Autocoding and Dictionaries on SMAP and MSL

Ed Benowitz
MSL/SMAP FSW Developer

Chris Swan
SMAP System Engineer

NASA/Jet Propulsion Laboratory
Caltech

http://smap.jpl.nasa.gov/
http://mars.jpl.nasa.gov/msl/

Copyright 2014 California Institute of Technology. Government sponsorship acknowledged.
Agenda

• Summary
• Mission Overview / Dictionary Overview
• MSL Dictionary/Autocoding Process
  – Commands/IPC Messages
  – Engineering Health and Accountability (EHA)
  – Event Record (EVR)
  – Data Products (DP)
  – Parameters
• MSL Lessons Learned
• SMAP Dictionary/Autocoding Process
  – Commands/EHA
  – EVR
  – Data Products
  – Parameters
• SMAP Lessons Learned
• MSL/SMAP compare contrast
Executive Summary

- Both SMAP and MSL
  - Used XML dictionaries for commands, telemetry and data products.
- The MSL and SMAP missions made extensive use of autocoding
  - Autocoding
    - Generating flight C code from XML
- MSL
  - Developers wrote the autocoder input XML by hand
- SMAP
  - Leveraged lessons learned from the MSL
  - Streamlined requirement sources
    - Used XML dictionaries for more types of specification
  - Used a web/database tool called Dictionary Management System (DMS) to:
    - Specify dictionary elements
    - Generate XML
      - Dictionaries
      - Autocoder’s XML inputs
Mission Scope/Overview

• MSL Overview
  – Latest Mars rover mission
  – Three mission phases
    • Cruise
    • Entry, Descent, and Landing
    • Surface
  – Highly redundant hardware
  – 10 instruments
  – Mobility system with autonomy
  – Drill
  – Robotic arm
  – Cameras

• SMAP Overview
  – Earth orbiter
  – Mostly single string with a few redundant devices
  – 2 instruments
  – Rotating spun section
  – Complex mechanical deployment

• Same flight software architecture for both missions
  – Written in C
  – Operating system: VxWorks
  – Architecture: Message passing via Inter-Process Communication (IPC)
Dictionary Overview

• Dictionary:
  – A machine-readable file that describes commands and telemetry formats
  – Allows ground tools to
    • Encode commands
    • Decode telemetry and data products
  – Is consistent between flight software and ground software
  – Contains human-readable command and telemetry descriptions
## Code / Dictionary Comparisons

<table>
<thead>
<tr>
<th></th>
<th>MSL</th>
<th>SMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw KLOC</td>
<td>Physical KLOC</td>
<td>Logical KLOC</td>
</tr>
<tr>
<td>Handwritten C</td>
<td>1,323 788 470</td>
<td>Handwritten C 490 252 155</td>
</tr>
<tr>
<td>Handwritten XML</td>
<td>409 368 277</td>
<td>Autogenerated C 142 117 95</td>
</tr>
<tr>
<td>Autogenerated C</td>
<td>4,049 2,563 1,101</td>
<td>Autogenerated XML 157 152 118</td>
</tr>
<tr>
<td>Total (XML and C)</td>
<td>7,629 5,369 3,107</td>
<td>Total (XML and C) 790 522 368</td>
</tr>
</tbody>
</table>

### MSL

<table>
<thead>
<tr>
<th>Commands</th>
<th>4000</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>EHA</td>
<td>19,600</td>
<td>3400</td>
</tr>
<tr>
<td>EVR</td>
<td>26,000</td>
<td>4200</td>
</tr>
<tr>
<td>Data Products</td>
<td>600</td>
<td>30</td>
</tr>
</tbody>
</table>
**MSL Dictionary Process**

- Systems Engineering “owns” the dictionary
  - Can make dictionary updates that do not affect FSW

- Requirements Sources
  - Command/EHA dictionary specified via spreadsheet by Systems Engineering
    - Converted to XML
  - Functional Description Document
    - Word document detailing behavior specification
    - Included requirements and command/telemetry specifications
  - Requirements
    - Via DOORS Database
    - Included requirements for specific dictionary elements
    - All sources are effectively incomplete (scope, level of detail)
    - FSW could self generate commands/telemetry independent of requirements sources

- All data stored and managed as flat files (Word, Excel, Text, XML)
  - Distributed among a large team

- FSW Change management
  - Items checked into Configuration Management (CM)
    - All autocoder inputs
    - All autocoder tools
    - All handwritten code
  - All autogenerated code was NOT checked into CM
    - Much developer time was spent regenerating code
MSL Autocoder: Command Process

- Commands
  - Opcode
  - Arguments
  - Argument ranges
- IPC Messages
- IPC Queues

• Systems Engineer
  - Systems Command Spreadsheet
  - System Command tool
  - Systems Command XML input

