designer_action Reference

10.1.5 additions to `designer_action` and related DesignerMCP tools — new `build` action parameters, compile feedback on write/read responses, and a `context_menu` action for grid right-click verification.

10.1.5 adds build-visibility signals to the DesignerMCP surface. The `build` action gains two parameters that mirror the `RuntimeBuildAndPublish` dialog, `write_objects` responses carry per-row `compile` feedback for the four tables that hold user-authored .NET, `get_objects(detail='full')` surfaces a `lastCompile` field on those same rows, and the solution context carries a compact `build_state` section.

The JSON shape returned by the build surface is defined once on MCP SDK Reference — the "`build` Block" section there is the authoritative shape. This page covers only the DesignerMCP-specific entry points.

---

designer_action('build', …) — new parameters

The `build` action invokes the same pipeline as the Designer "Build and Publish" dialog. In 10.1.5 it accepts two named parameters that mirror the dialog's checkboxes.

```
designer_action(
  action:             "build",
  rebuild_all:        bool = false,   // false = incremental, true = full rebuild
  validate_displays:  bool = true     // run Display compile pass in addition to Scripts
) -> { build: <block> }
```

Parameter	Default	Effect
rebuild_all	`false`	When `false`, only objects modified since the last build are recompiled; unchanged objects appear in `summary.skipped`. When `true`, every object in the four affected tables is recompiled regardless of modification state. Mirrors the *Rebuild All* checkbox in the Build and Publish dialog.
validate_displays	`true`	When `true`, `DisplaysList` code-behind is compiled alongside the three Scripts tables. When `false`, only Scripts are compiled (useful when iterating on script logic without wanting the Display compile cost). Mirrors the *Validate Displays* checkbox in the Build and Publish dialog.

The response is the shared `build` block. See MCP SDK Reference for the full shape.

```
# Fast path during iterative development
designer_action('build')
# == designer_action('build', rebuild_all=false, validate_displays=true)

# Pre-deploy check - recompile everything
designer_action('build', rebuild_all=true, validate_displays=true)

# Scripts-only run while debugging script logic
designer_action('build', validate_displays=false)
```

---

write_objects — per-row `compile` field

When a `write_objects` call touches a row in one of the four affected tables (`ScriptsTasks`, `ScriptsClasses`, `ScriptsExpressions`, `DisplaysList`), the response entry for that row carries a `compile` field populated from the save-time incremental compile.

```
{
  "results": [
    {
      "table": "ScriptsTasks",
      "name":  "Line1_Cycle",
      "status": "written",
      "compile": {
        "status": "error",
        "diagnostics": [
          { "line": 12, "msg": "The name 'tagx' does not exist in the current context" }
        ]
      }
    },
    {
      "table": "ScriptsClasses",
      "name":  "TankUtils",
      "status": "written",
      "compile": { "status": "ok", "diagnostics": [] }
    }
  ]
}
```

Writes to rows outside the four affected tables carry no `compile` field — those objects do not participate in the .NET compile pipeline.

The row-level `compile` reports only what the incremental pipeline saw for that single object. Cross-object breakage — for example, an edit to a `ScriptsClasses` row that breaks a `ScriptsTasks` row referencing it — is surfaced by a subsequent `designer_action('build')` call, not by the write response alone.

---

get_objects(detail='full') — `lastCompile` field

When `get_objects` is called with `detail='full'` against one of the four affected tables, each returned row carries a `lastCompile` field holding the most recent compile result for that object.

```
{
  "table": "ScriptsTasks",
  "name":  "Line1_Cycle",
  "code":  "...",
  "lastCompile": {
    "status": "ok",
    "at":     "2026-04-21T16:02:11Z",
    "diagnostics": []
  }
}
```

Field	Meaning
`lastCompile.status`	`ok` or `error`, matching the shared `build` block convention.
`lastCompile.at`	ISO-8601 timestamp of the compile that produced this result.
`lastCompile.diagnostics[]`	Same shape as elsewhere — `line` and `msg` per entry, empty when status is `ok`.

Calls with `detail='summary'` or the default detail level do not include `lastCompile` — it is only emitted on the full-detail path to keep the summary response compact.

SolutionContext / RefreshContext — `build_state` section

The context packages returned to the agent on connect and on refresh now include a compact `build_state` section. This gives the agent a one-shot view of compile health without needing to call `designer_action('build')` at session start.

```
{
  "build_state": {
    "built":     42,
    "failed":    1,
    "skipped":   0,
    "timestamp": "2026-04-21T16:02:11Z"
  }
}
```

The fields match the `summary` portion of the shared `build` block. Per-object diagnostics are not included here — callers that need them issue `designer_action('build')` or `get_objects(detail='full')` on the affected table.

