JSC-61777-models/dynamics/dyn_manager

JSC Engineering Orbital Dynamics Dynamics Manager Model

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

Package Release JEOD v5.3

Document Revision 1.2
February 2025

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC-61777-models/dynamics/dyn_manager

JSC Engineering Orbital Dynamics
Dynamics Manager Model

Document Revision 1.2
February 2025

David Hammen

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Executive Summary

The Dynamics Manager Model forms a component of the dynamics suite of models within JEOD v5.3. It is located at models/dynamics/dyn_manager.

This model manages the dynamic elements of a simulation. The dynamic elements in a simulation can be broadly categorized into two groups. Vehicular bodies are the primary focus of JEOD-based simulations. Forces and torques act to change the translational and rotational states of these vehicular bodies. These forces and torques lead to equations of motion that, when numerically integrated over time, yield the state history of these vehicular bodies.

The other broad category of dynamic elements are planetary bodies and related items. These too could be propagated via numerical integration. That is not the approach used in JEOD. Instead, JEOD uses ephemeris models to calculate the translational states of the planetary bodies and (somewhat) *ad hoc* rotation models to calculate the rotational states.

The model comprises three classes.

- The DynManager class manages the dynamic elements of a simulation. This is the main focus of this document.
- The DynManagerInit class contains data used to initialize a DynManager object. The DynManagerInit class was made distinct from the DynManager class to indicate to the user (and to the developers of the model) that the data in a DynManagerInit object are used at initialization time only. Changing the contents of the DynManagerInit object after using it to initialize the simulation's DynManager object has no effect on the simulation's DynManager object.
- The DynManagerMessages class classifies the types of errors, warnings, and other messages generated by the model.

Interactions With Other Models

The Dynamics Manager Model interacts with several JEOD models. Chief among these are interactions with

- The *Body Action Model* [2].
  The body actions registered with this model initialize the dynamic bodies in a simulation.

- The *Dynamic Body Model* [3].
  The dynamic bodies are the primary reason a simulation exists.

- The *DE4xx Solar System Ephemerides* [10].
  The ephemerides models describe the locations of the planets that are active in a simulation. Additionally, the DynManager class directly inherits from the class EphemeridesManager.

- The *Gravity Model* [11].
  The gravity models compute the gravitational accelerations on vehicles.

- The *Integration Model* [4].
  The integration model provides the ability to propagate the states of the simulation's dynamic bodies over time.

- The *Reference Frame Model* [8].
  The states of the vehicular and planetary bodies in a simulation are described in the form of reference frames. Additionally, the DynManager inherits the functionality of the RefFrameManager by virtue of its inheritance from EphemeridesManager.

- The *Time Model* [12].
  Time is the independent variable of the simulation.

Modes of Operation

The Dynamics Manager Model operates in one of three different modes. The typical mode of operation is the ephemeris mode. In this mode, multiple planetary bodies exert gravitational influences on the vehicles in a simulation. The states of these planetary bodies are determined by the simulation's ephemerides models. The ephemerides models, using methods provided by this model, build the base of the reference frame tree.

Simple problems should be simple to specify. The model offers two simple modes of operation that do not require an ephemerides model to be present. These are the single planet and empty space modes. In the single planet mode, the simulation has but one planetary body. A non-rotating frame with origin at the planet's center of mass is the simulation's inertial frame. Empty space mode is even simpler. There are no planetary bodies. The inertial frame has origin at some arbitrary point in space.

Object Registries and Reference Frame Services

The Dynamics Manager Model maintains searchable registries of objects of several types: planets, ephemeris items, reference frames, mass bodies, and dynamic bodies. Each registry is searchable by name. Note that much of this functionality is acquired through the inheritence of the EphemeridesManager [10], which in turn inherits from the RefFramemanager [8].

One of the key concepts in JEOD 2.0 is that of a reference frame. JEOD organizes reference frames in a tree structure. The Dynamics Manager Model manages the construction of the reference frame

tree. The root frame of the tree is the simulation's inertial reference frame. This functionality is inherited from the RefFrameManager [8].

In addition to the basic registry described above, the model provides the tools needed to build the base of the reference frame tree and to identify integration frames—frames that can be used as the basis for a propagating the state of a dynamic body. This functionality is another example of that inherited from the RefFrameManager [8].

Dynamic Bodies

Forces and torques, along with gravity, act on a dynamic body. These forces and torques lead to equations of motion that, when numerically integrated over time, yield the updated state of the vehicle. This model serves as the central hub for initializing and propagating the states of the simulation's active dynamic bodies.

Body Actions

The model maintains a list of body actions. External entities, typically a simulation user, adds elements to this list via an interface provided by this model. Each body action in the list has a subject mass body or dynamic body. The body action, when applied, will modify some aspect of this subject body.

This model uses the body action list at initialization time to initialize the states of the simulation's dynamic bodies. The same list is used during the course of the simulation run to asynchronously operate on these bodies.

State Propagation

The other key responsibility of this model with regard to dynamic bodies is to propagate their states over time. This model contains an integrator constructor that the dynamic bodies can use to create state integrators. This model computes the gravitational accelerations for each dynamic body in the simulation, invokes the dynamic bodies' force and torque accumulation methods to determine the equations of motion for those bodies, and invokes the bodies' state integrators to propagate state over time.

Contents

Chapter 1

Introduction

1.1 Purpose and Objectives of the Dynamics Manager Model

The Dynamics Manager Model manages the dynamic elements (vehicles and ephemerides) of a simulation and serves as a central repository for several kinds of named items such as vehicles, planets, and reference frames.

