Choosing a Software Architecture for Your Next Embedded Application

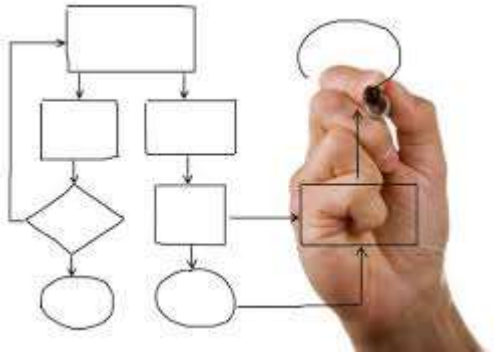
Liam Adams

Applications Engineering Specialist

NATIONAL INSTRUMENTS

Outline

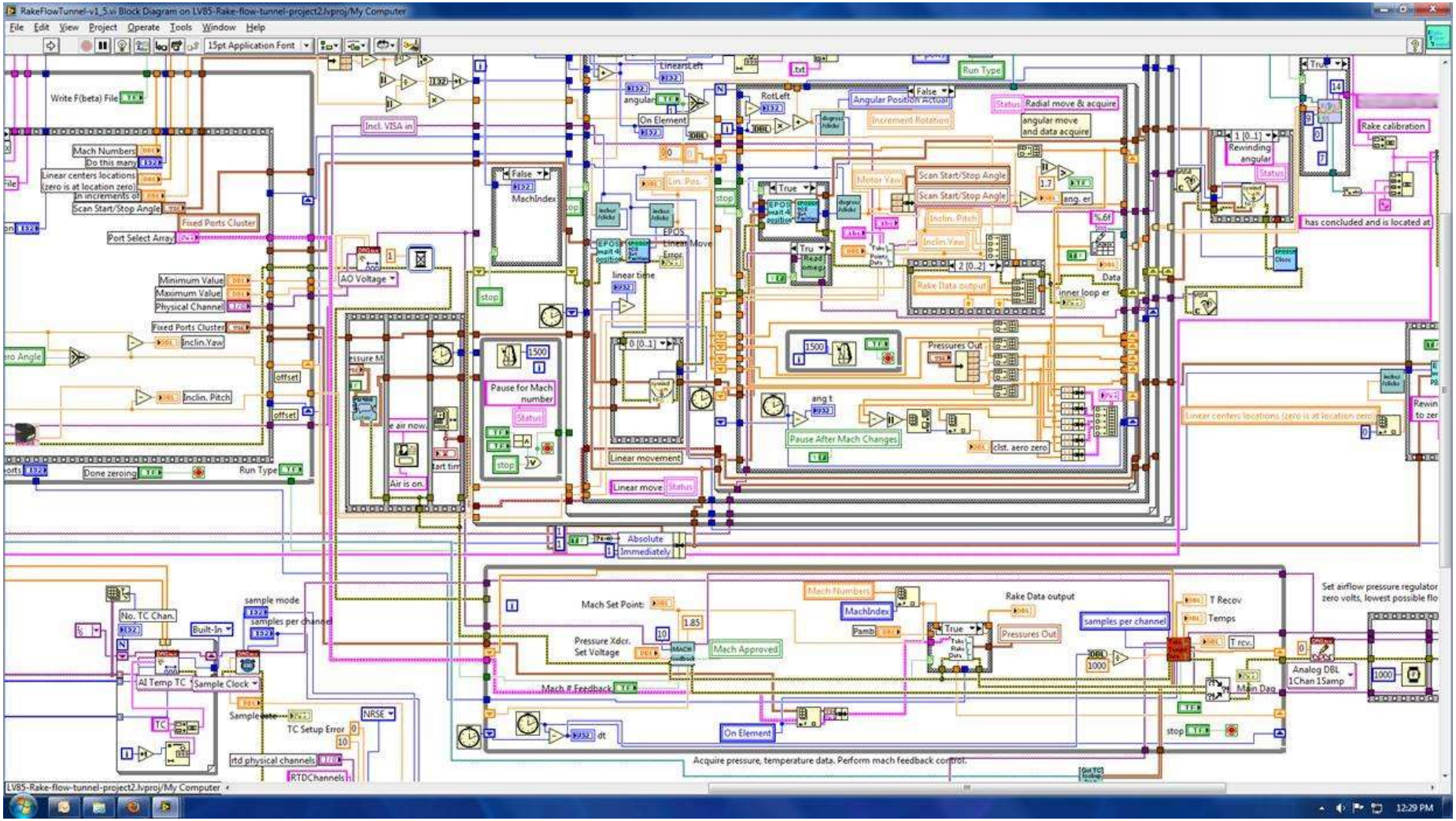
1. Make sure you have a plan



2. Plan for the right things

3. Study and reuse proven architectures

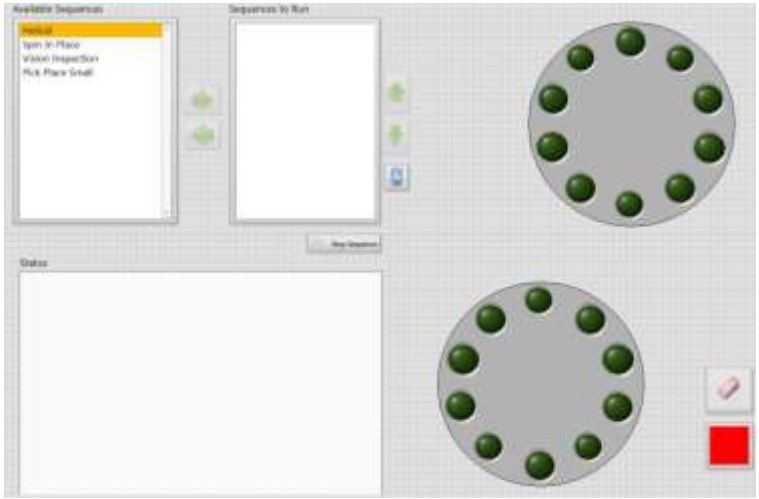
Make sure you have a plan



Common system planning tools

Hardware Diagrams

Connection Diagrams



UI Mockups

NATIONAL INSTRUMENTS

What is a Software Architecture?

