JSC-61777-models/dynamics/mass

# JSC Engineering Orbital Dynamics Mass Body Model

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

## Package Release JEOD v5.1

## Document Revision 2.0
## July 2023


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

JSC-61777-models/dynamics/mass

# JSC Engineering Orbital Dynamics
# Mass Body Model

**Document Revision 2.0**
**July 2023**

**Gary Turner**

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

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

**Abstract**

The Mass Body Model maintains the mass properties of all non-planetary massive objects in the simulation. Mass-Body objects that also have a dynamic state are represented as Dyn-Body objects, Dyn-Body being an inheritance from Mass-Body. Consequently, the most frequently encountered Mass Body objects are the Dyn-Body-based simulation vehicle(s); this model accurately maintains the mass properties of such entities throughout processes such as mass depletion due to fuel usage, the attaching and detaching of one Mass Body to/from another, and the relative motion of individual massive components of a composite body comprising multiple Mass Body elements (e.g. rotation of a solar panel on a vehicle).

When the dynamic state of a composite vehicle (one comprising multiple components) is integrated, only one integration is performed for the entire composite unit; the component parts are not integrated directly. It is the Mass Body Model that is responsible for computing the overall mass and inertia of the composite body that will be used in determining the dynamic response to applied forces and torques.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Purpose and Objectives of the Mass Body Model

The Mass Body Model maintains the mass properties of all non-planetary massive objects in the simulation. Mass-Body objects that also have a dynamic state are represented as Dyn-Body [3] objects, Dyn-Body being an inheritance from Mass-Body (i.e. a DynBody object is a MassBody object, but with additional descriptors). Consequently, the most frequently and obviously encountered Mass-Body objects are the Dyn-Body-based simulation vehicle(s); this model accurately maintains the mass properties of such entities throughout processes such as mass depletion due to fuel usage, the attaching and detaching of one Mass Body to/from another, and the relative motion of individual massive components of a composite body comprising multiple Mass Body elements (e.g. rotation of a solar panel on a vehicle).

When the dynamic state of a composite vehicle (one comprising multiple components) is integrated, only one integration is performed for the entire composite unit; the component parts are not integrated directly. It is the Mass Body Model that is responsible for computing the overall mass and inertia of the composite body that will be used in determining the dynamic response to applied forces and torques.

While the most obviously encountered Mass Body objects happen to be Dyn Body objects, the separation of mass properties from dynamic properties allows for a natural delineation between those objects for which dynamic state is important, and those for which it is not. When only the mass is important, a Mass Body is used; a Dyn Body is used when the object must also have a dynamic state.

For example, a user may choose to add a massive fuel tank to a vehicle with no intention of ever wanting to know how anything about the tank's position or velocity. Indeed, unless the tank is ejected at some point, then even if its dynamic state is desired, it can be obtained from the vehicle itself; maintaining its own state would be redundant.

It is therefore quite a natural development to build a dynamic body from a collection of non-dynamic massive bodies, each of which has its own unique (and possibly time-varying) mass properties. Keeping track of varying mass properties within a compound body is a non-trivial undertaking, so this model performs those tasks behind the scenes.

## 1.2 Context within JEOD

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [5]

The Mass Body Model forms a component of the dynamics suite of models within JEOD v5.1. It is located at models/dynamics/mass.

## 1.3 Document History

| Author | Date | Revision | Description |
|---|---|---|---|
| Gary Turner | January 2012 | 2.0 | Extensive revisions |
| Christopher Thebeau | Mar 2010 | 1.1 | Updated Description and Verification |
| Christopher Thebeau | Nov 2009 | 1.0 | Initial Version. |

## 1.4 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [6].

The document comprises chapters organized as follows:

**Chapter 1: Introduction** -This introduction describes the objective and purpose of the Mass Body Model.

**Chapter 2: Product Requirements** -The requirements chapter describes the requirements on the Mass Body Model.

**Chapter 3: Product Specification** -The specification chapter describes the architecture and design of the Mass Body Model.

**Chapter 4: User Guide** -The user guide chapter describes how to use the Mass Body Model.

**Chapter 5: Inspections, Tests, and Metrics** -The inspections, tests, and metrics describes the procedures and results that demonstrate the satisfaction of the requirements for the Mass Body Model.

# Chapter 2

# Product Requirements

*Requirement Mass_1: JEOD Requirements*

**Requirement:**
This model shall meet the JEOD project requirements specified in the JEOD v5.1 top-level document [5] .

**Rationale:**
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.1 release.

**Verification:**
Inspection

*Requirement Mass_2: Center of Mass Location*

**Requirement:**
The Mass Body Model shall correctly compute the center of mass location for the composite body and the individual components.

**Rationale:**
The center of mass is used elsewhere in the Mass Body Model and is used by many other spacecraft functions.