1.2 Context within JEOD

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [5]

The Dynamics Manager Model forms a component of the dynamics suite of models within JEOD v5.3. It is located at models/dynamics/dyn_manager.

1.3 Documentation History

Author	Date	Revision	Description
David Hammen	November, 2009	1.0	Initial Version
David Hammen	April, 2010	1.1	User guide and IVV chapters updated
Andrew Spencer	October, 2010	1.2	Reflected new inheritence tree

1.4 Documentation Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [7].

The document comprises chapters organized as follows:

Introduction - This introduction describes the objective and purpose of the Dynamics Manager Model.

Product Requirements - The requirements chapter describes the requirements on the Dynamics Manager Model.

Product Specification - The specification chapter describes the architecture and design of the Dynamics Manager Model.

User Guide - The user guide chapter describes how to use the Dynamics Manager Model.

Inspection, Verification, and Validation - The inspection, verification, and validation (IV&V) chapter describes the verification and validation procedures and results for the Dynamics Manager Model.

Chapter 2

Product Requirements

Requirement DynManager_1: Top-level Requirement

Requirement:
This model shall meet the JEOD project requirements specified in the JEOD v5.3 top-level document.

Rationale:
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.3 release.

Verification:
Inspection

Requirement DynManager_2: Registration and Lookup Services

Requirement:
The model shall provide the ability to register and lookup by name the following kinds of objects:

2.1 Planets. The model shall provide the ability to register and lookup objects of the class Planet (see *Planet Model* [6]).

2.2 Ephemeris items. The model shall provide the ability to register and lookup objects of the class EphemerisItem (see *DE4xx Solar System Ephemerides* [10]) and shall provide the ability to restrict the lookup to the JEOD-provided classes that derive from EphemerisItem.

2.3 Reference frames. The model shall provide the ability to register and lookup objects of the type RefFrame (see *Reference Frame Model* [8]) and shall provide the ability to restrict the lookup to the reference frames registered as integration frames (inertial frames or non-rotating frames that are subject to gravitation forces only).

2.4 Mass bodies. The model shall provide the ability to register and lookup objects of the type MassBody (see *Mass Body Model* [9]) and shall provide the ability to restrict the lookup to the the class DynBody (see *Dynamic Body Model* [3]).

Rationale:
The registration and lookup services are one of the driving requirements for the model.

Verification:
Inspection, Test

Requirement DynManager_3: Mode

Requirement:
The model shall provide the ability to operate in one of three modes:

3.1 Empty space mode. Vehicles fly as if they are in empty space; the sun and planets are not present. Ephemerides models are inactive.

3.2 Single planet mode. Vehicles fly around a single planet. The sun and other planets are not present. Ephemerides models are inactive.

3.3 Ephemeris mode. Vehicles fly in a system with multiple gravitational bodies. Ephemerides models are active and define the basis of the reference frame tree.

Rationale:
Simple problems should be simple to set up.

Verification:
Inspection, Test

Requirement DynManager_4: Ephemerides Initialization/Reinitialization

Requirement:
When operating in ephemeris mode, the model shall initialize the registered ephemerides models and shall provide these models with the ability to construct the base of the reference frame tree.

Rationale:
The task of creating the base of the tree belongs to the ephemerides models. This model's task is to provide a bridge between the ephemerides and reference frame models.

Verification:
Inspection, Test

Requirement DynManager_5: Ephemerides-Free Operations

Requirement:

When operating in empty space or single planet mode, this model shall define the base of the reference frame tree.

Rationale:

The ephemerides models are disabled (and may not be present) in these simple modes. The base of the reference frame tree still needs construction.

Verification:

Inspection, Test

Requirement DynManager_6: Body Actions

Requirement:

In conjunction with the *Body Action Model* [2], this model shall provide the ability to initialize and asynchronously modify aspects of a mass or dynamic body.

6.1 Action Registration. The model shall provide the ability to register a new body actions that is to be performed at the appropriate time.

6.2 Action Initialization. The model shall initialize each registered body actions at the appropriate time and shall initialize each registered action once.

6.3 Action Application. The model shall apply registered body actions at the appropriate time. Once applied the action shall be removed from the registered set of actions.

Rationale:

The Body Action Model and its interactions with other models, including this one, is one of the key innovations of JEOD 2.0.

Verification:

Inspection, Test

Requirement DynManager_7: Initialization Body Actions

Requirement:

7.1 Initialization Classes. The model shall initialize and apply registered BodyAction instances that derive from the MassBodyInit, MassBodyAttach, and DynBodyInit classes at simulation initialization time.

7.2 Processing Order. The model shall be able to properly initialize and apply a set of non-conflicting initialization actions regardless of the order in which the actions were registered.

Rationale:
The Body Action Model, along with its interactions with other models, was introduced primarily to solve the initialization problem.

Verification:
Inspection, Test

Requirement DynManager_8: Asynchronous Body Actions

Requirement:
The model shall provide the ability to initialize and apply registered BodyAction instances during the course of a simulation run.

Rationale:
This is a nice side benefit of the model.

Verification:
Inspection, Test

Requirement DynManager_9: Common Integration Scheme

Requirement:
The model shall provide the ability to use the same integration scheme for all registered dynamic bodies.

Rationale:
Unintentionally using different integration techniques introduces subtle errors into a simulation.

