Skill C# Foundations

title: "C# Foundations — Compile Rules, TK Toolkit, and Runtime Collection Access" tags: [csharp, vb, scripts, codebehind, tk, reference, foundations] description: "Reference for the FX-specific C# patterns: @ namespaces, compile subset rules, TK toolkit index, T.Modules cast catalog, runtime collection access, and Tag default-Value semantics." version: "1.0" author: "Tatsoft"

What This Skill Does

Reference for the FX-specific C# patterns that diverge from textbook .NET: the `@` namespaces, the compile-subset rules, the `TK` toolkit index, the `T.Modules` cast catalog, runtime collection access patterns, and Tag default-Value semantics. This is a lookup reference, not a how-to walkthrough. Companions:

- **Skill Scripts Expressions** — how to build Tasks, Classes, Expressions, and CodeBehind end-to-end
- **Skill Display Construction** — Display authoring and Display CodeBehind walkthroughs

When to Use This Skill

Use when:

- Before writing a Script Task, Class, or CodeBehind body — load this first so the compile-subset rules and the `@`-prefix rule are in mind
- When the compiler returns a misleading diagnostic (CS0106 on a member declaration, CS0103 on an `@`-prefixed identifier, "dynamic" rejection)
- When iterating a runtime `ListObj` collection or looking up by name
- When calling a `TK` toolkit helper and needing the surface index

Do NOT use for:

- How-to walkthroughs of Tasks / Classes / Expressions — see **Skill Scripts Expressions**
- Display drawing patterns or Display CodeBehind end-to-end — see **Skill Display Construction**
- ML.NET, MQTT, or other connector-specific C# — those skill families own their own code patterns

The @ Namespaces

Twelve runtime namespaces are reachable from any script body using the `@` prefix. The `@` is **mandatory** — without it, the compiler treats the identifier as a local variable and rejects member access.

Namespace	What It Holds	Where It Runs
`@Tag`	Tag values, qualities, timestamps, members	Server tasks and CodeBehind
`@Server`	Global server runtime properties	Server only
`@Info`	Solution metadata, build info, platform paths	Both
`@Alarm`	Alarm groups, items, areas	Server only
`@Client`	Client session: navigation, dialogs, query params, log-on	CodeBehind / client tasks only
`@Security`	Users, identity providers, sessions, policies	Most accessors server-only; `@Security.LogOnWithTokenAsync` is client-side
`@Display`	Display registry, layouts	Both
`@Script`	Script registry — reach Class methods via `@Script.Class.<Name>.<Method>`	Both
`@Dataset`	Query and table results, DB providers	Server only
`@Historian`	Historian groups and tags	Server only
`@Report`	Report forms, web-data forms	Server only
`@Device`	Device channels, nodes, points	Server only

Execution-scope decision rule. Picking the wrong scope fails at runtime, not compile-time — the compiler accepts the call; the runtime rejects it.

- **Server-only** (`@Alarm`, `@Device`, `@Dataset`, `@Historian`, `@Report`, `@Server`): callable from Tasks, Classes (Server domain), Expressions. Calling from CodeBehind fails.
- **Client-only** (`@Client`): callable from CodeBehind and client-domain Classes. Calling from a Server Task fails.
- **Both** (`@Tag`, `@Info`, `@Display`, `@Script`, `@Security`): work from either context, with minor caveats per namespace.

The Five Compile-Subset Rules

FrameworX scripting compiles a **subset** of .NET. Five constraints fail at compile or behave differently from textbook .NET:

1. **The `dynamic` keyword is not supported.** The compiler treats `dynamic` as `object` and rejects member access. Always use explicit `as <FullyQualifiedType>` casts (see Runtime Collection Access for the canonical pattern).
2. **The `@` prefix is mandatory.** When accessing FX runtime objects in code, the prefix is required. Without it the compiler treats the identifier as a local variable and CS0103 / CS0117 surface.
3. **Tag default property is Value.** `@Tag.X = 5` writes the value. `@Tag.X = @Tag.Y` *copies* the current value (not a reference, not a live binding). Other namespaces have no default property — `@Server.*`, `@Alarm.*`, `@Display.*`, `@Script.*` all require explicit property names.
4. **Iteration over runtime `ListObj` collections needs an explicit cast per element.** Bare `foreach (var x in @Alarm.Group) x.ActiveCount` fails — the loop variable is the base type and the member is not visible. See Runtime Collection Access below for the cast pattern.
5. **`async`/`await` is supported in Tasks and CodeBehind handlers, NOT in `ScriptsExpressions`.** Expressions are single-line VB.NET expressions — use synchronous patterns. The wrapper makes Task / CodeBehind bodies await-able implicitly.

Code Body Shape — What Goes Inside Contents

The single most common first-pass error is putting a `class`, `namespace`, method declaration, or `using` / `Imports` declaration inside `Contents` for the wrapped shapes. The FX compiler *wraps* your body in a generated class at compile time — with one exception.

Shape	What `Contents` Holds	Wrapped?
`ScriptsTasks`	Method body only — raw statements (`int x = 5; @Tag.Total = x;`)	Yes
`ScriptsClasses` (`ClassContent = Methods`, the default)	Method members only (`public static int Add(int a, int b) { ... }`) — no surrounding `class` declaration	Yes
`ScriptsClasses` (`ClassContent = Namespace`)	A FULL `namespace X { public class Y { ... } }` declaration. Inline `using` above the namespace is accepted (duplicates surface as CS0105 warning, not error).	No — the lone exception
`DisplaysList` / `DisplaysSymbols` (CodeBehind)	Method members of the generated Display class — lifecycle methods and click handlers	Yes
`ScriptsExpressions.Expression`	A single VB.NET expression that returns the value to assign to `ObjectName` (no semicolon, no `return` keyword)	Yes

Where do `using` declarations go? For wrapped shapes (everything except `ClassContent = Namespace`), put them in the `NamespaceDeclarations` field on the row — semicolon-separated, e.g. `System.Linq;System.Text;T.Toolkit.LocalAI`. **Never** inline inside `Contents` — the wrapper rejects them.

Diagnostic to recognize. Putting a method declaration like `public void Initialize() { ... }` inside a wrapped `Contents` fails with the misleading `CS0106 "modifier public is not valid for this item"`. That is the wrapper rejecting a nested method — the fix is to remove the method-declaration wrapper from your body, not to remove the `public` modifier.

Runtime Collection Access

Runtime collections like `@Security.IdentityProviders`, `@Alarm.Group`, `@Security.User`, `@Device.Channels` are typed `ListObj<T>`. Three access patterns cover every case.

Iterate with foreach + cast

LINQ (`.First()`, `.Where()`) does not work — `dynamic` is unsupported, so you must cast explicitly:

```
foreach (var item in @Security.IdentityProviders) {
    var idp = item as T.Modules.Security.SecurityIdentityProvider;
    if (idp == null) continue;
    if (idp.Active && idp.AuthType == eIdentityProviderAuthType.OIDC) {
        // use idp.Name, idp.Authority, idp.ClientId, etc.
    }
}
```

Always null-check the `as` cast — it returns null rather than throwing.

Direct lookup by name

The indexer returns the typed row directly. Returns null when the named row does not exist — null-check before accessing properties:

```
@Security.IdentityProviders["Keycloak"].Authority
@Alarm.Group["Critical"].ActiveCount
@Security.User["Operator1"].UserPermission
```

Row class namespaces

The full `T.Modules.<Module>` namespace is required because Display and Script CodeBehind compile *without* `using T.Modules.*` by default. Common row classes:

Namespace	Row classes

`T.Modules.Security`	`SecurityIdentityProvider`, `SecurityUser`, `SecurityRole`, `SecurityPolicy`
`T.Modules.Alarm`	`AlarmGroup`, `AlarmItem`, `AlarmArea`
`T.Modules.Device`	`DeviceChannel`, `DeviceNode`, `DevicePoint`
`T.Modules.Historian`	`HistorianTag`, `HistorianGroup`
`T.Modules.Dataset`	`DatasetQuery`, `DatasetTable`, `DatasetDBProvider`
`T.Modules.Display`	`Display`, `DisplayLayout`, `DisplaySymbol`

TK Toolkit Index

TK is the script toolkit class, reached directly from any script body as `TK.X(...)` — no `@Module` prefix. It hosts the canonical script-API surface for tag historian queries, tag and UDT operations, object-name resolution, UNS walk helpers, type conversion, and diagnostics.

