off). `Select all` exposes every tool in the group again; `Clear` switches off every tool in the group that may be switched off. Unchecking a tool denies it for this profile.

`server_info` cannot be switched off and shows a lock instead of a checkbox - it is the tool that reports server state and configuration drift, so a profile that went wrong stays diagnosable from the client.

Facet menus never repeat a navigation-core tool, so the facet counters and these counters never overlap.

**Sparse-selection warning.** When the profile would expose five or fewer tools, a warning banner appears:

> `This profile would expose only N tools. An agent connecting to it has almost nothing to work with: switch core tools back on, or enable a task facet.`

The warning is counted on the *effective* tool list, not on how many checkboxes are cleared - an enabled facet adds tools back. It is a judgment, not a hard limit: there is no tool count at which a profile provably stops working, so the warning names the actual number and leaves the verdict to you.

`Opt-in Tool Bundles` - project-type extras outside the task facets:

> `Project-type extras outside the task facets. Enabling a bundle adds its tools on top of the facet menus.`

Each bundle has a switch and a pill naming the number of tools it adds. The current catalog contains one opt-in bundle: `Public API compatibility`, which adds `compare_public_api` (breaking public-API changes between two built assemblies). Bundles are applied at server start; if the `AICB_MCP_TOOLS` environment variable is set, it still overrides the whole profile composition.

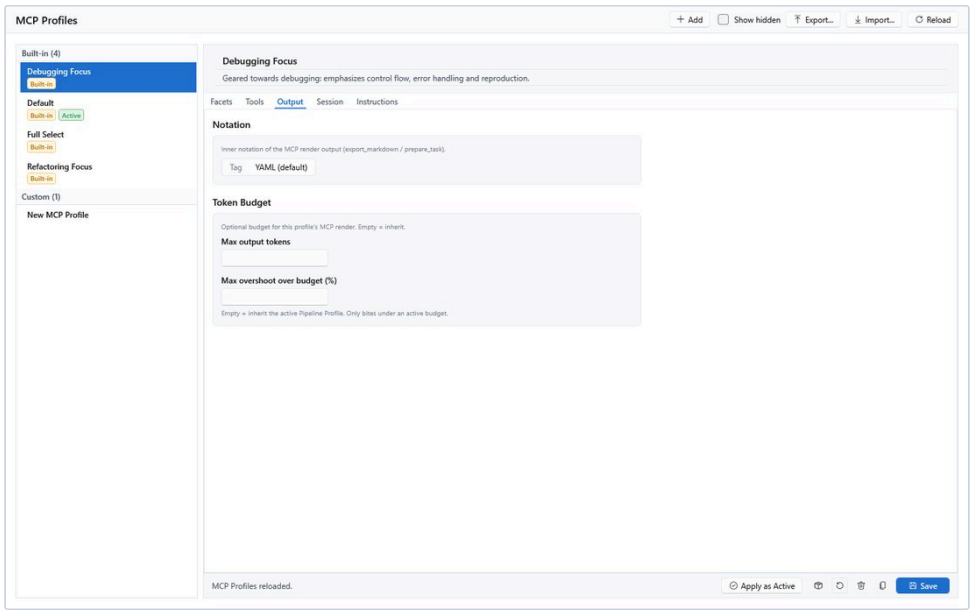
`Effective tools/list` - a read-only preview:

> `What this profile exposes to the MCP server on connect. Read-only.`

It shows a summary ( `N tools` ) and the exact tool functions the server will list for this profile, one per line. The preview is computed from the navigation core minus the tools you switched off, plus the union of the enabled facets' menus, plus any enabled bundle tools. Note: it cannot show the effect of `AICB_MCP_TOOLS`, because that variable belongs to the server process and is not readable here.

### Output tab - notation and budget

`Notation` - the inner notation of the MCP render output ( `export_markdown`, `prepare_task` ):




The Output tab

| Option | Meaning |
|---|---|
| `Tag` | `Render the established AI-Builder tag markdown (<CLASS>...</CLASS>). Byte-identical to the previous output.` |
| `YAML (default)` | `Render the content as idiomatic YAML. Lossless - only the notation differs.` |

Both are two ways of writing the same content; pick whichever the model or tool that reads the file handles better. An explicit `export_markdown(format=...)` call parameter wins over this profile default. The `.md` file name is unchanged.

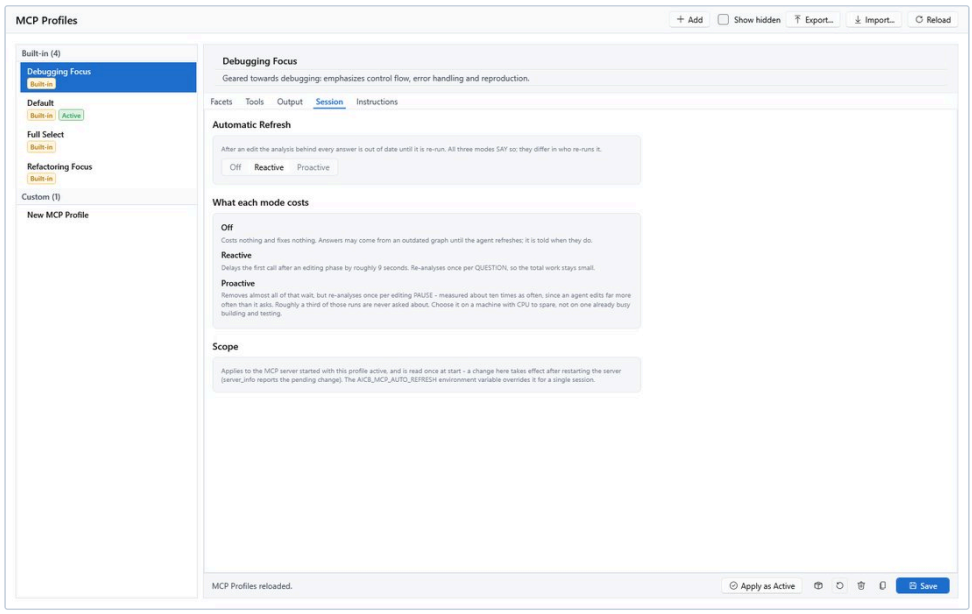
`Token Budget` - an optional budget for this profile's MCP render:

| Setting | Behavior |
|---|---|
| `Max output tokens` | Maximum output tokens for this profile's MCP render. Empty = inherit (the context template's budget, then the global one). Floored at 8000. A set value wins over the facet template's budget and forces trimming. Precedence: explicit call parameter > MCP profile > context template > global. |
| `Max overshoot over budget (%)` | How far the template render may exceed the budget before over-budget selected types are dropped, as a percentage of the budget (ceiling = budget × (1 + percent/100)). Empty = inherit the active Pipeline Profile's percent. `0` = a hard cap; a large value effectively never drops a selected type. Clamped to 0-1000. |

The overshoot setting applies to the `export_markdown` / `prepare_task` template render; the lean slice tools ( `get_context` , `pack_for_task` , `explain_symbol` ) keep their transport cap at the budget. Both settings affect the MCP render path only - GUI and CLI exports are unaffected. The panel adds: `Empty = inherit the active Pipeline Profile. Only bites under an active budget.`

### Session tab - how fresh the facts are

> `After an edit the analysis behind every answer is out of date until it is re-run. All three modes SAY so; they differ in who re-runs it.`




The Session tab with the three automatic-refresh modes

| Mode | Behavior | Cost |
|---|---|---|
| `Off` | `No automatic re-analysis. A drifted session is reported and left alone; the agent repairs it by calling refresh_session itself.` | `Costs nothing and fixes nothing. Answers may come from an outdated graph until the agent refreshes; it is told when they do.` |
| `Reactive` | `Repair when asked: the drift is noticed as a tool is about to answer, the re-analysis runs first, and the answer comes from the current graph. The caller waits for it - about 9 seconds on a solution this size.` | `Delays the first call after an editing phase by roughly 9 seconds. Re-analyses once per QUESTION, so the total work stays small.` |
| `Proactive` | `Repair ahead of the question: a file watcher starts the re-analysis once saving has gone quiet, so the next call usually finds a current graph and waits for nothing. The work does not disappear - it moves into the background, and there is measurably more of it.` | `Removes almost all of that wait, but re-analyses once per editing PAUSE - measured about ten times as often, since an agent edits far more often than it asks. Roughly a third of those runs are never asked about. Choose it on a machine with CPU to spare, not on one already busy building and testing.` |

`Reactive` is the shipped default: all four built-in profiles carry it, and every newly created profile starts with it. `Off` is what an absent value means - a profile that was written by a very old version or imported from one.

The panel lists all three costs at once on purpose: the choice is a trade-off between caller latency and background CPU, and you can only make it by comparing.

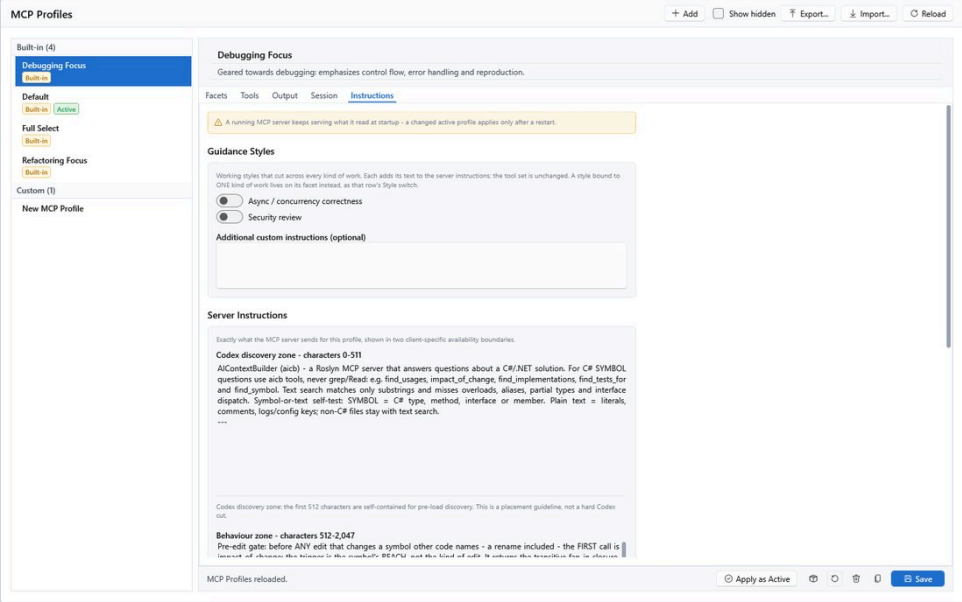
`Scope`:

> `Applies to the MCP server started with this profile active, and is read once at start - a change here takes effect after restarting the server (server_info reports the pending change). The AICB_MCP_AUTO_REFRESH environment variable overrides it for a single session.`

Accepted values of `AICB_MCP_AUTO_REFRESH`: `off` (also `0`, `false`, `no`), `reactive`, and `proactive` (also `1`, `on`, `true`, `yes`). An unrecognized value is ignored, so a typo cannot switch a mode on.

### Instructions tab - what the server sends on connect

The tab carries a warning banner:




The Instructions tab with the preview of the server instructions

> `A running MCP server keeps serving what it read at startup - a changed active profile applies only after a restart.`

`Guidance Styles` - working styles that cut across every kind of work:

> `Working styles that cut across every kind of work. Each adds its text to the server instructions; the tool set is unchanged. A style bound to ONE kind of work lives on its facet instead, as that row's Style switch.`

The two cross-cutting styles are `Async / concurrency correctness` and `Security review`. Hover a row to read its exact text. A style is guidance only: it never adds or removes tools - the `Facets` and `Tools` tabs do that. The six task-bound styles live on their facet rows, beside the tool menu and template of the same kind of work.

`Additional custom instructions (optional)` - a multi-line free-text field appended after the facet guidance and the checked styles. Write what they do not already say: a guidance text and a re-worded copy of it both reach the LLM, because only texts that match exactly are collapsed.

`Server Instructions` - a read-only preview of exactly what the MCP server sends for this profile, recomputed live as you change facets, styles, bundles or the free text:

> `Exactly what the MCP server sends for this profile, shown in two client-specific availability boundaries.`

The preview is split into three zones because different clients expose different amounts of it:

| Zone heading | Characters | Meaning |
|---|---|---|
| `Codex discovery zone - characters 0-511` | 0-511 | Self-contained identity and the symbol-vs-text rule for Codex before its deferred tool loading. A placement guideline, not a hard Codex cut; Codex receives the complete text after loading the tools. |

| Zone heading | Characters | Meaning |
|---|---|---|
| `Behaviour zone - characters 512-2,047` | 512-2,047 | Behavior-changing rules and the profile's working style inside the prefix Claude Code was measured to expose in prompt assembly. |
| `Post-prefix reference zone - from character 2,048` | from 2,048 | Reference and discovery material outside that measured prefix. It remains transported by the server and is available to Codex after tool loading. |

Where the text crosses the line, the panel draws a real boundary with this explanation:

> `Claude Code was measured to expose exactly the first 2,048 characters in prompt assembly. Text after this line remains available to Codex after its deferred MCP tool load; this is not a universal MCP-client limit.`

Below the zones, `Working style - this profile (facet guidance + styles + free text)` shows your profile's own contribution on its own: the facets whose `Style` switch is on, then the checked guidance styles, then the free text.

Two automatic notes belong to this tab:

1. **Is the working style being cut?** If part of the profile's working style falls below the line, the panel says so: `Part of this profile's working style falls below the line and is dropped by such a client. Exactly one guidance switch is guaranteed to arrive whole; beyond that it depends on their combined length, which the cut mark above shows.` Which blocks arrive is decided by composition order: facets first (in catalog order), then the cross-cutting styles, then the free text.
2. **Does the text name tools this profile does not offer?** Always shown, including the zero case: `Of the N tools named in the text above, this profile exposes M.` If names are withheld, the panel adds: `Not exposed here: <names>. An agent that follows these rules calls them and is refused, without being able to tell an unavailable tool from a wrong one.` Not necessarily a defect: a focus profile drops tools on purpose, and the rules describe the server in general. A block whose tools are *all* withheld is omitted from the sent text; a block backed by some exposed tools keeps its unreachable names.

The preview cannot mirror one thing: `AICB_MCP_TOOLS`. That is an override of the server process, which the GUI cannot read.

### Saving, activating and restoring

| Action | What it does |
|---|---|
| `Save` (`Ctrl+S`) | Persists the editor changes to the database. Editing a built-in sets its `Overridden` flag. |
| `Duplicate` (`Ctrl+D`) | Creates a custom copy of the selected profile (`<Name> (Copy)`) as a starting point for tweaks. |
| `Delete` | Built-ins are soft-hidden and recoverable via `Show hidden` + `Restore Built-In`; custom profiles are removed permanently. |
| `Restore Built-In` | Resets a built-in to its code defaults and clears the override and hidden flags. |
| `Export as Built-In` | Shows the entry as a C# snippet. Intended for the product's own development; it has no effect on your installation. |
| `Apply as Active` | Makes this profile the active MCP profile. The MCP server (render templates, guidance blocks, tool set) is driven by the active profile. A running server picks up the new tool set and instructions only after a restart. |
| `Add` (`Ctrl+N`) | Creates a new custom profile named `New MCP Profile`. |
| `Export...` / `Import...` | Writes or reads a JSON master-data file. Import skips built-ins contained in the file and asks how to resolve conflicts; the status line summarizes it as `Imported: n new, n replaced, n skipped.` |
| `Reload` (`F5`) | Discards unsaved edits and reloads all profiles from the database. |

Note: saving a built-in profile freezes its current facet menus into the stored row. For the `Default` profile this means the tool names it has today are pinned - a tool cataloged into a facet later no longer joins it by itself. If you want the `Default` profile to keep following the catalog, leave it untouched (or use `Restore Built-In`). Saving also marks the row `Overridden`, and the built-in seed then leaves it alone.

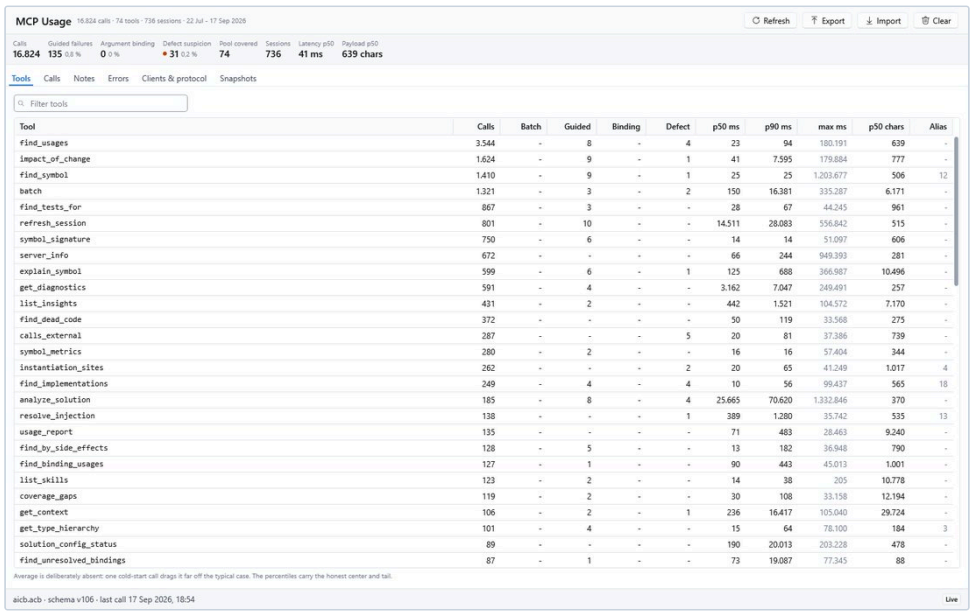
## 9.2 MCP Usage

**Where:** main menu → `MCP` → `MCP Usage`.

The page shows the telemetry the MCP server records for every handled call, read from the same database the app uses. It is not an editor: it is an evaluation view with a note function, export/import, and a controlled deletion path. The panel reloads whenever the tab is activated - the MCP server writes into the same database while the app runs, so a re-activated tab would otherwise keep showing the numbers from its first open.