Note:
Some users intentionally want to use different integration techniques or integration rates for different vehicles. The JEOD 2.0 implementation does not support this. Support for this capability is planned for JEOD 2.1.

Verification:
Inspection, Test

Requirement DynManager_10: State and Time Propagation

Requirement:
The model shall manage the propagation of the states of the registered dynamic bodies and of time from one dynamic time step to the next.

Rationale:
Coordinating the integration of state and time eliminates several subtle simulation errors.

Verification:
Inspection, Test

Requirement DynManager_11: Gravitation

Requirement:

The model shall provide the ability to compute the gravitational accelerations acting on all registered dynamic bodies.

Rationale:

Computing gravitational acceleration is an essential part of state propagation.

Verification:

Inspection, Test

Chapter 3

Product Specification

3.1 Conceptual Design

3.1.1 Overview

The Dynamics Manager Model manages the dynamic elements of a simulation. The dynamic elements in a simulation can be broadly categorized into two groups. Vehicular bodies are the primary focus of JEOD-based simulations. Forces and torques act to change the translational and rotational states of these vehicular bodies. These forces and torques lead to equations of motion that, when numerically integrated over time, yield the state history of these vehicular bodies.

The other broad category of dynamic elements are planetary bodies and related items. These too could be propagated via numerical integration. That is not the approach used in JEOD. Instead, JEOD uses ephemeris models to calculate the translational states of the planetary bodies and (somewhat) *ad hoc* rotation models to calculate the rotational states.

The model comprises three classes.

- The DynManager class manages the dynamic elements of a simulation. This is the main focus of this document.

- The DynManagerInit class contains data used to initialize a DynManager object. The DynManagerInit class was made distinct from the DynManager class to indicate to the user (and to the developers of the model) that the data in a DynManagerInit object are used at initialization time only. Changing the contents of the DynManagerInit object after using it to initialize the simulation's DynManager object has no effect on the the simulation's DynManager object.

- The DynManagerMessages class classifies the types of errors, warnings, and other messages generated by the model.

3.1.2 Interactions With Other Models

The Dynamics Manager Model interacts with several JEOD models. Chief among these are interactions with

- The *Body Action Model* [2].
  The body actions registered with this model initialize the dynamic bodies in a simulation.
- The *Dynamic Body Model* [3].
  The dynamic bodies are the primary reason a simulation exists.
- The *DE4xx Solar System Ephemerides* [10].
  The ephemerides models describe the locations of the planets that are active in a simulation. Additionally, the DynManager class directly inherits from the class EphemeridesManager.
- The *Gravity Model* [11].
  The gravity models compute the gravitational accelerations on vehicles.
- The *Integration Model* [4].
  The integration model provides the ability to propagate the states of the simulation's dynamic bodies over time.
- The *Reference Frame Model* [8].
  The states of the vehicular and planetary bodies in a simulation are described in the form of reference frames. Additionally, the DynManager inherits the functionality of the RefFrameManager by virtue of its inheritance from EphemeridesManager.
- The *Time Model* [12].
  Time is the independent variable of the simulation.

3.1.3 Modes of Operation

The Dynamics Manager Model operates in one of three different modes. The typical mode of operation is the ephemeris mode. In this mode, multiple planetary bodies exert gravitational influences on the vehicles in a simulation. The states of these planetary bodies are determined by the simulation's ephemerides models. The ephemerides models, using methods provided by this model, build the base of the reference frame tree.

Simple problems should be simple to specify. The model offers two simple modes of operation that do not require an ephemerides model to be present. These are the single planet and empty space modes. In the single planet mode, the simulation has but one planetary body. A non-rotating frame with origin at the planet's center of mass is the simulation's inertial frame. Empty space mode is even simpler. There are no planetary bodies. The inertial frame has origin at some arbitrary point in space.

3.1.4 Object Registries and Reference Frame Services

The Dynamics Manager Model maintains searchable registries of objects of several types: planets, ephemeris items, reference frames, mass bodies, and dynamic bodies. Each registry is searchable by

name. Note that much of this functionality is acquired through the inheritence of the Ephemerides-Manager [10], which in turn inherits from the RefFramemanager [8].

One of the key concepts in JEOD 2.0 is that of a reference frame. JEOD organizes reference frames in a tree structure. The Dynamics Manager Model manages the construction of the reference frame tree. The root frame of the tree is the simulation's inertial reference frame. This functionality is inherited from the RefFrameManager [8].

In addition to the basic registry described above, the model provides the tools needed to build the base of the reference frame tree and to identify integration frames—frames that can be used as the basis for a propagating the state of a dynamic body. This functionality is another example of that inherited from the RefFrameManager [8].

3.1.5 Dynamic Bodies

Forces and torques, along with gravity, act on a dynamic body. These forces and torques lead to equations of motion that, when numerically integrated over time, yield the updated state of the vehicle. This model serves as the central hub for initializing and propagating the states of the simulation's active dynamic bodies.

Body Actions

The model maintains a list of body actions. External entities, typically a simulation user, adds elements to this list via an interface provided by this model. Each body action in the list has a subject mass body or dynamic body. The body action, when applied, will modify some aspect of this subject body.

This model uses the body action list at initialization time to initialize the states of the simulation's dynamic bodies. The same list is used during the course of the simulation run to asynchronously operate on these bodies.

State Propagation

The other key responsibility of this model with regard to dynamic bodies is to propagate their states over time. This model contains an integrator constructor that the dynamic bodies can use to create state integrators. This model computes the gravitational accelerations for each dynamic body in the simulation, invokes the dynamic bodies' force and torque accumulation methods to determine the equations of motion for those bodies, and invokes the bodies' state integrators to propagate state over time.

