Refactoring and Quality Improvements for R2017

Overview

The GMAT project will spend a significant amount of time during our R2017 release development on code cleanup, developer documentation, quality improvements, and refactoring for improved long-term positioning.

When adding ideas, please put your initials in-line, so we know who to contact in case more information is needed.

Goals

- Ensure that GMAT can continue to respond to customer needs as the team grows and evolves
    - Make it easier for new developers to perform frequent developer tasks (i.e. new built-in functions, new propagator)
    - Improve documentation of key components (developer docs and code comments)
    - Ensure key components developed by those planned to retire or leave project are documented and code is cleaned up
    - Improve ability to add third-party plugins
- Improve system modularity and documentation
    - to support re-use of sub-components
    - to facilitate exposing of portions of GMAT through an API
- Address high value P2 improvements that often get overlooked

Key Developer Documentation

- (SPH) How to add new math and built-in GMAT functions.
- (SPH) How to add a new propagator
- (SPH) Overview of all architectural components
- (JJKP) How to add a new ephemeris format (reading/writing)
- (JJKP) Updated how to add a Plugin/Resource/Command docs

Code Cleanup

- (DJC)Document the code so that all methods are documented and no warnings are issued during builds.
- (DJC) If the Doxygen generator identifies missing blocks, fix them (Subheading: Follow the style guide!)
- (DJC) Warnings are being ignored. This is VERY dangerous – let's clean them up now, and fix them as they occur.
- (DJC) Complete the refactoring for the Propagate command and the PropagationEnabledCommand base class.
- (JJKP) Scrub the code against the Style Guide.
- (JJKP) Remove dead code
- (SPH) Remove items that have been deprecated for more than one release

Architectural Refactoring

- (DJC) Remove special case code from the Interpreter subsystem and Moderator(SPH)
- (DJC) Modularize libGmatBase into several libraries. The goal here is to fix the circular dependencies in the code, and as a side benefit to make some core components available as libraries. Having GMAT build with one library is okay, IMO, but the cross-dependencies make code maintenance more difficult.

Component Refactoring

Expansion to the Parameter Subsystem (DSC)

- (DJC) Add an option that does not require multiple inheritance
- (DJC) Add a registration method so that a class can register, rather than needing to create an object. Here's how I think we might do this -- but this doesn't work:

```
// register parameter names with info
ParameterInfo *pInfo = ParameterInfo::Instance();
pInfo->Add("AtmosDensity", Gmat::SPACECRAFT, Gmat::ORIGIN,
           "DefaultSC.AtmosDensity", GmatParam::ATTACHED_OBJ, true, true,
           false, false, "Atmospheric Density");
```

Ephemeris stuff (JJKP)

(If this isn't already being done)

(JJKP) Refactor the EphemerisFile and EphemerisPropagator subsystems to allow easy addition of new types (e.g. via a plugin). Similar to the work being done to parameters above.

SolarSystem (JJKP)

(JJKP) Update and clean up default SolarSystem values. Scrub for hard-coded values & replace with files/inputs where possible.

Existing High Value Quality Tickets

Tickets to be considered for quality improvements should be marked with the JIRA label "Quality". This filter tracks proposed quality improvements. Tickets prioritized as P1s tend to get addressed naturally. This is intended to address some high value P2s that have been ignored for too long.