If nothing has been recorded yet, the page shows:

> `No tool calls recorded yet. The MCP server records every handled call into this database; connect a client and this page fills up.`




MCP Usage page with the statistics strip and the Tools tab

### Header actions and statistics

| Action | Tooltip |
|---|---|
| `Refresh` ( `F5` ) | `Reload the usage data from the database (F5).` |
| `Export` | `Write the current usage report as JSON. The file carries exactly what the usage_report MCP tool returns, so it can be imported on another machine.` |
| `Import` | `Import an exported usage report as a named snapshot beside the live data. Imported sets are never merged into the live numbers.` |
| `Clear` | `Delete recorded calls (all, or older than a date). An export is saved first, and the deletion asks for a destructive confirmation.` |

Below the header, a statistics strip aggregates the **live** table:

| Tile | Meaning |
|---|---|
| `Calls` | Total recorded calls. |
| `Guided failures` | Calls that failed with `McpException`: the tool refused on purpose (an expired session, an unknown token). Not a defect. Shows a count and a share of all calls. |
| `Argument binding` | Calls the SDK's argument binding rejected because the caller named an argument the tool does not have. The call never reached the tool, so it is neither a defect nor a refusal. A count here means the argument surface confused somebody. |
| `Defect suspicion` | Calls that failed with any other exception type - a defect suspicion worth chasing. The `Errors` region breaks it down. |
| `Pool covered` | Tools of the active profile's pool with at least one recorded call, out of the pool's size (shown as a muted `of N`). Deliberately not the same figure as the header's tool count, which counts every tool the recording ever saw. Without an active profile the pool is unknown and the figure stands alone as the plain count of tools that were called. |
| `Sessions` | Number of distinct sessions that recorded calls. |
| `Latency p50` | Median latency over every call that recorded one. The median, not the average: one cold-start call drags an average far off the typical case. |
| `Payload p50` | Median result size over every call that recorded one. |

The strip is shown only while the live table holds at least one call. On a database that carries only an imported snapshot, every tile would aggregate an empty table, so the strip is hidden instead.

At the bottom, the status line names the database file, its schema version and the last recorded call (for example `aicb.acb · schema v105 · last call 30 Jul 2026, 18:41`); the full database path is its tooltip. A `Live` pill marks that the regions always show the live database - imported snapshots sit beside it in the `Snapshots` region.

### Tools tab

`Filter tools` filters the table by tool name (substring match); `Esc` clears the field. The table is sortable and lists every recorded tool, ranked by call count:

| Column | Meaning |
|---|---|
| `Tool` | The tool function name. Rows with no recorded call are muted. |
| `Calls` | How often the tool was called **directly**. Sub-queries inside a `batch` are not in here - they are the `Batch` column. A muted row ran neither way. |
| `Batch` | How often the tool was **dispatched as a sub-query inside a batch**. Those calls record no row of their own, so before this column a tool used only through `batch` read as never called. Read it as attempts, not runs: a sub-query the active profile refused, or one that failed on its arguments, is counted here too. `measure` is excluded - it runs the tool fully, but the caller consumes the size of the answer rather than the answer. |
| `Guided` | Failures the tool raised on purpose (`McpException`): an expired session, an unknown token. Not defects. |
| `Binding` | Calls rejected by argument binding: the caller named an argument this tool does not have, so the call never reached it. Counted out of `Defect`. |
| `Defect` | Failures with any other exception type: defect suspicions. The `Errors` region names each type. |
| `p50 ms`, `p90 ms` | Median and 90th-percentile latency. |
| `max ms` | The single worst call, usually a cold start. Muted on purpose: it is an outlier, not the typical case. |
| `p50 chars` | Median result size. |
| `Alias` | Calls that named a parameter by its accepted alias instead of its real name: how often the server silently corrected the caller. |

Columns with nothing to report show a dash rather than a `0`: `Guided`, `Binding`, `Defect` and `Alias` are blank when nothing happened. `Binding` is subtracted out of `Defect` - an argument-binding failure is the caller's mistake and must not keep suggesting the tool might be broken.

Below the table the panel states: `Average is deliberately absent: one cold-start call drags it far off the typical case. The percentiles carry the honest center and tail.`

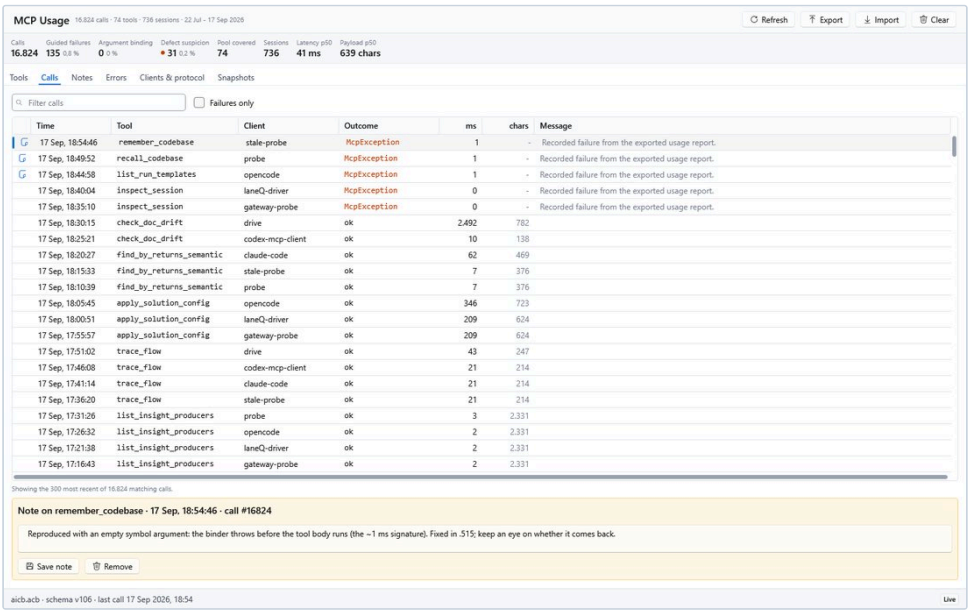
A sentence below the table names the rows of the active profile's pool that have no recorded call, for example `3 of the 54 tools in the active profile's pool have no recorded call. Their rows carry 0 calls.` It can also name tools that were called but are not in today's pool. The sentence is not shown when no active profile resolves, when every tool in the pool has been called, or while a filter is active (it counts the whole pool, and would otherwise stand under a filtered table).

Note: the muted styling of an unused row and that sentence use slightly different evidence. A row counts as unused only when both its direct and its `Batch` count are zero. The sentence, however, counts direct calls only - so a tool that was used exclusively through `batch` can appear in that list without being muted, and it shows its `Batch` count. The two sources disagree deliberately; read the `Batch` column before treating such a name as unused.

The ranked part of the table is capped at the 200 most-used tools (the same cap the `usage_report` tool accepts). Tools beyond that appear only in the pool sentence or in the exported report.

### Calls tab

This is the raw protocol: the individual recorded calls, newest first. It is the drill-down under the aggregates and the only place a note can be attached.




The Calls tab

- `Filter calls` filters by tool name (substring match); the filter is applied in the database, not on the visible slice, so you always see the newest matches of the whole table. `Esc` clears the field.
- `Failures only` restricts the log to calls that did not succeed: a thrown exception, or a result the tool itself flagged as an error.
- `Back to recent` appears only when a note-driven focus is active and drops it again, showing the most recent calls.

Columns:

| Column | Meaning |
|---|---|
| (marker) | A note is attached to this call. |
| `Time` | Local time, with seconds ( `30 Jul, 18:41:07` ). |

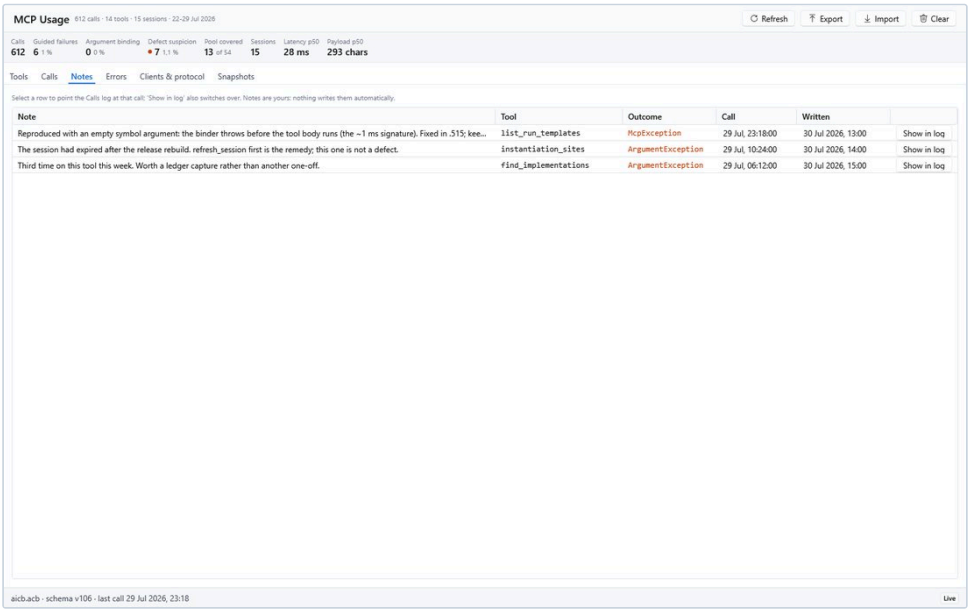
| Column | Meaning |
|---|---|
| `Tool` | The tool function name. |
| `Client` | The client that made the call, as it identified itself at connect time (name and version). A dash means the call was recorded before the server could tell. |
| `Outcome` | `ok`, or the exception type of a failure (shown in the error color). |
| `ms` | Duration, where recorded. |
| `chars` | Result size, where recorded. |
| `Message` | The failing exception's message, capped at 500 characters. Blank means either a successful call or one recorded before the server stored messages; the `Errors` region's `In version` column tells the two apart. |

If the log is longer than the window, a note says so, for example `Showing the 300 most recent of 1,029 matching calls.` The log always shows the 300 most recent matching calls. If no call matches the filter, the panel says `No recorded call matches this filter.`

Below the table is the note editor. Without a selection it reads: `Select a call above to write a note on it. Notes live beside the telemetry and are deleted with the call they belong to.` With a call selected, the heading names it in words - `Note on {tool} · {time} · call #{id}` - and you can write free text and use `Save note` or `Remove`. Saving again replaces the text and keeps the original date; removing deletes the note but keeps the call.

### Notes tab

`Select a row to point the Calls log at that call; 'Show in log' also switches over. Notes are yours: nothing writes them automatically.`




The Notes tab

| Column | Meaning |
|---|---|
| `Note` | Your text. |
| `Tool` | The tool of the annotated call. |
| `Outcome` | `ok` or the failure type of that call. |
| `Call` | The call's time. |

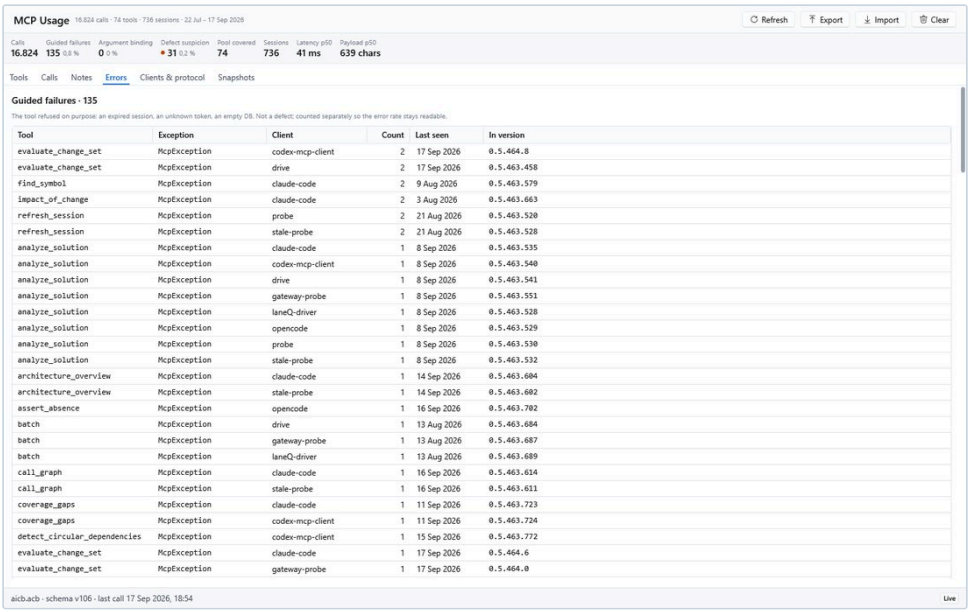
| Column | Meaning |
|---|---|
| `Written` | When you wrote the note. |
| `Show in log` | Switches to the `Calls` region with this call selected. |

Selecting a row already points the call log at that call (and loads the call in if the window does not reach it), but deliberately does not switch regions - otherwise the note list would not be browsable. If the call had to be loaded on its own, the log says so: `The selected note's call is not part of this view (older than the window, or filtered out); it was loaded in on its own.` `Show in log` also clears the log's filters, because a call excluded by a filter would otherwise not be visible.

If no notes exist: `No notes yet. Pick a call in the Calls region and write one; it stays attached to that call and shows up here.`

### Errors tab

Errors are never shown as one bare rate. The tab splits them into three groups, each with its own table:




The Errors tab

| Group | Explanation |
|---|---|
| `Guided failures` | `The tool refused on purpose: an expired session, an unknown token, an empty DB. Not a defect; counted separately so the error rate stays readable.` |
| `Argument-binding failures` | `The caller named an argument this tool does not have, so the call never reached it. That is neither a defect nor the tool refusing. Retrying unchanged fails identically; the error names the tool's real arguments. A count here means the argument surface confused somebody.` |
| `Defect suspicion` | `Any other exception type: a defect suspicion worth chasing. 'In version' names the server version of the LAST occurrence, so after a fix you can read off whether it still happens.` |

Each group's heading carries the total count, for example `Guided failures · 12`. Every table has the same columns:

| Column | Meaning |
|---|---|
| `Tool` | The tool function name. |
| `Exception` | The exception type. |

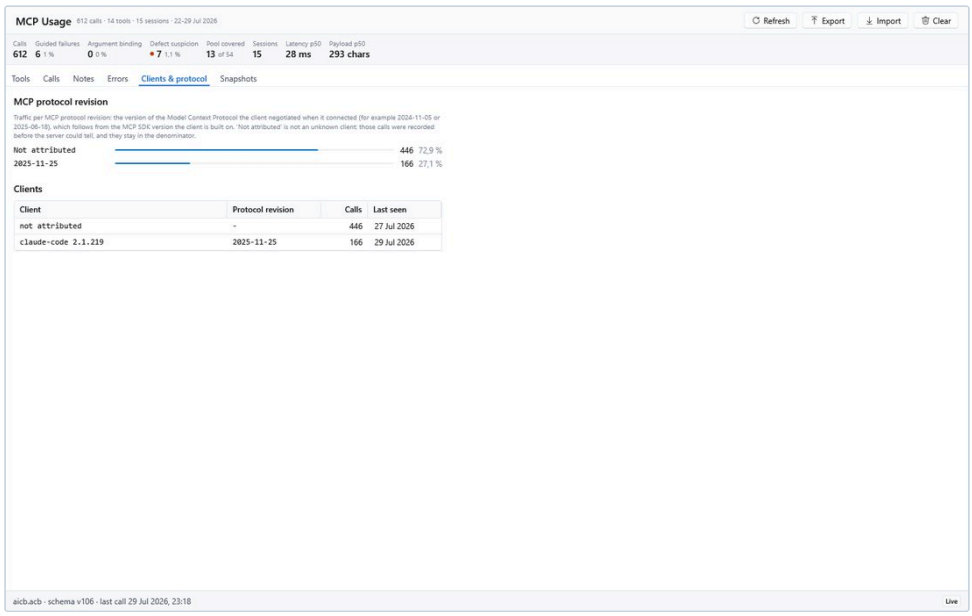
| Column | Meaning |
| --- | --- |
| `Client` | The client the errors came from, as it identified itself at connect time (name and version). A dash means the calls were recorded before the server could tell. |
| `Count` | How many times this combination occurred. |
| `Last seen` | Date of the last occurrence. |
| `In version` | The server version of the last occurrence. |

The client is part of the grouping: the same error from two clients is two rows.