- The high level software structure of a system, the discipline of creating such a high level structure, and the documentation of this structure.

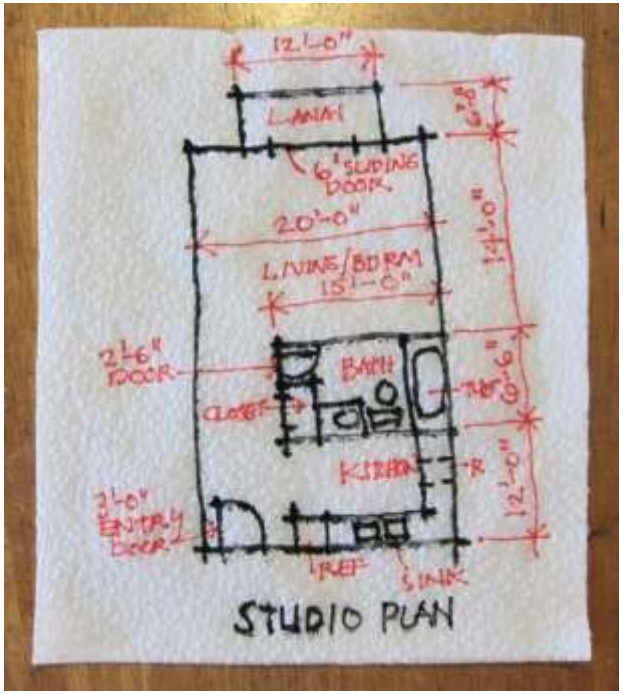
- An intellectually graspable abstraction of a complex system.

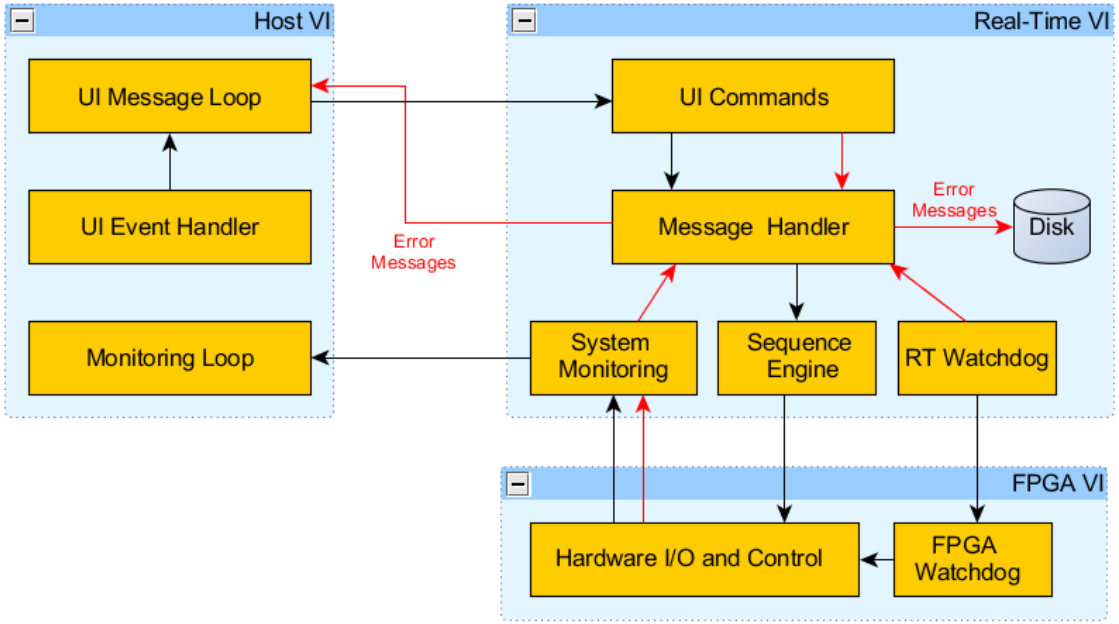
Why architectures are important

- Enable System Analysis
  - Abstraction of system complexity
  - Manifests early and major design decisions
  - Promotes modularity
- Simplify Communication Among Stakeholders
  - Development Team
  - Reviewers
  - Business Owners
  - Product Management
  - End Customer

Diagramming your system

- System diagrams are valuable tools for both designing and communicating your architecture





Example: Top-level software diagram

Execution Targets — *Real-Time, FPGA, Windows, Black Box*

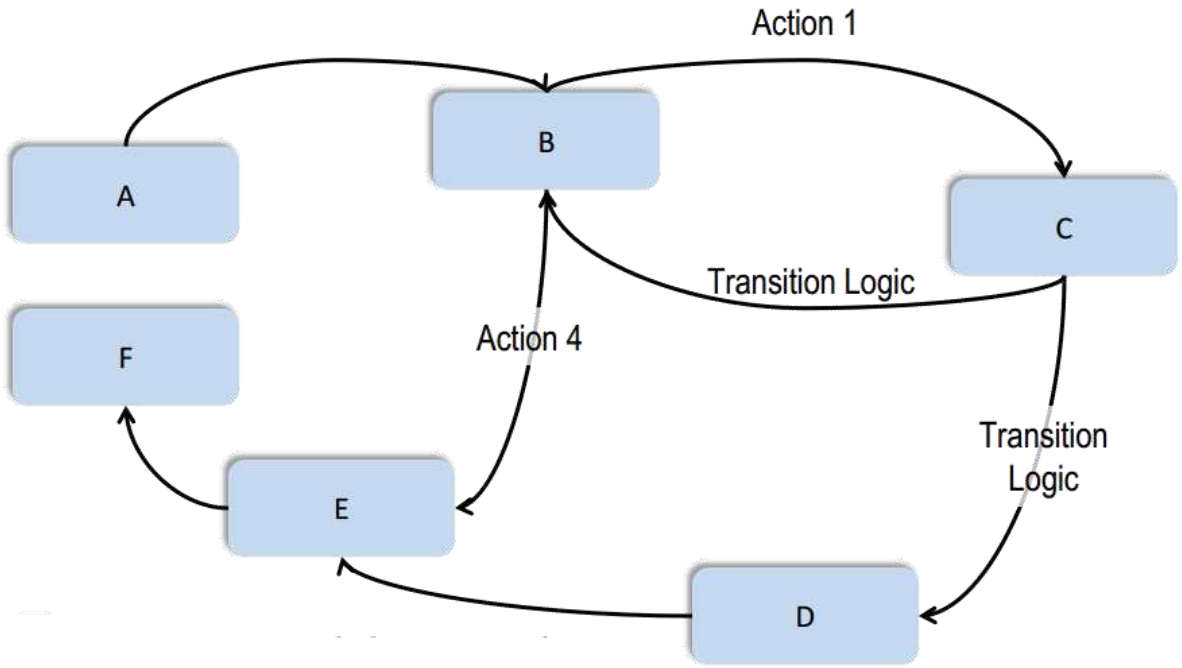
Loops/Threads/Processes — *UI Event Loop, RT Control, FPGA I/O*

