COSMOS Infrastructure

Eric J. Pilger

June 10, 2014

Contents

Chapter 1

Introduction

The primary goal of COSMOS is to draw together related systems into a loosely linked network, allowing for control, monitoring and communication on a network level. Toward this end, the COSMOS project has developed a paradigm for representing, storing and communicating the state of these systems and their elements so that they can interact with each other, and be managed, in standardized ways. This infrastructure is the combination of a method for encapsulating function in to distinct units of code; an approach for representing relevant aspects of each system in a unified parameter space; and mechanisms for mapping and transmitting this space between the code units and the outside world. This document defines the elements of this paradigm and how they interact.

Chapter 2

COSMOS JSON

JavaScript Object Notation(JSON) was chosen as the medium for exchange of information within COSMOS. It addresses the need to communicate various data types in a platform independent way, while remaining simple and direct. Additionally, we have shown that complex structures represented as JSON, and then compressed using GZIP, occupy no more more space than 50% of the original binary structure. The use of JSON therefore need not incur a significant cost in storage and/or transmission.

While pretty much the full range of JSON is supported in COSMOS, only a subset as it relates to the defined parameter space is practically useful. Additionally, COSMOS has extended the JSON specification internally by adding a variety of defined usages. This has placed limits on the range of JSON that actually makes sense within COSMOS, as well as generating JSON that would not be fully recognized by the specification.

In order to support the various data types a distinction is made between types of integer, and types of floating point. This distinction is then extended to the various data structures defined within COSMOS. This is all recognized and generated internally by the COSMOS JSON code. The details of this are elaborated on below.

In order to support the use of JSON to request values, support has been added for JSON Objects that do not include a value. This modified JSON Object, consisting only of a comma separated list of strings, can then be filled in with associated values, thereby generating a standard JSON Object. As an example, the truncated JSON object {"string1","string2","string3"} would be used to return the actual JSON {"string1":value1,"string2":value2,"string3":value3}.

Finally, in addition to the fixed arrays defined by the JSON standard, COSMOS JSON provides support for something akin to vectors which grow dynamically. This is implemented through the combination of a count variable, and a 3 digit numerical name extension. Multiple levels are supported through multiple numerical extensions.

Chapter 3

Hierarchical Organization

Each complex of systems in COSMOS should be considered a *Domain*. Within each Domain there will exist a variety of interacting physical *Nodes* along with their attendant data and functionality. Each Node represents an assemblage of linked hardware, driven by associated software, that is performing some common purpose. Each Node in a Domain has a unique name, and communication is possible between each Node by way of this name.

The functionality of each Node is implemented through a suite of programs, running on one or more processors, in direct network communication with each other. The software actions that perform the functionality of a Node are divided between one-off programs called *Commands*, and persistent programs called *Agents*. Both are considered to be COSMOS *Clients*, which means that they have access to the COSMOS parameter space and associated communication mechanisms. Agents are additionally considered to be *Servers*. As such, each Agent has its own name and can accept *Requests* over the network, as well as broadcasting relevant data about itself.

All communications are at the level of Node:Agent. They will consist of either a single packet message, or a file. Messages will either be Requests passed from one Client to a Node:Agent, or Broadcasts, which can be heard by any Client. Requests always generate a response, which will contain, at a minimum, "[OK]" or "[NOK]" with respect to the incoming Request. Files will be transferred within a directory structure that reflects the COSMOS hierarchy, some portion of which will be kept on any processor capable of supporting a file system. This directory structure will always start at what is referred to as *cosmosroot*. Within a Domain, cosmosroot will contain a folder for Domain resources, and a folder for Domain Nodes. Each Node folder will contain descriptive files for that Node, as well folders for incoming files, outgoing files, and data. The incoming and outgoing folders will be further divided into folders for each Agent.

Chapter 4

COSMOS Name Space

The suite of programs that make up each Node are tied together through a common pool of information that they share back and forth. Each piece of information in this pool is given a unique name with which that information can be accessed. These names exist in a Name Space hierarchy, subdivided by major divisional differences. Each name in the Name Space is tied to a type, a unit, a range of acceptable values, and a meaning. Increasing levels of complexity in the name hierarchy are represented by separate words in the name, demarcated by underscores.

One portion of this Name Space is dedicated to general system level information, while the other deals with Node specific information. Initialization files are provided for the Node specific information, which are stored at the Node level of cosmosroot. The following sections describe each of these divisions and their meaning.