Empty groups say so instead of showing an empty table: `No argument-binding failures recorded.` / `No defect suspicions recorded.` A corpus without defect suspicion is a good state, not a missing feature.

### Clients & protocol tab

The upper section `MCP protocol revision` shows a bar and a legend per protocol revision:



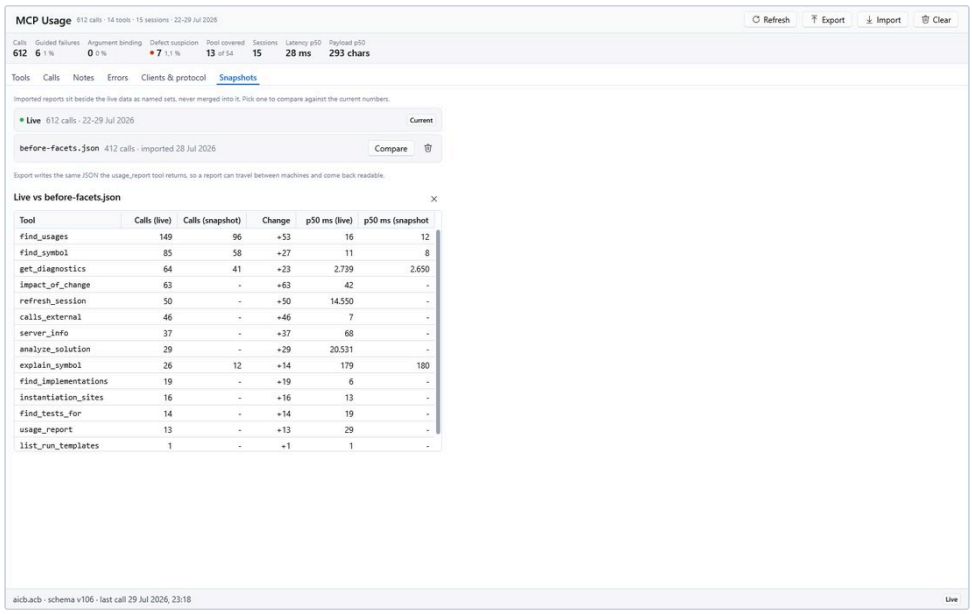

The Clients & protocol tab

> `Traffic per MCP protocol revision: the version of the Model Context Protocol the client negotiated when it connected (for example 2024-11-05 or 2025-06-18), which follows from the MCP SDK version the client is built on. 'Not attributed' is not an unknown client: those calls were recorded before the server could tell, and they stay in the denominator.`

The lower section `Clients` lists `Client`, `Protocol revision`, `Calls` and `Last seen`. A client that did not identify itself appears as `not attributed`. `Last seen` sorts by the raw timestamp, because the displayed text does not sort chronologically. The table shows up to 50 client-and-revision groups; if there are more, a note says how many of how many are shown.

### Snapshots tab

> `Imported reports sit beside the live data as named sets, never merged into it. Pick one to compare against the current numbers.`




The Snapshots tab with an imported usage report

The first card is the live table, marked with the `Current` pill and a green dot; its summary names the number of calls and the recording period. Each imported snapshot is a card with its file name, its summary ( `N calls · imported {date}` ), a `Compare` button and a delete button. The delete confirmation says: `Remove the imported snapshot '{name}'? The source file is not touched, but this stored set cannot be restored without re-importing it.`

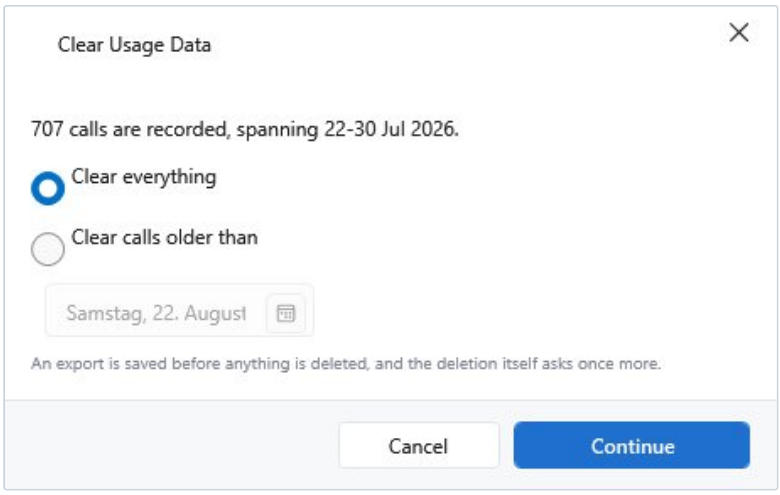
If nothing has been imported: `No imported reports yet. Import an exported usage-report JSON to keep it here as a named set.` The panel also states: `Export writes the same JSON the usage_report tool returns, so a report can travel between machines and come back readable.`

`Compare` opens a table titled `Live vs {snapshot name}` with the columns `Tool` , `Calls (live)` , `Calls (snapshot)` , `Change` , `p50 ms (live)` and `p50 ms (snapshot)` . A cross in the title closes the comparison again.

On a database whose live table is empty but which holds an imported snapshot, the page opens on the `Snapshots` region the first time, so you see the content that exists.

### Exporting, importing and clearing usage data

**Export** writes a fresh aggregate (not the cached view) with exactly the same content the MCP tool `usage_report` returns - one shared serializer for both sides. The save dialog is titled `Export Usage Report` , filters on `Usage report (*.json)|*.json|All files (*.*)|*.*` and suggests `usage-report-{yyyy-MM-dd}.json` . The file is written indented: same fields, same names, but a person opens this one. The `poolCoverage` block is included when an active profile resolves; otherwise it is omitted, exactly as on the server. The status line then shows `Exported to {file}` .




The Clear Usage Data dialog

**Import** opens a file, validates it and stores it as a named usage snapshot (the name is the file name), then switches to the `Snapshots` region and reloads. The status line shows `Imported {name}`. Two rejection reasons exist:

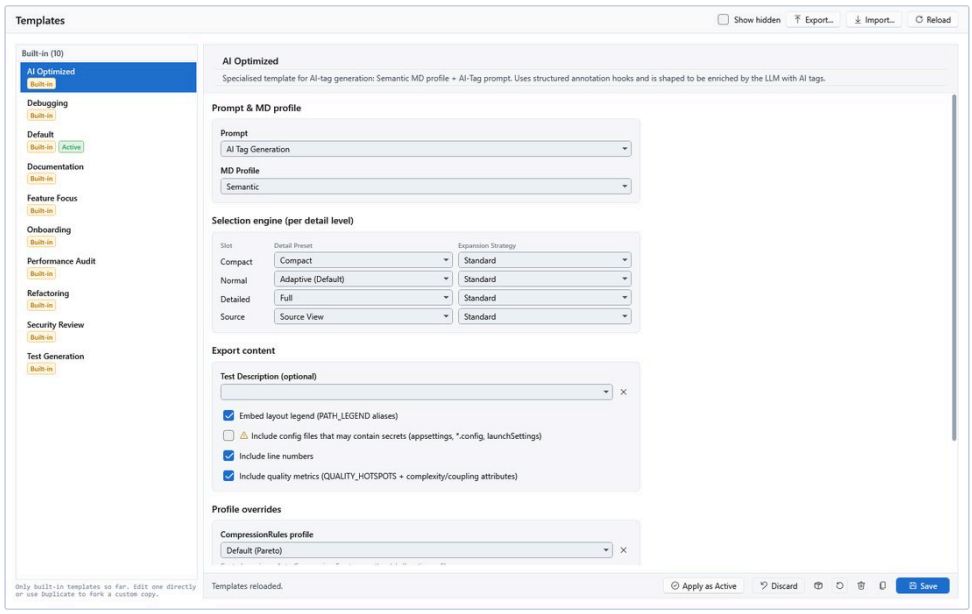
- `This file is not valid JSON.`
- `This file is not an exported usage report (expected the JSON the usage_report tool returns, with available=true and totals).`

**Clear** is the destructive path and runs in a fixed order:

1. A scope dialog titled `Clear Usage Data` shows how many calls are recorded and over which period, and offers `Clear everything` or `Clear calls older than` with a date picker (pre-filled with 30 days ago). `Clear calls older than` deletes calls recorded strictly before the chosen day; that day itself and everything after it stays. If nothing is recorded: `There are no recorded calls to clear.`
2. If the chosen scope matches nothing: `No recorded call is older than the chosen date.`
3. An export is forced. If you cancel the file dialog, the whole operation is cancelled: `Clear cancelled: no export target was chosen.` If the export cannot be written: `The export could not be written, so nothing was cleared: ...`
4. A destructive confirmation names the number of calls, the file - and the notes: `{n} notes written on those calls will be deleted with them, and the export does not contain them.` This warning matters: notes hang off the calls and are deleted with them, while the forced export contains only the aggregate. The notes are the one thing this operation destroys irrecoverably.
5. The calls are deleted and the status line shows `Cleared {n} calls (exported to {file} first)`.

## 9.3 Templates

**Where:** main menu → `Configuration` → `Templates`.




The Templates page

A context template is a reference container: it does not contain content itself, but selects the master data a context render uses. The page follows the master/detail layout; the list groups built-in and custom templates, and the editor on the right shows the selected template's selections, filled from the respective master-data lists.

The header offers `Reload`, `Import...`, `Export...` and `Show hidden`. The action bar offers `Apply as Active`, `Save`, `Duplicate`, `Delete`, `Restore Built-In`, `Export as Built-In` and `Discard`. Keyboard shortcuts: `Ctrl+S` saves, `Ctrl+D` duplicates, `Ctrl+N` creates a new, empty template, `F5` reloads.

Note: the header deliberately has no `Add` button, because a template carries a set of required selections. The hint under the list says: `Only built-in templates so far. Edit one directly or use Duplicate to fork a custom copy.` Use `Duplicate` to create a template: the copy is pre-filled with all values of the source.

Built-in templates are editable. Saving one sets the `Overridden` pill; `Restore Built-In` resets it to the code defaults. Deleting a built-in hides it softly (bring it back with `Show hidden` + `Restore Built-In`); custom templates are removed permanently. Deleting the active template repoints the active template to `default`. A custom template that other entries reference - run templates, an MCP profile's profile-wide template or one of its facet slots - shows an `IN USE · n` pill in the list. `Discard` reverts unsaved editor changes to the last saved values; it is available for custom templates.

If the selected template is used by one or more open sessions, a non-blocking hint banner appears: `1 open session uses this template. You can edit it - your changes take effect the next time that session runs.` (or the plural form for several sessions). The template stays fully editable.

The editor is organized into four sections:

`Prompt & MD profile`

| Field | Meaning |
|---|---|
| `Prompt` | The default prompt sent to the LLM when this template runs. The Context Builder tab can override it per run. |
| `MD Profile` | The MD profile defining which sections (FILE_INDEX, entry points, dependency graphs, and so on) appear in the export. |

`Selection engine (per detail level)` - a paired grid with one row per detail tier. The composer walker reads both columns per tree node according to the node's effective detail level:

| Slot | Detail Preset | Expansion Strategy |
|---|---|---|
| `Compact` | How much of each symbol (signature / members / body) is rendered for tree nodes of this tier. | How far the walker expands into neighbors (used types, callers, and so on) for tree nodes of this tier. |
| `Normal` | Same, for the Normal tier. | Same, for the Normal tier. |
| `Detailed` | Same, for the Detailed tier. | Same, for the Detailed tier. |
| `Source` | Same, for the Source tier. | Same, for the Source tier. |

The four slots are independent: you decide per tier which detail preset and which expansion strategy apply.

The Expansion Strategy column drives the GUI tree only. The MCP server and the CLI do not read it; for a `prepare_task` bundle they use `Expansion around the MCP start point` under `Profile overrides`. The slot tooltips say so.

`Export content`

| Option | Meaning |
|---|---|
| `Test Description (optional)` | An optional benchmark/test description attached to runs. Useful when the template feeds the SCE benchmark framework. A cross button empties the slot again. |
| `Embed layout legend (PATH_LEGEND aliases)` | Off for refactoring-style runs where the layer map dominates and aliases add overhead; on for graph-heavy analyses such as architecture review or onboarding. |
| `Include config files that may contain secrets (appsettings, *.config, launchSettings)` | **Off by default.** When enabled, secret-bearing configuration files - `appsettings.json` / `appsettings.*.json`, `*.config` (web/app/nuget.config) and `launchSettings.json` - are emitted verbatim in the `AUXILIARY_FILES` section, including any connection strings and API keys. Only enable it for a solution whose configuration holds no secrets, or review the output before sharing it. |
| `Include line numbers` | When on, every method and type tag in the MD export gets an additional `lines="START-END"` attribute (1-based, IDE convention). Useful when the LLM needs to point at specific source locations or when comparing line spans across runs. |
| `Include quality metrics (QUALITY_HOTSPOTS + complexity/coupling attributes)` | When on, the MD export adds a `QUALITY_HOTSPOTS` section ranking the most complex and most coupled methods and types, and annotates each method/type tag with `complexity=""` and `ce=""` (efferent coupling) attributes. Gives the LLM a numeric map of where the risk concentrates. Default off, which keeps the export byte-identical. |

Note: XAML, `.csproj` and other supporting files are included automatically by relevance; only the secret-bearing configuration subset stays gated by the option above.

`Profile overrides` - optional per-template overrides. An empty slot means "use the globally active profile" - for `Expansion around the MCP start point`, the MCP's own default. The three profile fields (`CompressionRules profile`, `Pipeline profile`, `Insights profile`) and `Expansion around the MCP start point` each have a cross button that empties the slot. `Token budget (MCP render)` has no cross; clear its text field to remove that override.

| Field | Meaning |
|---|---|
| `CompressionRules profile` | Controls scoring and Auto-Compression. Empty = use the globally active profile. |
| `Pipeline profile` | Controls the token-budget pipeline and snapshot auto-reload. Empty = use the globally active profile. |
| `Token budget (MCP render)` | Optional maximum output tokens when the MCP server renders through this template (floored at 8000). Positive whole numbers only; empty = inherit the global pipeline-profile budget. When set it forces trimming; a set MCP-profile budget wins over it. Affects the MCP render path only - GUI and CLI exports are unaffected. |

| Field | Meaning |
|---|---|
| `Expansion around the MCP start point` | How far the bundle grows around the symbols a task names when the MCP server or the CLI renders through this template ( `prepare_task` ). Empty = the MCP's own default. The GUI tree does not read it - it uses the Expansion Strategy slots of the selection engine, and those do not reach the MCP or the CLI. A whole-solution export is not affected. The CLI reads a change on its next run; an MCP server that is already running reads it only after a restart. Help line: `For the MCP and CLI only, not the GUI tree. Empty = the MCP default.` |
| `Insights profile` | Controls producer toggles, thresholds and ActionMode. Empty = use the globally active profile. |

**Required selections and save validation.** `Prompt` , `MD Profile` , all four detail presets, all four expansion strategies and the name are required. Saving fails with a specific message, for example `Save failed: Prompt is required.` , `Save failed: Compact Detail Preset is required.` , `Save failed: Compact Expansion Strategy is required.` or `Save failed: Name is required.` The name must be unique among templates of the same kind (built-in vs custom), compared case-insensitively: `Save failed: A template with name '{name}' already exists.`

## 9.4 Layer Profiles, Exclude Namespaces and Test Profiles appear twice

Three pages exist under the same names in two places, and they do different things:

| | Settings page | Solutions page |
|---|---|---|
| **Role** | The library editor: create, edit, delete, import, export entries. | The picker per solution: choose which entry applies to the selected solution. |
| **Reachability** | Settings container (for example `Settings` → `Layer Profiles` ). | The Solutions workspace, three pickers side by side. |
| **Without a selected solution** | Normally usable. | The picker is read-only and shows only the global default. |

The Settings pages write to the application settings; the Solutions pages write the per-solution choice into the solution record. The per-solution choice wins on all three axes:

| Axis | Resolution order |
|---|---|
| Test profile | Per-solution choice > global default ( `AppSettings.ActiveTestProfileId` ) > built-in default. |
| Exclusion list | Per-solution choice > global default ( `AppSettings.DefaultExclusionListName` ) > built-in `BclDefault` > empty. |
| Layer profile | Per-solution choice > app-global default; the configured name is matched by id **or** name, case-insensitively. |

The Solutions-side picker shows how many entries are available, points to the Settings library ( `Manage profiles in Settings > Layer Profiles.` and equivalents), and offers `Initialize Now` to restore the solution's axis from its sidecar file - no LLM, no analysis; it does nothing if the axis is already configured or the sidecar does not cover it.

The sidecar file is a separate thing from the per-solution setting. It is written as `<directory>\<SolutionName>.aicb.json` next to the solution, carries the rules embedded (no database id references, so it is portable in version control), and exists for hosts that run without a database. The order one level up is: explicit call parameter > database > sidecar > nothing - each host consults the sidecar only for an axis that would otherwise be empty.