In 10.1.5 the surface is organized into **capability groups**: `TK.Data`, `TK.Convert`, `TK.Run`, `TK.History`, `TK.Visual`, `TK.Util`. New code uses these grouped forms. The legacy flat `TK.<method>` shape remains supported for backward compatibility — existing customer scripts compile unchanged.

Bucket	What It Covers
`TK.Data`	Tag I/O (`GetObjectValue`, `SetObjectValue` sync+async), asset resolution (`Asset`, `AssetValue`, `AssetInt`, `AssetDouble`), tag/UDT bulk operations (`CopyTagToTag`, `CopyTagToDataTable`, `CompareTag`), members metadata (`GetMembers`, `GetMembersInfo`, `GetTagChildren`)
`TK.Convert`	Type conversion (`To<T>`, `ToInt`, `ToDouble`, `ToDateTime`, `ToTimeSpan`), XML/JSON serialization (`LoadFromXml`, `SaveToXml`, `JsonString`, `JsonValue<T>`), file/snapshot/zip/PDF (`SaveSnapshotTags`, `FileSaveAs`, `ZipFile`, `SaveImageAsPdf`), units conversion, image-bytes/stream
`TK.Run`	Expression engine (`CompileExpression`, `EvaluateExpression`), ScriptClass invoke (`ExecuteClassMethodOnServer` sync+async), Python shell, sync markers and task events
`TK.History`	Tag historian query (`GetTagHistorian`, `GetTagsHistorian`, `GetValueFromHistorian`, `GetValuesFromHistorian` — sync+async), `AddValuesToTagHistorian`, annotation read/write (`AddAnnotation`, `GetAnnotations`)
`TK.Visual`	Image-bytes-to-shape rendering (`ApplyImageBytesToShape`), UNS visual report (`GenerateUnsVisual`), alarm bulk-treat (`BulkTreatAlarm`)
`TK.Util`	Diagnostics (`Trace`, `LogException`), name helpers (`NormalizeObjectName`, `GetLastNode`, `DoubleQuotes`), randomness (`RandomNext`, `NextRandomDouble`), `GeneratePassword`, `GetCurrentPlugins`
`TK.UnsHelpers`	UNS table walk — `EnumerateUnsRows`, `ListUnsTagsOfType` (design-time metadata)
`TK.UnsReferences`	Runtime Reference traversal — `Reference.Link` / `LinkObj`, `ObjRef` cross-Reference walk. License-gated; ships with sector Packs.
`T.Toolkit.LocalAI.AI`	Local AI atomic LLM calls — `AI.Execute(query)`, `AI.ExecuteAsync(query)`. Add `T.Toolkit.LocalAI` to `NamespaceDeclarations` on the script row.

The legacy `TK.AIExecute` shim is deprecated and surfaces one obsolete warning per call site — new code uses `AI.Execute` from `T.Toolkit.LocalAI`.

Tag Default-Value Semantics

`@Tag.X` resolves to `@Tag.X.Value` by default in assignments and expressions — `@Tag` is the one namespace with a default property. Read and write both read from / write to `.Value`.

```
double level = @Tag.Plant/Tank1/Level;             // reads .Value
@Tag.Plant/Tank1/SetPoint = 75.0;                  // writes .Value
@Tag.Plant/Tank1/SetPoint = @Tag.Plant/Tank1/Level; // copies current .Value (not a binding)
```

To access other members, name them explicitly:

```
int quality = @Tag.Plant/Tank1/Level.Quality;
DateTime ts = @Tag.Plant/Tank1/Level.Timestamp;
@Tag.Plant/Tank1/Level.Forced = true;
```

Other namespaces (`@Server.*`, `@Alarm.*`, `@Display.*`, `@Script.*`, etc.) require explicit property names — there is no default. To get a *live binding* between two tags rather than a one-time copy, use a `ScriptsExpressions` row, not a Task assignment.

