Scripts Tasks Reference

Schedule and run script tasks.

Scripts Tasks (Reference) are automated program units that execute in response to events or on scheduled intervals, providing the core automation logic for your FrameworX solution. ScriptTask is a self-contained code blocks that:

- Execute when triggered by events or time intervals
- Run in their own threads for optimal performance
- Access all solution objects and namespaces
- Support VB.NET, C#, and Python languages

Unlike Script Classes, Tasks don't require method declarations - you write directly the code to be executed, similar to the contents of a method body.

Creating Script Tasks
Configuration Properties
Code Structure
Built-in Tasks
Task Triggers
Execution Model
Runtime Namespace
Using External Resources
Language-Specific Features
Best Practices
Troubleshooting

Creating Script Tasks

1. Navigate to **Scripts Tasks**
2. Click the **plus icon**
3. Configure:
    - **Name**: Task identifier
    - **Description**: Purpose of the task
    - **Language**: VB.NET, C#, or Python
    - **Code Source**: New or Import from Library
4. Click **OK**
5. Double-click the row to open Code Editor

Configuration Properties

Property	Description	Required
Name	Task identifier used in runtime references	Yes
Code	Programming language (VB.NET, C#, Python)	Yes
Domain	Execution location: Server (default) or Client	Yes
Trigger	Event that initiates task execution	No
Period	Time interval for periodic execution	No
InitialState	Enabled (ready to run) or Disabled	No
EditSecurity	Permission level required to modify	No
Description	object description	No

ReadOnly Properties

Property	Description
ID	Unique task identifier
VersionID	Modification version counter
Lines	Code line count
BuildStatus	? Green = Success, ? Red = Errors (Read Only)
BuildErrors	Compilation error count (Read Only)
BuildMessage	Compilation messages

Category	Task grouping category
LockState	Edit lock status
DateCreated	Creation timestamp
DateModified	Last modification timestamp

Code Structure

> **Important**: Script Tasks contain only the method body code - no function declarations or wrappers. The platform automatically handles method creation, exception protection, and thread management.

Correct Task Format

csharp

```csharp
// Direct code - NO method declaration
double temperature = @Tag.Temperature;
if (temperature > 100)
{
    @Tag.AlarmActive = true;
    @Alarm.Notify("High Temperature");
}
```

Incorrect Format

csharp

```csharp
// ? WRONG - Do not include method wrapper
public void ProcessTemperature()
{
    // Code here
}
```

Built-in Tasks

All solutions include predefined tasks:

Server Startup

- **Executes**: When TServer.exe starts
- **Purpose**: Initialize server-side components
- **Runs before**: All other server tasks
- **Use for**: Global initialization, connection setup

Server Shutdown

- **Executes**: When solution stops
- **Purpose**: Cleanup and finalization
- **Use for**: Closing connections, saving state

Client Startup

- **Executes**: When each client connects
- **Runs on**: TRichClient.exe, TSmartClient.exe, or Web browser
- **Purpose**: Initialize client-specific settings
- **Use for**: UI setup, local preferences

Client Shutdown

- **Executes**: When client closes
- **Purpose**: Client cleanup
- **Use for**: Saving user preferences, cleanup

Task Triggers

Event-Based Triggers

Tasks execute when specific events occur:

- **Tag Change**: When tag value changes
- **Alarm State**: When alarm activates/deactivates
- **Display Events**: User interactions
- **Custom Events**: Application-specific triggers

Time-Based Execution

Configure periodic execution:

- **Period**: Interval in milliseconds
- **Options**:
  - Fixed interval (e.g., every 1000ms)
  - Scheduled times (using expressions)
  - Calendar-based triggers

Execution Model

Threading

- Each task runs in its own thread
- Multiple tasks can execute simultaneously
- No blocking between tasks
- Automatic thread pool management
- On a license other than Enterprise, or when multi-threading is disabled, periodic tasks are queued instead of running concurrently

Calling Script Classes From Tasks

A Script Class is one shared instance within its Domain. When two tasks call the same class method at the same time, both calls run the method body on the same object, and the platform does not serialize them. Guard any state the method shares, such as a once-only load flag or a cached table, with `lock` in C#, `SyncLock` in VB.NET, or an atomic once-flag. The task queue on non-Enterprise licenses does not protect calls from expressions, events or other classes. See Scripts Classes Reference. This behavior applies to FrameworX 10.1.5 and 10.1.6.

Spawning Background Work From a Script Task

When a Script Task needs to fan out work to a background thread, prefer `TK.CreateTaskEvent` over directly calling `Task.Run(async () => ...)`.

Code launched with `Task.Run` from inside a Script Task runs on a thread that has not necessarily attached to the runtime context yet. Server-loaded properties such as `@Info.Solution.SolutionName` and other `@Info.*` values may read as empty strings (or stale defaults) for several hundred milliseconds - on large solutions this gap can exceed 2 seconds. The same warning applies to any namespace whose value is populated by the runtime context, not just `@Info.*`.

`TK.CreateTaskEvent` posts a parameter object into the named Script Task's own event queue. The platform invokes the task body only after the host context is ready and the property values it depends on have been resolved, so the body can read `@Info.*` and similar properties without the do-while-empty workaround.