Loop Communication Pathways

- **Current Values (Tags)** — *Latest value data, e.g., "Temp = 29° C"*
- **Messages/Commands** — *Intermittent data, low latency, e.g., "Stop"*
- **Stream** — *Continuous acquisition, e.g., "Waveform"*

Example: State Diagram

- A type of directed graph
- State is represented by a circle
- Transition, or actions, between the states is indicated by a directed line



Plan for the right things

Design Stage			
Application type (criticality, safety)			
Number of deployments			
Variations			

NATIONAL INSTRUMENTS

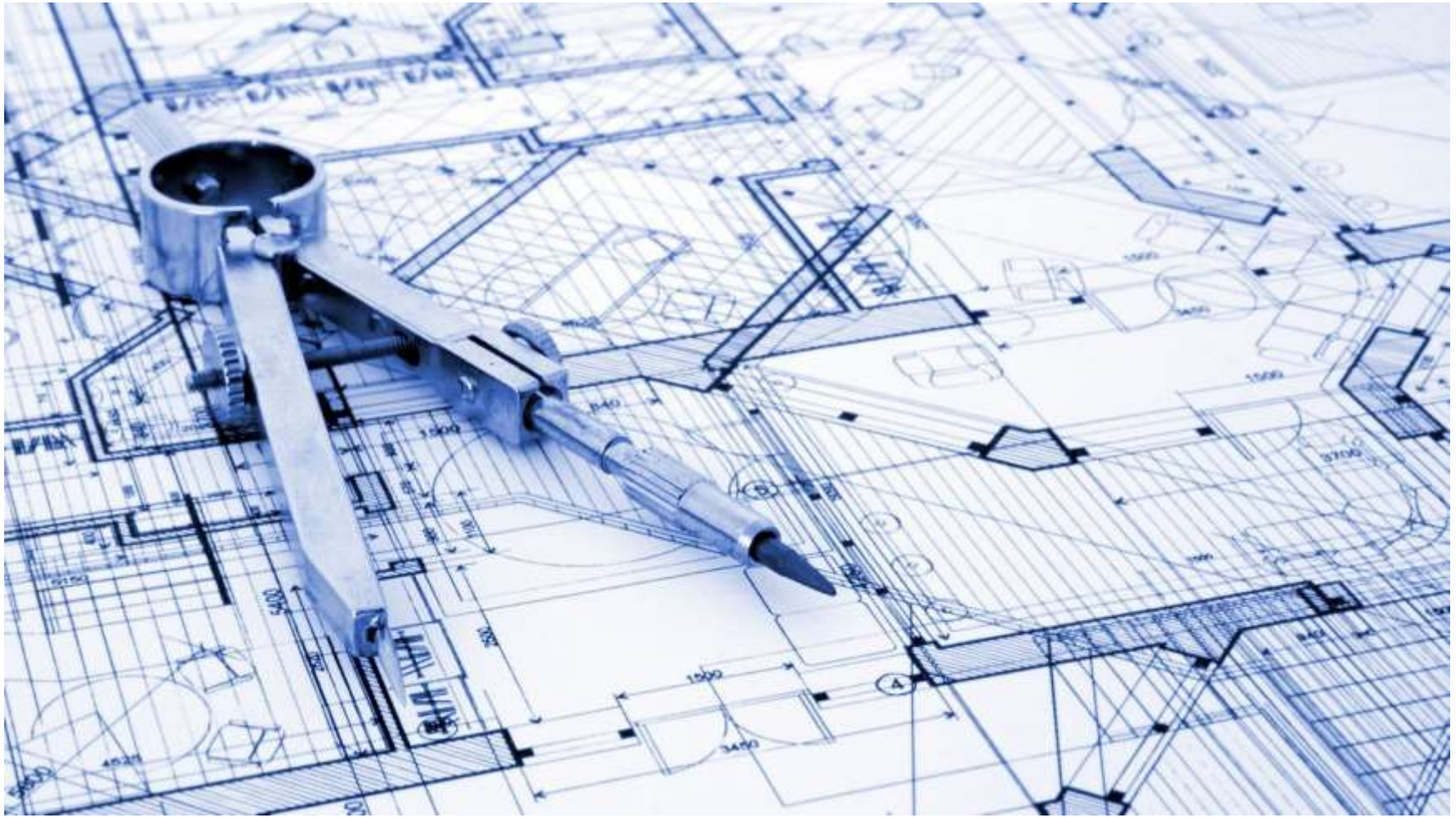
Think defensively

- Assume anything that can go wrong will go wrong
  - Expect the requirements to change
  - Minimize the possibility of unexpected behavior
  - Prepare for likely and/or costly failure conditions
  - Prove that it works
  - Reuse what has been proven

Defensive questions for your architecture

- Can it adapt to new requirements easily?
- Will it help me parameterize my solution?
- Does it help me prevent race conditions, deadlocks, etc.?
- Does it have robust error handling and recovery mechanisms?
- Is it something my development team will be successful implementing or plugging into?
- How easy is it to test and debug?
- How much of the code can be reused on the next project?
- Does this approach have proven success?
- IS IT WITHIN THE SCOPE OF THE PROJECT?!?!