**Verification:**
Inspection, Test

*Requirement Mass_3: Total Body Mass*

**Requirement:**
The Mass Body Model shall correctly compute the total mass of the composite body.

**Rationale:**
The spacecraft mass is an important variable used everywhere in spacecraft operations.

**Verification:**
Inspection, Test

*Requirement Mass_4: Composite Inertia Properties*

**Requirement:**
The Mass Body Model shall correctly compute the correct composite inertia properties for the body based on the individual components.

**Rationale:**
The spacecraft inertia is needed as part of the state integration process.

**Verification:**
Inspection, Test

*Requirement Mass_5: Non-Centroidal Inertia Properties*

**Requirement:**
The Mass Body Model shall be capable of interpreting the inertia properties of a body when specified in reference frames that have an origin displaced from the center of mass and/or axes that are rotated with respect to the body axes.

**Rationale:**
Inertia properties nay be specified in a variety of frames.

**Verification:**
Inspection, Test

*Requirement Mass_6: Mass Body Attachment*

**Requirement:**
The Mass Body Model shall be capable of attaching mass bodies to one another.

**Rationale:**
Many spacecraft have to dock or undock, combine or separate pieces of the total spacecraft on the fly.

**Verification:**
Inspection, Test

*Requirement Mass_7: Mass Body Detachment*

**Requirement:**
The Mass Body Model shall be capable of detaching mass bodies from one another.

**Rationale:**

Many spacecraft have to dock or undock, combine or separate pieces of the total spacecraft on the fly.

**Verification:**

Inspection, Test

*Requirement Mass_8: Reattach Child to Different Point*

**Requirement:**

The Mass Body Model shall be capable of taking a body already attached to the parent and moving it to a different attachment point.

**Rationale:**

For objects like robot arms that change their attachment points, this can be an important function.

**Verification:**

Inspection, Test

*Requirement Mass_9: Model Extensibility*

**Requirement:**

The Mass Body Model shall be extensible.

**Rationale:**

Other objects and classes should be able to derive from the Mass Body Model.

**Verification:**

Inspection, Test

# Chapter 3

# Product Specification

## 3.1 Conceptual Design

### 3.1.1 The Basic MassBody

The Mass Body Model must maintain an up-to-date record of the mass and inertia tensor (the mass properties) of massive objects. The inertia tensor is referenced to a coordinate system with origin at the center of mass and set of axes, the *body-axes*, oriented as desired. Because a stand-alone massive block is neither very interesting nor useful, additional characteristics are incorporated to enhance the Mass Body, while maintaining its fundamental purpose.

### 3.1.2 Adding Dynamic State Information

First, it is often desirable, though not essential, to monitor and propagate the dynamic state of a massive object. This feature is provided through inheritance, whereby a MassBody object can also be a DynBody object [3], with the dynamic state features incorporated through the sub-class additions. Once again though, having a dynamically active massive object that just moves subject to central forces is not particularly interesting or useful.

JEOD must be able to realistically propagate the response of a vehicle to external forces and torques. The rotational response to an applied torque, and the translational response to an applied force are both easy, but the rotational response to an applied force requires knowledge of the relative position vector between the center of mass of the object and the point at which the force is applied. Conventionally, locations of objects within a vehicle are defined in the *structural* reference frame. Consequently, a MassBody also defines a point (a MassBasicPoint, see *Points of Interest* (on page 9)) called the *structure_point*, relative to which all positions can be defined or determined. A new coordinate system, with origin at the structure-point and with a set of axes called the *structural-axes*, is used for enumeration of the position vectors. The structural-axes and body-axes are oriented and positioned relative to one another.

**Aside on Structure-Body Orientation** Typically, this relative orientation is straightforward, such as by having the axes aligned, or with a 180-degree yaw, or 90-degree pitch, or similar. See

*Composite and Core Properties* (on the next page) for more details.

**Aside on Axes and Reference Frames** The two coordinate systems now defined are both further developed into full-up reference frames – with states all of their own – in the DynBody [3] class. There, the structure-axes and body-axes become known as the structural and body reference frames (respectively), and retain their orientations and their origins at the structure-point and center of mass (respectively).

### 3.1.3 Mass-Mass Attachments

Even with a dynamic state, the object is still very limited in its versatility. A very significant additional feature, which by itself adds sufficient complexity to justify the need for a separate model just to maintain the mass properties, is the ability to connect two (or more) massive objects to make compound objects. When each of these massive objects can be attaching or detaching to/from the compound object, or changing its mass properties, or moving with respect to the other mass entities in the compound object, then the maintenance of the compound mass properties becomes non-trivial and necessitates a dedicated model for proper treatment.

To handle multiple massive objects connected together, we use a tree structure, refered to as the *mass-tree*. Every MassBody is in one – and only one – tree; some trees are "atomic", having only one member, and some are more complex. Each tree represents the elemental massive objects that are physically connected together to make a compound massive object. There may be more than one tree in any simulation, it is not necessary that all MassBody objects be connected.