## 9.5 Where settings are stored and when they take effect

Storage is hybrid, and the boundary is not arbitrary: whatever is needed *before* the database is open must live in the JSON file.

| What | Where |
|---|---|
| Storage settings (base path, database path, backup) | JSON: `%APPDATA%\AIContextBuilder\app-settings.json` |
| Recent solutions and recent databases | The same JSON file |

| What | Where |
|---|---|
| Everything else from the application and general settings | SQLite, key/value under `general/...` |
| All master data (prompts, profiles, schemas, templates, MCP profiles, and so on) | SQLite, in their own tables |
| MCP telemetry and notes | SQLite tables `tool_calls`, `tool_call_notes`, `usage_snapshots` |
| API keys of Model Profiles | Windows Credential Manager - not in the database, not in the JSON file, not in a master-data export |

The base path defaults to `%APPDATA%\AIContextBuilder`, and the database defaults to `{BasePath}\user-data\aicb.acb`.

Settings keys stored in the database include `general/settings` (the complete general-settings block), `general/active-context-template-id`, `general/default-layer-profile-name`, `general/default-exclusion-list-name`, `general/default-expansion-strategy-id`, `general/default-graph-layout-enabled`, `general/active-mcp-profile-id`, `general/active-pipeline-profile-id`, `general/active-quality-profile-id`, `general/active-compression-rules-profile-id`, `general/active-test-profile-id`, `general/tooltips-enabled`, `general/tooltip-show-delay`, `general/persist-insights-with-session`, `general/auto-trigger-insights-on-load`, `general/enable-load-perf-logging`, `general/has-run-first-analyze-once`, `general/last-selected-settings-sub-tab`, and `layout/settings` for the splitter positions.

When a change takes effect:

| Effect | Examples |
|---|---|
| Immediately on save | Font family, font size/zoom, theme, the tooltip switch and the tooltip delay. |
| At the next start of the app | Base path, LLM retry values, the theme in some individual panels. |
| At the next start of the MCP server | The MCP tool set, the MCP instructions, the auto-refresh mode. |
| Live per MCP call | A profile's context-template assignment. |
| The next time a tab is opened | Default layer profile, default exclusion list, default graph layout. |

Note: the application database is shared with the standalone MCP server. A schema update is applied by whichever host opens the database first after the update, and it applies to all instances that use that database at once. If you keep long-lived copies of the database, make sure they are backed up before you update.

# 10 Dialogs

AIContextBuilder does not use the plain Windows message boxes. Every dialog is a themed window of the app itself, and this chapter describes each one: when it appears, what it shows, what each button does, and what happens to your data on each path.

## 10.1 How dialogs behave

All dialogs share the same behavior:

- **Modal.** While a dialog is open, the main window behind it does not accept input. You answer or close the dialog first.
- **Centered on the main window.** If a dialog appears before the main window exists - for example an error during startup - it is centered on the screen instead.
- **Themed title bar, no window management.** Dialogs have no minimize and no maximize button and never appear in the taskbar. The title bar carries only the close button (X).
- **Escape always closes a dialog** and answers with the dialog's dismiss value - the cancel answer wherever the dialog has a Cancel option.
- **Enter activates the default button** (the highlighted one) - the button the dialog expects you to choose.
- **The X behaves like Escape.** It returns the same dismiss value as Escape, so closing a dialog with the X never counts as a confirmation.

Most dialogs are fixed in size: they grow to fit their content and cap their height, so long messages and lists scroll inside them. These five open at a default size and can be resized with the mouse:

| Window | Default size |
|---|---|
| Diff window | 900 x 600 (minimum 640 x 420) |
| `Keyboard shortcuts` | 760 x 660 (minimum 520 x 360) |
| `Export as Built-In Code` | 720 x 600 (minimum 560 x 400) |
| `Pick namespaces` | 560 x 560 (minimum 420 x 360) |
| `Session opened - missing nodes` | 560 x 480 (minimum 520 x 360) |

The dialogs at a glance:

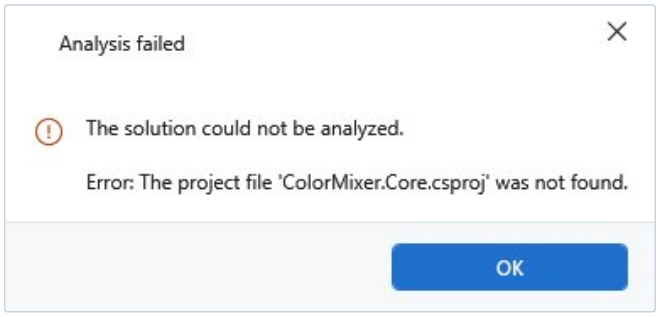
| Dialog | Appears when |
|---|---|
| Message dialog | The app reports something or asks a simple question |
| Text input | One short piece of text is needed, typically a name |
| `Save Session` | You save a session, or an unnamed session is saved while closing |
| `Relocate solution` | A stored solution file can no longer be found at its path |
| `Unsaved tabs` | You close the app while tabs still hold unsaved work |
| `Session opened - missing nodes` | A session refers to tree nodes that no longer exist |
| `Import master entities` | An imported file contains entries that already exist with different data |
| `Clear Usage Data` | You clear recorded MCP usage data |
| `Queue as work` | You queue an insight detail line as work to do |
| `Pick namespaces` | You ask an editor to suggest rules from a solution |
| `Export as Built-In Code` | You export an edited entry as built-in source code |
| `Keyboard shortcuts` | You press `F1` |

| Dialog | Appears when |
|---|---|
| Diff window | You compare two snapshots or review the result of a solution reload |

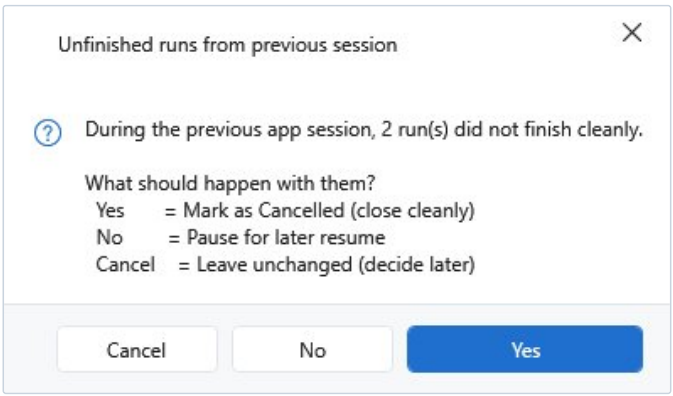
## 10.2 Message dialogs

### Message dialog

The message dialog replaces the native Windows message box. It shows a title, a message, a status icon and one to three buttons, and it is used wherever the app has something to report or needs a simple decision.




A message dialog of type error




A message dialog with three choices: Unfinished runs from previous session

The icon depends on the kind of message:

| Kind | Icon |
|---|---|
| Information | Blue accent information circle |
| Question | Blue accent question mark circle |
| Warning | Amber warning triangle |
| Error | Red error circle |
| None | No icon |

Long messages scroll; the dialog grows to fit its content up to a maximum height.

The buttons depend on the question. Three slots exist - a primary button, an optional `No`, and an optional `Cancel` - and the following table shows which combination is used:

| Button set | Primary button | `No` visible | `Cancel` visible | Escape answers | X answers |
|---|---|---|---|---|---|
| OK | `OK` | no | no | `OK` | `OK` |

| Button set | Primary button | `No` visible | `Cancel` visible | Escape answers | X answers |
|---|---|---|---|---|---|
| OK / Cancel | `OK` | no | yes | `Cancel` | `Cancel` |
| Yes / No | `Yes` | yes | no | `No` | `No` |
| Yes / No / Cancel | `Yes` | yes | yes | `Cancel` | `Cancel` |

Normally the primary button holds the focus and answers Enter, so Enter confirms. **Destructive confirmations deliberately turn this around**: there `No` holds the focus and answers Enter, and the primary button is drawn in the red danger style. A reflexive Enter on a dialog that has just appeared therefore never deletes anything, and both Escape and the X answer `No`.

The app uses this destructive form only where something is actually lost. The actions behind it are: clearing recorded usage data, deleting a usage snapshot, closing a file with unsaved changes, discarding unsaved files, overwriting a file that someone else changed, resetting node overrides, deleting a session, deleting a snapshot, exporting a sidecar configuration, and removing a solution from the list.

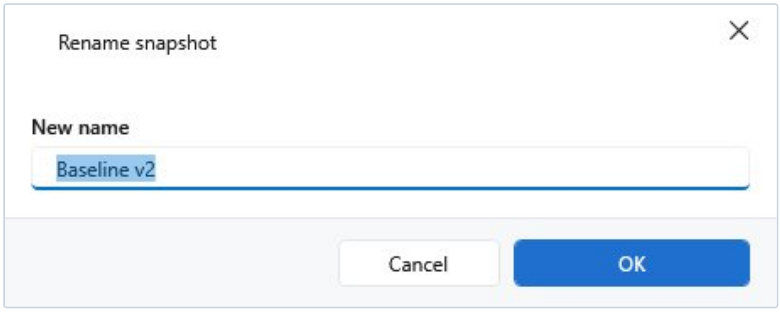
Ordinary OK/Cancel questions appear, for example, before switching the database location, before an insight action whose active quality profile asks for a confirmation first, and for a run whose estimated size exceeds its budget.

Plain information, warning and error dialogs report the outcome of an action and carry a single `OK`. Errors that occur so early that the main window does not exist yet use the same dialog, centered on the screen.

## 10.3 Input dialogs

### Text input

The text input dialog is the smallest dialog: a title, a label and a single-line text field. It is used wherever the app needs one short piece of text, typically a name.




The text input dialog, here for renaming a snapshot

- The window title is `Input` unless the caller supplies its own.
- The label is supplied by the caller. The tooltip of the field reads `Input field - Enter confirms, Escape cancels.`
- When the dialog opens, the text field has the focus and its content is selected - typing replaces the suggestion.
- Buttons: `Cancel` (tooltip `Close the dialog without input (or press Escape).`) and `OK` (tooltip `Confirm the input and close the dialog (or press Enter).`). Enter inside the field is handled directly, so it works without leaving the field.
- There is **no validation** in the dialog. An empty text is an accepted answer and is returned as an empty string; whatever check exists belongs to the caller.

The places it is used, with the title and label each one supplies:

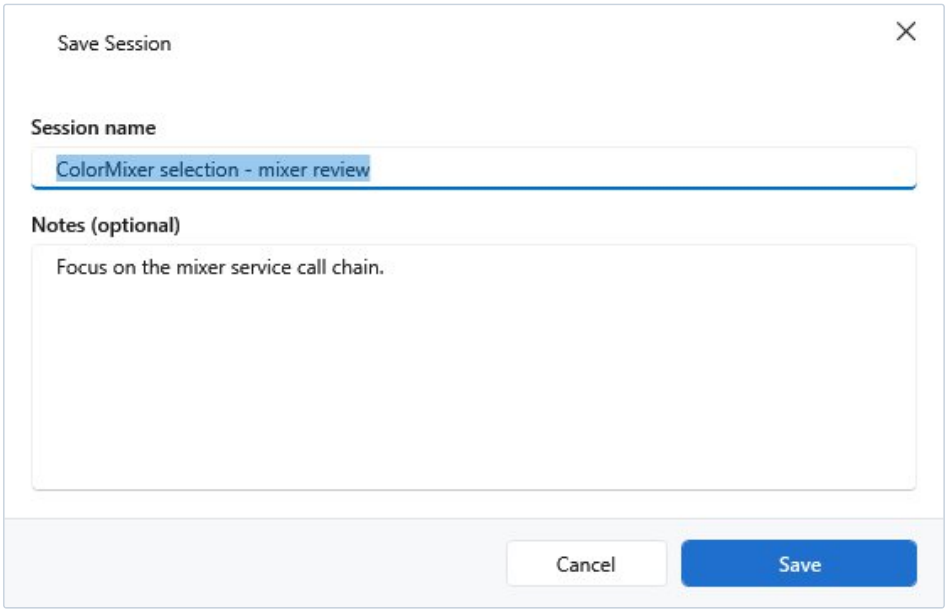
| Used for | Title | Label | Prefilled with |
|---|---|---|---|
| Create a manual snapshot (Snapshots tab) | `Create Manual Snapshot` | `Name for the snapshot:` | empty |
| Rename a snapshot (Snapshots tab) | `Rename Snapshot` | `New name:` | the current name |

| Used for | Title | Label | Prefilled with |
|---|---|---|---|
| Pin a snapshot (Solutions tab) | `Pin Snapshot` | `Snapshot name (manual snapshots are kept longer):` | the latest snapshot's name, or `Pinned <date>` |
| Rename a snapshot (Solutions tab) | `Rename snapshot` | `New snapshot name:` | the current name |
| LLM Layer Wizard (Layer Profiles) | `LLM Layer Wizard` | `Describe your architecture (layers / conventions), or leave blank to let the model infer it:` | empty |

Note: `Cancel`, Escape and the X give the caller nothing, and the operation is abandoned. Confirming an empty field gives the caller an empty string: the first four flows above treat that as "no change", while the LLM Layer Wizard uses it as "no description - let the model infer the architecture".

### Save Session

This dialog collects the name and the notes for a session that is about to be saved. It opens when you save a session in the Context Builder (for example with `Ctrl+S`), and it also opens from the save-all that `Save and close` performs in the `Unsaved tabs` dialog when a session has no record yet.




The Save Session dialog

Fields:

| Field | Type | Tooltip |
|---|---|---|
| `Session name` | single line | `Unique display name for this Session. Appears in the Session list and the tab header.` |
| `Notes (optional)` | multi-line, wraps, scrolls | `Free-text notes about the Session (context, reason, findings). Appears in the Session list tooltip.` |

The dialog is prefilled: the name with the session's current name, or with `<solution file name> - <date time>` for a new session; the notes with the existing notes. The focus starts in the name field with the text selected.

Buttons:

- `Cancel` - tooltip `Close the dialog without saving the Session.`
- `Save` - default button, tooltip `Save the Session under the given name. Enabled as soon as the name is not empty.` It is disabled while the name is empty or contains only spaces. There is **no error text**; the button simply stays disabled.

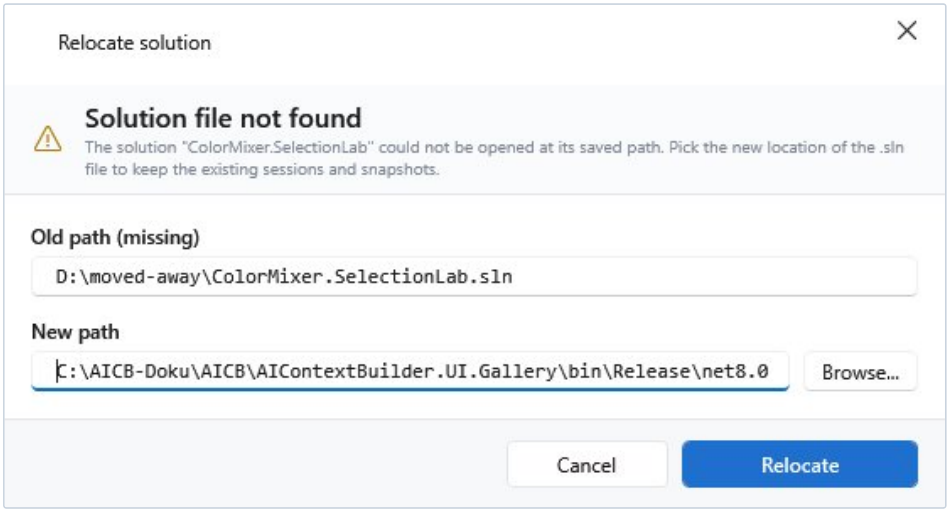
What happens with your answer:

- `Save` stores the session under the trimmed name. You may keep the suggested name unchanged.
- `Cancel`, Escape and the X write nothing and abandon the save.
- If another session in the same solution already carries the name, the app asks again: `A session named "<name>" already exists in this solution (last modified: <date>). Save anyway? Both sessions will then appear with the same name in the list. Click 'No' to return to the save dialog and pick a different name.` (title `Duplicate Session Name`, warning icon). `Yes` saves the duplicate name; `No` abandons the save, and you can start it again to pick a different name.
- If the session's state is identical to the state at the last save, the app asks first: `Current state is identical to the last saved session "<name>" (same state hash). Overwrite anyway?` (title `Save Session`, question icon). `Yes` saves anyway, `No` abandons the save.
- In the close flows (`Save and close` on app shutdown, closing a Context Builder tab with unsaved sessions), sessions that already have a record are updated silently under their current name and notes. Only a session without a record opens this dialog. Cancelling that dialog aborts the whole close, so the app or the tab stays open.

## 10.4 Recovery and safety dialogs

### Relocate solution

This dialog restores a stored solution whose `.sln` file can no longer be found at its saved path - typically after the database was copied to another computer. It is shown when you open a session whose solution is missing, and after a failed attempt to open a solution.




The Relocate solution dialog

It shows a warning icon with the heading `Solution file not found` and the subtitle `The solution "<name>" could not be opened at its saved path. Pick the new location of the .sln file to keep the existing sessions and snapshots.`

Fields:

| Field | Behavior | Tooltip |
|---|---|---|
| `Old path (missing)` | read-only, monospace; selectable so you can copy it | `The saved path where the .sln is expected but currently missing (read-only; select to copy).` |
| `New path` | editable, monospace | `Full path to the new .sln file (type it or pick via Browse).` |

`Browse...` opens the Windows file picker titled `Select relocated solution` with the filter `Solution files (*.sln)` and `All files (*.*)`. It starts in the folder of the old path when that folder still exists. Choosing a file fills `New path`; cancelling the picker changes nothing. The focus starts in the `New path` field.

