Assignment to Cloned Objects in Command Mode

Darrel J. Conway
Thinking Systems, Inc.

March 14, 2012

Abstract

This document describes the structures and interfaces used in the General Mission Analysis Tool (GMAT) to support parameter setting on configured objects that are cloned as members of other objects. The document is written and formatted in a style that facilitates inclusion in the GMAT Architectural Specification.

1 Introduction

When GMAT loads and runs a script, it performs this action is several distinct stages. The script loading process takes a text description of GMAT's objects needed for the mission and converts that serialized version of the mission into the objects needed to model the mission. The script file is usually configured with the descriptions of mission resources at the top of the file, followed by a description of the mission time line in the form of a sequence of commands comprising the Mission Control Sequence. The script parsing runs in two separate modes, called "Object Mode" and "Command Mode." The distinction is made between these modes so that parameter settings that should only be applied at a specific point inside of the Mission Control Sequence can be applied at that scripted location in the time line. As an example, consider the following partial script:

```
Create Spacecraft sat;
Create Propagator prop;

BeginMissionControlSequence;

Propagate prop(sat) {sat.Periapsis};
sat.VX = 2.0;
Propagate prop(sat) {sat.Apoapsis};
sat.VX = -1.5;
Propagate prop(sat) {sat.ElapsedDays = 2};
```

For this scripting, the user clearly wants the value of sat.VX to change at specific times on the time line, and not to be applied as values on the Spacecraft object when the script is translated into GMAT objects.

When a script is deserialized into GMAT objects, those objects are placed in separate collections in memory. Named GMAT Resources are stored in a table managed by GMAT's configuration manager. Commands in the Mission Control Sequence are stored in a linked list associated with the current Sandbox. The linkages in the list are set sequentially as ordered in the script. Some command allow for subsequences; those commands manage the subsequence in a separate linked list, and execute them based on branching criteria associated with the command that manages the separate list.

When a mission is run, the resources defined in the configuration manager are all cloned into a GMAT Sandbox. These clones are places in containers called the Local Object Store and the Global Object Store for the Sandbox. Each GMAT Sandbox has its own set of object stores.

Once in the Sandbox, linkages between resources are established. Object to object linkage usually takes the form of setting pointer references on the objects that call others. In a few cases, the reference is not made using a pointer, but rather by creating a locally managed clone of the object that is referenced. Then the objects in the object stores are initialized. After the object stores are initialized, pointers to each object store are passed to the commands in the Mission Control Sequence, and each command is initialized. When a command is initialized, it locates all of the objects it needs to perform its function, and either saves the pointers to those objects or makes clones of the objects for use in the command.

During a nominal run, execution of the Mission Control Sequence proceeds by starting at the first node on the linked list of commands and calling its Execute() method. Upon return from the call, the Sandbox retrieves the next node that need to be executed. The Sandbox loops through this process until it received a NULL node as the next node to execute. Receiving NULL terminates the run.

There are three types of object references that GMAT uses when running a script. These types are

1. **Referenced objects**: Objects that are accessed by pointer to the object in the Sandbox's Global or Local Object Store.

   An example of this type of connection is the relationship between the Maneuver command, its ImpulsiveBurn object, and the Spacecraft that received the impulse. The command sits inside of the Mission Control Sequence, the Maneuver in the Sandbox's Local Object Store, and the Spacecraft in a separate slot in the object store. When the Maneuver executes, it uses the data from the ImpulsiveBurn to change the value of the velocity of the Spacecraft. That effect can be observed by examining the data on the Spacecraft object in the object store.

2. **Owned objects**: Objects that are entirely encompassed by another object.

   An example of this type of object is the integrator inside of PropSetup (scripted as "Propagator") object. The user specified the class of integrator required, but never directly script the integrator. GMAT's integrators are always hidden inside of a PropSetup.

3. **Cloned objects**: Objects that have a pristine version in an object stores, but that are used in the command or other object by creating a clone.

   An example of the use of cloning in this context is the hardware attached to a Spacecraft. The user configures FuelTank and Thruster objects directly, and then attaches those objects to a Spacecraft. When the Spacecraft is passed a pointer to a hardware element, it creates a clone of that element, which it then manages locally. This design was constructed specifically with formations of spacecraft in mind. GMAT users can configure groups of identical spacecraft by specifying the details of the hardware on one, and then attaching that hardware to each member of the formation.

The third category of object to object relationships requires a bit of finesse in the context of parameter specification for the cloned objects. Consider the following scripting:

```
...
BeginMissionSequence
Propagate prop(sat) {sat.ElapsedDays = 5.0};
% Start slowly at the next step!
prop.InitialStepSize = 3.0;
Propagate prop(sat) {sat.ElapsedDays = 5.0};
...
```

The user intends for the propagation step at the start of the second Propagate command to be 3 seconds long. However, the Propagate command has already been initialized before the Assignment command ("`prop.InitialStepSize = 3.0;`") is executed. That initialization creates a clone of the PropSetup named prop which is hidden inside of the Propagate command, and not visible in the scope of the Assignment command using the design as described so far. This document presents the design for passing data into these hidden clones of the objects that are configured in the object stores.

2 Overview of the Design

3 Elements Facilitating Cloned Object Parameter Management

4 Flow in the Sandbox