Working with Signals: Integration Patterns

In this system, there are three main ways to interact with signals. Understanding these patterns helps you write reliable and maintainable code when handling data streams and signal management.

# 1. Direct Creation

If you create a new signal directly in your code using the Signal constructor, the signal is automatically registered in the SignalPool and becomes globally available throughout the system.

Example:

public Signal MySignal = new Signal(  
 name: "mySignal",  
 text: "MySignal",  
 unit: "°C",  
 format: "0.00",  
 value: 23.5  
);

# 2. Registration via AClient

When using an AClient, your code delegates the registration to the SignalManager, which then adds the signal to the SignalPool. No explicit Signal instance exists in your code—all registration and management is handled centrally.  
  
The SignalManager will also normalize the signal name to PascalCase. For example: myClient.Sig1 will become MyClient\_Sig1.

Example:

MyAClient.Push(new Signal(  
 name: "myClient.Sig1",  
 unit: "°C",  
 format: "0.00",  
 value: 23.5  
));

# 3. Access by Name

You can retrieve a reference to any already-registered signal by its name from the SignalPool. This does not create a new signal instance; it simply provides a reference (pointer) to the existing signal object.

Examples:

// Static access  
public Signal MySignal = SignalPool.MyClient\_Sig1;  
  
// Dynamic access  
public Signal MySignal = SignalPool["MyClient\_Sig1"];

Summary:  
- Direct creation provides you with a code-visible signal instance.  
- AClient registration manages signals centrally, without direct instances in your code.  
- Access by name is used to get a reference to any existing signal in the pool.

(Optional: Add a diagram or table to visualize these relationships if needed.)