4.1 System Level

4.1.1 User

Allows the definition of Users within the system. This allows the implementation of an authentication mechanism, as well as tracking and assignment.

Base Name	Data Type	Units	Meaning
user_cpu	name		CPU assigned to
user_name	name		Name of User
user_node	name		Node of User
user_tool	name		Tool currently being used

4.1.2 Agent

Information concerning the internal functions of each Agent.

Base Name	Data Type	Units	Meaning
agent_addr	string		Network address
agent_aprd	double	seconds	Activity period
agent_beat	hbeat		Heartbeat contents
agent_bprd	double	seconds	Heartbeat period
agent_bsz	uint16		Request buffer size
agent_node	name		Node assigned to
agent_ntype	uint16		Network protocol type
agent_pid	int32		Process ID
agent_port	uint16		Network port
agent_proc	name		Name
agent_sohstring	string		JSON State of Health to output
agent_stateflag	uint16		Agent state
agent_user	string		User assigned
agent_utc	double	MJD	Timestamp

4.1.3 Event

4.1.4 Physics

4.2 Node Specific

4.2.1 Node

All names beginning with *node_* present information about the Node as a whole. The initialization file is name *node.ini*. This file typically contains the name of the node, its type, and counts for the numbers of entries in the other divisions. Currently supported names are:

Base Name	Data Type	Units	Meaning
node_name	string		Name of Node
node_type	uint16		Type of Node
node_hcap	float		Heat Capacity
node_mass	float	kg	Mass
node_area	float	m sq	Area
node_battcap	float	amp hr	Battery Capacity
node_battlev	float	amp hr	Battery Level
node_powuse	float	watts	Power Use
node_powgen	float	watts	Power Generation
node_utc	double	modified julian day	Coordinated Universal Time
node_loc	locstruc		Complete Position and Attitude
node_loc_att	attstruc		Complete Attitude
node_loc_pos	posstruc		Complete Position
node_loc_att_geoc	qatt	quaternion, meters, seconds	0th, 1st, 2nd derivative of geocentric attitude
node_loc_att_icrf	qatt	quaternion, meters, seconds	3 derivatives of ICRS based attitude
node_loc_att_lvlh	qatt	quaternion, meters, seconds	3 derivatives of LVLH based attitude
node_loc_att_selc	qatt	quaternion, meters, seconds	3 derivatives of selenocentric based attitude
node_loc_pos_bary	cartpos	meters, seconds	3 derivatives of barycentric based position
node_loc_pos_eci	cartpos	meters, seconds	3 derivatives of ECI based position
node_loc_pos_geoc	cartpos	meters, seconds	3 derivatives of geocentric based position
node_loc_pos_geod	geoidpos	radians, meters, seconds	3 derivatives of geodetic based position
node_loc_pos_sci	cartpos	meters, seconds	3 derivatives of SCI based position
node_azfrom	float	radians	Azimuth from nearest target
node_azto	float	radians	Azimuth to nearest target
node_elfrom	float	radians	Elevation from nearest target
node_elto	float	radians	Elevation to nearest target

4.3 Piece

Under the *Node* level is the *Piece* level. This deals with all the physical elements of the *Node*. Any other levels below this will be tied to some physical *Piece*. Each *Piece* is given a name and a type, along with physical dimensions, the meaning of which vary with type. Currently supported types are:

- Dimensionless: 1 point, location only
- Internal Panel: n points defining perimeter of panel, dimension indicating thickness
- External Panel: same as internal but cross product of any 3 sequential points is external direction
- Box: 8 points, representing 2 parallel sides with points in same order, first side cross product points out, dimension indicates wall thickness
- Cylinder: 3 points, 1st at vertex, 2nd in center of mouth, 3rd on perimeter, dimension indicates wall thickness
- Cone: 3 points, one end, other end, perimeter of other end, dimension indicates wall thickness
- Sphere: 2 points, 1st at center, 2nd on radius, dimension indicates wall thickness

Additional static traits are heat capacity, emmissivity, and absorptivity. Dynamic traits are temperature, and heat content.

A subset of *Parts* will have some further special qualities and be listed as *Components*. In this case, a *Component* index will be listed as well. *Parts* without a *Component* aspect will be given a *Component* index of -1.