Buttons:

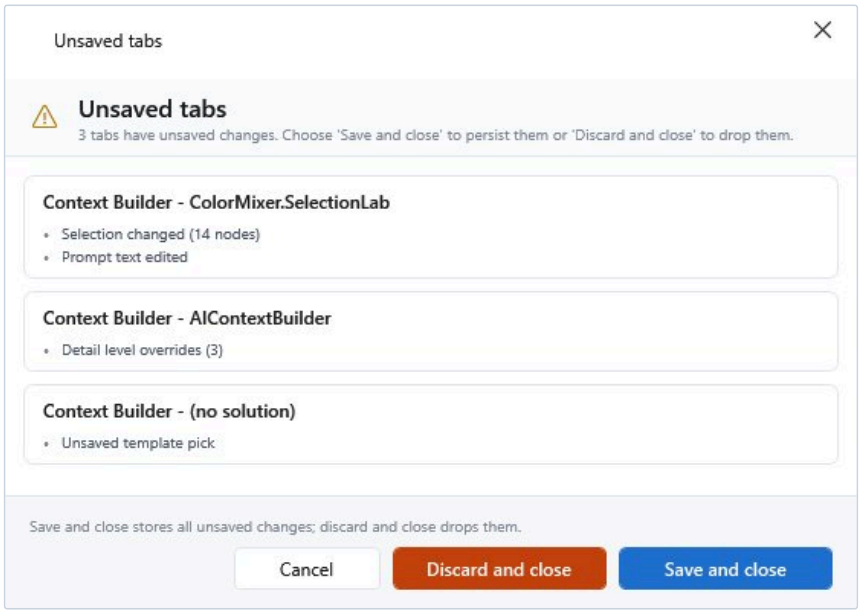
- `Cancel` - tooltip `Cancel the relocate operation - the Solution stays in the DB with the old (missing) path.`
- `Relocate` - default button, tooltip `Update the Solution path in the DB - all Sessions/Snapshots are preserved.` It stays disabled until the path is non-empty **and** the file actually exists on disk. Note: no message explains why it is disabled - it simply stays grayed out until the file is found.

What happens with your answer:

- `Relocate` updates only the stored path. **Sessions and snapshots are kept** - they belong to the solution entry, not to its path, which is the whole point of the dialog. If the stored path cannot be updated, an error dialog with the title `Relocate solution` appears and the operation stops.
- `Cancel`, Escape and the X change nothing: the solution keeps the old, missing path and the open is abandoned.
- If the freshly chosen file disappears between confirming and loading it, an error dialog says `Solution file still not accessible at: <path>`. The stored path has already been updated at that point.

### Unsaved tabs

This dialog guards the exit of the application. It appears when you close the app while tabs still hold unsaved work, and it lists one card per affected tab, each with the tab header and a bullet list of what changed. If no tab has unsaved changes, the dialog does not appear and the app closes directly.




The Unsaved tabs dialog

It shows a warning icon with the heading `Unsaved tabs` and a count-aware subtitle:

- One tab: `1 tab has unsaved changes. Choose 'Save and close' to persist them or 'Discard and close' to drop them.`
- Several tabs: `<n> tabs have unsaved changes. Choose 'Save and close' to persist them or 'Discard and close' to drop them.`

A footer hint reads `Save and close stores all unsaved changes; discard and close drops them.`

Buttons, from left to right:

| Button | Style | Tooltip | Effect |
|---|---|---|---|
| `Cancel` | secondary | `Cancel the close operation - tabs stay open with unsaved changes.` | The app stays open; every tab keeps its changes. |
| `Discard and close` | danger (red) | `DESTRUCTIVE: discard unsaved changes and close the tabs. Cannot be undone.` | The app closes and all unsaved changes are dropped, without a second confirmation - this dialog is the confirmation. |

| Button | Style | Tooltip | Effect |
|---|---|---|---|
| `Save and close` | accent, default | `Save all unsaved changes and close the tabs.` | Saves everything, then closes. |

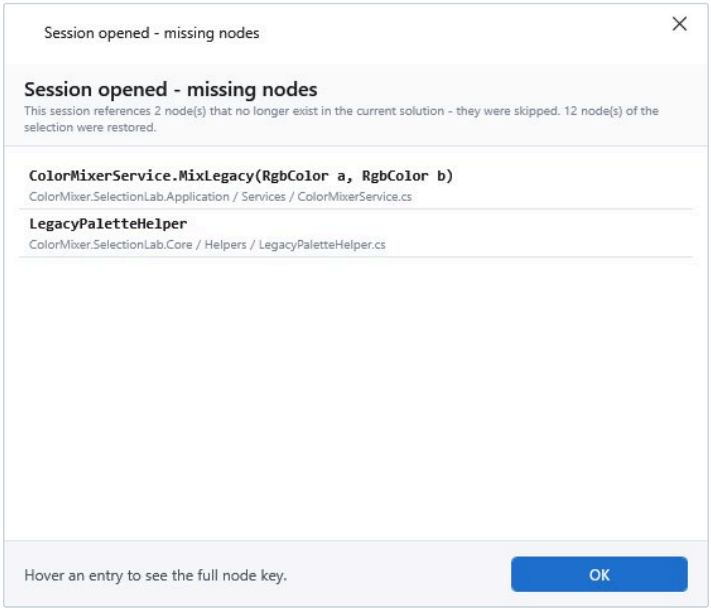
Escape and the X both cancel - the safe answer is preselected, so a reflexive Escape or X keeps your work.

Note: `Save and close` performs a **quiet** save-all. Sessions that already have a record are updated without asking; only a session without a record opens the `Save Session` dialog for its name. If a tab remains unsaved afterwards - for example because you cancelled that name dialog - the close is cancelled and the app stays open. The button label promises the close, but keeping the app open protects your work; no additional message is shown in that case.

If the check for unsaved changes fails for any reason, the close is cancelled and an error dialog with the title `Close Application` appears: `Could not check for unsaved changes, so the app was kept open to protect your work: <message> Save your sessions manually, then close again.`

### Session opened - missing nodes

This window reports the result of opening a session whose saved tree selection refers to nodes that no longer exist in the freshly loaded solution - typically after a refactoring renamed or removed types.




The window Session opened - missing nodes

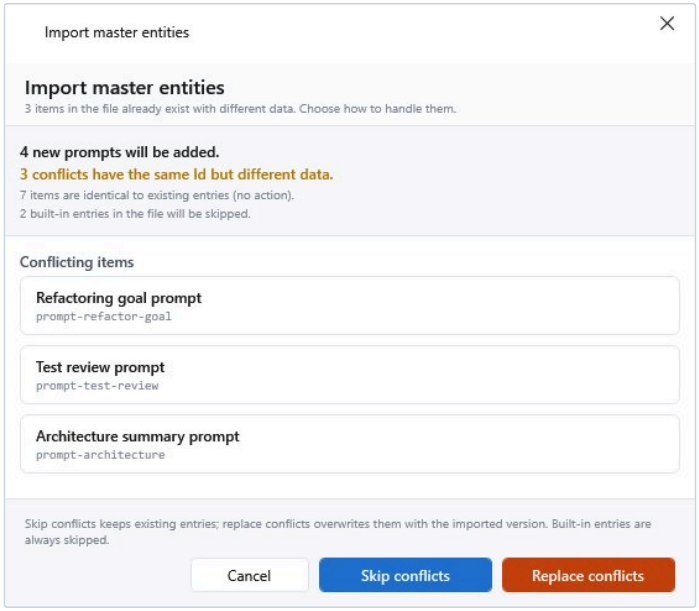
- Heading and title `Session opened - missing nodes`.
- Summary: `This session references <n> node(s) that no longer exist in the current solution - they were skipped. <m> node(s) of the selection were restored.`
- One row per missing node: a bold monospace line with the type or `Type.Method`, a muted second line with the project, folder and file, and the full node key as the row tooltip.
- Footer hint: `Hover an entry to see the full node key.`
- A single `OK` button, default and cancel at once, tooltip `Close the orphan list. Orphaned nodes are removed from the Session on the next Save.`

Data effect: the window only reports. The missing nodes were skipped when the session was applied, and because a save collects the selection from the tree as it is now, they are gone from the session after the next save. There is no way to keep them from this window.

## 10.5 Import and data dialogs

### Import master entities

This dialog resolves conflicts when an imported file contains entries whose ID already exists with different data. It is used by the master data imports and by the run template import. **It appears only when the import preview found conflicts** - a file without conflicts is imported directly, without any dialog.




The Import master entities dialog with conflicts

- Title and heading: `Import master entities`.
- Subtitle: `1 item in the file already exists with different data. Choose how to handle it.` or `<n> items in the file already exist with different data. Choose how to handle them.`
- Four summary lines:
- `<n> new <entities> will be added.` (emphasized)
- `<n> conflicts have the same Id but different data.` (warning color)
- `<n> items are identical to existing entries (no action).` (muted)
- `<n> built-in entries in the file will be skipped.` (muted)
- A scrollable `Conflicting items` list: one card per conflicting entry with its display name and, in monospace, its ID. Built-in entries are deliberately not listed because they are always skipped.
- Footer hint: `Skip conflicts keeps existing entries; replace conflicts overwrites them with the imported version. Built-in entries are always skipped.`

Buttons, from left to right:

| Button | Style | Tooltip | Effect |
|---|---|---|---|
| `Cancel` | secondary, Escape | `Cancel the import - no changes to the repository.` | Nothing is written; the status line reads `Import cancelled.` |
| `Skip conflicts` | accent, default | `Conflicting entries stay unchanged; only new entries are imported. Built-Ins are always skipped.` | New entries are added; every conflicting entry keeps its existing data. |
| `Replace conflicts` | danger (red) | `DESTRUCTIVE: existing entries are overwritten with the imported version. The provenance stamp is kept. Built-Ins are always skipped.` | New entries are added; every conflicting entry is overwritten with the version from the file. |

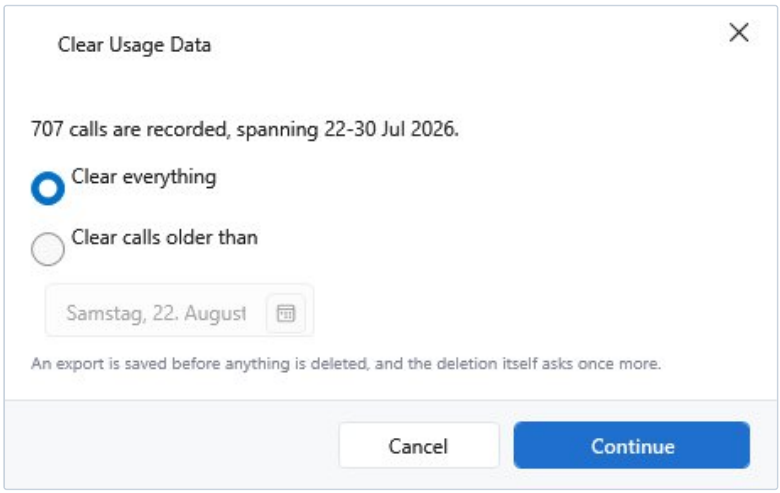
In both modes, entries that are identical to existing ones and built-in entries found in the file are counted as skipped and never written.

Note: `Replace conflicts` writes entry by entry without a transaction. If a write fails halfway through, the import reports the error and leaves the entries already written in place - there is no rollback and no undo. Afterwards the status line reads `Imported: <n> new, <n> replaced, <n> skipped`. Master-data imports add `(<n> error(s) - see log)` when something failed; the Run Templates import uses `(<n> error(s))` without `- see log`.

Errors found while reading the file (wrong format, unreadable JSON) are handled before the dialog: the import aborts with the status line `Import failed: <first error>` and no dialog appears. A file that contains only new entries is written without a confirmation dialog; the status line afterwards reports what was written.

### Clear Usage Data

This is the first step of clearing recorded MCP usage data. It only chooses the **scope** and is deliberately not the destructive confirmation. Open it with the `Clear` button in the `MCP Usage` panel (tooltip: `Delete recorded calls (all, or older than a date). An export is saved first, and the deletion asks for a destructive confirmation.`).




The Clear Usage Data dialog

- Title `Clear Usage Data`.
- A figures line built from the current data: `<n> calls are recorded, spanning <period>.` (or `1 call is recorded, ...`). The period is written like `22-30 Jul 2026`, `22 Jun - 30 Jul 2026` or `28 Dec 2025 - 3 Jan 2026`.
- Two options in the `ClearScope` group:
- `Clear everything` - selected by default. Tooltip: `Delete every recorded call.`
- `Clear calls older than` - tooltip: `Keep the recent calls and delete only those older than the chosen date.` It is followed by a date picker (tooltip: `Calls recorded before this day are deleted; the day itself and everything after it stays.`). The picker starts at today minus 30 days and is disabled until this option is selected.
- Help text: `An export is saved before anything is deleted, and the deletion itself asks once more.`
- Buttons: `Cancel` (tooltip `Close without clearing anything (or press Escape).`) and `Continue` (accent, default; tooltip `Continue to the export step. Nothing is deleted yet.`). `Continue` is enabled when `Clear everything` is selected or a date has been chosen.

The date you pick is a local calendar day: "older than" means strictly before midnight at the start of that day, in your local time.

What happens after `Continue` - the full flow:

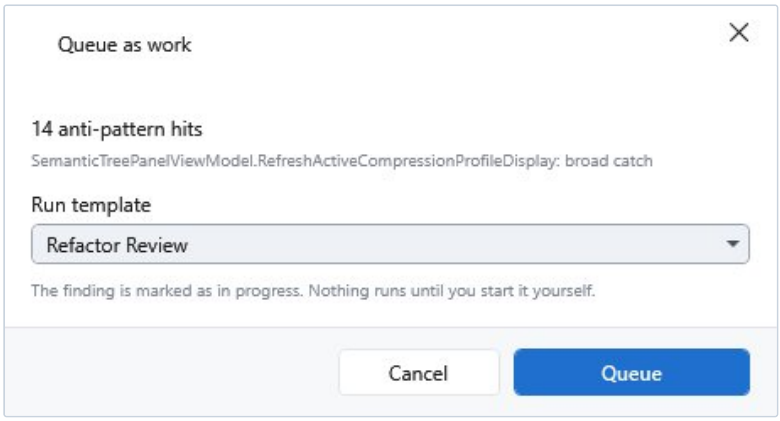
1. If no calls are recorded at all, an information dialog says `There are no recorded calls to clear.` and nothing happens.
2. The scope dialog returns your choice. Cancel, Escape or the X stop the flow.
3. The app counts the affected calls. If none are older than the chosen date, an information dialog says `No recorded call is older than the chosen date.` and nothing is deleted.
4. A save dialog titled `Export Before Clearing` appears (suggested file name `usage-report-<yyyy-MM-dd>.json`, filter `Usage report (*.json)` and `All files (*.*)`). The export is **forced**: cancelling the file choice aborts the whole flow, with the status line `Clear cancelled: no export target was chosen.` If the export cannot be written, an error dialog says `The export could not be written, so nothing was cleared: <message>`.

5. The destructive confirmation appears: `Delete all <n> recorded calls?` or `Delete the <n> recorded calls older than <date>?`, followed by `This cannot be undone. An export was saved to <file> first.` When notes are affected, it adds `<n> note(s) written on those calls will be deleted with them, and the export does not contain them.` As with every destructive confirmation, `No` holds the focus and answers Enter, and `Yes` is the red button. Anything other than `Yes` stops here.
6. The calls are deleted and the panel reloads. The status line reads `Cleared <n> calls (exported to <file> first)`.

Note: the only thing that cannot be recovered is the notes written on those calls. They are deleted together with the calls, and the export contains the aggregate report, not the notes - which is exactly what the confirmation tells you.

**Queue as work**

This dialog picks a run template for one insight detail line. It chooses and nothing else: it does not start anything. Open it with `Queue as work` on a detail line (tooltip: `Queue as work - record this line as something to fix, with a run template of your choice. Nothing is started; the line stays visible and reports 'In progress'.`).




The Queue as work dialog

- Title `Queue as work`.
- The insight's title, then the line's own text in the help-text style.
- `Run template` - a dropdown listing the available run templates. Tooltip: `The run template this piece of work should be handled with. It is recorded with the item; nothing is started now.` The list opens on the send template configured by the active quality profile, or on the first template when that one is not in the list. The related card-level `Apply + send to LLM` path is not exposed by the shipped producers.
- Help text: `The finding is marked as in progress. Nothing runs until you start it yourself.`
- Buttons: `Cancel` (tooltip `Close without queueing anything (or press Escape).`) and `Queue` (default; tooltip `Record this finding as work to do, with the chosen run template.`), disabled while no template is selected.

