F Prime Architecture

Timothy Canham
NASA Jet Propulsion Laboratory
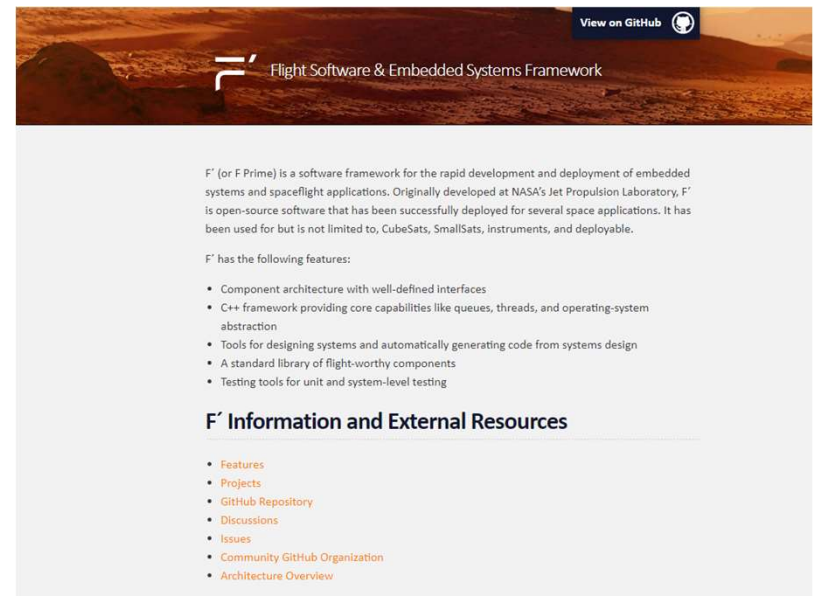**October 17, 2022**

Jet Propulsion Laboratory
California Institute of Technology

Background

- F` was developed as part of a technology task at JPL
  - Explore new flight hardware
  - Explore new software approaches
  - Targeted at smaller projects like instruments, Cubesats, and Smallsats
  - Sparser processor resources (e.g. 2MB memory, 128K program space)
    - TI MSP430, ARM-M*, LEON3
 - Clusters of interconnected processors
- Goals were to show:

Goal	Explanation
Reusability	Frameworks and adaptations readily reusable
Modularity	Decoupled and easy to reassemble
Testability	Components easily isolated for testing
Adaptability	Should be adaptable to new contexts and bridge to inherited
Portability	Should be portable to new architectures and platforms
Usability	Should be easily understood and used by customers
Configurability	Facilities in the architecture should be scalable and configurable
Performance	Architecture should perform well in resource constrained contexts. Should be very compact.

View on GitHub

F' Flight Software & Embedded Systems Framework

F' (or F Prime) is a software framework for the rapid development and deployment of embedded systems and spaceflight applications. Originally developed at NASA's Jet Propulsion Laboratory, F' is open-source software that has been successfully deployed for several space applications. It has been used for but is not limited to, CubeSats, SmallSats, instruments, and deployable.

F' has the following features:

- Component architecture with well-defined interfaces
- C++ framework providing core capabilities like queues, threads, and operating-system abstraction
- Tools for designing systems and automatically generating code from systems design
- A standard library of flight-worthy components
- Testing tools for unit and system-level testing

F' Information and External Resources

- Features
- Projects
- GitHub Repository
- Discussions
- Issues
- Community GitHub Organization
- Architecture Overview

F`: A Component Architecture

- Definition: The F` Component Architecture is a design pattern based on an architectural concept combined with a software architectural framework.

- Not just the concepts, but framework classes and tools are provided for the developer/adapter.

- Implies patterns of usages as well as constraints on usage.

- Centered around the concept of "components" and "ports"

- Uses code generation to produce code to implement common framework logic
    - Developer specifies in FPP (F Prime Prime) DSL (Domain Specific Language)

- Developer writes implementation classes to implement interfaces.