Study and reuse proven architectures



NI Reference Design Portal

Home > Community > Reference Design Portal

Reference Design Portal

Latest Activity: 17 hours ago

Overview | Members (139) | Discussions (2) | Documents (25) | Blog | Polls

Reference Designs and Reuse Code are code components and templates, that are useful in the development of new applications and systems. They are typically developed and published by the Systems Engineering group at National Instruments. more

The purpose of the Reference Design Portal is to:

- Provide visbility to available reference designs and reuse code that has been found to be useful by developers in a wide range of applications.
- Foster discussion and collaboration on future reference design development, specifically in the areas of design patterns, frameworks and advanced reference applications and architectures.

If you'd like to start a conversation to discuss a new reference design or start a collaborative development effort, please create a discussion item or document in this group, throw out your thoughts or a brief proposal, and see who else is interested in contributing.

Other Content:
Examples and IP for Software-Designed Instruments and NI FlexRIO
LabVIEW Tools Network
Instrument Driver Network (IDNet)
LabVIEW FPGA IP (IPNet)

Reference Designs and Reuse Code

Category	Topic
All	Communication
Applications and Frameworks	Data
Templates, Design Patterns and Reference Designs	User Interface
Libraries and Examples	
Beta or Prerelease Versions	

Featured Reference Designs

- LabVIEW Replication And Deployment (RAD) Utility
- NI CompactRIO Waveform Reference Library
- Asynchronous Message Communication (AMC) Reference Library
- LabVIEW XML Data (GXML) Reference Library

Actions

Since this group is members only, you must first join before posting.

Join this group

Notifications

Receive email notifications

Group feeds

www.ni.com/referencedesigns

The two categories of architecture software

	Reference Architecture	Framework
Provides …	Examples (Best Practices)	Abstractions (Rules)
Intended for…	Teaching	As-Is Reuse
Applied through…	Modification	Extension
Maintained by…	Each User (code is branched)	Framework Dev. Team
Updated…	Manually (code is branched)	By Installing New Releases
Can be **built** from…	Reuse Components (APIs, etc.)	Reuse Components (APIs, etc.)

Relevant Architectures and Frameworks

Reference Architectures and Applications	

• LabVIEW Sample Projects
  • cRIO Waveform
  • FPGA Control
	Shipping Examples

• cRIO Vibration Data Logger Reference Application • Machine Control Reference Architecture	Online Examples

Frameworks	

• State Machine
• Queued Message Handler
• Actor Framework
  • Application Agnostic
  • Messages
	NI Product

• Tag Bus Data Framework
  • Control
  • Current Value Data (Tags)
	Community Owned

NI InsightCM™ Enterprise software suite

A deployment software solution with tightly integrated hardware options for online condition monitoring that allows companies to gain insight into the health of rotating machinery for operations and maintenance programs.