Result:

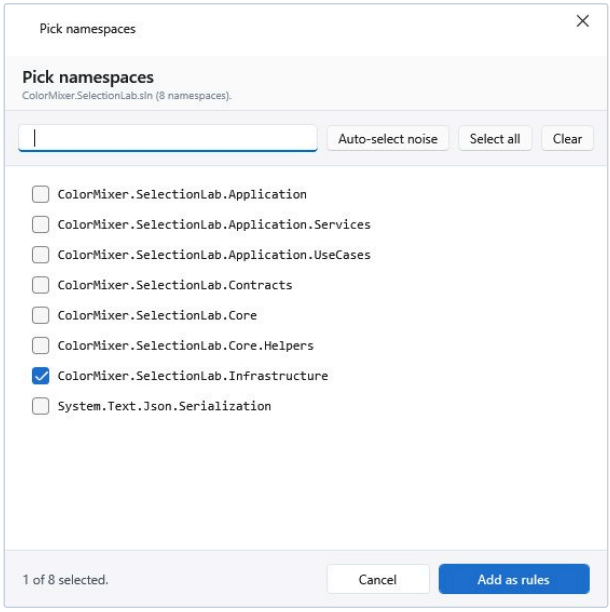
- `Queue` records the line as work to do together with the chosen template's ID and name. The line stays visible and reports "in progress"; nothing is started automatically.
- If work is already queued for that line, the status line says `Work is already queued for this line.` - the existing entry is not replaced.
- On success the status line says `Queued as work to do.`; if the entry cannot be stored, `Could not queue the work item.` and the line keeps its previous state.
- `Cancel`, Escape and the X change nothing.

Note: if the database contains no run templates at all, the dialog does not open and a status message says `No run templates are available to queue against.`

## 10.6 Editors

### Pick namespaces

This dialog offers a filterable, checkable list of the namespaces found in a solution, and turns your selection into rules. It opens from `Suggest...` in the Layer Profiles editor and in the Namespace Exclusions editor. Both first ask for a solution file (title `Pick a solution to read namespaces from`, filter `Solution files (*.sln)` and `All files (*.*)`), read the namespaces, and then show this picker. Layer Profiles reads the namespaces the solution declares; Namespace Exclusions reads the namespaces the solution references.




The Pick namespaces dialog

- Title and heading `Pick namespaces`.
- Subtitle: `<source> (<n> namespaces).` - or `<n> namespaces available.` when no source is supplied. The source label is `Namespaces in <file>` (Layer Profiles) or `Referenced namespaces in <file>` (Namespace Exclusions).
- Toolbar, right-aligned:
- `Clear` - removes every checkmark, filtered or not. Tooltip: `Remove all selection checkmarks - no namespaces become rules.`
- `Select all` - checks every namespace that passes the current filter. Tooltip: `Add all filtered namespaces to the selection.`
- `Auto-select noise` - visible only in the Namespace Exclusions editor. Tooltip: `Check every namespace that matches the BCL / framework heuristic (System., Microsoft., Windows., Syncfusion., ...)`. It checks every matching entry regardless of the filter.
- A filter field - tooltip: `Filter the list by Namespace prefix (live, case-insensitive).` The list is filtered as you type. Note: the match is a case-insensitive **substring** match, so `Core` also finds `MyApp.Core.Something` and `Foo.ScoreKeeper`.
- The list shows one checkbox per namespace, in monospace. Tooltip: `Check to turn this namespace into a new exclusion rule when you confirm.`
- A live status line at the bottom left reads `No namespaces selected.` or `<selected> of <total> selected.` The total is the whole list, not the filtered subset.
- Buttons: `Cancel` (tooltip `Close the dialog without changes to the exclusion rules.`) and `Add as rules` (default; tooltip `Create all selected namespaces as new NamespaceExclusion rules.`). The focus starts in the filter field, so you can type immediately.

Note: `Select all` and `Clear` are deliberately not symmetric - `Select all` respects the filter, `Clear` does not. If you filter, select all, filter differently and then clear, the earlier selection is cleared too. The tooltips state this.

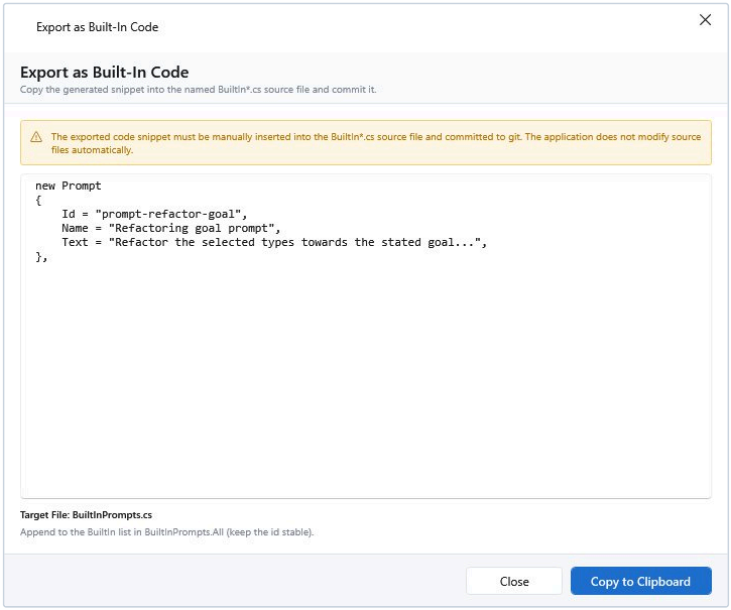
What confirming does:

- Layer Profiles: every selected namespace becomes a new layer rule with the match type `StartsWith` and an empty layer; you assign the layer per row afterwards. The status line says `Added <n> suggested rule(s). Assign a layer per row.`
- Namespace Exclusions: every selected namespace becomes a new exclusion rule with the match type `StartsWith`. The status line says `Added <n> suggested exclusion(s).`
- The selection is returned in display order and includes entries the current filter hides - filtering never unchecks anything.
- `Cancel`, Escape and the X create no rules.
- Note: confirming without a single checkmark is a valid answer that creates nothing; the caller treats an empty selection like a cancel.

If no namespaces are found, the picker does not open. An information dialog says `No C# namespaces were found in the selected solution.` (Layer Profiles) or `No referenced namespaces were found in the selected solution.` (Namespace Exclusions). If the file cannot be read, an error dialog with the title `Suggest from Solution` reports the reason.

### Export as Built-In Code

This window shows the C# snippet that would turn your edited entry into a shipped built-in. Open it with the export-as-built-in icon in the action bar of a master data entry or a run template (tooltip: `Export as Built-In: show the C# snippet for this entity to paste into the BuiltIn*.cs source file - promotes the custom value to a permanent Built-In after build + restart.`). **The app never writes source files itself.** Note: this function is intended for the product's own development; the snippet only has an effect in AICB's source code, not in your installation.




The Export as Built-In Code window

- Title and heading `Export as Built-In Code`; subtitle `Copy the generated snippet into the named BuiltIn*.cs source file and commit it.`
- A permanent warning banner: `The exported code snippet must be manually inserted into the BuiltIn*.cs source file and committed to git. The application does not modify source files automatically.`
- A red banner when the entry is already a built-in with an active override: `This item is already a Built-In with an active override. Exporting creates a new candidate Built-In; merging into the original source file is your responsibility.`
- A red banner when the entry may contain solution-specific data: `This entity may contain solution-specific paths or namespaces. Review the snippet before committing to ensure it generalizes correctly.`
- A read-only monospace text box with the snippet. It does not wrap and offers both scrollbars; tooltip: `Generated C# snippet (read-only). Copy it and paste into the target BuiltIn*.cs file - the app never writes source files itself.`
- `Target File: <name>`, or `Target File: (unknown)` when no file name is known.
- The insertion hint for the snippet (for example where in the file it belongs).

- Buttons: `Close` (tooltip `Close the dialog. The snippet was not written to the source file automatically.`) and `Copy to Clipboard` (accent, default; tooltip `Copy the C# snippet to the clipboard. Then paste it manually into the relevant BuiltIn*.cs file and commit.`).

Behavior of the buttons:

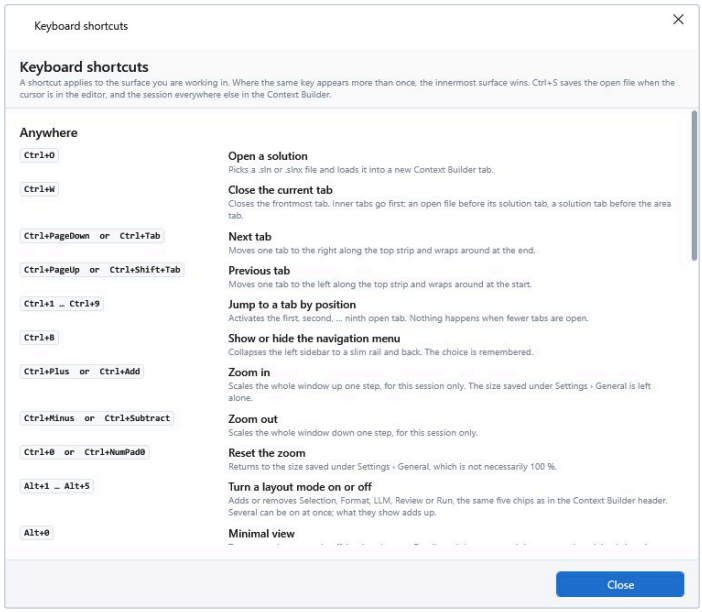
- `Copy to Clipboard` copies the snippet and **leaves the dialog open**. It shows no success or error message. Note: if another program is holding the clipboard at that moment, the copy can fail silently. Paste into a text editor first to check what is on the clipboard before you paste it into a source file.
- `Close`, Escape and the X close the window.
- Note: `Copy to Clipboard` is the default button, so Enter copies rather than closing the dialog.

Data effect: none on the database and none on disk. The only side effect is the clipboard.

## 10.7 Information windows

### Keyboard shortcuts

Press `F1` to open this read-only, scrollable overview of every keyboard shortcut, grouped by the surface it applies to. The app has no menu bar where a shortcut would surface on its own, so this window is the central place to discover them.

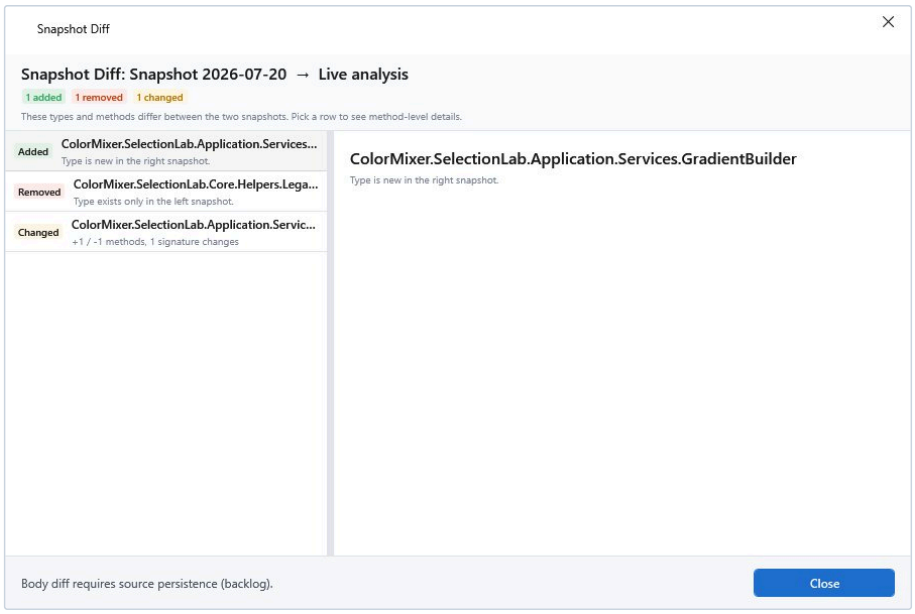



The Keyboard shortcuts window

- Title and heading `Keyboard shortcuts`.
- Subtitle: `A shortcut applies to the surface you are working in. Where the same key appears more than once, the innermost surface wins. Ctrl+S saves the open file when the cursor is in the editor, and the session everywhere else in the Context Builder.`
- The content is generated from the app's own shortcut list, so the overview cannot describe a shortcut the app does not have.
- Groups appear in this order: `Anywhere`, `Context Builder`, `Code editor`, `Settings and master data`, `Lists`. Empty groups are omitted.
- Each line shows the gesture or gestures in a monospace pill, the action in bold and a short description. When an action has more than three gestures (for example `Ctrl+1` through `Ctrl+9`), the pill shows a range like `Ctrl+1 … Ctrl+9`; up to three gestures are listed as `A or B`.
- A single `Close` button (default and cancel at once; tooltip `Close this overview.`). The window is resizable.

### Diff window

One window serves two comparisons: snapshot against snapshot, and the previously loaded tree against a reloaded solution. It is resizable and opens at 900 x 600, with the list on the left and the details on the right separated by a splitter you can drag.




The diff window comparing two snapshots

The two flows differ only in their texts:

| | **Snapshot comparison** | **Solution reload** |
|---|---|---|
| Window title | `Snapshot Diff` | `Solution reload - changes` |
| Heading | `Snapshot Diff: <left> → <right>` | `Solution reload - changes in <solution>` |
| Subtitle when nothing differs | `Both snapshots have identical class/method structure.` | `No changes detected. The reloaded tree is structurally identical to the previously loaded one.` |
| Subtitle when something differs | `These types and methods differ between the two snapshots. Pick a row to see method-level details.` | `These types and methods differ between the previously loaded tree and the freshly analyzed one. Pick a row to see method-level details.` |
| Summary for an added type | `Type is new in the right snapshot.` | `Type is new in the reloaded tree.` |
| Summary for a removed type | `Type exists only in the left snapshot.` | `Type existed only in the previously loaded tree.` |
| Footer hint | A note that comparing method bodies is not available. | `The tree above already reflects the new state. Close this window when you are done reviewing.` |

Content:

- Three count pills are always visible in the header: `<n> added` (green), `<n> removed` (red) and `<n> changed` (amber). Tooltips: `Number of added types in this diff.`, `Number of removed types in this diff.`, `Number of changed types in this diff (method/signature changes).`
- The list on the left is flat, in a fixed order: added types first, then removed types, then changed types. Each row carries a kind pill (`Added`, `Removed` or `Changed`), the type name and a summary. For a changed type the summary reads `+<n> / -<n> methods, <n> signature changes`. The first row is selected automatically. If the diff is identical, the list stays empty and the three pills all show `0`.
- The detail pane on the right shows up to three sections for the selected row: `Added methods` (each line `+ <name>`), `Removed methods` (each line `- <name>`) and `Changed signatures` (the method name plus the old and new signature as `- <left>` and `+ <right>`; tooltip: `Methods with a changed signature - Left = previous state, Right = new state.`). With no row selected it reads `No diff entry selected.`
- A single `Close` button (default and cancel at once; tooltip `Close the diff window.`).

When a solution reload finds no changes at all, the app shows an information dialog instead of this window: `No structural changes detected. The reloaded tree is identical to the previously loaded one.` (title `Solution reload`).

Note: the comparison is structural. It reports types and methods that were added or removed and signatures that changed. Method bodies are not compared.

## 10.8 Native file dialogs

Where the app asks you for a file, it uses the standard Windows dialogs, owned by the main window so they cannot disappear behind it.

- Open dialogs take their title and their file type filter from the action that opened them. Where no filter is supplied, all files are offered. A starting folder is used when the action supplies one.
- Save dialogs take their title, filter and a suggested file name from the action. Windows asks before overwriting an existing file.
- Note: the app does not set a default file extension. If you type a bare name without an extension, the file may be saved without one unless the chosen filter adds it. Type the extension as part of the name if you are unsure.

Open dialogs you can meet:

| Opened from | Title | Filter |
|---|---|---|
| Open a solution (Welcome, Solutions, Context Builder) | `Open solution` | `Solution files (*.sln;*.slnx;*.slnf)` |
| Layer Profiles - `Suggest...` | `Pick a solution to read namespaces from` | `Solution files (*.sln)` and `All files (*.*)` |
| Namespace Exclusions - `Suggest...` | `Pick a solution to read namespaces from` | `Solution files (*.sln)` and `All files (*.*)` |
| Layer Profiles - LLM Layer Wizard | `Pick a solution for the LLM layer wizard` | `Solution files (*.sln;*.slnx)` |
| `MCP Usage` panel - `Import` | `Import Usage Report` | `Usage report (*.json)` |
| Run templates - import | `Import RunTemplates` | `JSON files (*.json)` |
| Master data - import | `Import <entities>` | `JSON files (*.json)` |
| Constellations - import | `Choose a constellation` | `JSON files (*.json)` |
| Solutions - import sidecar configuration | `Import solution config (sidecar)` | `AICB sidecar (*.aicb.json)`, `JSON files (*.json)`, `All files (*.*)` |
| Storage settings - switch database | `Select database to switch to` | `AICB database (*.acb)` |

Save dialogs you can meet:

| Opened from | Title | Filter | Suggested name |
|---|---|---|---|
| `MCP Usage` panel - export | `Export Usage Report` | `Usage report (*.json)` | `usage-report-<yyyy-MM-dd>.json` |
| `MCP Usage` panel - `Clear` (forced export) | `Export Before Clearing` | `Usage report (*.json)` | `usage-report-<yyyy-MM-dd>.json` |
| Run templates - export | `Export RunTemplates` | `JSON files (*.json)` | `RunTemplate-export-<yyyyMMdd-HHmm>.json`; the export contains the complete list, not only the selection |
| Sessions - export | `Export session` | `JSON files (*.json)` | a suggested name |
| Master data - export | `Export <entities>` | `JSON files (*.json)` | a default name |

| Opened from | Title | Filter | Suggested name |
|---|---|---|---|
| Constellations - export | `Export constellation` | `JSON files (*.json)` | `constellation-<yyyyMMdd-HHmm>.json` |
| Storage settings - `Create new database` | `Create new database` | `AICB database (*.acb)` | `aicb.acb` |
| Context Builder - save the context document | `Save context document` | `Markdown files (*.md)` | a name derived from the solution |

Cancelling any of these dialogs returns you to the app without a change.

