AmiumScripter – Architecture and Concept Description

# 1. Purpose

AmiumScripter is a modular, dynamic UI and signal framework for technical and scientific applications. It focuses on maximum flexibility, runtime dynamism, and a clean separation of logic, visualization, and control.

# 2. Component Overview

* a) Projects & Pages

- A project consists of multiple "Pages" – functional units with their own logic and interface.  
- Each page lives in its own directory (Pages/PageName/) and consists of three files:

• Page.cs: Workflow, business logic, signal definitions (no UI)  
 • controls.cs: Visual controls, pure layout (generated/editable in the editor)  
 • view.cs: UI logic (behavior, events, visual logic)

* b) Central Control

- Project.cs: Manages all Pages and Views, instantiation, initialization, start/stop.  
- UIEditor: Visual editor for controls, tab management, drag & drop.  
- SignalManager/DataStorage: Central management and pooling of all signals.

* c) Dynamic Build System

- Uses Roslyn to compile and load Pages and Controls at runtime.  
- Dynamic assemblies allow new/edited Pages to be available immediately in the running system.

* d) Thread, Task and Class Management

- AThread and ATask provide centralized management and lifecycle control for background jobs. They enable monitoring, registration, and clean termination of all running threads and tasks.  
- The TokenManager enables controlled, global cancellation of background operations during project rebuilds or unloads.  
- ClassManager (BaseClass):  
 - The ClassManager automatically registers all classes that inherit from BaseClass (such as workers, managers, or other runtime objects).  
 - All active BaseClass instances are centrally listed and managed, making it possible to enumerate, monitor, and clean up all objects during destroy/rebuild cycles.  
 - During destroy or rebuild, ClassManager ensures that all BaseClass instances are notified and can release resources, stop operations, or close open handles.

# 3. Core Principles & Advantages

- Separation of logic, layout, and UI behavior: increases maintainability, testability, and reusability.  
- Signal-based data model: All data flows via centrally managed signals.  
- Runtime dynamism: Changes to controls, pages, layouts possible without app restart.  
- Clean resource management: Centralized handling of all tasks, threads, and tokens.  
- Automated refactoring enforcement: Build checker prevents critical coding patterns.

# 4. Technical Interplay

Loading process:  
1. Build/Compile: All pages/views/controls are compiled dynamically.  
2. Project.cs: Instantiates objects, adds them to dictionaries, calls Initialize()/Run().  
3. UIEditor: Binds the views into the UI (tabs).  
4. SignalManager: Provides the data pools.  
5. Background threads are started via AThread/ATask and can be stopped centrally at any time.

# 5. Typical Data and Signal Flow Diagram

The following diagram shows the interaction between Page, View, Controls, and the central SignalPool:

TBA

# 6. Extensibility

- New pages can be added via editor or code.  
- Controls can be added or modified via drag & drop, code, or editor.  
- Signal connectivity can be integrated via custom SignalClients/AClients.  
- Robustness through hot-rebuilds, central unload, and build checker.

# 7. Best Practices

- All logic involving threads/tasks/signals should use the helpers (AThread, ATask, TokenManager).  
- Dumb controls, smart views, intelligent pages.  
- No business logic in the UI, no UI logic in the page.  
- For "while(true)", always use IsRunning or CancellationToken.

# Conclusion

AmiumScripter offers a highly flexible, robust, and team-ready architecture for modular, dynamic applications with clear separation of all responsibilities.