---

designer_action('context_menu', …) — grid right-click verification

The `context_menu` action drives a row's right-click context menu through the same UI surface a Designer operator would use. It serves automated verification workflows that previously had to flag *MANUAL-QA-BENCH-REQUIRED* for any menu-driven step (Tag editor rows, Security Secrets, Devices / Datasets grids, and every other table whose row carries actions accessible only via right-click).

Two modes, discriminated by the presence of a `|` separator in the options string:

```
# Inspect: list the items the context menu would show for this row.
designer_action('context_menu', '<TableType>.<RowName>')
# Invoke: fire the named menu item on this row.
designer_action('context_menu', '<TableType>.<RowName>|<MenuAction>')
```

Inspect form

Returns the snapshot of the menu items after the page's `ContextMenu.Opened` gates fire against the selected row. Items reflect the post-gate state, so visibility and enablement match what a user would see on right-click.

```
{
  "mode": "inspect",
  "table": "SecuritySecrets",
  "row":   "AdminPass",
  "items": [
    { "header": "Copy SecretValue to clipboard",
      "tag":    "TDEV1412_CopySecretValue",
      "isEnabled": true,
      "isVisible": true }
  ]
}
```

Invoke form

Matches `MenuAction` against **`MenuItem.Tag`** first (case-sensitive — Tags are programmer-set stable identifiers), then against **Header text** (case-insensitive). Fires `RaiseEvent(MenuItem.ClickEvent)` on the matched item; the response carries any error the Click handler raised.

```
{
  "mode": "invoke",
  "table": "SecuritySecrets",
  "row":   "AdminPass",
  "match": { "matchedBy": "tag", "tag": "TDEV1412_CopySecretValue" },
  "status": "ok"
}
```

Prefer Tag matching for stable AI scripts — Header text often carries localization or accelerator markers that drift release to release. Tags are author-set in code-behind and stay stable across cosmetic UI changes.

> **Click handlers must not call `ShowDialog` inline.** The verb invokes the Click handler synchronously on the WPF Dispatcher. A handler that opens a modal dialog inline deadlocks the MCP pipe thread. Day-1 wired items (e.g., `SecuritySecrets.CopySecretValue`) only mutate clipboard / object state — safe. If a future Click handler needs to surface a modal, refactor it to `BeginInvoke` the dialog so the menu invocation completes first.

Page coverage (v1)

v1 wires only pages that expose the canonical `GridControlObj` element name — every `W3ContentsControl`-derived grid (Tags, Devices, Alarms, Datasets, Security, Reports, etc.). Display editors and the project tree return `PAGE_NOT_SUPPORTED`; future releases extend the verb's surface area.

Error Codes

Code	When
`MISSING_PARAMETER`	The options string is missing.
`INVALID_PARAMETER`	Options string does not match `<TableType>.<RowName>[|<MenuAction>]`.
`ROW_NOT_FOUND`	The named row does not exist in the table on the active solution.
`NO_DISPATCHER`	WPF dispatcher unavailable (Designer not interactive).
`NO_ACTIVE_PAGE`	No page is currently active in Designer for that table.
`PAGE_NOT_SUPPORTED`	Page does not expose the canonical `GridControlObj` grid (Display editors, project tree, etc.).
`NO_DATA_GRID`	The page exposes `GridControlObj` but it does not host a data grid.
`NO_CONTEXT_MENU`	The grid does not have a context menu attached.
`MENU_ITEM_NOT_FOUND`	Invoke form: no menu item matched `MenuAction` by Tag or Header.
`MENU_ITEM_GATED`	Invoke form: matched item is disabled or hidden under the active permissions. Response includes `isEnabled` / `isVisible` for diagnostics.
`CLICK_HANDLER_EXCEPTION`	The Click handler raised an exception. The original message is surfaced in the response.
`EXCEPTION`	Any other unexpected failure.

Error Codes

Code	When	Recovery
`BUILD_IN_PROGRESS`	Another `build` call is already running for the active solution.	Wait for the in-flight call to complete. The second call does not queue.
`INVALID_PARAMETER`	`rebuild_all` or `validate_displays` is not a boolean.	Response includes an `examples` array with the correct call form.

In this section...