3.2 Mathematical Formulations

N/A

3.3 Detailed Design

See the Reference Manual[1] for a description of classes that comprise the model and for descriptions of the member data and member functions defined by these classes.

3.4 Inventory

All Dynamics Manager Model files are located in `${JEOD_HOME}/models/dynamics/dyn_manager`. Relative to this directory,

- Header and source files are located in the model `include` and `src` subdirectories. Table 3.1 lists the configuration-managed files in these directories.
- Documentation files are located in the model `docs` subdirectory. See table 3.2 for a listing of the configuration-managed files in this directory.

Table 3.1: Source Files

File Name
include/base_dyn_manager.hh
include/class_declarations.hh
include/dyn_manager.hh
include/dyn_manager_init.hh
include/dyn_manager_messages.hh
include/dynamics_integration_group.hh
src/cmake_file_list.cmake
src/dyn_bodies_primitives.cc
src/dyn_manager.cc
src/dyn_manager_messages.cc
src/dynamics_integration_group.cc
src/gravitation.cc
src/initialize_dyn_bodies.cc
src/initialize_model.cc
src/initialize_simulation.cc
src/integ_group_primitives.cc
src/mass_bodies_primitives.cc
src/perform_actions.cc

Table 3.2: Documentation Files

File Name
docs/dyn_manager.pdf
docs/refman.pdf
docs/tex/change_history.tex
docs/tex/dyn_manager.bib
docs/tex/dyn_manager.sty
docs/tex/dyn_manager.tex
docs/tex/exec_summary.tex
docs/tex/intro.tex
docs/tex/ivv.tex
docs/tex/makefile
docs/tex/reqt.tex
docs/tex/spec.tex
docs/tex/user.tex

Table 3.3: Verification Files

File Name
verif/SIM_removable_body_action/README
verif/SIM_removable_body_action/S_define
verif/SIM_removable_body_action/S_overrides.mk
verif/SIM_removable_body_action/Title
verif/SIM_removable_body_action/Log_data/log_aero.py
verif/SIM_removable_body_action/Log_data/log_atmos.py
verif/SIM_removable_body_action/Log_data/log_constants.py
verif/SIM_removable_body_action/Log_data/log_earth_RNP.py
verif/SIM_removable_body_action/Log_data/log_ephemerides.py
verif/SIM_removable_body_action/Log_data/log_force_torque.py
verif/SIM_removable_body_action/Log_data/log_grav.py
verif/SIM_removable_body_action/Log_data/log_orbelem.py
verif/SIM_removable_body_action/Log_data/log_state.py
verif/SIM_removable_body_action/Log_data/log_suite.py
verif/SIM_removable_body_action/Log_data/log_time.py
verif/SIM_removable_body_action/Modified_data/aero_drag.py
verif/SIM_removable_body_action/Modified_data/grav_controls.py
verif/SIM_removable_body_action/Modified_data/integration.py

Continued on next page

Table 3.3: Verification Files (continued from previous page)

File Name
verif/SIM_removable_body_action/Modified_data/mass.py
verif/SIM_removable_body_action/Modified_data/solar_flux.py
verif/SIM_removable_body_action/Modified_data/state.py
verif/SIM_removable_body_action/Modified_data/time.py
verif/SIM_removable_body_action/Modified_data/uniform_wind.py
verif/SIM_removable_body_action/SET_test/common_input.py
verif/SIM_removable_body_action/SET_test/RUN_1/Title
verif/SIM_removable_body_action/SET_test/RUN_1/input.py
verif/SIM_removable_body_action/SET_test_val/RUN_1/input.py
verif/SIM_removable_body_action/SET_test_val/RUN_1/log_ constants.header
verif/SIM_removable_body_action/SET_test_val_rh8/common_input.py
verif/SIM_removable_body_action/SET_test_val_rh8/RUN_1/input.py
verif/SIM_removable_body_action/bin/JEOD_1.5_to_2.0
verif/SIM_removable_body_action/docs/DYNCOMP.pdf
verif/SIM_removable_body_action/docs/tex/DYNCOMP.bib
verif/SIM_removable_body_action/docs/tex/DYNCOMP.sty
verif/SIM_removable_body_action/docs/tex/DYNCOMP.tex
verif/SIM_removable_body_action/docs/tex/DYNCOMPAbstract.tex
verif/SIM_removable_body_action/docs/tex/model_name.mk
verif/SIM_removable_body_action/docs/tex/figs/derivative.png
verif/SIM_removable_body_action/docs/tex/figs/initialization.png
verif/SIM_removable_body_action/docs/tex/figs/scheduled.png
verif/SIM_removable_body_action/docs/tex/paper/siw.pdf
verif/mock/dyn_manager_mock.hh
verif/mock/dynamics_integration_group_mock.hh
verif/unit_tests/CMakeLists.txt
verif/unit_tests/dyn_bodies_primitives_ut.cc
verif/unit_tests/dyn_manager_init_ut.cc
verif/unit_tests/dyn_manager_ut.cc
verif/unit_tests/dyn_manager_ut.hh
verif/unit_tests/dynamics_integration_group_ut.cc
verif/unit_tests/gravitation_ut.cc
verif/unit_tests/initialize_dyn_bodies_ut.cc
verif/unit_tests/initialize_model_ut.cc

Continued on next page

Table 3.3: Verification Files (continued from previous page)

File Name
verif/unit_tests/initialize_simulation_ut.cc
verif/unit_tests/integ_group_primitives_ut.cc
verif/unit_tests/makefile
verif/unit_tests/mass_bodies_primitives_ut.cc
verif/unit_tests/perform_actions_ut.cc

Chapter 4

User Guide

This chapter describes how to use the Dynamics Manager Model from the perspective of a simulation user, a simulation developer, and a model developer.

