Installed: Git, CMake and Python

Notes: The Discerning User's Guide to F´ | F´ (nasa.github.io)

F Prime – A frame Work for development and deployment of embedded system and spaceflight

,Used on CubeSats (Small Cube like Satellites) and SmallSats, Component Architecture, C++ Framework with queues, threads and operating system abstractions.

Projects and Deployment

Projects – Can consist of many deployments which can share component designs

Deployments – contain the system to create and build f prime framework

Topology – interconnected components that can list connections between the ports.

Core Constructs: Ports, Components, and Topologies

CommandDispatchPort, Guarded and Kind ports

Ports – Point of network interconnection for components, different types of ports but they can only connect to the same type. Port taking to a port is called invocation, datatypes. Can specify zero or many arguments of data (int,float, U8) *Careful with memory management* *A single output port can be connected to only one input port at a time**Distinguish port design and instantiation

Port Serialization – makes arguments to data buffers

Component Types

Passive Component - has no threads and cannot support asynchronous ports or commands

Active Component – has a thread of execution and a queue

Queued component – has no thread but does have a queue

Threads – used to implement active components

Fprime::Component – creates an active component, Start() – when component first activated, dispatch()- whenever component receives message to process

Division of Component Implementation

Core Framework Class- the base class of components and may inherit from active, passive and queued classes

Generated Component-Specific Base – the class is the descendant of the framework that is automatically generated

Component-Specific Developer Implementation Class – inherits from the generated component-specific base class and only contains user specific implementation

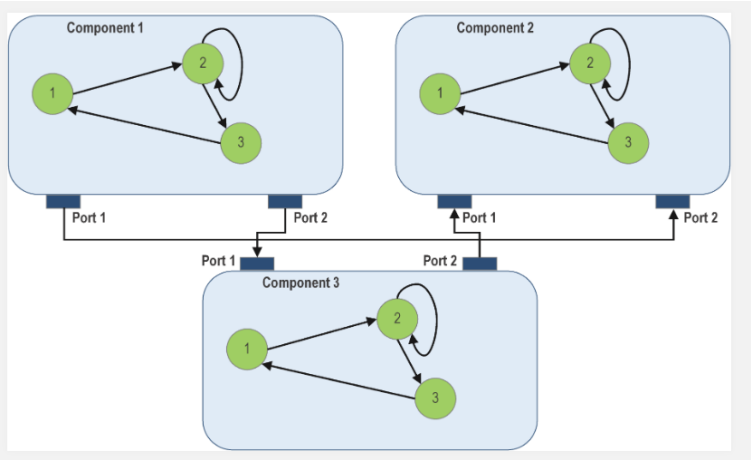
Port Kinds

Quick Look at Component Types and Supported Ports

Component Type	Output	Guarded / Synchronous Ports	Asynchronous Ports
Passive	0 or more	0 or more	Not available
Queued	0 or more	1 or more	1 or more
Active	0 or more	0 or more	1 or more

Topology: f' Application

Components and topology graph are instantized at runtime but the actual connection of ports are established during construction and setup



Data Types and Data Structures: Primitive Types, Enums, Arrays, and Serializable

Primitive Types

F´ Type	C/C++ Type	Description
BOOL	bool	C++ boolean
I8	int8_t	signed 8-bit integer
I16	int16_t	signed 16-bit integer
I32	int32_t	signed 32-bit integer
U8	uint8_t	unsigned 8-bit integer
U16	uint16_t	unsigned 16-bit integer
U32	uint32_t	unsigned 32-bit integer
F32	float	32-bit floating point
F64	double	64-bit floating point
NATIVE_INT_TYPE	int	architecture dependent integer
NATIVE_UINT_TYPE	unsigned int	architecture dependent unsigned integer
POINTER_CAST		integer of sufficient size to store a pointer for the architecture