At the base (or top, depending on how it is visualized) of the tree is the *root* body, which (unless it is atomic) has one or more *children*. Each child has one (exactly one) *parent*, and possibly one or more children of its own. There are two important restrictions on how these objects are positioned in the tree:

- Because the dynamic state is propagated from parent to child, a dynamic MassBody (i.e. a DynBody) can never be a child of a non-dynamic MassBody (i.e. a base-class MassBody).
- There can be no circular attachments (e.g. a situation in which A is a child of B, which is a child of C, which is a child of A, is forbidden.)

An important consequence is that the mass tree cannot always reflect reality. It is entirely feasible that two objects directly connected on the mass tree have no physical connection; the mass tree is purely a mathematical construct and should never be relied upon as an illustration of physical connectivity.

Recall that each MassBody has a coordinate system associated with its structure-point; when two MassBody objects are joined, those coordinate systems should be linked. Therefore, each structure-point is given an orientation and position specification that identifies its relative state with respect to its parent's structure-point axes. The orientation is represented as a quaternion and as a transformation matrix.

Note that the root MassBody objects have no parent, thus it would correctly be inferred that they have no defined orientation or position. However, realize that if a base-class MassBody (as opposed

to a DynBody) were at the root of the tree, then the entire tree must comprise base-class MassBody objects (as opposed to DynBody objects) since DynBody objects cannot be children of MassBody objects. Thus, the entire tree has no state, thus absolute orientation and position are not defined, and all that is necessary is relative orientation and position of the elemental MassBody objects with respect to one another. When (as is typical) the root is actually a DynBody, the structural reference frame that is developed from the structure-point is given a parent, thus the orientation and position of the structural reference frame, and thus of the structure-point, are defined.

### 3.1.4 Core and Composite Properties

Consider a compound object, comprising multiple MassBody objects. The Mass Body Model must provide the mass properties of this single entity, but it must also retain the mass properties of the individual components, and the mass properties of all sub-trees:

- Mass-loss would be applied to elemental components, and then incorporated into the compound object, so elemental properties must be retained so that they can be manipulated.
- Motion or detachment of elemental components – or of a section of the mass tree – requires that the mass properties of that moving entity be known so that the mass and inertia can be subtracted out (and added back in the case of motion).

Thus, every parent object in the tree must, at all times, keep information on itself, and on that part of the tree of which it is the head (i.e. itself, and all of its children, children of children, etc.). Thus, we keep two sets of mass properties associated with each MassBody:

1. **Core Properties** are those properties associated with the center of mass of the elemental body.
2. **Composite Properties** are those properties associated with the center of mass of that part of the mass tree of which it is the head.

To simplify the determination of whether a MassBody needs a set of composite properties, we just provide this capacity to all MassBody objects; in the case of a body without children, the core properties and composite properties are equivalent.

With each set of properties comes a different center of mass, and consequently a different set of body-axes. A Mass Point is automatically created for each center of mass, thereby setting the relative position of each set of body-axes relative to the structural axes, and the relative orientation of each of the body-axes sets with respect to the structural-axes. Note that both sets of body-axes have the same orientation.

**Aside on locations of variables** The relative position is stored as the respective properties' position (e.g. *composite_properties.position*). The relative orientation of each of the body-axes sets with respect to the structural-axes is specified as a transformation matrix or quaternion-set (e.g. *core_properties.T_parent_this*). The rationale for this choice is expounded in *Detailed Design* (on page 13).

**Aside for DynBody extension** In the DynBody extension, there are three distinct reference frames - the core-body frame, the composite-body frame, and the structural frame. The structural frame has its origin at the structure-point, while the body frames have their origin at the respective centers of mass. Therefore, the position of a body frame with respect to the structural frame in a DynBody is equal to the respective properties' position value (e.g. *core_properties.position*). By analogy, the orientation of the body frames with respect to the structural frame is the same as that of the body-axes with respect to the structural-axes. For example, the transformation from the structural frame to the composite-body frame is *composite_properties.T_parent_this*.

### 3.1.5 Points of Interest

There are numerous instances where some point on a body is of particular interest, and its position needs to be well defined - the location of an antenna, or a sensor, the point at which another object is attached, etc. In many applications, it is also desirable to specify an axes-set at the point so that other positions can be defined with respect to the point. Consequently, we include an orientation in the defining data of each point.

We have two types of points - the basic *MassBasicPoint*, and the more commonly used *MassPoint* (which is, essentially, a MassBasicPoint with a name; the name provides the user with a way to identify the mass point).

Every MassBody starts with three MassBasicPoints, and we have already considered them:

1. The structure-point; axes are the structure-axes.
2. The composite-properties point; axes are the body-axes with origin at the composite center of mass.
3. The core-properties point; axes are the body-axes with origin at the core center of mass.