# 11 Troubleshooting the desktop app

This chapter is the reference for the desktop app's error behavior: what to check on a first start, where the app reports problems, where it stays silent, what its log file contains, and what to do about each problem you can actually hit. The CLI and the MCP server keep their own diagnostic channels and are covered in their own chapters.

## 11.1 First start checklist

### What the app needs

The desktop app is self-contained - no .NET *runtime* is required to start it. Analysis is different: opening a solution runs MSBuild to resolve references, exactly like `dotnet build` does, so the machine needs a **.NET SDK or a Visual Studio installation** with the `.NET desktop development` or `MSBuild Tools` workload. The app checks this before anything else and refuses to start without it (see "MSBuild not found" below). Only analyze solutions you trust, because loading one executes its build logic.

The desktop app comes in two forms:

| Form | What it is |
|---|---|
| `AIContextBuilder-Setup-<version>.exe` | Installer, needs administrator rights. It puts the `aicb` command on `PATH` unless you untick that option. Consoles, editors and agents that were already open see the new `PATH` only after a restart. |
| `AIContextBuilder-<version>-win-x64.zip` | Portable, no installation. `gui\aicb-ui.exe` starts the desktop app; `cli\aicb.exe` is the CLI and MCP server. |

From version 0.5.503.1 on, the installer and the programs are code-signed; the publisher is `Gregor Dadera`. A new release can still make Windows SmartScreen show "Windows protected your PC" until it has built up reputation; if the dialog names that publisher, *More info* and then *Run anyway* continues. Every release lists SHA-256 checksums for its files.

### What happens when you start the app

The startup order is fixed, and each step assumes the previous one:

1. The splash window appears with `Starting up…` and the line `100 % local analysis - nothing is sent without your action.` It runs on its own thread, so it keeps animating while the rest of the startup blocks the main thread.
2. MSBuild is located (via the .NET SDK or Visual Studio). Without it: dialog `MSBuild not found`, then the app exits.
3. The service container is built and validated, then the database schema is migrated. On a fresh installation the database file is created here.
4. Unfinished runs from the previous session are looked for. Only if any exist do you get the `Unfinished runs from previous session` dialog.
5. The shell is initialized, the theme is applied, the main window appears, and the splash closes.
6. The automatic backup starts in the background if it is enabled and due (on by default for a new installation); it does not delay the start.

A start can take several seconds, and longer when the database is large. That is normal, not a hang; the splash exists to make the wait visible.

### What you see on the Start page

After the start, **no tab is open and no sidebar entry is selected** - the Start page stays visible until you navigate yourself.

| Element | What it does |
|---|---|
| Entry text | `Turn your C# solution into a semantically compressed context for LLMs. Open a solution or load a previous session to get started.` |
| `New Solution` card | Opens the file dialog for a `.sln`, `.slnx` or `.slnf` file and loads it. |

| Element | What it does |
|---|---|
| `Open Solution` card | Opens the same file dialog. Both cards behave identically today; the tooltips describe a difference that does not exist yet. |
| `Load Session` card | Opens the `Workspace` tab, where you pick a saved session. |
| `Recent Solutions` list | The solutions you opened most recently (up to 8), with an `Open` button and double-click. A warning icon marks an entry whose `.sln` no longer exists at the saved path. |
| `Recent Sessions` list | Your most recently saved sessions (up to 5). |
| `Refresh` button | Reloads both lists. |

On the very first start both lists are empty and show `No recent solutions yet.` and `No saved sessions yet.`; the footer says `Tip: Click an area in the left sidebar to open it as a tab.`

The left sidebar carries four groups with eight entries (group headers are not clickable):

| Group | Entries |
|---|---|
| `Quick Access` | `Start` |
| `Workspace` | `Context Builder`, `Workspace`, `Run Templates` |
| `Configuration` | `Templates`, `Settings` |
| `MCP` | `MCP Profiles`, `MCP Usage` |

### Opening your first solution

1. Click `New Solution` or `Open Solution`. The file dialog is titled `Open solution` and filters on `Solution files (*.sln;*.slnx;*.slnf)`.
2. The solution is analyzed in a new `Context Builder` tab. A load screen reports the phases with a progress bar: `Opening workspace`, `Building project tree`, `Analyzing code` with a `N / M files` counter, `Saving snapshot`, `Done`. The bar becomes determinate with the first progress report, `Opening workspace`, rather than waiting for a per-file report. A pinned-snapshot restore starts with `Restoring snapshot` and remains indeterminate.
3. When the load finishes, four things exist that did not before: a row for the solution in the workspace, an automatic snapshot of the analyzed state, an entry in the recent-solutions list, and the solution tree in the Context Builder.
4. On a **fresh installation** the first analysis also runs the insights producers once automatically, so the Insights tab is not empty on first contact. Later loads do not do this unless you switch on `Auto-trigger insights on solution load` in `Settings > General`.

A bundled sample solution is available too: `ColorMixer.SelectionLab.sln` lives under `{exe}\SampleSolutions\`. It is offered by the `Open sample Solution` button in the empty state of the Insights tab, which is only reachable once a solution is open. The Start page does not offer it.

### What is remembered, and what is not

| Remembered | Not remembered |
|---|---|
| Recently opened solutions (up to 8) and databases | Open tabs - the app always starts on the Start page |
| The last selected Settings sub-tab | Window size and position - the window always starts maximized, centered, at a fixed default size |
| Theme, font family, font size, UI zoom | Anything you did not save as a session |
| The active templates and profiles | |

| Remembered | Not remembered |
|---|---|
| Saved sessions (they reappear in the `Recent Sessions` list) | |

The `On startup` choice in `Settings > General` (`Restore last tabs`, `Restore last session`, `Sessions overview`, `Empty session`) is stored but currently has **no effect**; restoring the previous session is a planned feature. Until then, saving a session is the only way to carry work across a restart. See "Workspace, sessions and snapshots" for saving and resuming sessions.

### Where your data lives

| What | Default location |
|---|---|
| Database | `%APPDATA%\AIContextBuilder\user-data\aicb.acb` |
| Settings file | `%APPDATA%\AIContextBuilder\app-settings.json` |
| Support log | `%APPDATA%\AIContextBuilder\aicb.log` (plus rotated copies) |
| Load timings | `%APPDATA%\AIContextBuilder\load-perf.log` |
| Automatic backups | `<BasePath>\backups\`, by default `%APPDATA%\AIContextBuilder\backups\` |

`Settings > Storage` shows the active database file (`Current file`), the settings file the app actually uses (`App settings file`) and the `Base path`. The default database and the base path can be changed there; the settings file itself always stays at `%APPDATA%\AIContextBuilder\app-settings.json`, even if you change the base path. Deleting `%APPDATA%\AIContextBuilder\` removes your data along with the app's settings.

## 11.2 Where errors appear

The desktop app reports problems through exactly three channels, plus the splash window:

1. **Modal dialogs** for anything that needs an answer or that you must see.
2. **Status lines** inside panels for the outcome of an action.
3. The **startup notice bar** for problems that happened before any window existed.

There are no toasts, no snackbars and no info bars. A problem that reaches none of these three channels leaves no visible trace at all - see "Silent failures" below.

### Modal dialogs

All message dialogs are themed windows of the application, not Windows message boxes. They come with four button sets: OK, OK-Cancel, Yes-No, and Yes-No-Cancel.

- `Esc` and the window's close button return the same answer as the dismiss button: Cancel where the dialog has one, otherwise No, otherwise OK.
- `Enter` answers the primary button - except in a **destructive** confirmation (for example discarding unsaved changes). There the `No` button holds the focus and answers `Enter`, so a reflexive key press cannot delete anything, and the dangerous button is styled as dangerous.

Typical startup dialogs and their titles:

| Title | Meaning |
|---|---|
| `MSBuild not found` | No usable MSBuild installation; the app exits. |
| `Database is newer than this build` | The database was written by a newer version; the app exits rather than write to it. |
| `Database migration failed` | The schema could not be updated; the app exits. |
| `Unfinished runs from previous session` | Runs from the last session never reached a final state. |

| Title | Meaning |
|---|---|
| `Crash recovery` | The recovery check itself failed; the app starts anyway. |
| `Unexpected error` | An unhandled error; the app stays open. |

### Status lines

Panels report the result of an action in a status line - usually at the bottom of the panel's action bar. These messages are **not modal and easy to miss**, and the next action can overwrite them. Read the status line right after the action that produced it.

| Message pattern | Where |
|---|---|
| `Save failed: <reason>` | Settings panels, run templates, session save |
| `Export failed: <reason>` | Settings panels, run templates |
| `Import failed: <reason>` | Run templates, master data |
| `Check failed: <reason>` | Code editor in the Details tab |
| `Send failed: <reason>` | Context Builder, under `MD Input` |
| `Snapshot history could not be read` | Workspace, snapshot list |
| `Statistics could not be read` | Workspace, overview cards |
| `Not analyzed yet` | Workspace, overview cards - a statement, not an error |
| `- runs` | A session row whose run count could not be read (a dash, not a zero) |
| `Auto-save could not save one of your open sessions. ...` | Dialog (once per session), see "Auto-save and unnamed tabs" |

### The startup notice bar

The startup notice bar sits at the top of the content area, above the Welcome page or the tab strip. It appears only when something went wrong **before any window existed**, so it has exactly two producers:

- the automatic backup when the archive is incomplete, could not be written at all, or was written successfully but its completion timestamp could not be saved (a fully recorded clean backup, and a start on which none was due, stay silent);
- an unreadable application settings file.

Messages are prefixed with their source: `Automatic backup at startup - ...` or `Application settings - ...`. If both happen on one start, both lines appear.

The dismiss button hides the bar for the current session only. Nothing is saved: if the problem is still there, the next start reports it again.

### The splash window

The splash (`Starting up…`) is not an error channel. It exists so that a slow start does not look like a crash - which is exactly why a long-running splash is not a symptom by itself.

### Silent failures

Some failures produce no message at all; the app simply continues with a plausible-looking result. This is the deliberate trade-off behind the "the app stays usable" policy, and it is the hardest class to diagnose, because nothing looks broken. The rule of thumb: **if an area is unexpectedly empty or unexpectedly unchanged, refresh it explicitly (`Refresh`, `Analyze Solution`) and look into `aicb.log`.**

| Symptom | What actually happened |
|---|---|
| The Insights panel is empty and shows `No insights to show here - pick a category on the left, or relax the principle/severity filters.` | Reading the stored insights failed; the empty-state text sends you to the filters while the cause is a database read error. |
| The `Solutions` tab shows old data that looks current | The refresh of the tab failed and the previously loaded rows stayed on screen. |
| The model profile picker is empty although profiles exist | Reading the profiles failed. |
| The list of recent runs is empty | Reading the run history failed - indistinguishable from "no runs yet". |
| A dismissed insight comes back after the next refresh | The database write for the dismissal failed; the card is removed from the list anyway. |
| The counter-clockwise-arrow button in the Insights toolbar (tooltip beginning `Reset what you hid - ...`) seems to do nothing | The database write failed and the command returned. |
| Copying to the clipboard does nothing, in any panel | The clipboard was held by another process (for example an RDP session or a virus scanner). Copy retries three times with a 50 ms pause and then gives up without a message. Paste reads once, catches an error and silently returns no text; it is not retried. |
| A splitter position is forgotten after a restart | Saving the layout is best-effort and silently skipped on failure. |
| The prompt fields are unchanged after opening a session | The stored prompt payload could not be parsed; the current values were kept. |
| The unsaved-changes summary is missing the tree-selection line | It is not missing: the line reads `Tree selection: could not be read - it may hold unsaved overrides`. |
| The tab strip does not react to clicks | A run is active; see "The tab strip is locked while a run is active". |

## 11.3 The `Unexpected error` dialog

An unhandled error in a command, an event handler or a background callback is caught by the app's global exception handlers and reported in a dialog titled `Unexpected error`:

```
An unexpected error occurred.

The application is still running, but its state may be inconsistent. Save any open sessions and restart.

Details: <Type>: <message> -> <inner type>: <message> -> ...
```

The `Details:` line unwraps up to three nested exceptions, because the outermost one is often a meaningless wrapper.

**Why the app keeps running.** Several Context Builder tabs can hold unsaved sessions in memory. Ending the process would discard them without asking. Staying alive gives you the one thing you cannot get afterwards - the chance to save. The price is a possibly inconsistent state, and the message says so instead of pretending the error was handled.

Two rules keep this dialog from taking over the app:

- **At most three dialogs per session.** A failing layout or render path throws on every frame; an unconditional dialog would lock you out of your own application. After the third dialog, further errors are written to the log only.
- **Each distinct error is shown once.** Errors are de-duplicated by type, message and top stack frame, so the same bug does not repeat while a different bug with the same message still gets through.

What to do: save your open sessions, restart the app, and include the `Details:` line in a support request. The suppressed repeats after the third dialog are still in `aicb.log`.

A failure inside a background task that nobody awaits does **not** produce a dialog: since .NET Core it no longer ends the process, and the app logs it as a diagnostic. If something behaves oddly after a long-running action, the log is the place to look.

## 11.4 The log file `aicb.log`

The desktop app writes its own warnings and errors to a rotating local file:

| Property | Value |
|---|---|
| Path | `%APPDATA%\AIContextBuilder\aicb.log` |
| Levels | `Warning`, `Error` and `Critical` only - the normal flow is not recorded |
| Format | ISO 8601 timestamp (UTC), level in brackets, logger category, message; the full exception follows on the next lines when one is attached |
| Rotation | at 1 MiB, keeping three archives: `aicb.log.1`, `aicb.log.2`, `aicb.log.3`; the oldest is dropped |
| Open handle | none - you can copy the file while the app is running |

Notes:

- The file is created with the first warning or error. On a healthy installation it may not exist at all.
- A second running instance may lose a single entry when it writes at the exact moment another process holds the file; the entry is dropped so that logging can never turn a recoverable error into a failed start.
- The log contains the unhandled errors of the `Unexpected error` dialog, including the repeats that were suppressed after the third dialog.
- A few deliberately non-critical startup steps - applying the theme, the font and the UI zoom - are caught **without** writing a log entry. If the app starts in the wrong theme, the log is silent about it; a restart is the first thing to try.
- Logging is best-effort: if the file cannot be written, the error is still shown to you, it is just not recorded.

`load-perf.log`

A second, separate file records solution-load phase timings:

- Location: `%APPDATA%\AIContextBuilder\load-perf.log`
- One line per solution load, prefixed with a local timestamp, in the form `[PerfLoad] {context} total=...ms Phase1_LeaseAcquire=... Phase2_TreeBuild=... Phase3_Analyze=... Phase4_Snapshot=...`; only `total` carries the `ms` suffix
- Switch: `Settings > General` → `Record solution-load performance timings` (on by default)
- It contains **times, not errors**, and is not the file to send for a malfunction report.

### What to send with a support request

| Item | Where to find it |
|---|---|
| The log and its archives | `%APPDATA%\AIContextBuilder\aicb.log`, `.1`, `.2`, `.3` |
| A screenshot of the dialog | The `Details:` line carries the exception type and message |
| The app version | `Settings > About` |
| Settings and paths | `%APPDATA%\AIContextBuilder\app-settings.json` |
| Load timings (only if the start or a load is the problem) | `%APPDATA%\AIContextBuilder\load-perf.log` |
| For MCP problems | The stderr log of your MCP client - the only place the MCP server leaves diagnostic text |

Bugs and feature requests go to GitHub Issues; the `About` panel shows the Issues URL as plain, non-clickable text in its support card.

## 11.5 Problems and their solutions

### The app does not start

`MSBuild not found`

**Symptom.** A dialog titled `MSBuild not found` appears right after the splash. It ends with `The application will exit.`, and the app closes (exit code 1).

**Cause.** The app found no compatible MSBuild installation. It tries, in this order: the MSBuild locator's defaults, the installed Visual Studio instances, `vswhere.exe`, and the newest .NET SDK under `%ProgramFiles%\dotnet\sdk`. Without MSBuild no Roslyn solution can be loaded, so a window without a working feature would be the worse answer.

**What to do.** Install the .NET SDK or Visual Studio with the `.NET desktop development` or `MSBuild Tools` workload, then start the app again. The dialog names the probes it ran, including the PowerShell commands to reproduce them:

```
& "C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.exe" -latest -products * -requires
Microsoft.Component.MSBuild -property installationPath
dotnet --list-sdks
```

`Database is newer than this build`

**Symptom.** A dialog titled `Database is newer than this build` shows the database schema version and the version this build knows; the app exits (exit code 2).