= some state machine in the component
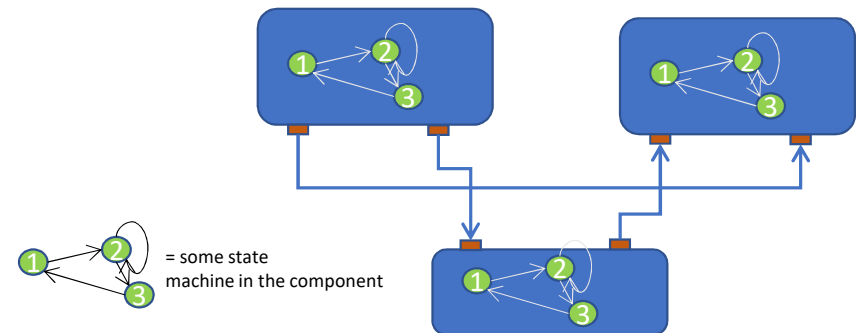
Characteristics of Components

- Encapsulates behavior
- Components are not aware of other components
- Localized to one compute context
- Interfaces are via strongly typed ports
  - Ports are formally specified interfaces
  - No direct calls to other components
- Context for threads
- Executes commands and produces telemetry

Component

Port 1   Port 2

Jet Propulsion Laboratory
California Institute of Technology
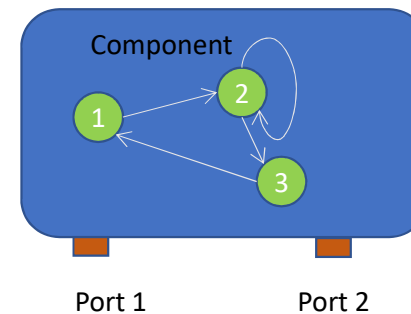
Characteristics of Ports

- Encapsulates typed interfaces in the architecture.
  - Think C++ class with one interface method

- Point of interconnection in the architecture.

- Ports are directional; there are input and output ports
  - Direction is direction of *invocation*, not necessarily data flow. Ports can retrieve data.

- Ports can connect to 3 things:
  - Another typed port
    - Call is made to method on attached port
  - A component
    - Incoming port calls call component provided callback
  - A serialized port
    - Port serializes call and passes as data buffer (more to come)

- All arguments in the interface are serializable, or convertible to a data buffer. There are built-in types supported by the framework; user can write custom types. (see later slides).

- Ports can have return values, but that limits use
  - Only return data when component has synchronous interface
  - No serializable connections since serialization passes a data buffer but does not return one (see later slides for explanation)

- Pointers/references allowed for performance reasons

- Multiple output ports can be connected to a single input port

serialize()
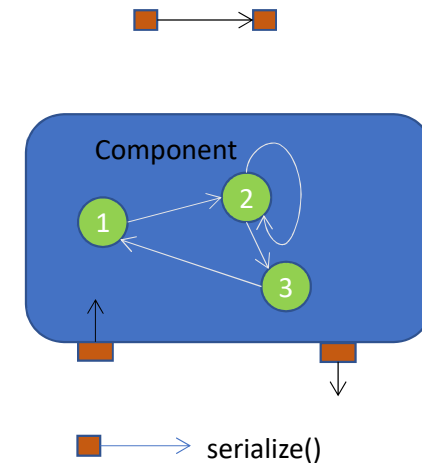
Jet Propulsion Laboratory
California Institute of Technology

- Components are instantiated at run time

- They are then connected via ports into a *Topology*, or a specific set of interconnected components

- There are no code dependencies between components, just dependencies on port interface types

- Alternate implementations can easily be swapped
 - E.g. simulation versions
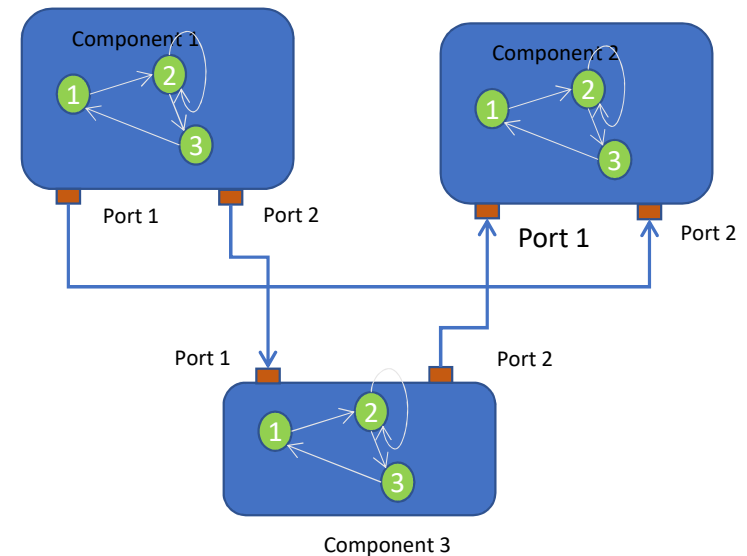
Component Type Hierarchy