Any MassBody can then be given any number of additional, user-specified, MassPoint instances. Every MassPoint gets added to the mass-tree as a child of the structure-point of the same MassBody (obviously, excepting the structure-point itself, which is a child of the structure-point of the parent MassBody, and thereby a sibling to other MassPoint instances of the parent MassBody).

### 3.1.6 Summary

A MassBody comprises three components:

1. A MassBasicPoint, called the *structure_point* that provides:
   (a) A position and orientation relative to the parent body (if it exists)
   (b) A reference point and axes (*structural-axes*), from which further measurements may be made (such as to define the location of the center of mass).
2. A MassProperties object, *core_properties*, that provides:
   (a) Mass

(b) Inertia tensor, referenced to a specified axes-set, the *body-axes*.

(c) Position of the center of mass with respect to its structure-point, expressed in the structural-axes.

(d) The orientation of the body-axes with respect to the structural-axes.

for the stand-alone MassBody.

3. An additional MassProperties object, *composite_properties*, that provides the same set of properties for the compound object comprising the MassBody and everything attached **to** it in the mass tree (not including objects that it is attached to, the hierarchy in the mass tree is very important).

## 3.2 Mathematical Formulations

### 3.2.1 Mass

The core and composite masses are found at **_properties.mass*.

#### 3.2.1.a Core Property

This value has to be set externally (e.g. by the user).

#### 3.2.1.b Composite Property

This value is computed trivially,

$$M_{composite} = M_{core} + \sum_{children} M_{composite,i} \tag{3.1}$$

where $M_{composite,i}$ is the composite mass of each of the children of this Mass Body. Clearly, generation of the composite mass is an iterative procedure, requiring first the calculation of the corresponding value for each of the children, which require the same for theirs, etc.

### 3.2.2 Center of Mass

The position of the core and composite centers of mass are found at **_properties.position*; the value is expressed in the structural axes.

#### 3.2.2.a Core property

This value has to be set externally (e.g. by the user).

#### 3.2.2.b Composite Property

The position of the composite center of mass is derived from the respective positions of all of the components of the sub-tree originating with this body, using classical mechanics:

$$M_{composite} \cdot \vec{x}_{composite} = M_{core} \cdot \vec{x}_{core} + \sum_{children} M_{composite,i} \cdot \vec{y}_{composite,i} \tag{3.2}$$

where $\vec{y}_{composite,i}$ is the position of the respective composite center of mass for each of the children, expressed with respect to, and in, the structural axes of this mass body.

$$\vec{y}_{composite,i} = \vec{x}_i + \boldsymbol{T}_{i \to this} \left( \vec{x}_{composite,i} \right) \tag{3.3}$$

with $\vec{x}_i$ the position of the child's structure point expressed in, and with respect to the structural axes of this body, and $\boldsymbol{T}_{i \to this}$ is the transformation matrix from the structural axes of the child body to the structural axes of this body.

Again, this is clearly an interative process.

### 3.2.3 Inertia Tensor

The inertia tensor for the core and composite bodies are found at **_properties.inertia*; the value is referenced to the respective body axes. The diagonal elements are positive moments of inertia, while the off-diagonal elements are negative products of inertia.

#### 3.2.3.a Core Property

This value has to be set externally (e.g. by the user).

#### 3.2.3.b Composite Property

Computation of the composite body inertia tensor is a multi-step process:

1. Compute the inertia tensor for the core-body, referenced to the composite-body body-axes, rather than the core-body body-axes. Since the two sets of axes are aligned, we can use the parallel axis theorem:

$$\boldsymbol{I}_{core:comp} = \boldsymbol{I}_{core:core} + M_{core} \begin{bmatrix} y^2 + z^2 & -xy & -xz \\ -xy & x^2 + z^2 & -yz \\ -xz & -yz & x^2 + y^2 \end{bmatrix} \tag{3.4}$$

   where $x$, $y$, and $z$ represent the position of the core center of mass relative to the composite center of mass, expressed in the composite body-axes.

2. For each child, transform a copy of the child's composite-body inertia tensor so that it is referenced to this (i.e. the parent) body's composite-body body-axes, rather than its own. This is a multi-step process:

(a) Compute the position of the origin of the child's composite-body-axes relative to that of this body.

(b) Compute the orientation of the child's composite-body-axes relative to that of this body.

(c) Use the orientation data to apply a rotational transformation to the child's composite-body inertia tensor such that it references the parent's composite-body-axes. This step is necessary because the composite-body-axes for the child body are, in general, not aligned with those for the parent body. Consider

$$\tau = \boldsymbol{I}\alpha$$

Hence, $\tau_{parent}$ can be expressed as

$$\tau_{parent} = \boldsymbol{I}_{parent}\alpha_{parent}$$

and also expressed as a transformation of the same expression in the child frame:

$$\tau_{parent} = T_{child\to parent}\left(\boldsymbol{I}_{child}\left(T_{parent\to child}\left(\alpha_{parent}\right)\right)\right)$$

Consequently,

$$\boldsymbol{I}_{parent} = T_{child\to parent}\boldsymbol{I}_{child}T_{parent\to child} \tag{3.5}$$

This term represents the inertia tensor in a set of axes aligned with the parent composite-body-axes, with an origin that still matches that of the child composite-body-axes.