4.1 Analysis

This section addresses use of the Dynamics Manager Model from the perspective of a simulation analyst. The model largely works behind the scenes. With the exception of the body action list described below, little interaction between the model and the typical simulation user is required.

4.1.1 The BodyAction List

The recommended practice regarding the initialization of DynBody objects in a JEOD-based Trick simulation is to use the *Body Action Model* [2] in conjunction with the Dynamics Manager Model. The simulation's DynManager object maintains a list of Dynamics Manager Model objects. Calling the DynManager's `add_body_action()` method adds an item to the list. The DynManager draws objects from this list at the appropriate time and only does so when the objects indicate that they are ready to be executed. This mechanism disconnects the execution order of the enqueued model objects from the order in which the simulation objects are declared and from the order in which the objects are added to the list.

This separation between initialization and declaration order was first addressed in JEOD 1.4/1.5, but the C-based solution involved some rather convoluted and very inflexible code. The C++ based JEOD 2.0 solution to this problem makes for a very extensible and simpler set of code. Users are strongly recommended to take advantage of this flexibility.

4.1.2 Initialization

The DynManager member function `initialize_model()` takes two arguments: a pointer to a Trick INTEGRATOR structure and a reference to a DynManagerInit object. The DynManagerInit is a simple object. It contains three data members that are used to set values in the simulation's

dynamic manager that must remain constant for the duration of the simulation run. These three data members are:

`EphemerisMode mode` Specifies the mode in which the dynamics manager will operate. The default is ephemeris-driven mode, and this is the context under which realistic simulations will operate. Following the guideline that simple problems should be simple to specify, JEOD offers two other simplified modes, empty space and single planet mode. There is but one integration frame in these two simple modes. The two differ in that the planet in single planet mode presumably has a gravitational field that affects dynamics while gravitation is nonexistent in empty space mode.

`std::string central_point_name` This parameter names the central point and is used only when the manager operates in empty space or single planet mode.

`IntegratorConstructor * integ_constructor` The integrator constructor specifies the integration technique that is to be used to integrate all of the dynamic bodies in a JEOD simulation. See the *Integration Model* [4] for details.

If the DynManagerInit object's `integ_constructor` pointer is null (which it is by default), the dynamics manager uses the Integration Model factory constructor to create an integrator constructor based on the `option` in the INTEGRATOR structure pointer passed to the `initialize_model()`.

4.1.3 Controlling Dynamics

The DynManager `deriv_ephem_update` data member determines whether ephemerides are updated at the derivative rate or only as scheduled. Setting this element to true will cause the ephemeris models to be updated at the derivative rate. Doing so makes the calculations of the perturbative gravitational accelerations more accurate but at the expense of an increased computational burden. The default setting for this data member is false, which means ephemerides are updated at the scheduled rate as specified in the simulation's S_define file.

4.2 Integration

This section addresses use of the Dynamics Manager Model, first from the perspective of a simulation integrator and then from the perspective of a model developer who wants to use some of the functionality provided by this model.

4.2.1 S_define-Level Integration

The DynManager class provides 48 publicly accessible methods, 50 counting the default constructor and destructor. Only eight of these methods are expected to be used in a simulation S_define file. These eight methods are listed in table 4.2.1.

Table 4.1: DynManager S_define-level Methods

Method	Job Class	Description
`add_body_action`	environment[a]	Adds a BodyAction to the dynamic manager's list of body actions. The dynamics manager will eventually process this newly-queued action.
`compute_derivatives`	derivative[b]	Causes each root dynamic body registered with the dynamics manager to formulate the equations of motion for that body. This includes accumulating the collected forces and torques that on the body. If your simulation has gravitational forces, make sure you call `gravitation()` before calling `compute_derivatives()`.
`gravitation`	derivative[b]	Causes the gravity model to compute the gravitational acceleration and gravity gradient for each root dynamic body registered with the dynamics manager.
`initialize_model`	initialization	Prepares the dynamics manager for the initializations that are to follow. This method should run fairly early in the initialization process.
`initialize_simulation`	initialization	Initializes the ephemerides and performs the initialization-time body actions registered with the dynamics manager.
`integrate`	integration	Causes each root dynamic body registered with the dynamics manager to integrate the equations of motion for that body and updates time to reflect the time at the end of the intermediate step.
`perform_actions`	environment	Performs all of the body actions registered with the dynamics manager that are ready to be applied.
`update_ephemerides`	environment	Makes each ephemeris model registered with the dynamics manager update itself to reflect the current simulation time.

[a] Typically specified as a zero-rate job. Zero-rate jobs are not be called on a regular basis. They can however be called from the input file, which is the intended use of this method.

[b] It may be desirable to also call this as scheduled jobs if you are logging the accelerations of a DynBody object and want to see the derivatives reflect the current time.