4.4 Component

A *Component* represents the electrical nature of a *Part*. Each *Component* is assigned to a Power *Bus* index. It is statically assigned a static nominal amperage and voltage. It is also dynamically assigned an instantaneous amperage and voltage, and whether it is on or not. Each Component also has an index back to the *Part* that it is tied to.

A subset of *Components* represent more complex *Devices*. These are then given a *Device* type and index. *Components* without a *Device* aspect will be given a *Device* type and index of -1.

4.5 Devices

A different *Device* type exists for each unique type of *Component*. Each *Device* type has its own set of static and dynamic data elements, as well as an index back to the *Component* it is tied to. Currently supported *Device* types are:

- Payload: Each payload can define up to 10 keys as paired names and floating point values.
- Sun Sensor: Four quadrant sun sensor. Static values are the alignment of the device in the body frame. Dynamic values include the voltages on the four quadrants and the derived azimuth and elevation.
- Inertial Measurement Unit: 3 axis accelerometer, gyros and magnetometer. Static values are the alignment of the device in the body frame. Dynamic values can be first three derivatives of position, first three derivatives of the attitude, and a magnetic field vector.
- Reaction Wheel: Static values are the alignment of the device in the body frame, the moments of inertia of the device, and its maximum angular speed. Dynamic values can include the instantaneous angular speed and acceleration.
- Magnetic Torque Rod: Static values are the alignment of the device in the body frame and the magnetic moment of the rod. Dynamic values include the magnetic field.
- Central Processing Unit: Static values are the maximum disk and memory available. Dynamic values are the instantaneous load, disk and memory usage.
- Geographic Position System: Dynamic values include the first two derivatives of position.
- Antenna: Static values include the antenna pattern?
- Receiver:
- Transmitter:
- Transceiver:
- String: Serial string of photovoltaic cells. Static values include the area, absorptivity, maximum power production and the efficiency response to temperature. Dynamic values are power.
- Battery: Static values of charging efficiency. Dynamic values of instantaneous level.
- Heater:
- Motor:
- Rotator:
- Temperature Sensor:
- Thruster:
- Propellant Tank:
- Switch:

Chapter 5

COSMOS Data Structure

The *COSMOS Data Structure* has a space reserved to represent any data value needed to support any of the elements described previously. At its top level, it is divided into a Static and a Dynamic part. The Static part, **cosmosstruc_s**, contains elements that are not subject to significant change over short periods of time. While not necessarily completely static, these values change only infrequently, and a record of their past values is not required. The Dynamic part, **cosmosstruc_d**, is meant for values that are in constant flux. Because these values change so frequently it is expected that you might want to store 1000's of copies, or more, for quick access.

Within each of **cosmosstruc_s** and **cosmosstruc_d** are parts for the major divisions introduced above. **nodesstruc** stores information concerning *Nodes*, **agentstruc** data concerning *Agents*, and **eventstruc** data concerning *Events*.

The **eventstruc** contains:

- a time stamp
- an event name
- an event flag
- an event type
- an event body (contents dependent on event type)

The **agentstruc** contains:

- agent internals
- a name
- a time stamp
- the period of the agents heartbeat
- the size of the agents transfer buffer

Chapter 6

COSMOS Name Space

In order to provide a bridge between the COSMOS Data Environment and the internal data environment of program code, COSMOS has created a uniform space of names that represent each of the elements listed above. By providing a map between these names and internal data elements, values can be imported and exported without the need for either recompiling, or accompanying meta data.

The naming scheme follows the hierarchical nature of the data environment, with each level being represented in the name. As a compromise between human readability/writeability and programming efficiency a number of rules and exceptions have been defined:

- Words are separated by "_"
- Words are kept to a minimum of 2 characters and a maximum of 4
- Redundant leading words are stripped
- Arrays of similar items are indicated with a trailing 3 digit zero based number
- All such number are appended at the end, separated by "_"

The resulting name space is defined as follows:

6.1 Node

6.2 Part

Base Name	Data Type	Level	Units	Meaning
part_abs	double	1		
part_area	double	1		
part_cidx	int16	1		
part_cnt	int16	0		
part_com	rvector	1		
part_dim	double	1		
part_emi	double	1		
part_hcap	double	1		
part_hcon	double	1		
part_heat	double	1		
part_mass	double	1		
part_name	string	1		
part_pnt	rvector	2		
part_pnt_cnt	int16	1		
part_temp	double	1		
part_type	uint32	1		
part_tidx	int16	1		

Chapter 7

COSMOS Communications and File Hierarchy