(d) Evaluate and add the parallel-axis theorem addition term in equation 3.4, such that:

$$\boldsymbol{I}_{child:parent} = T_{child\to parent}\boldsymbol{I}_{child:child}T_{parent\to child} + M_{core}\begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} \tag{3.6}$$

where $x$, $y$, and $z$ represent the position of the child composite-body center of mass relative to the parent composite-body center of mass, expressed in the parent composite-body-axes. $M_{child}$ is the mass of the child composite-body.

3. Add to the adjusted parent body inertia tensor (equation 3.4) the resulting inertia tensors for each of the children (equation 3.6)to give the composite body inertia tensor for the parent.

$$\boldsymbol{I}_{comp:comp} = \boldsymbol{I}_{core:comp} + \sum_{children} \boldsymbol{I}_{child:parent} \tag{3.7}$$

### 3.2.4 Inverse Inertia

The inverse-inertia tensor for a body is found at *inverse_inertia*; the value is referenced to the respective body axes.The inverse inertia is only needed where torques are going to be applied to a MassBody, and then only if the MassBody has a dynamic state to respond to those torques (i.e. if it is a DynBody). Furthermore, since torques (and forces) are only applied to a mass tree in its entirety, the only inverse inertia that is needed is that of the composite properties of the root body. Consequently, the inverse inertia is only computed if the MassBody is a DynBody, if it is at the root of its tree (including single-entity trees), and then only the composite inertia is inverted.

#### 3.2.5 Transformation from Structure to Body

The transformation from the strucutral-axes to the body-axes is found at **_properties.T_parent_this* and at **_properties.Q_parent_this* (for a transformation matrix and quaternion set, respectively). This value is set at initialization for the core-properties;

##### 3.2.5.a Core Property

This value has to be set externally (e.g. by the user) at initialization; it is fixed for the duration of the simulation.

##### 3.2.5.b Composite Property

Since both sets of body axes are aligned, the value set for the core-properties gets copied into the composite-properties at initialization.

#### 3.2.6 Attaching Bodies

The process by which bodies are attached is outlined in the Detailed Design section (3.3.3)

### 3.3 Detailed Design

#### 3.3.1 API

Follow this link for the *Mass Body Model API* [2].

#### 3.3.2 Class Overview

The MassBody is the basic class for a Mass Body. It comtains (this is not an exhaustive list):

- *core_properties*, an instance of MassProperties, that describes the properties of the MassBody as a single entity.
  - A MassProperties class is a MassBasicPoint which also provides data elements for mass and the inertia tensor.
    - A MassBasicPoint is a MassPointState that provides the linkages within the mass tree (to its children, parent, etc.). A mass tree comprises the collection of MassBasicPoint instances that are physically connected at some point.
      - A MassPointState provides data elements for the position of the point, and the orientation of the axes associated with the point, with respect to some parent.
    - A related item, a MassPoint, is simply a MassBasicPoint with a name, allowing for it to be found easily.

- *composite_properties*, another instance of MassProperties, that describes the properties of the subtree of which the body is the head.
- *structure_point*, an instance of MassBasicPoint (see above).

Note that since a MassProperties is a MassBasicPoint, it must have a parent for evaluation of its position and orientation. The parent of both instances of Mass Properties (*core_properties* and *composite_properties*) is the MassBasicPoint *structure_point*; the position and orientation are stated relative to the structural axes.

### 3.3.3 Attaching Bodies

Attaching two bodies together is typically performed with a Body Action. A rough outline of the algorithm is presented here, instructions for implementation are contained in the Body Action document ([3]).

The following rules provide the restrictions on which bodies can be commanded attached to which bodies.

- A DynBody (extension of MassBody) can never be a child of a basic MassBody.
- There can be no circular attachments.

IMPORTANT NOTE: Only root bodies may actually attach to another body. While it may be legal to command the attachment of body A to body B (subject to the rules above), the attachment in the mass tree will be represented as the root of the tree containing body A attaching to body B.

#### 3.3.3.a Attaching using Points

The easiest implementation of attaching two bodies is to define an attach point (a MassPoint) on each body, and make those points coincident, with their z-axes aligned, and their x- and y-axes both anti-aligned. Because each MassPoint will have position and orientation defined with respect to its respective body's structural axes, it is a striaghtforward undertaking to obtain the orientation and position of the child body's structural axes with respect to the parent body's structural axes. Then, the general attachment method can be implemented, and the parent will have as children its MassPoint used for attachment and the child structure-point; the child will have as a child its MassPoint used for attachment.

