

TK Annotation Helpers Reference

Reference page for the 10.1.5 server-side script-API helpers `TK.AddAnnotation` and `TK.GetAnnotations` — the headless read/write surface for annotation tables.

[How-to Guides](#) [Solution Examples](#) [Feature Examples](#) [User Interactions Examples](#) [TrendChart Annotations Example](#) TK Annotation Helpers Reference

Version 10.1.5+

Overview

`TK.AddAnnotation` and `TK.GetAnnotations` are static helper methods on the Toolkit class `TK`. They give server-side scripts a headless read/write surface against an annotation `Dataset.Table` — no operator UI, no chart wiring required.

Typical use cases:

- **AI-generated annotations** from a `Server Script.Task` — the script computes an explanation or anomaly note and pins it on the affected tag.
- **Report-time annotations** from a generator that walks a time range and writes summary commentary.
- **Batch annotation imports** from external maintenance, MES, or work-order systems.
- **Programmatic read-back** for analytics, audit, or downstream visualization that does not use the platform's `TrendChart`.

The helpers operate on the same `Dataset.Table` that the `TrendChart` and `DrillingChart` bind through `AnnotationsSource`. Annotations written through `TK.AddAnnotation` render on charts wired to the same source on the next refresh.

TK.AddAnnotation

Inserts a single annotation row into the resolved `Dataset.Table`.

```
public static int AddAnnotation(  
    string    tagName,  
    DateTime  utc,  
    double    value,  
    string    contents,  
    string    source          = "Annotations", // Dataset.Table object name  
    bool      isEvent         = false,  
    double    durationSeconds = 0,  
    string    color           = null,  
    string    title           = null);
```

Parameters

Name	Type	Description
<code>tagName</code>	<code>string</code>	The tag the annotation pins to. Used by the chart at read-back time to decide which pen the pin belongs to. May be empty for chart-level (non-pen-bound) notes.
<code>utc</code>	<code>DateTime</code>	The annotation's timestamp. UTC is required — if a non-UTC value is passed, it is converted before insert.
<code>value</code>	<code>double</code>	The Y-axis value the annotation refers to. Typically the tag's value at <code>utc</code> .
<code>contents</code>	<code>string</code>	The annotation body. Free-form text; renders inside the pin tooltip.
<code>source</code>	<code>string</code>	Optional. <code>Dataset.Table</code> object name. Defaults to "Annotations" — the auto-created table inside the <code>TagHistorian</code> SQLite database. Pass an explicit name (e.g. "MyAnnotations") to write to a customer-managed table.
<code>isEvent</code>	<code>bool</code>	Optional. <code>true</code> marks the row as an event-range annotation; combined with <code>durationSeconds</code> renders a span instead of a pin. Defaults to <code>false</code> .
<code>durationSeconds</code>	<code>double</code>	Optional. Duration in seconds for event-range annotations. Defaults to 0 (point-in-time pin).
<code>color</code>	<code>string</code>	Optional. Pin color hint (e.g. "#FFAA00"). When null, the chart's default annotation color applies.
<code>title</code>	<code>string</code>	Optional. Short title shown on the pin in addition to <code>contents</code> . When null, only <code>contents</code> render.

Return value

Returns 0 on success, non-zero on failure (most commonly: source unresolved, target Dataset.Table not writable, SQL error). The helper **never throws** — it logs the failure to the platform trace log at Warning level and returns the failure code. Callers should check the return value before assuming the annotation persisted.

Example: AI-generated annotation from a Server Script.Task

```
// Server-side AI-generated annotation -- triggered when the anomaly score
// crosses a threshold. Runs in a Server Script.Task whose Trigger is set to
// the AnomalyScore tag.

string aiText = "Anomaly detected: temperature drift exceeds normal range.";
double currentTemp = Tag.Mixer.Heater.Temperature;

int status = TK.AddAnnotation(
    "Mixer.Heater.Temperature", // pin to the temperature pen
    DateTime.UtcNow,
    currentTemp,
    aiText);

if (status != 0)
{
    TK.LogException(new Exception("AddAnnotation failed status=" + status));
}
```

TK.GetAnnotations

Reads annotations from the resolved Dataset.Table for a given tag and time window.

```
public static DataTable GetAnnotations(
    string tagName,
    DateTime startUtc,
    DateTime endUtc,
    string source = "Annotations");
```

Parameters

Name	Type	Description
tagName	string	Filter to annotations bound to this tag. Pass null or empty to retrieve all annotations in the time window across all tags.
startUtc	DateTime	Inclusive lower bound on the annotation timestamp. UTC required.
endUtc	DateTime	Inclusive upper bound on the annotation timestamp. UTC required.
source	string	Optional. Dataset.Table object name. Defaults to "Annotations".

Return value

Returns a DataTable in the projected consumer column shape (the same shape the chart's read-back path consumes). Returns null when the source cannot be resolved (typo in name, Dataset.Table not configured, etc.); returns an empty DataTable when the source resolves but no rows match. Always check for null before iterating.

Example: read-back for an audit report

```

DateTime end    = DateTime.UtcNow;
DateTime start = end.AddHours(-24);

DataTable rows = TK.GetAnnotations(
    "Mixer.Heater.Temperature",
    start,
    end);

if (rows == null)
{
    TK.LogException(new Exception("GetAnnotations: source unresolved"));
    return;
}

foreach (DataRow r in rows.Rows)
{
    DateTime ts      = (DateTime)r["DateTime"];
    string contents = r["Contents"] as string;
    // ... emit to report
}

```

Async overloads

Both helpers expose `async/await` overloads for callers running in async contexts (e.g., async `Script.Tasks`, async event handlers):

```

public static Task<int>      AddAnnotationAsync(
    string tagName, DateTime utc, double value, string contents,
    string source = "Annotations", bool isEvent = false,
    double durationSeconds = 0, string color = null, string title = null);

public static Task<DataTable> GetAnnotationsAsync(
    string tagName, DateTime startUtc, DateTime endUtc,
    string source = "Annotations");

```

The synchronous overloads are wrappers around the async ones using the platform's standard sync-over-async helper (no `.Result / .GetAwaiter()` / `GetResult()` deadlock risk).

Default source behavior

When `source` is omitted (or passed as `"Annotations"`), the helper resolves the auto-created `Annotations` table inside the `TagHistorian SQLite` database. This is the same table the `TrendChart`'s zero-config quick-start path uses — pins written from a `Script.Task` render on charts wired to `AnnotationsSource = "Annotations"` on the next refresh, with no extra wiring.

Customers managing their own annotation table pass the `Dataset.Table` object name explicitly:

```

// Write to a customer-managed annotation table
TK.AddAnnotation(
    "Reactor.PressureSetpoint",
    DateTime.UtcNow,
    Tag.Reactor.PressureSetpoint,
    "Setpoint adjusted per shift handover note",
    source: "MaintenanceAnnotations");

```

Cross-reference

For chart-side configuration — the `AnnotationsSource` binding, the `AnnotationsEnabled` opt-out, the four `*Method` override hooks, and the auto-created table's full 26-column SQL schema — see [Trend Annotations Reference](#).

In this section...