Initialization Jobs

As noted in table 4.2.1, the model provides two methods that are expected to be used in an S_define file as integration class jobs. These are `initialize_model()` and `initialize_simulation()`. The first method, `initialize_model()`, prepares the simulation's DynManager object for all of the other initializations that will soon follow. See section 4.1.2 for a description of this method. The second method, `initialize_simulation()`, takes no arguments. This method initializes the gravity controls for each of the simulation's dynamic bodies, completes the initialization of the simulation's ephemeris models, and initializes and applies body actions that have been registered with the dynamics manager and are ready to run at initialization time.

The only initializations that should occur prior to the call to `initialize_model()` are initializations of the time model. Initializations that depend on dynamic bodies have fully-initialized states should be performed after the call to `initialize_simulation()`. The bulk of the calls to initialization class jobs should be sandwiched between the calls to the two DynManager initialization-time methods. The best way to ensure that jobs will take place as described above in a Trick-based simulation is to use the Trick prioritization scheme.

Two issues dictate this arrangement:

- Those intermediate initialization jobs' use of the dynamics manager requires that the dynamics manager itself be properly initialized. That means these tasks must be performed after the call to `initialize_model()`.

- Which planetary ephemerides are active is determined by those frames that have a non-zero subscription count. Initialization-time jobs that subscribe to ephemerides-based reference frames must be performed prior to the call to `initialize_simulation()` to ensure that the simulation's reference frame tree is constructed as planned.

Derivative and Integration Jobs

The model provides two methods, `gravitation()` and `compute_derivatives()`, that should be called as derivative class jobs and one method, `integrate()`, that should be called as an integration class job. The call to `gravitation()` must precede the call to `compute_derivatives()` to ensure proper computation of accelerations.

As noted in table 4.2.1, it may be desirous to also call the two derivative class jobs as scheduled or logging class jobs. Doing so will not affect simulation dynamics, but it will make the logged accelerations be consonant with the other logged values.

Scheduled Jobs

The model provides three methods that are expected to be called as scheduled jobs. The method `update_ephemerides()` updates the simulation's ephemeris models to reflect the current time. All simulations that involve multiple gravitational bodies should include a call to this method in the simulation's S_define file.

The method `perform_actions()` performs queued body actions that are ready to be applied. Some simulations only use the Body Action Model to initialize a simulation's dynamic bodies. These simulations do not need to call `perform_actions()`. A call to this method is needed if the Body Action Model is to be used to perform asynchronous operations.

Finally, the mechanism by which body actions are registered with the simulation's dynamics manager is via calls to `add_body_action()`. JEOD models do not invoke this method. The recommended practice is to call this method as a zero rate scheduled job in the simulation's S_define file. Because the calling rate is zero, the generated `S_source` file will not call this method. The method can however be invoked from a simulation input file, and this is exactly how this method is intended to be used.

4.2.2 Using the Model in a User Defined Model

The public DynManager methods that are intended for use by model developers is more-or-less orthogonal to the set intended for use at the S_define level. In particular, using `add_body_action()` and `perform_actions()` as methods invoked by a user model is not a recommended practice.

The methods intended for use in user-defined methods fall into one of three categories.

- Lookup services. The lookup methods include
  - `find_ephem_item()`. Finds the EphemItem registered with the dynamics manager whose name matches the supplied value.
  - `find_ephem_angle()`, `find_ephem_point()`, and `find_ephem_planet()`. Specializations of `find_ephem_item()` that restrict the lookup to specializations of EphemItem.
  - `find_planet()`. Finds the Planet registered with the dynamics manager whose name matches the supplied value.
  - `find_ref_frame()`. Finds the RefFrame registered with the dynamics manager whose name matches the supplied value.
  - `find_dyn_body()`. Finds the DynBody registered with the dynamics manager whose name matches the supplied value.
- Registration services. Most of the registration methods are intended for use within JEOD. The one exception is `add_ref_frame()`. JEOD cannot possibly anticipate every reference frame that users might want to use. For example, JEOD does not support the Mean of 1950 (M50) reference frame. A user-defined model can implement this frame and register it with the dynamics manager.
- Frame subscription services. The frame subscription methods are
  - `subscribe_to_frame()` Increments a frame's subscription count.
  - `unsubscribe_to_frame()` Decrements a frame's subscription count.
  - `frame_is_subscribed()` Checks whether a frame's subscription count is positive.

  Two versions of each method are defined. One takes a name of a reference frame as an argument; the other takes a reference to a RefFrame object.

4.3 Extension

The model is not designed to be extensible.

Chapter 5

Inspection, Verification, and Validation

5.1 Inspection

This section describes inspections of the Dynamics Manager Model with respect to the requirements level on the model.

Inspection DynManager_1: Top-level Requirements

This model does not yet comply with requirement DynManager_1; it has not been adequately tested. The code body does comply with the top-level requirement.

Inspection DynManager_2: Design Inspection

By inspection, the Dynamics Manager Model provides

- Registration and lookup services as specified by requirement DynManager_2. This requirement is also partially met through the DynManager's inheritence of the EphemeridesManager [10], which inherits from the RefFrameManager [8].
- Operation modes as specified by requirement DynManager_3.
- Ephemeris initialization capabilities as specified by requirement DynManager_4.
- Ephemerides-free operations as specified by requirement DynManager_5.
- Support for body actions as specified by requirement DynManager_6.
- Body initializations as specified by requirement DynManager_7.
- Asynchronous actions as specified by requirement DynManager_8.
- A common integration scheme as specified by requirement DynManager_9.

- State and time propagation as specified by requirement DynManager_10.
- Gravitational computations as specified by requirement DynManager_11.