#### 3.3.3.b Attaching using Offset and Orientation

This more general form of the attach process requires knowledge of the orientation and relative position of the child body's structural-axes with respect to those of the parent body, which may be provided directly or from the previous method.

If the child body is not the root of its own tree, its root is first found, then the relative position and orientation of that root body's structure-axes relative to the structure-axes of the parent body are determined.

With the root state known, the attachment of the root of the child's tree (the child-root) to the parent can proceed in three steps:

1. **Validate:** Is this a valid attachment which for the Mass Body Model, means it follows a tree structure and avoids invalid circular attachments.
2. **Establish Links:** The child-root MassBody is required to establish the links to the parent MassBody that define their relationship in the mass tree.
3. **Update Properties:** The parent MassBody updates its composite properties to reflect the addition of the child-root MassBody. The composite properties of the child-root remain unchanged.

A few examples will help illustrate the concept of attachments and we will start by supposing that we have three MassBodys called A, B, and C respectively.

In the first example we decide to attach MassBody C to MassBody B thereby making C a child of B. Given that no other attachments exist in this example this is a valid attachment so we will pass step one(Validate). In step two(Establish Links) the child will update the links between the two MassBodys including MassPointLinks and MassBodyLinks. In step three(Update Properties) the parent will update its composite properties (mass, inertia, center of mass) to represent the combined state.

Building on the new mass tree created with the attachment of MassBody C to MassBody B, we will now attach MassBody B to MassBody A. MassBody B will then become a child of MassBody A, and MassBody C will remain a child of MassBody B. In this attachment process, the validity is checked – since A is not already attached to B or C this is approved. MassBody B now performs the Establish Links step configuring all MassPointLinks and MassBodyLinks with MassBody A. Then MassBody A uses the composite properties of MassBody B to perform the Update Properties step. It should be noted that the composite properties of MassBody B already include the properties of MassBody C so, following the update, the composite properties of A represent the entire tree. Even though the properties of A do contain information for the entire tree, MassBody A has no knowledge that MassBody C exists. MassBody A only knows about MassBody B.

Consider another situation: return to the situation where only MassBody B and MassBody C are attached and C is a child of B. Suppose we then want to attach MassBody C to MassBody A. The Mass Body Model recognizes that MassBody C is not a root body, so instead attaches MassBody B to MassBody A. The offset and rotation of B with respect to A are set such that the offset and rotation of C with respect to A are as requested. Again, the composite mass properties of A will include those of C through the composite properties of B.

In both situations, the mass tree looks identical.

### 3.3.4 Detach Overview

In the detach scenario, the links between the body to be detached and its parent are severed, and the composite properties of the parent are updated to reflect that one of its children has been removed. The composite properties of the child body are unchanged.

## 3.4 Inventory

All Mass Body Model files are located in `${JEOD_HOME}/models/dynamics/mass`. Relative to this directory,

- Model header and source files are located in model `include` and `src` subdirectories. See table **??** for a list of these configuration-managed files.
- Model documentation files are located in the model `docs` subdirectory. See table **??** for a list of the configuration-managed files in this directory.

# Chapter 4

# User Guide

The User Guide is divided into three instruction sets:

1. **Instructions for Simulation Users**. This instruction-set contains a description of how to modify Mass Body Model variables after the simulation has compiled, including an in-depth discussion of the input file; an overview of how to interpret (but not edit) the S_define file; and a sample of some of the typical variables that may be logged.
2. **Instructions for Simulation Developers**. This instruction-set builds on the previous set, and adds information on the necessary configuration of the Mass Body Model within an S_define file, and the creation of standard run directories.
3. **Instructions for Model Developers**. This instruction-set builds on the previous set, and adds information on the potential for extending the model to perform tasks that are beyond its current abilities.

## 4.1 Instructions for Simulation Users

### 4.1.1 Defining the Mass Body

The following values for each MassBody are typically specified in the input file:

1. Mass
2. Position of the center of mass with respect to its structural-axes (often also known as the structural frame or structure frame)
3. The orientation of the body-axes with respect to the structural-axes
4. Inertia tensor.

This specification is usually performed through the use of the MassBodyInit Body Action (this is strongly recommended). The Body Action document [4] has its own User Guide specifically for the MassBodyInit action, duplicating some of the examples presented here.

In the following example we have a simulation object called *sim_object_A*, which contains a MassBody called *body*, and an instance of the Body Action MassBodyInit called *mass_init*.

Example code is given for Trick10 users; non-Trick users should follow a similar pattern using their appropriate syntax. We start with the trivial declarations that tell *mass_init* that it is initializing body, that the body has a mass of 1.0 $kg$, and specifies the position of the center of mass of the body with respect to its structure-point.