- Hierarchy consists of:
  - core framework classes
  - generated classes that implement architecture features
  - Developer written classes that implement interfaces and project-specific logic


Component Class Hierarchy
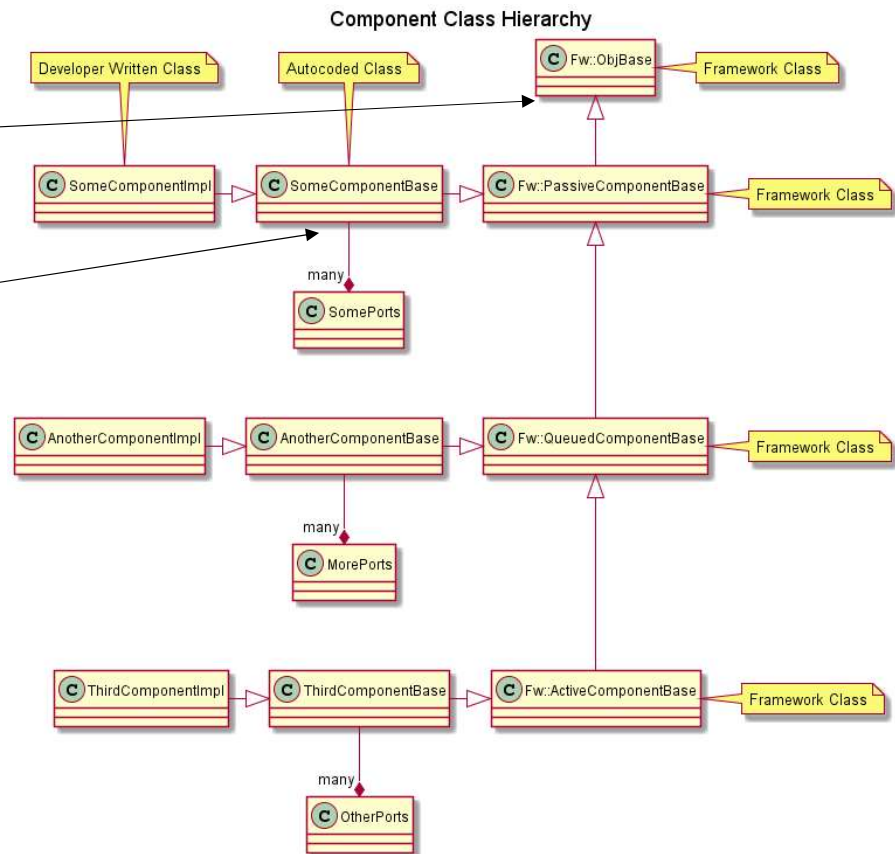
Component Types

- User specifies type of component in FPP. Types are:

- Passive Component
  - No thread
  - Port interface calls are made directly to user derived class methods

- Queued Component
  - No thread
  - A queue is instantiated, and asynchronous port calls are serialized and placed on queue.
  - Implementation class makes call to base class to dispatch calls to implementation class methods for asynchronous ports
    - Can be made from any implementation class function
    - Thread of execution provided by caller to a synchronous port

- Active Component
  - Component has thread of execution as well as queue
  - Thread dispatches port calls from queue as it executes based on thread scheduler

- Calls to output port are on thread of implementation functions
  - Thread making call is dependent on port type (see next slide)

Passive

Queued

Active
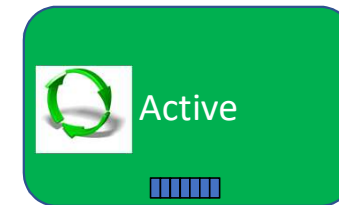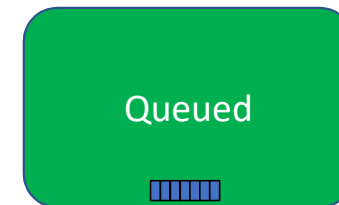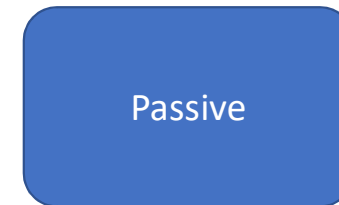
Port Characteristics

- The way incoming port calls are handled is specified by the component FPP.

- Input ports can have three attributes:
  - Synchronous – port calls directly invoke derived functions without passing through queue
  - Guarded – port calls directly invoke derived functions, but only after locking a mutex shared by all guarded ports in component
  - Asynchronous – port calls are placed in a queue and dispatched on thread emptying the queue.