Acquire Dynamic & Static Data


Manage Data


Analyze Waveform Data


Configure and Monitor Nodes

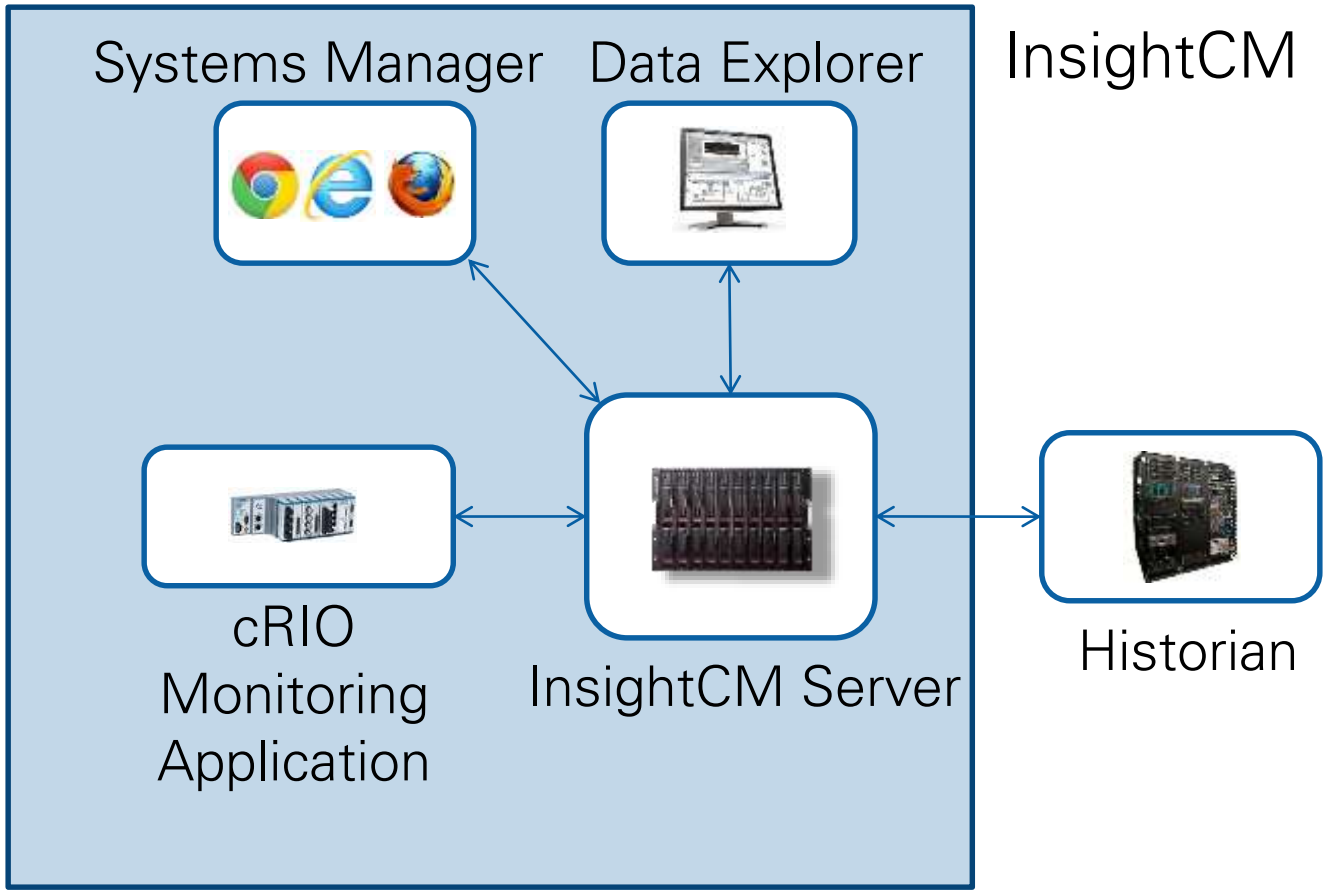

Visualize Raw Data & Results


Authenticate Users & Devices


Generate & Manage Alarms

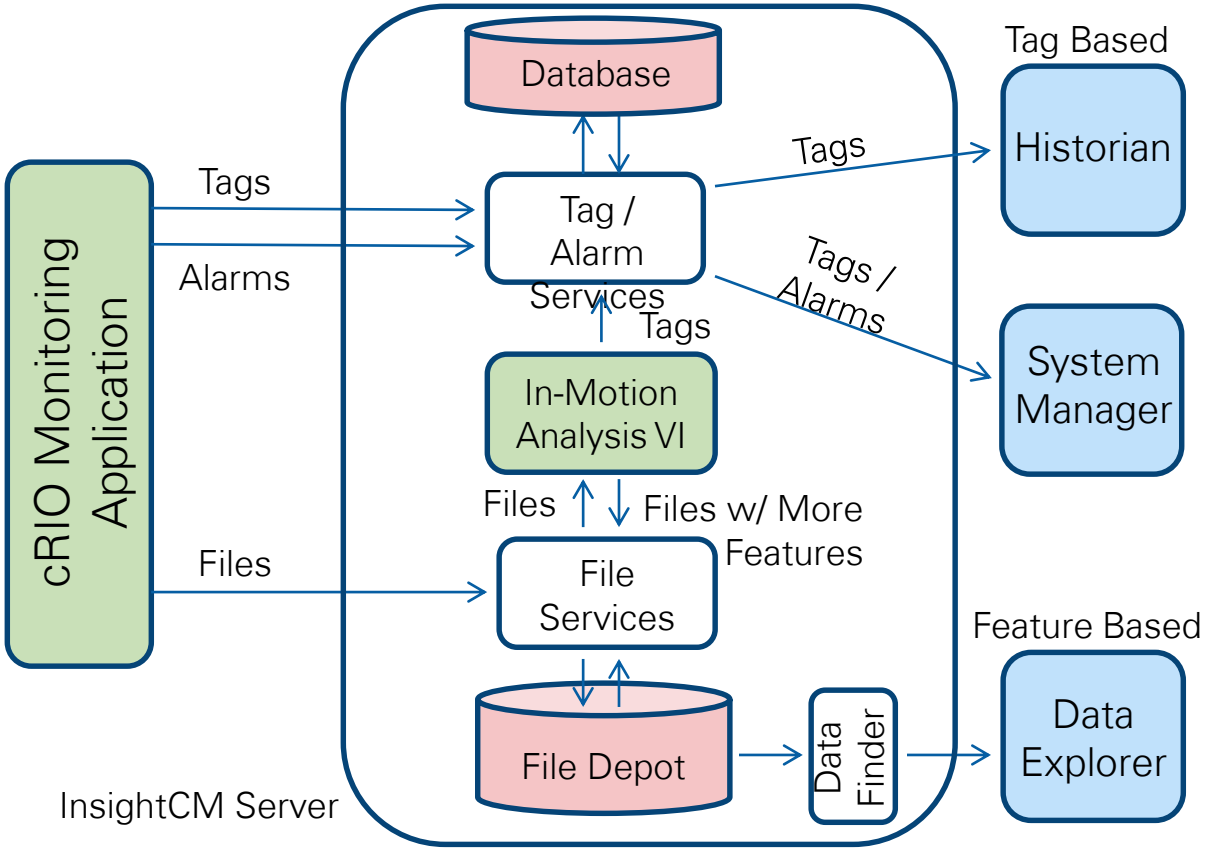

Integrate with IT Infrastructure

InsightCM Enterprise Components



NATIONAL INSTRUMENTS

Dataflow within InsightCM



NATIONAL INSTRUMENTS

Monitoring vs. Control

OR

What is the Tag Bus Data Framework (TBD)?

An open-source LabVIEW framework for creating configurable, latest value, data engines that can acquire data from multiple input sources, process that data, and then route it back to outputs or to data services.