**Trick 10**

```
sim_object_A.mass_init.set_subject_body(sim_object_A.body);
sim_object_A.mass_init.properties.position = trick.attach_units("M",[1.0, 0.0,
0.0])
sim_object_A.mass_init.properties.mass     = trick.attach_units("kg",1.0)
```

The orientation of the body-axes with respect to the structure-axes must also be specified. Note that this orientation will be applied equally to the composite_properties body-axes and to the core_properties body-axes.

There are several methods by which the orientation can be specified, as detailed in the Orientation Model document [7]. There are also two ways to interpret the specified rotation – either as body to structure, or as structure to body (if none is specified, the default value is structure-to-body).

The following values are all equivalent, and all mean structure-to-body:

- 0
- StructToBody
- StructToCase
- StructToPoint
- StructToChild

The following values are all equivalent, and all mean body-to-structure:

- 1
- BodyToStruct
- CaseToStruct
- PointToStruct
- ChildToStruct

**Aside on Naming** While the names are functionally equivalent, they have specific meanings, hence the number of equivalent options:

1. *Struct* refers to the structure-point.
2. *Body* typically refers to the body-axes associated with the MassBasicPoints *core_properties* and *composite_properties*.
3. *Case* is typically used to represent the case frame of a sensor, or similar, located at a Mass-Point.
4. *Point* is a generic term, used for any mass point.
5. *Child* is also a generic reference that recognizes the mass-tree structure, wherein all mass points for a MassBody are children of the structure-point (obviously, excepting the structure-point itself).

In the example below, we specify structure-to-body, and use eigen-vector rotation of 90 degrees about the z-axis (this would be the z-axis in the structure-axes)

**Trick 07**

```
sim_object_A.mass_init.properties.pt_frame_spec =
MassPointInit::StructToBody;
sim_object_A.mass_init.properties.pt_orientation.data_source =
                                        Orientation::InputEigenRotation;
sim_object_A.mass_init.properties.pt_orientation.eigen_angle {d} = 90.0;
sim_object_A.mass_init.properties.pt_orientation.eigen_axis[0]   = 0.0, 0.0,
1.0;
```

This example, in Trick10 gives a transformation matrix from body-axes to structure-axes, with a 180-degree rotation about the y-axis.

**Trick 10**

```
sim_object_A.mass_init.properties.pt_frame_spec =
                                          trick.MassPointInit.BodyToStruct;
sim_object_A.mass_init.properties.pt_orientation.data_source =
                                                  trick.Orientation.InputMatrix;
sim_object_A.mass_init.properties.pt_orientation.trans[0] = [ -1.0,  0.0,  0.0]
sim_object_A.mass_init.properties.pt_orientation.trans[1] = [  0.0,  1.0,  0.0]
sim_object_A.mass_init.properties.pt_orientation.trans[2] = [  0.0,  0.0, -1.0]
```

Note that the orientation of the body with respect to any other object has not been specified. That is only relevant under one of the following situations:

1. The body is attached to another body; in this case the relative orientation between the two bodies will be specified as part of the attachment definition.

2. The absolute orientation of the body is useful in some sense; in this case, the body needs a state, and so should be a DynBody. The initialization of a DynBody does include orientation of its reference frames (which a MassBaody does not have) with respect to their parent (which is the inertial reference frame). By orienting the DynBody reference frames, the MassBody axes (which are aligned with the respective DynBody reference frames) also get aligned.

The final initialization item is the inertia tensor, and there are several methods for specifying this value. The variable *mass_init.properties.inertia_spec* can take one of the following values:

- NoSpec The inertia tensor is not specified. Any attempt to specify it in the *mass_init* object will be ignored.
- Body (default) The inertia tensor is specified using the body-axes.
- StructCG The inertia tensor is specified using axes oriented with the structure-axes, but based at the center-of-mass.
- Struct The inertia tensor is specified using the structure-axes.
- SpecCG The inertia tensor is specified using axes oriented with some specified orientation, based at the center-of-mass.
- Spec The inertia tensor is specified using axes oriented with some specified orientation, based at a specified origin.

For options *Spec* and *SpecCG* ONLY, the value *mass_init.properties.inertia_orientation* must be specified. This is an instance of the Orientation class, see the Orientation document for full details [7]. The value specified is the process by which the orientation of the axes in which the inertia tensor is specified may be transformed into the orientation of the body-axes (e.g. $T_{spec \to body}$. It is analogous to the specification of the body-axes orientation relative to the structural-axes orientation.

For option *Spec* ONLY, the value *mass_init.inertia_offset* must also be specified. This is a simple 3-vector, expressed in the axes associated with the *mass_init.properties.inertia_orientation* specification.

**Trick07**

```
sim_object_A.mass_init.properties.inertia[0][0] {kg*M2} = 1.0, 0.0, 0.0;
sim_object_A.mass_init.properties.inertia[1][0] {kg*M2} = 0.0, 2.0, 0.0;
sim_object_A.mass_init.properties.inertia[2][0] {kg*M2} = 0.0, 0.0, 3.0;
```

In this example, the axes are switched to specify the inertia on the structure-axes.