Note: C/C++ types come from `stdint.h` and `stdbool.h`. The last three types above are not of set size but are architecture-dependent. Should a project's architecture not support all these types, see: Configuring F´: Architecture Supported Primitive Types

Polymorphic Type

PolyType object can assign a value by a constructor or a equals sign. Ex:

PolyType myInt(123),

or

Poly Type myfloat;

myFloat = 123.03

Getting a polytype can be in two different ways

U32 val = (U32)pt;

Or

U32 Val;

Pt.get(val);

Checking value of polytype uses the is function ex:

PolyType p(123);

If(p.isU32()){

}

Enums: Fixed set of values that can be referred to by name but are stored and transmitted as an int

Arrays: Fixed length contains of other types

Serializable: are field-value composition of other types and can be type-heterogenous. May contain any other type as a value

C++ classes may be used when interacting with software not within f' design layer using subclasses of Fw::Serializable and with methods serialize and deserialize

Data Constructs: Commands ,Events ,Channels, and Parameters

Events -EVR, Telemetry Channels – EHA

Components have set commands, commands are for user interaction.

Command Properties:

Opcode – a number representing a command

Mnemonic – a text value representing a command

Arguments- data types supplied to the command handler for execution

Synchronization "kind" – controls which execution the commands runs, Sync and guarded, guarded being protected by reentrancy, commands run on command dispatcher, Async commands execute on the threads

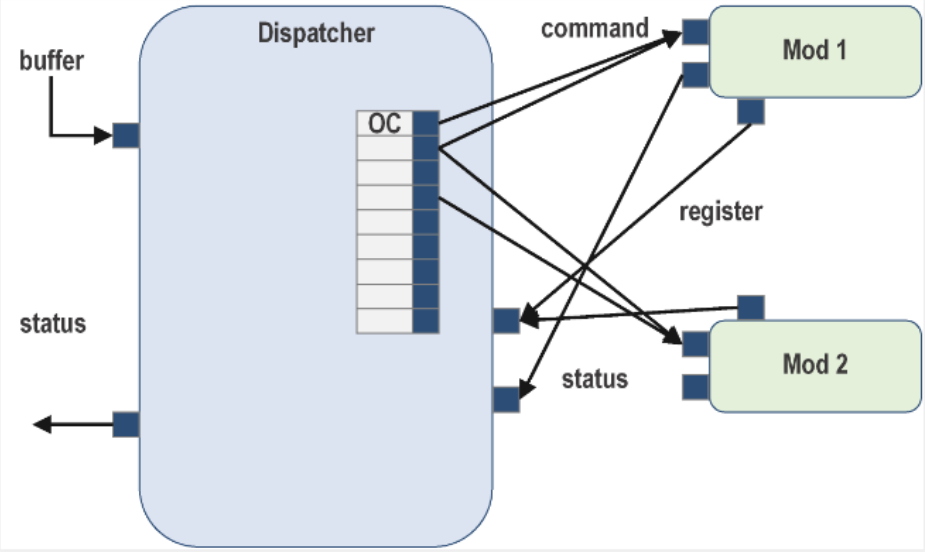


Figure 4. Command dispatcher. The command dispatcher receives the raw buffer containing the command and arguments. The command opcode is extracted, and a lookup table is used to find the handling component. The argument buffer is then passed to the component, and the command dispatcher waits without blocking for the component to return status..

Command Sequencing

Reads a defined sequence of commands and sends it to the dispatcher then status is returned. Sending command buffers to the command dispatcher through sequencer is an alternative from sending them externally from the ground