- A passive component can have synchronous and guarded ports, but no asynchronous ports since there is no queue. Calls execute on the thread of the calling component.

- A queued component can have all three port types, but it needs at least one synchronous or guarded port to unload the queue and at least one asynchronous port for the queue to make sense.

- An active component can have all three varieties, but needs at least one asynchronous port for the queue and thread to make sense.

- Designer needs to be aware of how all the different call kinds interact (e.g. reentrancy)

- Output ports are invoked by calling generated base class functions from the implementation class.

Developer Written Implementation Class

Jet Propulsion Laboratory
California Institute of Technology
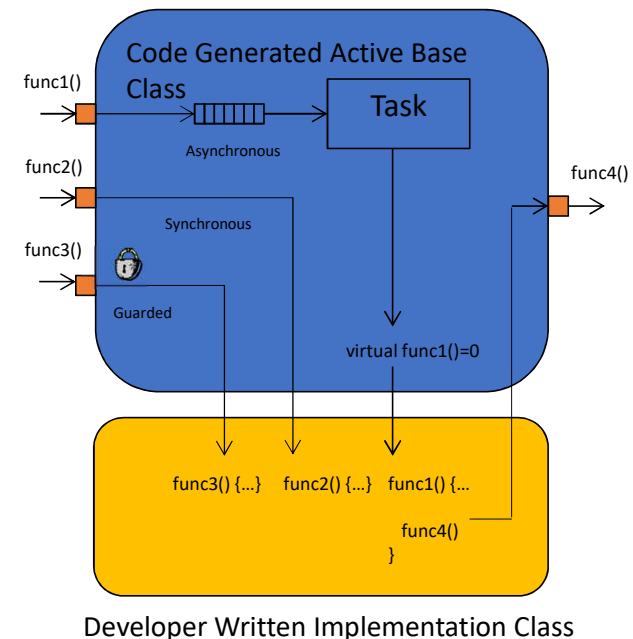
Serialization

- Serialization is a key concept in the framework

- Definition: Taking a specific set of typed values or function arguments and converting them in an architecture-independent way into a data buffer

- Port calls and commands and their arguments are serialized and placed on message queues in components

- Command arguments and telemetry values are passed and encoded/decoded into this form

- Components that store and pass data can use this form and not require knowledge of underlying types

- User can define arbitrary interface argument types and framework automatically serializes

- User can define complex types in FPP and code generator will generate classes that are serializable for use internally and to and from ground software

32 bit integer

1000

| 0	0	0	0	0	3	E	8
4	0	4	9	0	F	7	C

3.14157

32 bit floating point

Serialization Ports

- A special optional port that handles serialized buffers

- Takes as input a serialized buffer when it is an input port, and outputs a serialized buffer when it is an output port.

- Can be connected to *any* typed port (almost).
  - For input port, calling port detects connection and serializes arguments
  - For output port, serialized port calls interface on typed port that deserializes arguments
  - Not supported for ports with return types

- Useful for generic storage and communication components that don't need to know type
  - Allows design and implementation of C&DH (command and data handling) components that can be reused.
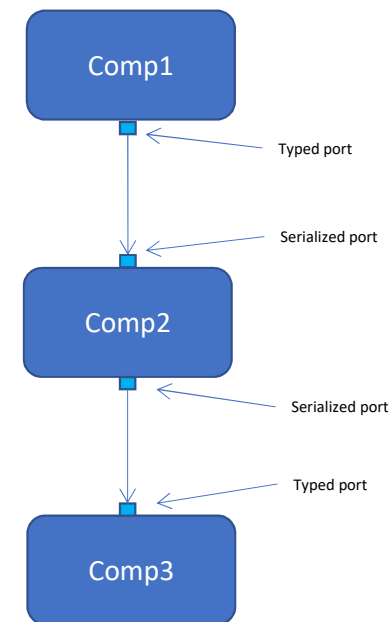
Commands, Telemetry, Events and Parameters

- The code generator provides a method of implementing commands, telemetry, events (AKA JPL EVRs), and parameters.

- Component FPP specifies arguments and types.

- Data for service is passed in serialized form.

- The code generator parses arguments and types and generates code to convert arguments to serialized data or vice versa.