Preferred

csharp

```csharp
// Inside a Script Task: package the inputs and queue an event on a target task.
var parameters = new { tableName, update };
// "Threads" is the name of an existing Script Task that does the actual work.
TK.CreateTaskEvent("Threads", parameters, false);
```

Avoid

csharp

```csharp
// Anti-pattern: launching work directly with Task.Run from a Script Task.
// The task body may execute before @Info.* properties are loaded on this thread,
// returning empty strings or stale values for hundreds of ms (more on large solutions).
tasks.Add(Task.Run(async () =>
{
    // @Info.Solution.SolutionName may still be empty here.
    DataTable table = await LoadTableAsync(tableName, update);
}));
```

Method Signature

csharp

```
public static int CreateTaskEvent(
    string taskName,
    object obj,
    bool isSequential = true,
    bool addToLast = true,
    int priority = 0);
```

Parameter	Description
taskName	Name of the Script Task that will receive the event. Pass the leaf name (the trailing segment after the last `.` is used).
obj	Anonymous or named object whose fields are passed to the receiving task. Retrieve inside the task body with `TK.GetTaskEvent ("<taskName>")`.
isSequential	When `true`, the event runs only after the previously queued event for this task finishes. When `false`, events for this task may interleave on the thread pool.
addToLast	When `true`, append to the tail of the queue; when `false`, insert at the head.
priority	Thread priority. `0` = Normal, `1` = AboveNormal, `2` = Highest.

Return value	Meaning
0	Event queued.
1	Client-domain Script Tasks cannot create events; call from a server-domain task.
2	Script Task Server is not running.

> **When the Task.Run pattern is fine.** Pure CPU-bound work that does not touch `@Info.*`, `@Tag.*`, or other runtime-context-loaded values can still use `Task.Run`. The hazard is specifically reading runtime properties from a thread the platform has not yet bound to the runtime context.

Exception Handling

- Platform provides automatic exception protection
- Errors logged but don't crash the application
- Task continues next execution cycle
- Access error details via BuildErrors

Domain Execution

> **Default Configuration**: Client-side tasks are disabled by default in Solution Settings because web clients don't support them yet. Most solutions use only server-side tasks.

Server Domain (Default):

- Executes on server computer (TServer.exe)
- Global scope across all clients
- Access to all server resources
- Typical use: Business logic, calculations, data processing

Client Domain (When Enabled):

- Executes on each client independently
- Local scope per client
- Access to display context
- Only for WPF clients (not web)

Runtime Namespace

Access task runtime information via `@Script.Task` namespace:

csharp

```
// Access task properties
int count = @Script.Task.MyTask.ExecutionCount;
TimeSpan cpu = @Script.Task.MyTask.LastCPUTime;
bool enabled = @Script.Task.MyTask.Enabled;

// Control task execution
@Script.Task.MyTask.Run();  // Force execution
@Script.Task.MyTask.Enabled = false;  // Disable task
```

Task Runtime Properties

Property	Type	Description
`ExecutionCount`	Integer	Number of executions
`LastCPUTime`	TimeSpan	CPU time of last execution
`Disable`	Boolean	Task disable state
`IsRunning`	Boolean	Currently executing
`LastError`	String	Last execution error

Using External Resources

Calling Script Classes

csharp

```
// Direct access to Script Classes
var result = @Script.Class.Calculations.Process(100);
```

Using External DLLs

1. Add reference in Scripts References
2. Add namespace via Namespace Declarations button
3. Use in task code

Accessing Solution Objects

csharp

```
// All namespaces available
@Tag.Temperature = 25;
@Alarm.Reset("Area1");
var data = @Dataset.Query("SELECT * FROM Production");
@Display.MainScreen.Open();
```

Language-Specific Features

VB.NET

vbnet

```
' Use Exit Function to stop execution
If temperature > 100 Then
    Exit Function
End If
```

C#

csharp

```
// Use return to stop execution
if (temperature > 100)
{
    return;
}
```

Python

python

```python
# Standard Python syntax
if temperature > 100:
    return
```

Best Practices

- [ ] **Keep tasks focused** - Single responsibility per task
- [ ] **Handle exceptions** - Even with automatic protection, validate inputs
- [ ] **Use appropriate triggers** - Event-based for reactive, time-based for polling
- [ ] **Monitor performance** - Check ExecutionCount and LastCPUTime
- [ ] **Document purpose** - Use Description field
- [ ] **Test thoroughly** - Verify BuildStatus before deployment
- [ ] **Consider threading** - Tasks run independently, avoid shared state issues. Guard state shared through Script Classes

Troubleshooting

Task not executing:

- Check InitialState is Enabled
- Verify trigger configuration
- Review BuildStatus for errors
- Check Domain setting matches deployment

Performance issues:

- Monitor LastCPUTime for slow tasks
- Check ExecutionCount for excessive runs
- Consider longer Period for time-based tasks
- Review threading conflicts

Compilation errors:

- Double-click BuildStatus red X
- Check namespace declarations
- Verify referenced objects exist
- Ensure no method wrappers included

In this section...