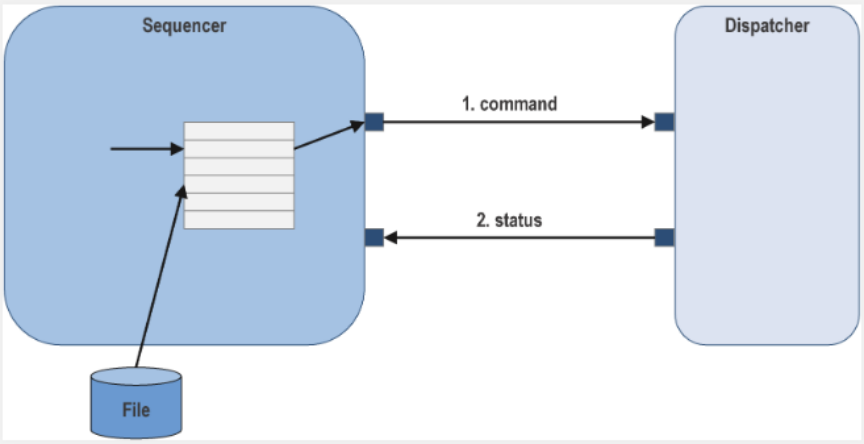


Figure 5. Command sequence. The command sequencer loads a sequence file from the file system, sends the command, and waits for the response for each command in the sequence. A failed response terminates the sequence, while a successful response moves to the next command in the sequence.

Events are a log of activities and are defined per component and are typically used to capture what the component is doing. Can occur sporadically

Event Properties:

ID – a number to define the Event

Name – a text identifier for the event

Severity – a text defining the severity of the event, Values are:

1. DIAGNOSTIC: akin to debug messages. Usually not sent to the ground.
2. ACTIVITY_LO: akin to fine info messages these typically come from background tasks
3. ACTIVITY_HI: akin to info messages these typically come from foreground tasks
4. WARNING_LO: less severe warning events
5. WARNING_HI: high-severity warning events, although the system can still function
6. FATAL: fatal events indicating that the system **must** reboot
7. COMMAND: events tracing the execution of commands

Arguments – are primitive and complex types that represent the variable date asscociated with the event

Format String – string used to reconstruct a text version of event

Two ports are created for the component-specific generated base which are the binary log output port for sending outside the system and the text log output port for on-board consoles

Event Logging:

Events acquire a time tag to know when event occurs, Svc::ActiveLogger processes event and recognizes and begins responses for fatal severity events

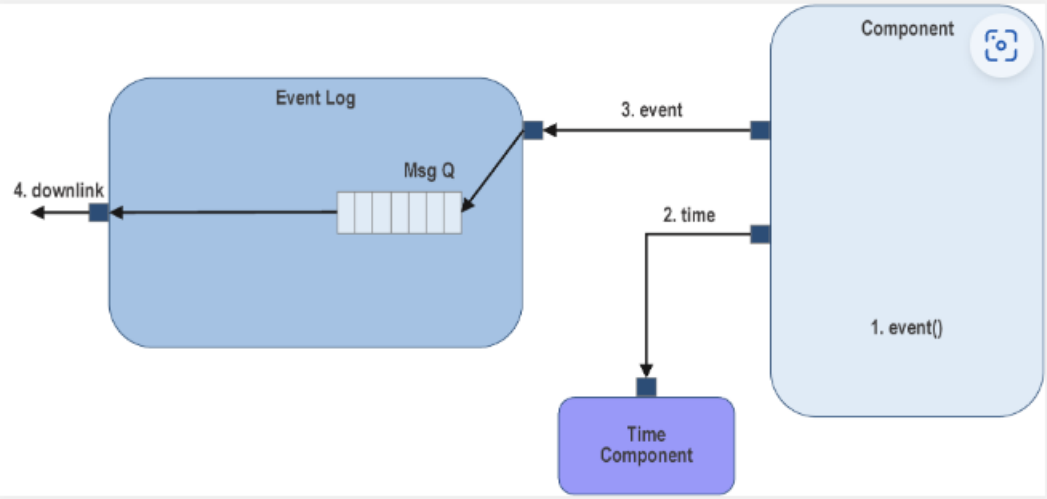


Figure 6. Event log. The component implementation calls a function to generate the event. The base class retrieves the time tag from the time source component. The component sends the event to the event log component, which reads it from the port queue and sends it to the ground.