CodeBehind Lifecycle (Quick Reference)

Display CodeBehind runs on the client. Lifecycle methods — declare as `void` members of `Contents` with these exact signatures:

Method	When It Runs

`void DisplayOpening()`	Once, before the page becomes visible — init, pre-load
`void DisplayIsOpen()`	Periodically while open (interval = `IsOpenInterval` ms, default 1000)
`void DisplayClosing()`	Once, when the page is closing — cleanup, save
`void DialogOnOK()`	Dialog-mode display only — user clicks OK

Inside any of these, `this` is the Display instance — the wrapper exposes `public TDisplayControl CurrentDisplay`. Reach controls placed on the display via:

```
var chart = this.CurrentDisplay.GetControl("<Uid>") as T.Wpf.RunControls.Charts.TTrendChart;
if (chart != null) { /* ... */ }
```

Controls are **not** auto-exposed by Name — `GetControl` uses the element `Uid` (visible in the Designer property panel). Element click handlers: declare as `public void OnButtonClick() { ... }` in `Contents`, then wire from the element via `ActionDynamic { ActionType: 'RunScript', ObjectLink: 'OnButtonClick' }`.

For the full Display authoring workflow, see **Skill Display Construction**.

Common Pitfalls

Pitfall	Symptom	Fix
`public void Foo() { ... }` inside S`criptsTasks.Contents`	CS0106 "modifier public is not valid for this item"	Remove the method-declaration wrapper. `Contents` is the body ONLY for wrapped shapes.
Inline `using System.Linq;` in `Contents`	Compile fails — wrapper rejects the directive	Move usings to the `NamespaceDeclarations` field (semicolon-separated).
`@Alarm.Group["X"].ActiveCount` without null-check	`NullReferenceException` when "X" does not exist	Capture and null-check: `var g = @Alarm.Group["X"]; if (g == null) return;`
`foreach (var x in @Alarm.Group) x.ActiveCount`	Compile fails — base type lacks `Activ`eCount	Cast each iteration: `var g = x as T.Modules.Alarm.AlarmGroup; if (g != null) { ... g.ActiveCount ... }`
`@Alarm.Group.Where(g => g.ActiveCount > 0)`	LINQ on `ListObj` fails — `dynamic` unsupported	Iterate with `foreach` and `as` cast.
`@Tag.X = @Tag.Y` expecting a live binding	Copies the current value only — not a reference	Use a `ScriptsExpressions` row for a live binding.
`dynamic obj = TK.Data.GetObjectValue("Tag.X")`	Compiler treats `dynamic` as `object`, member access rejected	Cast explicitly: `var v = TK.Data.GetObjectValue("Tag.X") as double?;`
Calling `@Client.OpenDisplay(...)` from a server `Task`	Runtime fails — `@Client` is client-only	Move the call into CodeBehind, or set a tag the client watches.
Calling `@Alarm.Group[...]` from CodeBehind	Runtime fails — `@Alarm` is server-only	Move logic to a Server Task or Script Class; surface the result via a tag.
Tag path with `/` in Python	Python rejects `/` in identifiers	Use _ instead: `Tag.Plant_Tank1_Level.Value`
`TK.AIExecute(query)` on new code	Obsolete warning per call site	Use `AI.Execute(query)` with `T.Toolkit.LocalAI` in `NamespaceDeclarations`.

Related Skills

- **Skill Scripts Expressions** — end-to-end Tasks, Classes, Expressions, and CodeBehind walkthroughs
- **Skill Display Construction** — Display authoring, layout, and the full CodeBehind workflow
- **Skill New Solution** — starter solution with tags, simulator, historian, alarms, dashboard