• FSW Developer
  - IPC/Command Input XML
  - Module IPC/Command Autocoder
  - Module IPC and Command C code

• Updated command descriptions

• Systems Merge tool
  - Final XML Command Dictionary

• Module Command XML output
  - FSW Command Merge tool
  - FSW XML Command Dictionary

• Ground System Software
MSL Autocoder: EHA Process

1. Systems Engineer
   - Systems EHA Spreadsheet
   - Systems EHA tool

2. FSW Developer
   - FSW EHA Input XML
   - FSW EHA Autocoder
     - Module EHA C code

3. Systems Merge tool
   - Derived channel XML
   - Systems EHA XML input
   - FSW EHA Merge tool
     - Module EHA XML output

4. Final XML EHA Dictionary
   - Ground System Software

5. FSW XML EHA Dictionary

Flow:
- Systems Engineer uses Systems EHA Spreadsheet to generate Systems EHA tool.
- FSW Developer uses FSW EHA Input XML to generate FSW EHA Autocoder.
- Systems Merge tool receives derived channel XML and Systems EHA XML input to generate Systems EHA tool.
- FSW EHA Merge tool receives Module EHA XML output to generate Final XML EHA Dictionary.
- Ground System Software uses Final XML EHA Dictionary.
**MSL Autocoder: EVR Process**

- Developers write EVRs in their C Code
  - An EVR works similarly to a printf
- To specify enumerations
  - Developers map by hand in XML
    - C-enumerations data types to EVR arguments
  - This is the EVR enumeration XML file
**MSL Autocoder: Data Products**

- A data product is a structured binary file onboard the spacecraft
- Given a dictionary, the ground tool can decode any data product into a human readable text form
- To accomplish this on flight and test platforms with difference endian-ness and compilers
  - The encoding, alignment, and endian-ness of all data products is fully specified in a machine-readable consistent way
- Data products can contain multiple Data Product Objects (DPO)
  - Did NOT specify or restrict what DPOs might be in a given data product
- DPOs allow specification of
  - Primitive types
  - Fixed size multidimensional arrays
  - C-like structs
  - Top level variable-sized arrays
  - Strings
- Data product autocoder
  - FSW developers specify data products and DPOs in XML
  - The DP autocoder generates C code to marshal C data structures to data products
    - Performs packing, byte swapping across platforms
- XML representation
  - All data products names/ids are merged into one XML
  - Each DPO is kept separately in its own XML file in the final dictionary
MSL Autocoder: Data Product Process

FSW Developer → FSW Data Product / DPO Autocoder

FSW Data Product and DPO C code

DPO XML output

Module Data Product XML output

FSW XML Data Product Dictionary

Ground System Software
MSL Autocoder: Parameter Process

- Parameters are
  - Saved onboard settings that are changed by command
- Every parameter has
  - A set of generated commands
  - Non-volatile storage
  - A data product
MSL Lessons Learned

• ID Assignment and Locking
  – Late requirement
  – FSW autocoder tools had responsibility
    • IDs were tracked across XML and text files
  – Added implementation overhead in earlier versions of FSW
    • Translation in flight code of locked ids to FSW enumerations
  – Consider tools outside of FSW to track IDs in the future missions

• When to autocode
  – Autocode flight/ground interfaces when
    • The generated code is well defined, repetitive
    • There is a need to synchronize definitions between flight and ground

• Running all the tools to build the dictionary is labor intensive

• Parameters
  – There is no centralized parameter dictionary
    • Caused additional burdens for ground tools tracking parameters over time

• Keeping FSW and Systems in sync on definitions was a challenge

• Requirements sources
  – Requirements in Doors, requirements in Excel, systems command definitions in Excel, FDD word documents
  – Too many sources for the same information (or subsets of the information)

• Overhead
  – Running autocode tools for every checkouts increased build times dramatically
SMAP: Dictionary Process

- Systems Engineering “owns” the dictionary
  - Can make dictionary updates that do not affect FSW
- All data stored in a centralized database and exported as XML
  - SMAP Dictionary process is focused around a web application called DMS (Dictionary Management System).
    - DMS collects input from FSW and Systems engineering, validates, merges, and tracks it.
    - DMS also managed unique IDs of all dictionary elements
  - SMAP elected to incorporate Parameters and Fault Protection Monitors and Responses into the dictionary.
    - This was driven by their interrelation with other dictionary elements and a desire to reduce requirements sources.
- Requirements Sources
  - To streamline requirement sources the SMAP project opted to treat the dictionary as requirements and to centralize them via DMS.
- FSW Change management
  - Items checked into CM
    - All autocoder inputs
    - All autocoder tools
    - All handwritten and autogenerated code
Dictionary Interconnectivity