Inspection DynManager_3: Test Inspection

One reason this document does not comply with requirement DynManager_1 is that no unit tests are supplied with the model. That said, all of the JEOD-level and many of the model-level tests use the Dynamics Manager Model. These other tests collectively demonstrate that the model does satisfy requirements DynManager_3 to DynManager_11.

5.2 Metrics

Table 5.1 presents coarse metrics on the source files that comprise the model.

Table 5.1: Coarse Metrics

	Number of Lines			
File Name	**Blank**	**Comment**	**Code**	**Total**
include/base_dyn_manager.hh	31	133	38	202
include/class_declarations.hh	11	57	9	77
include/dyn_manager.hh	70	180	108	358
include/dyn_manager_init.hh	23	127	39	189
include/dyn_manager_messages.hh	24	98	23	145
include/dynamics_integration_group.hh	41	149	51	241
src/cmake_file_list.cmake	2	0	17	19
src/dyn_bodies_primitives.cc	26	59	85	170
src/dyn_manager.cc	31	90	131	252
src/dyn_manager_messages.cc	16	27	15	58
src/dynamics_integration_group.cc	48	142	194	384
src/gravitation.cc	23	70	86	179
src/initialize_dyn_bodies.cc	51	144	198	393
src/initialize_model.cc	27	70	99	196
src/initialize_simulation.cc	24	62	66	152
src/integ_group_primitives.cc	16	46	35	97
src/mass_bodies_primitives.cc	23	58	69	150
src/perform_actions.cc	13	35	24	72
Total	500	1547	1287	3334

Table 5.2 presents the cyclomatic complexity of the methods defined in the model.

Table 5.2: Cyclomatic Complexity

Method	File	Line	ECC
jeod::DynManager::is_initialized ()	include/dyn_manager.hh	129	1
jeod::DynManager::std::get_dyn_bodies ()	include/dyn_manager.hh	176	1
jeod::DynManager::compute_derivatives ()	include/dyn_manager.hh	215	1
jeod::DynManager::reset_integrators (Dynamics IntegrationGroup & integ_group)	include/dyn_manager.hh	228	1
jeod::DynManager::integrate (double to_sim_time, Time Manager &)	include/dyn_manager.hh	237	1
jeod::DynamicsIntegration Group::is_empty ()	include/dynamics_integration_group.hh	127	1
jeod::DynManager::find_dyn_body (const std::string & body_name)	src/dyn_bodies_primitives.cc	50	3
jeod::DynManager::is_dyn_body_registered (const Dyn Body * dyn_body)	src/dyn_bodies_primitives.cc	81	1
jeod::DynManager::add_dyn_body (DynBody & dyn_body)	src/dyn_bodies_primitives.cc	91	6
jeod::DynManager::Dyn Manager ()	src/dyn_manager.cc	62	1
jeod::DynManager::~Dyn Manager ()	src/dyn_manager.cc	81	1
jeod::DynManager:: timestamp ()	src/dyn_manager.cc	95	1
jeod::std::name ()	src/dyn_manager.cc	104	1
jeod::DynManager::shutdown ()	src/dyn_manager.cc	113	1
jeod::DynManager::set_gravity_manager (Gravity Manager & gravity)	src/dyn_manager.cc	121	4

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::DynManager::add_body_action (BodyAction & body_action)	src/dyn_manager.cc	164	3
jeod::DynManager::remove_body_action (const std::string & action_name_in)	src/dyn_manager.cc	201	4
jeod::DynManager::reset_integrators ()	src/dyn_manager.cc	224	4
jeod::DynamicsIntegrationGroup::DynamicsIntegrationGroup ()	src/dynamics_integration_group.cc	51	1
jeod::DynamicsIntegrationGroup::DynamicsIntegrationGroup (JeodIntegrationGroupOwner & owner, er7_utils::IntegratorConstructor & integ_cotr, JeodIntegratorInterface & integ_inter, JeodIntegrationTime & time_mngr)	src/dynamics_integration_group.cc	60	1
jeod::DynamicsIntegrationGroup::register_base_contents ()	src/dynamics_integration_group.cc	77	1
jeod::DynamicsIntegrationGroup::~DynamicsIntegrationGroup ()	src/dynamics_integration_group.cc	87	1
jeod::DynamicsIntegrationGroup::create_group (JeodIntegrationGroupOwner & owner, er7_utils::IntegratorConstructor & integ_cotr, JeodIntegratorInterface & integ_inter, JeodIntegrationTime & time_mngr)	src/dynamics_integration_group.cc	95	1
jeod::DynamicsIntegrationGroup::register_group (DynManager & dyn_manager)	src/dynamics_integration_group.cc	112	2
jeod::DynamicsIntegrationGroup::initialize_group (DynManager &)	src/dynamics_integration_group.cc	127	4