**Cause.** The database was written by a newer version of the application. An older build would silently save the outdated shape back, so it refuses to open the file at all.

**What to do.** Update the application, or point the database path at a different file in `Settings > Storage`. Note that the installer and an unpacked ZIP share the same `%APPDATA%` folder: a newer unpacked copy migrates the shared database, after which an older installed copy refuses to start.

`Database migration failed`

**Symptom.** A dialog titled `Database migration failed` shows the error; the app exits (exit code 2).

**Cause.** The schema migration threw - typically the database file is not reachable or not writable, is locked, or the disk is full.

**What to do.** Check the database path shown in `Settings > Storage` for reachability and write permission, fix the cause, and start again. If the settings file itself is unreadable, the app runs on defaults and may point at a different database - see "The settings file could not be read".

`Unfinished runs from previous session`

**Symptom.** A three-way dialog at startup lists runs that did not finish cleanly, up to five entries plus `... and N more`. Each line shows the run name, its type and its start time.

**Cause.** The previous session ended without the run reaching a final state (crash, hard kill, power loss). The app finds runs still marked as running and asks what to do with them.

**What to do.**

| Answer | Effect |
|---|---|
| `Yes` | Mark the runs as `Cancelled` and set their finish time to now. |
| `No` | Pause them for a later resume; no finish time is set. |
| `Cancel` / window close | Leave them unchanged - they stay visible and the question returns at the next start. |

If the recovery check itself fails, a warning dialog titled `Crash recovery` appears, the app starts anyway, and the runs stay in the running state.

### The splash stays longer than expected

**Symptom.** `Starting up…` stays on screen for several seconds before the main window appears.

**Cause.** The shell builds all of its page view models in one synchronous call; a large database makes this slower. The splash runs on its own thread precisely so it keeps animating during that time.

**What to do.** Wait. If the splash never closes and no main window appears, see the next entry.

### No window appears after an `Unexpected error` during startup

**Symptom.** An `Unexpected error` dialog appears while the app is starting. After you dismiss it, no main window opens, but the process keeps running.

**Cause.** An exception escaped the startup path. The global handler keeps the process alive and reports the error, but if it happened before the main window was created, there is no window to return to.

**What to do.** End the process ( `aicb-ui.exe` ) in Task Manager and start the app again. If it happens repeatedly, include the `Details:` line and `aicb.log` in a report.

### Auto-backup repeats at every start

**Symptom.** The startup notice bar reports a failed backup, and it does so again at every start.

**Cause.** Auto-backup is enabled and due, and its run failed. A **failed** run never advances the last-backup timestamp, so it is due again at the next start until the cause is fixed. The run itself happens in the background after the main window appears and does not delay the start.

**What to do.** Read the reason in the notice ( `BasePath does not exist: '...'` , `Could not create backup folder: ...` , `Backup failed: ...` ), fix it, or switch auto-backup off in `Settings > Storage` . `Backup Now` runs a backup immediately and shows its result in the panel's status line. Note that an archive that is missing a file that could not be read is written with a `-partial` suffix and **does** advance the timestamp - so it does not repeat every start.

### The settings file could not be read

**Symptom.** The startup notice bar shows `Application settings - Your settings file could not be read, so this session is running on DEFAULTS ...` . Your sessions and solutions may look missing, and the theme may be the default one.

**Cause.** `%APPDATA%\AIContextBuilder\app-settings.json` is corrupt or could not be opened. The app continues with default settings - and the defaults include the database path, so it may open the default database instead of yours. The settings file is never moved by the `Base path` setting; it always lives at the default location.

**What to do.** The message names a preserved copy of the unreadable file, `app-settings.json.corrupt-<timestamp>` , next to the original. Repair that copy and restore your paths, or set the correct database path again in `Settings > Storage` . If no copy could be made, the message says so and warns that the next solution you open will overwrite the file - copy it elsewhere before you continue.

### The app starts with the wrong theme or font size

**Symptom.** The theme or the font size differs from the configured value, although `Settings > General` shows the right setting.

**Cause.** Theme, font and UI zoom are applied during startup as best-effort steps; a failure is caught and **not** reported anywhere. The splash itself always follows the Windows app theme, so it can briefly differ from the app's own theme even on a healthy start.

**What to do.** Restart the app. If the setting is still not applied, re-select it in `Settings > General` (standard controls switch immediately; some custom panels apply fully only after a restart).

### Opening and loading solutions

#### A solution was moved or renamed

**Symptom.** Opening a solution from the recent lists fails with a dialog titled `Open solution` and the text `Solution file not found:` followed by the path.

**Cause.** The saved path no longer exists on this machine.

**What to do.** The app then opens the `Relocate solution` dialog: `Browse...` to the new location of the `.sln` file, or type the path, then `Relocate`. The stored path is updated and all sessions and snapshots of that solution are preserved. `Cancel` leaves the solution in the database with the old path.

#### Choosing a file in the `Open solution` dialog does nothing

**Symptom.** You pick a file, and neither a tab opens nor a message appears.

**Cause.** The chosen path does not exist at the moment it is checked (a typed path, or a network share that went away between selection and check). This path returns silently.

**What to do.** Check the file is really there and try again. The recent-solutions path does better: it reports `Solution file not found` and offers the relocate dialog.

#### Drag and drop does nothing

**Symptom.** Dropping a file onto the window has no effect.

**Cause.** The drop accepts exactly **one** file, and only with the extension `.sln`, `.slnx` or `.slnf`, and only if it exists on disk. Anything else - several files at once, another extension - is discarded without a message.

**What to do.** Drop a single solution file, or use the file dialog. If the drop fails while opening, a dialog titled `Open Solution` reports `Could not open solution:` with the reason.

#### Symbols are missing from the analysis

**Symptom.** After loading a solution, types or projects that exist in the code do not appear in the tree, or queries report fewer symbols than expected.

**Cause.** Roslyn load problems - an unsupported project type, a broken project file, a missing target framework - are reported as warnings to the log, not to the interface. The affected project quietly drops out of the analysis.

**What to do.** Look for `[Workspace]` lines in `aicb.log`. Fix the project so it loads (the same way it would have to load for `dotnet build`), then reload the solution. If the solution defines no `Any CPU` platform and AICB did not load it at the platform it builds at, name that platform in the `designTimeBuildPlatform` key of its `<SolutionName>.aicb.json` file (see Step 3 in "Configuring AICB for a team") and reload. Answer `Yes` when the reload asks whether to re-analyze anyway: the file is not counted as a code change.

Two further causes leave symbols out without a message in the app. A git submodule that was never initialized leaves an empty folder, so the sources a project compiles from it are missing: run `git submodule update --init` in the repository root and reload. A file that a pre-build step or a code generator writes is missing when that step has not run: build the solution once and reload. The MCP tool `get_diagnostics` names both cases (`uninitializedSubmodules`, `missingCompileItems`).

### Working in the app

#### The tab strip is locked while a run is active

**Symptom.** Clicking another Context Builder tab does nothing; the whole tab strip is unresponsive.

**Cause.** Tab switching is disabled while any run is active, so a run state cannot be lost by switching away. The tooltip on the locked strip says `Tab switching locks while a run is active.`

**What to do.** Wait for the run to finish, or cancel it with the run's `Cancel` button; the tab strip unlocks afterwards.

### No red border on an invalid input

**Symptom.** You enter something invalid, click `Save`, and nothing seems to happen.

**Cause.** The app has no field-level validation feedback: no field is outlined or marked. Numeric fields simply refuse non-numeric input as you type. Everything else is validated **when you save**, and the reason is written to the panel's status line, at the bottom of the action bar.

**What to do.** After clicking `Save`, read the status line. A rejected save leaves the value unsaved and shows a message such as `Save failed: Name is required.` The same applies to the Settings pages: the message appears in the action bar, not next to the field.

### Copy does nothing

**Symptom.** `Copy to Clipboard` or `Ctrl+C` appears to succeed, but the clipboard is unchanged.

**Cause.** Another process holds the clipboard (RDP sessions and virus scanners are common causes). Copy retries three times with a 50 ms pause and then gives up silently. Paste is different: it reads once and silently returns no text if that read throws.

**What to do.** Try again a moment later, or close the process that holds the clipboard. There is no message, because the copy path has no return value to report.

### A splitter position is not remembered

**Symptom.** You drag a splitter between panels, restart the app, and the position is back to the default.

**Cause.** Splitter positions are saved on every drag, but the write is best-effort: a failure (missing service, database not migrated, write error) is swallowed. The saved position also takes effect only the next time the view is loaded.

**What to do.** Nothing to repair - re-drag the splitter. If it never survives a restart, check `aicb.log` for write errors on the settings side.

### Closing the app: `Could not check for unsaved changes`

**Symptom.** You close the window and get a dialog titled `Close Application` with the text `Could not check for unsaved changes, so the app was kept open to protect your work:` followed by the error and the advice `Save your sessions manually, then close again.` The app stays open.

**Cause.** The check for unsaved tabs threw. WPF would still close the window after such an error, which would discard every unsaved session - so the app cancels the close instead. Cancelling is the only safe answer to "we could not establish that closing is safe".

**What to do.** Save your sessions manually (see "Workspace, sessions and snapshots" in the desktop app manual), then close the app again.

### The unsaved-changes summary mentions a tree selection that could not be read

**Symptom.** The `Unsaved tabs` dialog lists a line `Tree selection: could not be read - it may hold unsaved overrides`.

**Cause.** Collecting the tree selection failed while the summary was being built. The failure is reported as a named line instead of a missing entry, so the summary can never look complete while being short.

**What to do.** Treat the dialog as a warning that the tree selection may hold unsaved overrides; `Cancel` keeps the tab open so you can save it explicitly.

### Auto-save and unnamed tabs

**Symptom.** Auto-save is enabled, but a tab with unsaved work was not saved.

**Cause.** Auto-save saves tabs that have a session ID. A brand-new tab without a session is skipped, but the LLM send path creates an `Untitled - <solution file> - <date and time>` session automatically; that session has an ID and is covered by later auto-saves even if you never renamed it.

**What to do.** If the tab has never been sent or saved, use `Save Session` to create its session record. If a send already created an `Untitled ...` session, no extra naming step is required for auto-save. If auto-save fails for a session, a dialog titled `Auto-save` reports it once per session, and the status says the changes remain unsaved.

#### The `On startup` setting has no effect

**Symptom.** You choose `Restore last tabs` (or another option) in `Settings > General`, but the app always starts on the Start page.

**Cause.** The setting is stored, but restoring the previous session has not been implemented yet. This is a planned feature.

**What to do.** Nothing yet - save your session before closing if you want to continue where you left off.

### Insights and panels that can look empty or stale

#### The Insights panel is empty and points at the filters

**Symptom.** The Insights tab shows `No insights to show here - pick a category on the left, or relax the principle/severity filters.`

**Cause.** Either there really is nothing to show, or reading the stored insights failed. The empty-state text is the same in both cases, and it names the filters while the cause may be a database read error.

**What to do.** Click `Analyze Solution` to re-evaluate, then check `aicb.log`. Insights run on demand by default; `Auto-trigger insights on solution load` in `Settings > General` switches them to automatic.

#### A dismissed insight comes back

**Symptom.** You dismiss a finding, it disappears, and after the next refresh it is listed again.

**Cause.** The database write for the dismissal failed. The card is removed from the list anyway, so the dismissal looks successful until the list is rebuilt.

**What to do.** Dismiss it again; if it keeps coming back, check `aicb.log` for write errors. Note that a finding the producers mark as persistent cannot be dismissed at all - its dismiss buttons are hidden.

#### The reset button in the Insights action bar seems to do nothing

**Symptom.** The restore button in the Insights toolbar has no visible effect.

**Cause.** The database write failed and the command returned without a message. A successful restore makes dismissed findings and intentionally accepted findings visible again, re-evaluates all sections immediately, and leaves suppressions declared in the committed `<Solution>.aicb.json` sidecar in force.

**What to do.** Trigger it again; if it still does nothing, check `aicb.log`.

#### The model profile picker or the recent runs list is empty

**Symptom.** The model profile dropdown in the Context Builder contains only its neutral default entry, or the recent runs list stays empty.

**Cause.** Reading the profiles or the run history failed; the failure is swallowed and the list stays empty. An empty recent-runs list is indistinguishable from "no runs yet".

**What to do.** Switch to another tab and back to reload, and check `aicb.log`. The send path is unaffected by the empty picker: it falls back to the standard model resolution.

#### The `Solutions` tab shows old data

**Symptom.** The solution list and its counts look unchanged although something was created or deleted in another tab.

**Cause.** The refresh of the tab failed; the previously loaded data stays on screen and looks current.

**What to do.** Use the tab's `Refresh` button, or reopen the tab. If the problem persists, check `aicb.log`. Note that the neighboring labels `Snapshot history could not be read` and `Statistics could not be read` are deliberate and different from `Not analyzed yet`: they say that the read failed, not that nothing was analyzed.

#### The `MCP Usage` panel misses the active-pool marking or stops updating

**Symptom.** In `MCP Usage`, the marking of tools that are in the active pool is missing, or the call log does not refresh.

**Cause.** Reading the active MCP profile or the call log failed; the panel logs the error and keeps its previous state. The panel deliberately reloads on every activation because the MCP server writes into the same database while the app runs.

**What to do.** Reopen the page; if it stays stale, check `aicb.log`.

#### Auto-compression and other settings quietly fall back to defaults

**Symptom.** Node compression does not follow your configured rules, snapshots are not reloaded before applying, or a preselection overview is rendered with compact defaults.

**Cause.** If the provider for the compression rules or the active pipeline profile fails, the app falls back to the built-in heuristic defaults and logs a warning. The overview renderer also falls back to Compact defaults when its configured detail preset cannot be loaded, but that particular fallback is silent and writes no warning.

**What to do.** For compression-rule or pipeline-profile failures, check `aicb.log` for the fallback warning, reopen the settings, and try again. A missing overview detail preset leaves no log entry, so inspect that configuration directly. The fallback keeps the feature working; it does not apply your configuration.

#### The prompt fields stay unchanged after opening a session

**Symptom.** You open a saved session, but the prompt fields still show the previous content.

**Cause.** The stored prompt payload could not be parsed. The fields are deliberately left untouched rather than cleared.

**What to do.** Re-enter the prompt, save the session again, and check `aicb.log`.

### Database, backup and paths

#### Switching the database fails: target is newer than the app

**Symptom.** In `Settings > Storage`, `Switch Database...` refuses with `Target DB schema version <n> is newer than this app's latest (<m>). Upgrade the app or pick a different DB.`

**Cause.** The selected file was written by a newer version of the application.

**What to do.** Update the application, or pick another database file.

#### Switching the database fails: migration error

**Symptom.** The switch fails with `Migration failed: ...` and, depending on the case, `JSON rolled back to previous path.` or `No previous path to roll back to - JSON points to the new (broken) target.` followed, in the worst case, by `Rollback itself failed: ...`.

**Cause.** The pending migrations on the target file could not be applied. The app tries to restore the previous database path in the settings file.

**What to do.** Read the message: it tells you exactly which state the settings file is in. If the rollback succeeded, the previous database is active again; if not, set the path manually in `Settings > Storage`. Background work (runs, analyses) must be idle before a switch.

`Cannot ... while N background task(s) are active`

**Symptom.** `Switch Database...`, `Create New DB...` or saving the base path is refused with `Cannot switch DB while 1 background task(s) are active.` (the action name varies).

**Cause.** A background task is still running - a run or an analysis in the Context Builder, or a backup. The message names the count, not the location.

**What to do.** Finish or cancel the run in the Context Builder, or let the backup finish, then repeat the action.

### A backup archive is named `-partial`

**Symptom.** A file such as `backup-20260921-101500-partial.zip` appears in the backup folder, and the notice or status line lists skipped files.

**Cause.** A file under the base path could not be read and was skipped; skipped files are named in the message (up to five, then `and N more`). The database is not the cause: it is copied as a snapshot while other programs keep it open.

**What to do.** Close the program that holds the file, or accept the partial archive - it still contains everything else. A partial run does advance the backup timestamp, so it does not repeat at every start.

### Changing `Base path` does not move the settings file

**Symptom.** You change `Base path` in `Settings > Storage`, the app accepts it, and `app-settings.json` stays in `%APPDATA%\AIContextBuilder`.

**Cause.** This is by design: the settings file must be readable before the configured path is known, so it always lives at the default location. The base path is honored for the database, templates, node overrides and backups, and it takes effect after a restart.

**What to do.** Use `Settings > Storage` to see which file the app actually uses (`App settings file`). If you want your data somewhere else, change `Base path` and restart; the settings file itself stays where it is.

### The GUI and the MCP server share one database

**Symptom.** The `MCP Usage` page changes on every visit, or a schema migration takes effect immediately in a running process.

**Cause.** The desktop app and the MCP server can point at the same database file, and both write to it while they run. This is the normal case and explains both behaviors: the usage panel reloads on every activation, and a migration applies to every running process at once.

**What to do.** Nothing to fix - but if you want the two fully independent, point them at different database files. Note that the MCP server tolerates a database written by a newer version and reports the drift, while the desktop app refuses to start on one.