Author: Linda Jun

Created: March 25, 2016

Include statement in GMAT script

1. Purpose

The purpose of this document is to describe how loading script capability will be implemented in current GMAT. Load or Include script allows user to import one script into another script.

1. User requirements

Add a new feature to load other objects and settings so that shared configurations can be used across scripts. Both SDO and TESS need the ability to load script "snippets" to help with automation, and to avoid duplicating configuration that is used across multiple scripts. For example, SDO has three main scripts that perform targeting, and they all load tanks/thruster configurations from the same file.

Reference: <http://li64-187.members.linode.com:8080/browse/GMT-5526>

1. Functional Specification

3.1. Syntax

#Include 'PathToScriptFile\ScriptFilename.script'

* Path is optional and can be absolute or relative.
* If path is not provided or relative path is specified, use standard file path search order.

3.2. Include Statement Behavior

* Include statements can appear anywhere in the main script and/or GMAT Functions. It can appear anywhere before or after BeginMissionSequence.
* Objects and commands created in the include script should be populated in the GUI Resource and Command trees.
* Nesting includes are allowed, but it should trap cyclic includes.

3.3. GUI will not save objects or commands created from Include files, but #Include statement should be preserved.

* GUI will save the last value set to objects created in the main script.

For example in the main script:

Create Spacecraft mySat;

mySat.X = 5000;

#Include "InitializeMySat.script"

mySat.X = 7000;

In "InitializeMySat.script"

mySat.X = 6000;

From the example above, it will save 6000 as mySat.X when GUI saves the script.

1. Design

4.1. Syntax

Add an entry GMAT\_INCLUDE\_PATH to the FileManager and startup file. Update FileManger to allow multiple path just like GMAT\_FUNCTION\_PATH and MATLAB\_FUNCTION\_PATH.

4.2. Include Statement Behavior

Design approach:

To meet the Include statement behavior, the ScriptInterpreter will treat Include script files as additional lines to the main script, GMAT function script or another Include file. So when #Include statement is encountered, the scriptInterpreter will open the Include file and switch the ScriptInterpreter to parse include file until end of file is reached. When it returns from Include file parsing, the ScriptInterpreter will continue to parse from the next line where it was left off.

Implementation for Include statement:

In ScriptInterpreter:

Change scriptFilename to mainScriptFilename to avoid confusion

Add a member data to hold current script file being parsed

Add a stack to hold Include files and last file position read

Remove - Add a vector to add Include files for checking cyclic includes

Add a new function ParseIncludeBlock() to parse #Include line

This function will do:

Find full path filename of include file from the FileManager

Throw an error if Include file not found

Add a new function InterpretIncludeFile().

This function will do:

Check for self-include (A includes A)

Check for circular-include file (A includes B and B includes A)

Throw an error if self or cyclic includes found

Push current Include script and instream pointer to Include stack

Call SetInStream() with current include file

Call ReadScript()

If include stack is not empty, pop include and instream stack

In TextParser:

Add a new INCLUDE\_BLOCK to BlockType

Update EvaluateBlock()

To return INCLUDE\_BLOCK for Include statement

To save Include statement as special comment so that when script is saved, the Include statement can also be written out

Update ChunkLine()

To appropriately chunk #Include line as <#Include> <’IncludeScriptFileName’>

In GmatBase:

Add a new member data to hold script file name where the object was created from such as main script, function, or include script file name.

std::string scriptCreatedFrom;

Add a member flag indicating object is created form the main script to avoid string comparison.

bool isCreatedFromMainScript;

Add a member function to access this new member data

void SetScriptCreatedFrom(const std::string &script);

std::string GetScriptCreatedFrom();

void SetIsCreatedFromMainScript(bool flag);

bool IsCreatedFromMainScript();

In Moderator:

Change currentScriptFileName to mainScriptFileName to avoid confusion

Update ScriptInterpreter::ReadScript() to handle INCLUDE\_BLOCK:

Read header comment on option

Read first block

While current block is not empty

If current block is INCLUDE\_BLOCK

Set currentScriptBeingRead to current include file

Call ParseIncludeFile()

Call InterpretIncludeFile()

ElseIf current block is DEFINITION\_BLOCK

…

…

ElseIf current block is COMMAND\_BLOCK

….

Check for multiple BeginMissionSequence command

Throw an exception if BeginMissionSequence already in the command list.

…

ElseIf

...

EndIf

Read next local block

EndWhile

4.3. Save will not write objects or commands created from Include files, but #Include statement should be preserved

Update TextParser::EvaluateBlock() to save #Include statement as special comment so that when script is saved, the #Include statement can also be written out in the same position in the script.

Questions:

1. Do we allow user to change objects created in the Include file in the GUI or make them read-only?
2. Initial version will save the last value set to the object when saving script from the GUI. Is this reasonable? If not, what users expects to happen when saving script?
3. Initial version will not show the contents of Include files in the GUI ScriptEditor. Is it needed to add this capability in the initial version?