DMS validates all interconnectivity (on input and for every release) and makes it transparent to the team.
SMAP Autocoder: Command Process

FSW developer

FSW developer

System Engineer

DMS

FSW Command
Merge Tool

FSW Command Autocoder

Module Command XML

Module Command C code

Command XML dictionary (Sandbox)

Command XML dictionary (Official)

FSW developer

Ground System Software

FSW developer
SMAP Autocoder: EHA Process

FSW developer

System Engineer

DMS

Module EHA XML

FSW EHA Autocoder

FSW EHA Merge Tool

EHA XML dictionary (Sandbox)

Module EHA C code

EHA XML dictionary (Official)

Ground System Software
SMAP Autocoder: EVR Process

- Unlike MSL, the SMAP autocoder is able to fully extract enumerations by determining the enum types of all EVR arguments automatically.
SMAP Autocoder: Data Product Process

- Like MSL, given a dictionary, the ground tool can decode any data product into a human readable text form
- To accomplish this on flight and test platforms with difference endian-ness and compilers
  - The encoding, alignment, and endian-ness of all data products is fully specified in a machine-readable consistent way
- SMAP allows nesting and repetition
  - Primitive types
  - C-like structs
  - Arrays
  - But there are redundant ways to specify nested structures in XML
- Unlike MSL, SMAP fully specifies data product contents
  - No DPOs
- XML representation
  - All data products names/ids are merged into one XML
  - Each Data Product’s contents is kept separately in its own self-contained XML file
- SMAP does not have a data product autocoder
  - Developers manually write code for
    - Data product ids
    - Marshal and byte swap data products
SMAP Autocoder: Data Product Process

Flight Software Developer → Data Product C code

System Engineer → DMS → Data Product XML

Ground System Software
• Unlike MSL, SMAP does not have a parameter autocoder
• Shows parameter state to the ground via EVR when possible
**SMAP: Lessons Learned**

- **Single Source = Big Win**
  - Migrating to a single source for all dictionary specification was major improvement.
  - Dramatically reduced mechanical effort required of collecting/merging/analyzing individual files (also reduced error)
  - Enabled the validation of inputs across dictionaries and enhanced rule based validation on inputs (beyond the limits of XML schema)
  - Allowed for rich metadata to follow dictionary elements (including test history)
  - Reduced Systems / FSW dictionary inconsistencies

- **Accessibility and Ease of Change brought challenges**
  - Configuration Control
    - SMAP dictionaries followed standard JPL project CM/CC which is primarily a paper process.
    - Sync’ing up the paper for a given change with the digital change code could be challenge
  - Work Flow
    - During development inputs would often stream in over multiple weeks
      - What was ready for FSW?
    - Users want more focused view of their work
    - Management wanted to be able to track the work better
    - Systems wanted to know what they could test

- **Auto generated code checked in (not regenerated) saved on build time.**
Compare Contrast MSL and SMAP

• Single Source (DMS) of information was a major improvement
  – Addressed ID assignment issues, FSW/System Sync Issues
  – Combined with the web application this saved multiple FTE per year
  – Made development progress transparent and helped forecast work to go
  – Enabled easy access to rich meta data (V&V history, cognizant engineer)
  – Validation of input helped improve resulting dictionary (and underlying FSW code)
  – Collapsed several disparate tools into a single tool

• Dictionary as requirements
  – Forced the dictionary to be better (FSW codes to it)
  – Supported the single source

• Autocoding
  – IPC autocoder was useful on MSL, might have been beneficial on SMAP.
  – Data Product autocoder was not needed on SMAP but that was due to scope and number of data products.

• MSL parameters went overboard
  – This had ripple effects across the dictionary (commands, data products, etc)
  – Lack of an up front dictionary resulted in problems tracking them
    • Traceability and visibility issues regarding the default values

• MSL top level Data Product structure was not specified
  – Resulted in difficulty understanding what a given product would contain
MSL Autocoder: Alarms and Ground Derived Channels

- Channel information independent of flight software
  - Alarms
    - When a channel goes out of range, alert the ops team
  - Derived Channels
- System engineers specify in a spreadsheet
- Autocoder converts alarm, channel spreadsheets into XML for use by the ground system
  - This also needed to validate that the alarm was valid
**MSL Autocoder: Fault Protection**

- **Terminology**
  - Monitors
    - Identify a persistent problem
  - System response
    - Makes large system level changes to address a fault condition
    - Enlist multiple modules
- **Autocoding fault protection information**
  - Only a small subset of fault protection was autocode
    - Restricted to hardcoded tables and enumerations
      - Enumerations of monitors and responses
      - Arrays mapping monitors to responses
  - System engineers provided a csv text spreadsheet file
  - The autocoder generated C code from the .csv file