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::DynamicsIntegration Group::add_dyn_body (Dyn Body & dyn_body)	src/dynamics_integration_ group.cc	162	7
jeod::DynamicsIntegration Group::delete_dyn_body (DynBody & dyn_body)	src/dynamics_integration_ group.cc	217	4
jeod::DynamicsIntegration Group::prepare_for_integ_ loop (double sim_endtime)	src/dynamics_integration_ group.cc	254	1
jeod::DynamicsIntegration Group::gravitation (Dyn Manager & dyn_manager, GravityManager & gravity_ manager)	src/dynamics_integration_ group.cc	266	5
jeod::DynamicsIntegration Group::collect_derivatives ()	src/dynamics_integration_ group.cc	295	3
jeod::DynamicsIntegration Group::reset_body_ integrators ()	src/dynamics_integration_ group.cc	316	2
jeod::er7_utils::integrate_ bodies (double cycle_dyndt, unsigned int target_stage)	src/dynamics_integration_ group.cc	332	5
jeod::DynManager::initialize_ gravity_controls ()	src/gravitation.cc	43	3
jeod::DynManager::reset_ gravity_controls ()	src/gravitation.cc	76	5
jeod::DynManager:: gravitation ()	src/gravitation.cc	116	5
jeod::DynManager::initialize_ dyn_bodies ()	src/initialize_dyn_bodies.cc	52	5
jeod::DynManager::initialize_ dyn_body (DynBody & body)	src/initialize_dyn_bodies.cc	92	1
jeod::DynManager::perform_ mass_body_initializations (MassBody * body)	src/initialize_dyn_bodies.cc	111	6
jeod::DynManager::perform_ mass_attach_initializations ()	src/initialize_dyn_bodies.cc	167	4

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::DynManager::perform_dyn_body_initializations (DynBody * body)	src/initialize_dyn_bodies.cc	216	19
jeod::DynManager::check_for_uninitialized_states ()	src/initialize_dyn_bodies.cc	334	8
jeod::DynManager::JEOD_DECLARE_ATTRIBUTES (EmptySpaceEphemeris)	src/initialize_model.cc	52	1
jeod::DynManager::initialize_model (JeodIntegrator Interface & integ_if, Dyn ManagerInit & init, Time Manager & time_mngr)	src/initialize_model.cc	69	1
jeod::DynManager::initialize_model_internal (Dyn ManagerInit & init, Time Manager & time_mngr)	src/initialize_model.cc	86	11
jeod::DynManager::initialize_simulation ()	src/initialize_simulation.cc	43	6
jeod::DynManager::initialize_integ_groups ()	src/initialize_simulation.cc	97	3
jeod::DynManager::update_integration_group (Jeod IntegrationGroup & group)	src/initialize_simulation.cc	122	4
jeod::DynManager::is_integ_group_registered (const DynamicsIntegrationGroup * integ_group)	src/integ_group_primitives.cc	48	1
jeod::DynManager::add_integ_group (Dynamics IntegrationGroup & integ_group)	src/integ_group_primitives.cc	58	4
jeod::DynManager::find_mass_body (const std::string & body_name)	src/mass_bodies_primitives.cc	44	3
jeod::DynManager::is_mass_body_registered (const MassBody * mass_body)	src/mass_bodies_primitives.cc	75	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::DynManager::add_mass_body (MassBody & mass_body)	src/mass_bodies_primitives.cc	85	4
jeod::DynManager::add_mass_body (MassBody * mass_body)	src/mass_bodies_primitives.cc	124	2
jeod::DynManager::perform_actions ()	src/perform_actions.cc	38	3

5.3 Requirements Traceability

The IV&V artifacts that demonstrate the satisfaction of the requirements in chapter 2 are depicted in table 5.3.

Table 5.3: Requirements Traceability

Requirement		Artifact
DynManager_1	Top-level requirement	Insp. DynManager_1 Top-level requirements Failed
DynManager_2	Registration and lookup services	Insp. DynManager_2 Design inspection
DynManager_3	Mode	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_4	Ephemerides operations	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_5	Ephemerides-free operations	Insp. DynManager_2 Design inspection
DynManager_6	Body actions	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_7	Initialization body actions	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_8	Asynchronous body actions	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_9	Common integration scheme	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_10	State and time propagation	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection
DynManager_11	Gravitation	Insp. DynManager_2 Design inspection Insp. DynManager_3 Test inspection

Bibliography

[1] (autogenerated). *Dyn Manager Reference Manual*. NASA, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, February 2025.

[2] Hammen, D. Body Action Model. Technical Report JSC-61777-dynamics/body_action, NASA, Johnson Space Center, Houston, Texas, February 2025.

[3] Hammen, D. Dynamic Body Model. Technical Report JSC-61777-dynamics/dyn_body, NASA, Johnson Space Center, Houston, Texas, February 2025.

[4] Hammen, D. Integration Model. Technical Report JSC-61777-utils/integration, NASA, Johnson Space Center, Houston, Texas, February 2025.

[5] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, February 2025.

[6] Morris, J. Planet Model. Technical Report JSC-61777-environment/planet, NASA, Johnson Space Center, Houston, Texas, February 2025.

[7] NASA. NASA Software Engineering Requirements. Technical Report NPR-7150.2, NASA, NASA Headquarters, Washington, D.C., September 2004.

[8] Spencer, A. Reference Frame Model. Technical Report JSC-61777-utils/ref_frames, NASA, Johnson Space Center, Houston, Texas, February 2025.

[9] Thebeau, C. Mass Body Model. Technical Report JSC-61777-dynamics/mass, NASA, Johnson Space Center, Houston, Texas, February 2025.

[10] Thompson, B. DE4xx Solar System Ephemerides. Technical Report JSC-61777-environment/ephemerides, NASA, Johnson Space Center, Houston, Texas, February 2025.

[11] Thompson, B. and Morris., J. Gravity Model. Technical Report JSC-61777-environment/gravity, NASA, Johnson Space Center, Houston, Texas, February 2025.

[12] Turner, G. Time Model. Technical Report JSC-61777-environment/time, NASA, Johnson Space Center, Houston, Texas, February 2025.