Scalable and extensible plug-in architecture

Develop with plug-in templates

Reuse existing plug-ins

Define and configure plug-in parameters

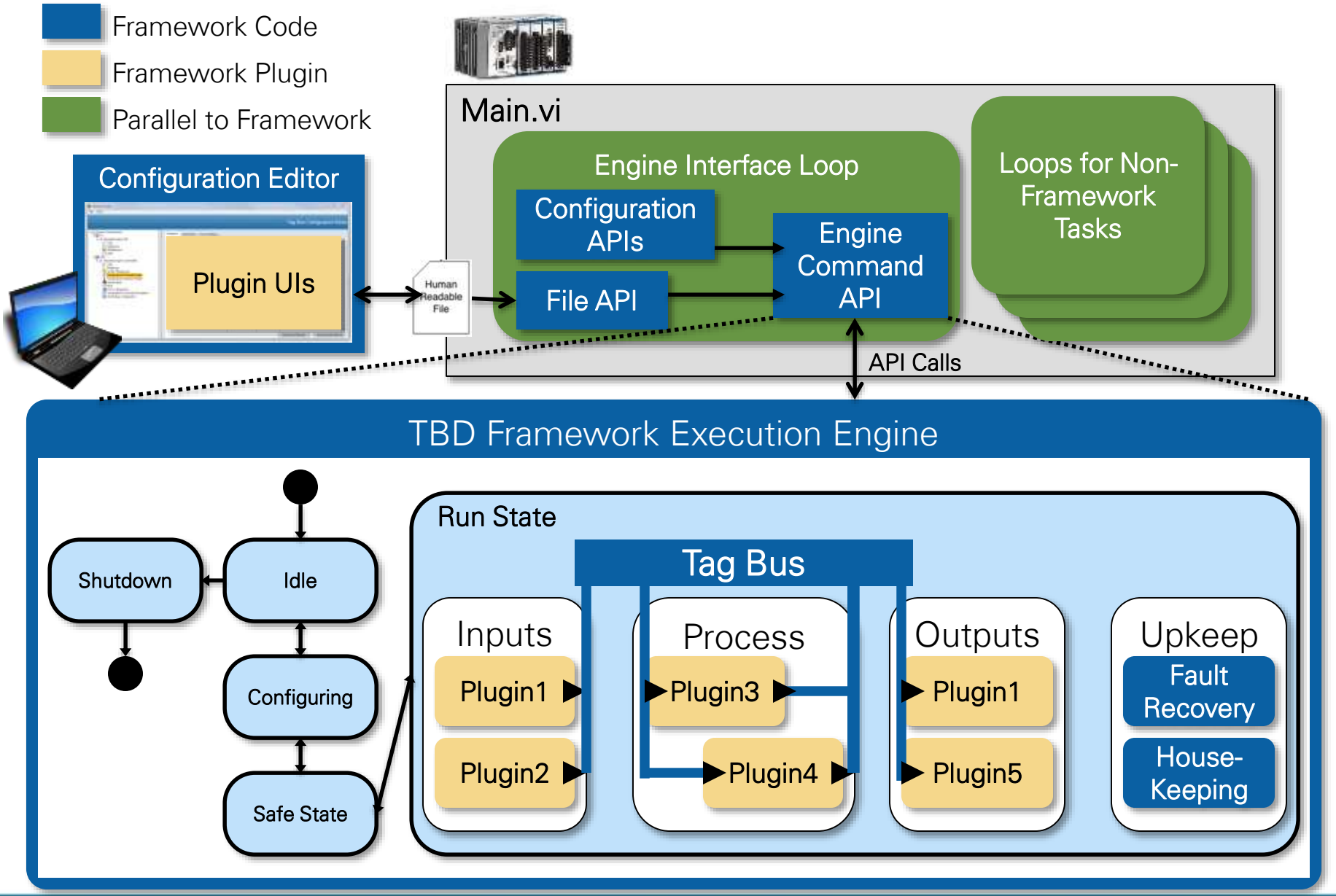
Single point I/O, processing, and data services

Enable multi-developer teams

Configure timing and error handling

Optimized for real-time execution

TBD Application Architecture Diagram



The TBD Workflow

1. 

Identify Req. Plugins

- Separate functionality
- Plugin or not?

2. 

Reuse Existing Plugins

- Research on community
- Reuse as-is or as example

3. 

Develop New Plugins

- Define I/O and parameters
- Use available templates

4. 

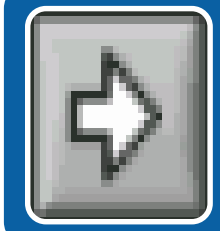
Validate Plugins

- Test plugins standalone
- Iterate until valid

5. 

Configure/Integrate

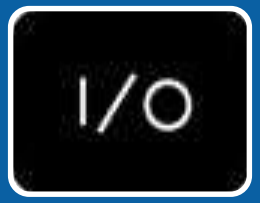
- Use config. editor
- # plugins and parameters

6. 