![Diagram](image_url)
Lines of Code

• Raw
  – Count every line once, regardless of its content

• Physical
  – Raw minus blank and comment only lines

• Logical
  – The number of executable statements, as computed by JPL’s SLIC program counter
MSL Autocoder: Commands and IPC Process

• System engineers describe commands in a spreadsheet
  – Specify command restrictions and ranges
  – A systems autocoder converts systems inputs from Excel to XML
• The FSW developer creates XML describing the following per module
  – Commands
    • Opcode
    • Arguments
    • Argument ranges
  – IPC Messages
  – IPC Queues
• The IPC/command autocoder (per module)
  – Generates
    • C Code for decoding commands
    • C Code for sending and receiving IPC messages
    • A module level output command XML
  – Tracks and preserves command opcodes across builds
• The FSW command merge tool concatenates all module command XML, creating the command dictionary
MSL Autocoder: EHA Process

- System engineers describe EHA in a spreadsheet
  - Assign ids and descriptions
  - Create derived channels
    - For the ground, not used by FSW
- The FSW developer creates XML describing the following per module
  - Channel id
  - Data type
- The EHA autocoder combines FSW module XML and system input XML to produce
  - C Code for pushing channels
- The FSW EHA merge tool concatenates all module EHA XML, creating the EHA dictionary
  - Checks that FSW implemented all channels systems requested
    - Using the systems EHA XML input
MSL Autocoder: EVR Process

- Developers write EVRs in their C Code
  - An EVR works similarly to a printf
- To specify enumerations
  - Developers map by hand in XML C-enumerations data types to EVR arguments
  - This is the EVR XML file
- The EVR autocoder
  - Extracts EVRs from the C code
    - Data types based on the format specifier
  - Assigns EVR ids
  - Generates a header file with all EVR ids
  - Extracts enumerations and matches them to EVR arguments based on the EVR XML
  - Outputs the module EVR XML file
- An EVR XML merge tool
  - Merges all module level EVR XML files into one large XML file
MSL Autocoder: Statecharts

- MSL had two statechart autocoders
  - Graphical autocoder
    - Developers draw statecharts in the MagicDraw UML graphical editor
    - The autocoder generates C code from the MagicDraw XML file
    - Each event maps to a function
    - Switch statement on the state

  - Text autocoder
    - Uses Samek’s Quantum Framework statechart C code library
    - Developers write a text file
    - The autocoder converts the text file into C code using the quantum framework
**SMAP Autocoder: Command & EHA Process**

- **Command process**
  - System engineers specify commands in DMS
  - FSW developers export their module commands from DMS as XML
  - The SMAP command autocoder generates code containing command arguments and opcodes only
    - Code is generated per module
    - Module XMLs are merged into a command dictionary usable in sandboxes during development
  - The FSW developer manually writes IPC code

- **EHA Process**
  - System engineers describe EHA in DMS
  - FSW developers export their module EHA from DMS as XML
  - The SMAP EHA autocoder generates code for pushing EHA
SMAP Autocoder: EVR Process

- Developers write EVRs in their C Code
  - An EVR works similarly to a printf
- To specify enumerations
  - Unlike MSL, the SMAP autocoder is able to fully extract enumerations by determining the enum types of all EVR arguments automatically
- The EVR autocoder
  - Extracts EVRs from the C code
  - Unlike MSL, the developer manually specifies an EVR ID
- EVRs are imported into DMS
MSL Autocoder: Parameter Process

- Parameters are
  - Saved onboard settings that can be changed by command
- Parameters are specified in structured text file
  - Containing: Name, Data type, Range, Default value
  - One level of C-like structs and arrays are allowed
  - For large sets of default values, the user can use a .csv file as input
- Every parameters has
  - A set of generated commands for
    - Change the parameter value
      - in RAM or in non-volatile storage
    - Dump the parameter to a data product
  - Non-volatile storage
- The parameter autocoder generates
  - Code to save and read from non-volatile storage
  - XML command definitions for input to the command and data product autocoders
  - Command implementations
- Typical parameter commands change a group of parameters at once
SMAP Autocoder: Statecharts

- Graphical autocoder
  - Developers draw statecharts in the MagicDraw UML graphical editor
  - The autocoder generates C code from the MagicDraw XML file
  - The generated C code uses the Quantum Framework
SMAP Autocoder: Parameter Process

• Unlike MSL, SMAP does not have a parameter autocoder
• Parameters are specified by system engineers in DMS
  – Associated commands are specified in DMS
• FSW developers manually write code that
  – Saves and retrieves parameters from non-volatile storage
    • Uses the specified default value if the parameter cannot be retrieved
  – Implements parameter commands
  – Shows parameter state to the ground
    • Via EVRs when possible
    • With data products otherwise
  – Checks parameter ranges
• DMS exports a parameter dictionary as XML