- Calls implementation functions with deserialized arguments (commands) or provides base class functions to implement calls (telemetry, events and parameters)

- Code generator uses port types that are specified in XML themselves. These ports can be then used in components that provide interfaces for transporting or storing data for those services.
  - Since data is handled in serialized form, don't need to know specific argument types
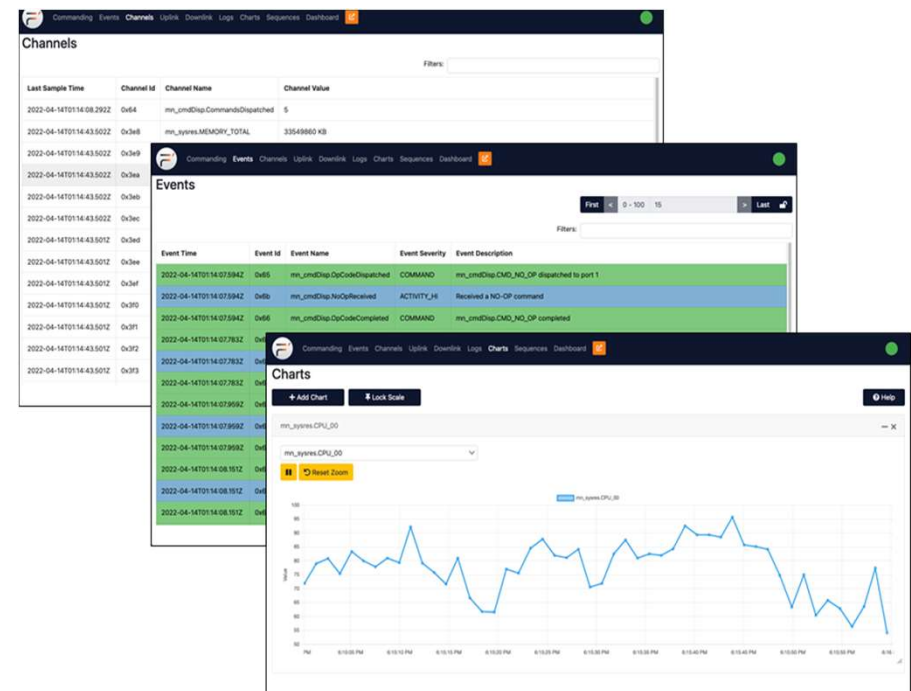
Commands
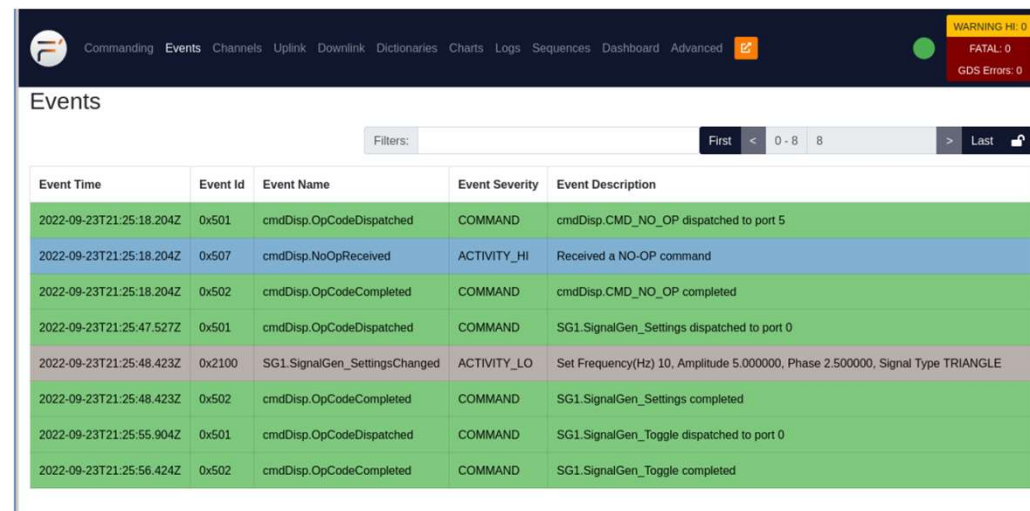
- Commands are sent from the ground system to the spacecraft to initiate activities

- Component command FPP specifies:
  - Opcode, mnemonic, and arguments
    - Arguments can be any built-in type or external XML complex type
    - Complex type can be a single argument
    - Can define enumerations
  - Synchronization attribute
    - Sync, async, or guarded
    - Same meaning as ports
      - Async can specify message priority

- Implementation class implements function for each command
  - Framework code deserializes arguments from argument data buffer

- Autocoder automatically adds ports for registering commands, receiving commands and reporting an execution status.