Channels – known as Telemetry Channels, represents current reading of portion of the system state

Channels Properties:

ID- number of channel

Name – text name of channel

Data-type – type of the value pf channel

Update – to update only when the value changes and omitted to always downlink

Telemetry Database – acts as a double buffer storage for telemetry values

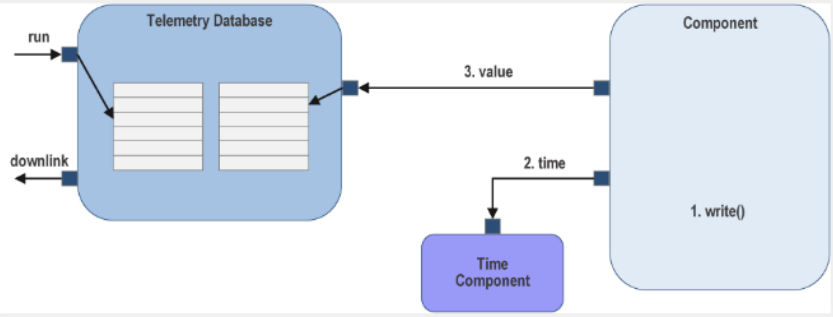


Figure 7. Telemetry database. The telemetry database has a double-buffered array of telemetry buffers. The base class function retrieves the time tag from the time source component and then writes the updated value to the telemetry database component. The telemetry database is called periodically to send the current set of telemetry to the ground.

Parameters – are ways of storing non- volatile state in the embedded system.

Parameter Properties:

Id- number of the parameter

Name - text name for the parameter

Data type: primitive or complex type that represents value stored

Default type – used in case the parameter can't be retrieved

Parameter database – provides ports to set and get parameters

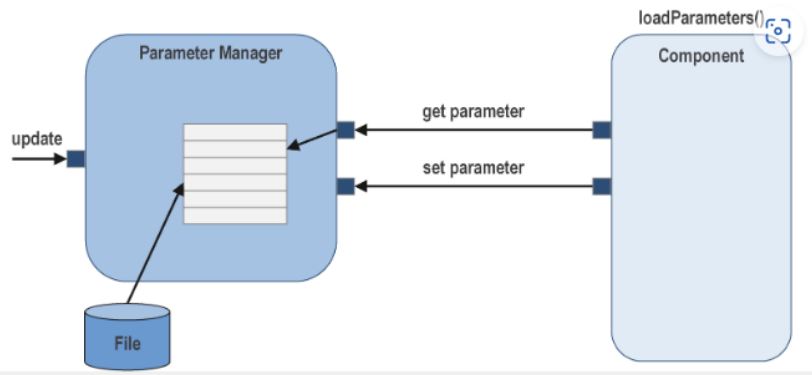


Figure 8. Parameter manager. The parameter manager or database loads the file containing parameters from the file system during initialization. The initialization subsequently calls *loadParameters()* on components with parameters. Components can set and retrieve parameters. The parameter manager saves the updated values to the file system via the set and save commands auto-generated for every parameter; the set command updates the value of the parameter locally within the component that owns it, and the save command pushing the current value of the parameter to non-volatile storage, meaning it will persist within the files of the system across system resets.

Unit Test in F'

Two phases of testing unit testing and integration testing. Unit tests out individual units while integration testing test the integrated systems

TesterBase – base class for testing a component and provides a harness for unit testing. For each output port there is input port called "from port", For each input port there is output port called "to port", for each "from port" there is history of data

GTestBase – includes headers with F' specific macros, derived from TesterBase. The macros check the telemetry received from the ports, the events received from the ports and the data received from ports.

Tester – derived from GTestBase, contains the component under test as a member

Build Unit test – fprime-util generate –ut

Run Unit Test – fprime-util check [parameter flags]

Other checks:

--all – run all unit tests

--coverage – check for code coverage in unit test

--leak check for memory leaks in unit test