Execute System

- Read and execute config.
- Add other functionality

Identifying and reusing Framework Plugins



I/O

- Scan Engine/Variable Engine
- Modbus
- Profibus
- EthernetIP

Data Services

- TDMS Data Logger
- Web Service
- UDP Engine to Engine Communication
- CVT

Data Processing/Application Logic

- User Control Module
- Simple Sequencer
- Shared Library

Protocol Gateways

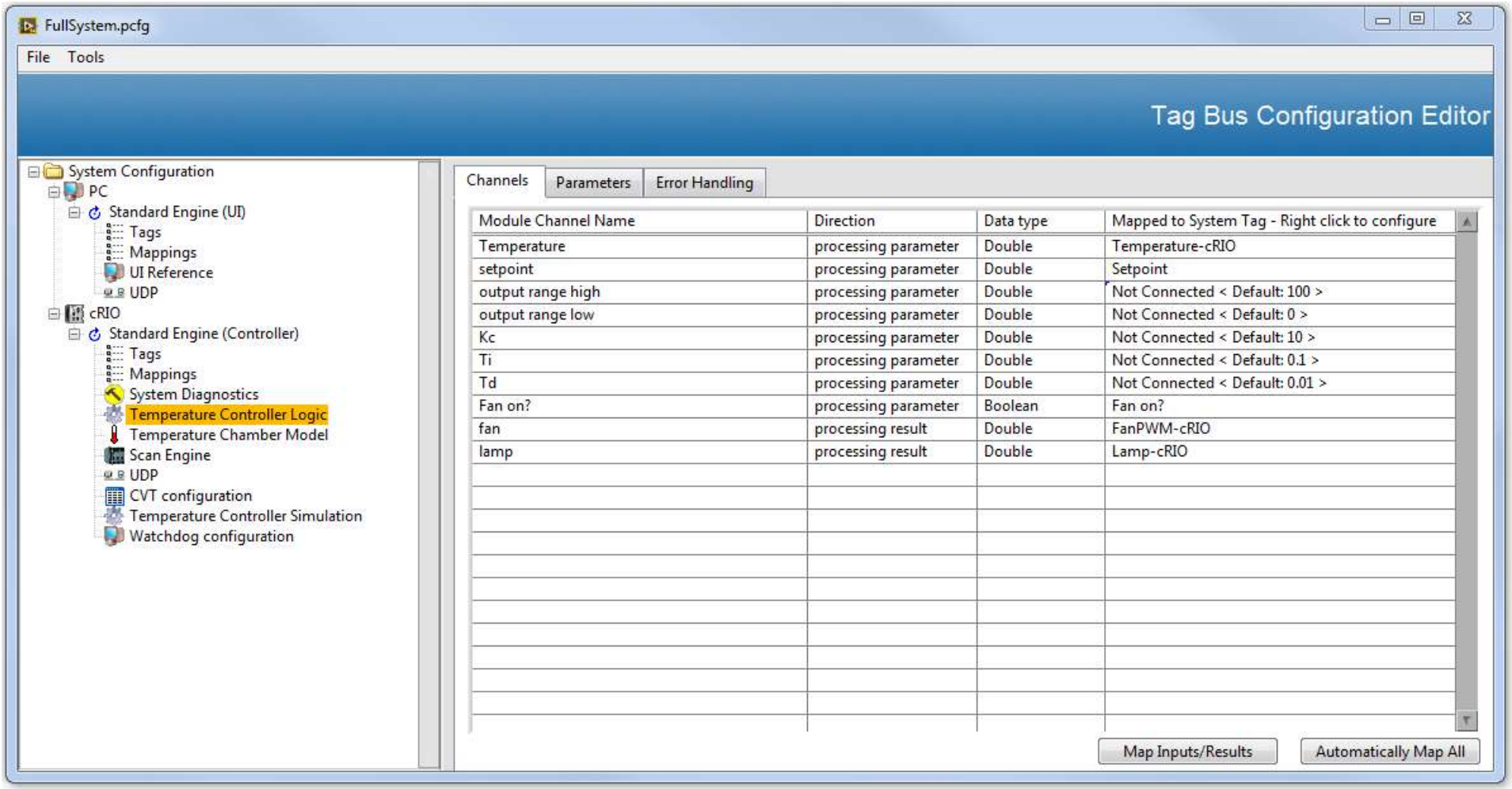
Pre-built Services

Hardware Abstraction

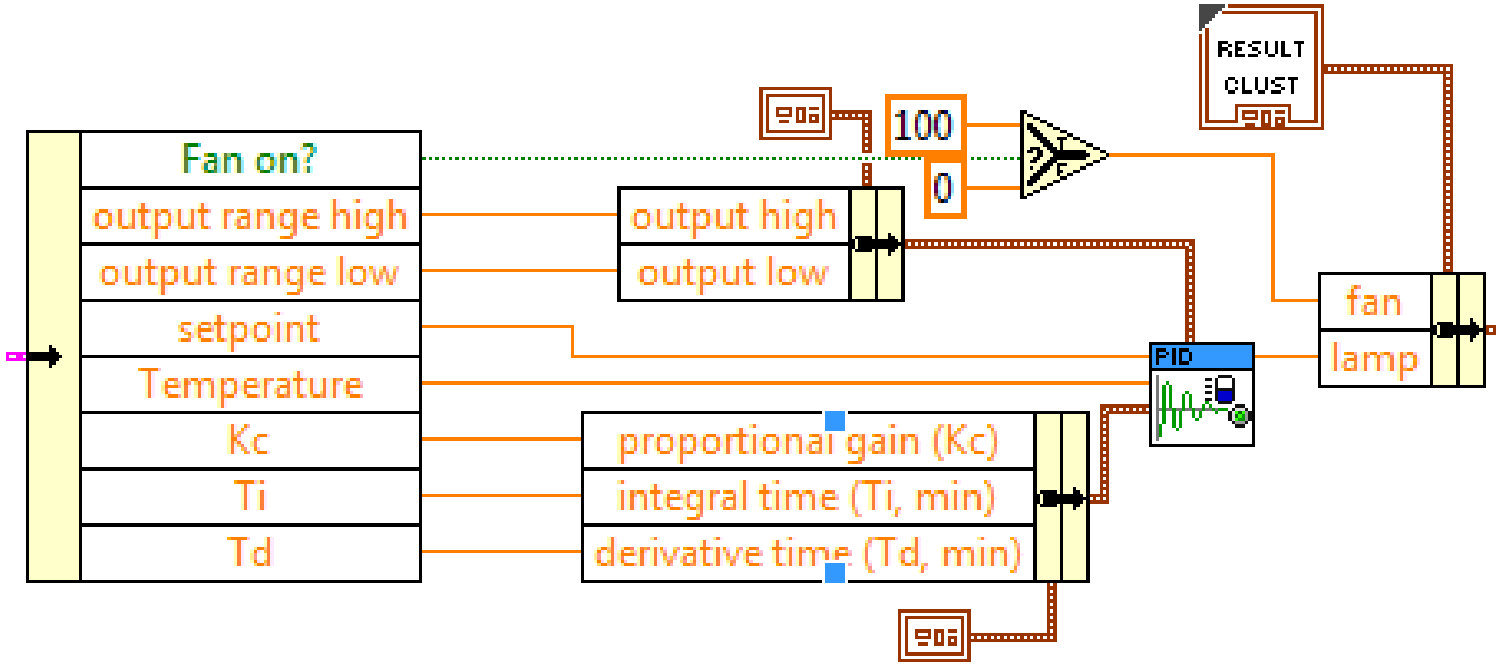
Simplified Creation of Application Logic

NATIONAL INSTRUMENTS

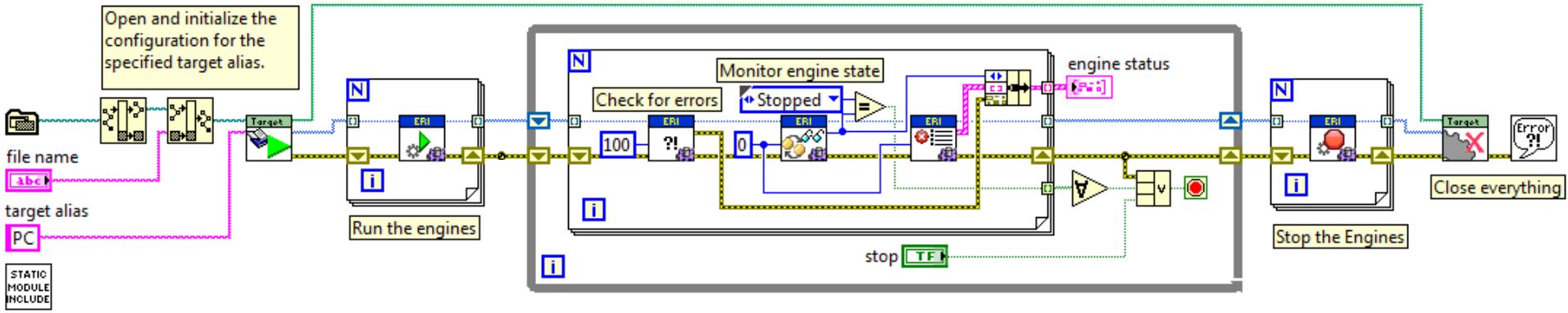
Configure Plugins with the TBD Configuration Editor