Ingenuity Uplink Team

Events

- Events are asynchronous notifications of spacecraft activity
- Component event FPP specifies:
  - ID, name, severity and arguments
    - Arguments can be any built-in type or XML complex type
    - Complex type can be a single argument
    - Can define enumerations
  - Format specifier string
    - Used by ground software and optional on-board console to display message with argument values
    - Follows C format specifier syntax
- Code generated base class provides function to call for each event with typed arguments
  - Provides stronger type checking at compile time than MER/MSL EVR macros
  - Called by implementation class
- Code generator automatically adds ports for retrieving time tag and sending event
  - Two independent ports for sending events
    - A binary version with serialized arguments for transport to ground software
    - A text port that sends a string version of the event (using the format specifier) that can be sent to a console
      - Can be globally disabled via architecture configuration macros to save execution time and code space

Telemetry

- Telemetry is data periodically sampled and emitted
- Component telemetry FPP specifies channels that have:
  - ID, name, and data type
    - Data type can be any built-in type or external XML complex type
    - Can define enumerations
  - Format specifier string
    - Used by ground software and optional on-board console to display message with argument values
- Code generated base class provides function to call for each channel with typed argument
  - Called by implementation class
- Code generator automatically adds ports for retrieving time tag and sending channelized data

Parameters

- Parameters are a means of storing non-volatile state
  - Allows modification of software behavior without a software update
    - Ex. Thermal alarm limits
- Component FPP specifies parameters that have:
  - ID, name, and data type
    - Data type can be any built-in type or external XML complex type
    - Can define enumerations
  - Optional default value
    - In the event the parameter cannot be retrieved, assigns default value to parameter
- Code generator automatically adds port for retrieving parameters
- During initialization, a public method in the class is called that retrieves the parameters and stores copies locally
  - Can be called again if parameter is updated
- Code generated base class provides function to call for each parameter to retrieve stored copy
  - Implementation class can call whenever parameter value is needed
- Framework provides code generation to manage, but projects must write specific storage component
  - Basic file-based storage provided by F Prime core component

F Prime Core Components

- The F Prime repository has a set of tested and flight-proven components to do common spacecraft functions

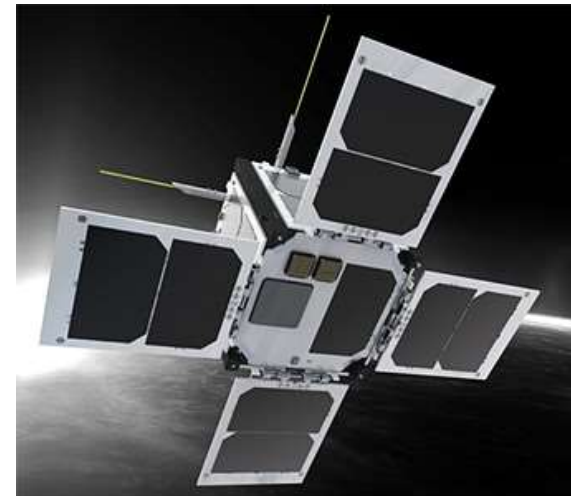| Component	Description
Command Dispatcher	Dispatches uplinked commands to components
Command Sequencer	Executes a set of commands uplinked as a file
Telemetry Processing	Gets sampled telemetry from components for downlink
Event Logging	Gets asynchronous event notifications from components for downlink
File Uplink/Downlink	Send and receive files
File Manager	File management including listing, moving and deleting files
Parameter Database	Database to store non-volatile parameter values for components
Value Database	Allows components to exchange run-time data values
Rate Group Execution	Allows a set of components to run sequentially at a certain rate
Uplink/Downlink Framing	Generic components for framing data for telecommunications
Buffer Management	Pre-allocates a set of buffers for dynamic memory requirements for components

Code Scaling

- Framework code is very compact
- Generated code is also compact
    - Demo application for TI microcontroller was about 15K
- Native type sizes can be configured
    - e.g. some microcontrollers have 16-bit/8-bit only support
- Features can be added or removed depending on resources
    - Object naming
    - Port execution tracing
    - Serialization of ports
        - Single node systems don't really need
        - This is not data serialization but the use of serialized ports
    - Object naming/registry
    - Component connection tracing
    - Text logging
- For very compact processors with no OS, developers can choose non-active components

Summary

- F Prime provides flight software-in-a-box
- Projects benefit from:
  - A ready-to-go framework
  - Flight heritage
  - Open source support
  - Designed for embedded
- Also good for testbeds or labs
  - Test script interface
  - Quickly add new commands and telemetry

Georgia Tech GT-1