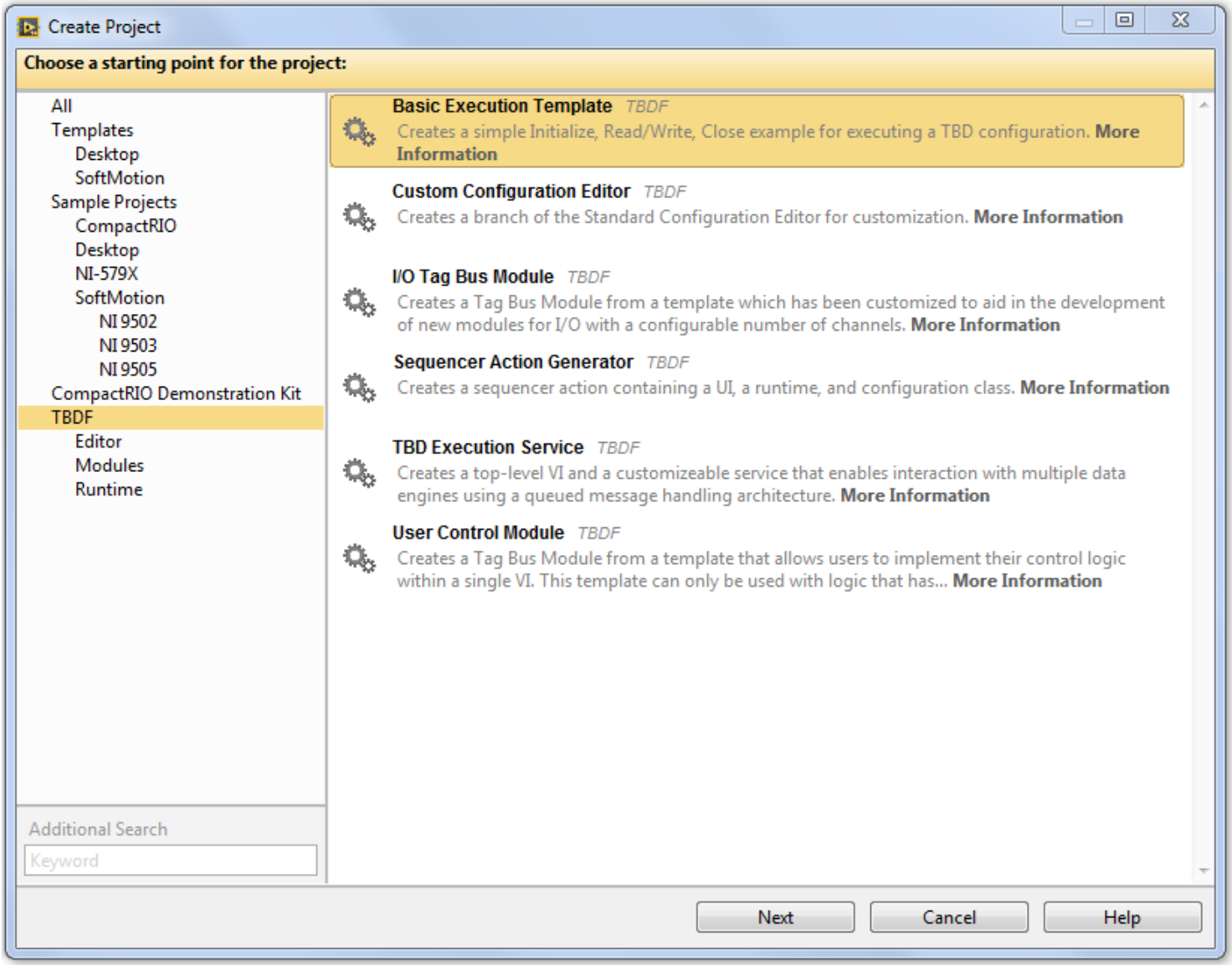
Building and validating a PID plugin



TBD ‘Main.vi’


Open and initialize the configuration for the specified target alias.
file name
target alias
PC
STATIC MODULE INCLUDE
Target
Run the engines
Monitor engine state
Check for errors
Stopped
100
0
engine status
stop
TF
Stop the Engines
Close everything
Error

TBD Project Templates



NATIONAL INSTRUMENTS

TBD Benefits Summary: Productivity and Quality

Build better embedded control software in less time!

Productivity

- Use framework functionality
  - Execution timing
  - Error handling
  - State management
  - Data communication
  - System configuration
- Reuse framework plugins
- Configure system changes
- Develop plugins in parallel
- Enable new developers
- Switch between simulation and real world execution

Quality

- Prevent unexpected behavior
  - No race conditions
  - No deadlocks
  - No memory allocations
- Update framework code
- Reuse proven plugins
- Encourage testable code
- Visualize system configuration
- Open source
- Scale from prototype to deployment

Getting Started with TBD

Learn more about TBD from the TBD community:

- Download the .vipc file and double-click to install
- Community includes documentation, hands-on, examples, and more!
- The .vipc package includes a shipping example and sample projects

Can also be found from www.ni.com/referencedesigns

This is not a product, it is an open source project. Find support on the community and submit issues to GitHub.

- https://github.com/TBDFramework

Contributors welcome!

Review

- Take time to plan up front
- Think defensively
- Study existing architectures to use or to improve your own designs