**Trick10**

```
sim_object_A.mass_init.properties.inertia_spec =
trick.MassPropertiesInit.Struct
sim_object_A.mass_init.properties.inertia[0]  =
                              trick.attach_units( "kg*m2",[ 1.0,  0.0,   0.0])
sim_object_A.mass_init.properties.inertia[1]  =
                              trick.attach_units( "kg*m2",[ 0.0,  2.0,   0.0])
sim_object_A.mass_init.properties.inertia[2]  =
                              trick.attach_units( "kg*m2",[ 0.0,  0.0,   3.0])
```

### 4.1.2 Adding Mass Points

Mass points are useful for providing additional sets of axes from which positions and orientations can be determined. Perhaps the most useful application of MassPoint objects is in attaching mass bodies to one another. Mass points are typically defined using the same body action item, and at the same time that the rest of the MassBody is being defined. The following items define a MassPoint:

1. Name
2. Position with respect to the structural-axes
3. The orientation of the point-axes with respect to the structural-axes.

We have already covered the initialization of both of these elements for the MassBody, and the same system is followed for the MassPoint. The position is straightforward again, and the orientation can be specified in any one of the many ways available from the Orientation model [7], and interpreted as either structure-to-body or body-to-structure, just as outlined above.

The MassPoint instances do need allocation. In the following examples, notice that the *pt_frame_spec* value is not specified, so takes the default value *StructToPoint*. Notice also that I have chosen to illustrate two different ways of specifying the orientation in the two examples; this does not mean that one is prefered in Trick07 and one in Trick10, they are jsut illustrative examples.

**Trick07**

```
sim_object_A.mass_init.num_points = 1;
sim_object_A.mass_init.points = alloc( 1 );

sim_object_A.mass_init.points[0].set_name( "interesting_name" );
sim_object_A.mass_init.points[0].pt_frame_spec =
MassPointInit::StructToPoint;
sim_object_A.mass_init.points[0].position[0] {M}  = 2.0, 0.0, 0.0;
sim_object_A.mass_init.points[0].pt_orientation.data_source =
```

```
                                                    Orientation::InputEigenRotation;
sim_object_A.mass_init.points[0].pt_orientation.eigen_angle {d} = 180.0;
sim_object_A.mass_init.points[0].pt_orientation.eigen_axis[0]   = 0.0, 0.0,
1.0;
```

**Trick10**

```
sim_object_A.mass_init.num_points = 1
sim_object_A.mass_init.points =
                           trick.sim_services.alloc_type( 1, "jeod::MassPointInit" )
sim_object_A.mass_init.points[0].set_name( "interesting_name" )
sim_object_A.mass_init.points[0].position  =
                                      trick.attach_units( "m",[ 2.0, 0.0, 0.0])
sim_object_A.mass_init.points[0].pt_orientation.data_source =
                                                    trick.Orientation.InputMatrix
sim_object_A.mass_init.points[0].pt_orientation.trans[0]  = [ -1.0,  0.0, 0.0]
sim_object_A.mass_init.points[0].pt_orientation.trans[1]  = [  0.0, -1.0, 0.0]
sim_object_A.mass_init.points[0].pt_orientation.trans[2]  = [  0.0,  0.0, 1.0]
```

### 4.1.3 Log Files

The output from the Mass Body Model is generally simple information (mass, center of mass, inertia matrix). It is possible to output this information for all of the bodies in a simulation.

Logging of the composite-properties is more common than the core-properties, because the core-properties are set by the user (so are presumably already known), whereas the composite-properties are generated by the Mass Body Model.

**Trick07**

```
"sim_object_A.body.composite_properties.mass";
"sim_object_A.body.composite_properties.position[0]";
"sim_object_A.body.composite_properties.inertia[0][0-2]";
"sim_object_A.body.composite_properties.inertia[1][0-2]";
"sim_object_A.body.composite_properties.inertia[2][0-2]";
```

**Trick10**

```
dr_group.add_variable("sim_object_A.body.composite_properties.mass")
for ii in range(0,3) :
  dr_group.add_variable("sim_object_A.body.composite_properties.position["+st
  r(ii)+"]")
```

etc.

## 4.2 Instructions for Simulation Developers

### 4.2.1 S_define File

Below is an example of how to include the Mass Body Model in an S_define file

#### 4.2.1.a Instantiation

Instantiation of a MassBody is straightforward:

**Trick07**

```
dynamics/mass:    MassBody       body;
```

**Trick10**

```
MassBody       body;
```

Typically, the body requires some state information, in which case the DynBody is used:

**Trick07**

```
dynamics/dyn_body:    DynBody       body;
```

**Trick10**

```
DynBody       body;
```

#### 4.2.1.b Initialization

To initialize the mass properties of a mass-body it is STRONGLY recommended that the Body Action MassBodyInit be used. Examples of usage are given in the previous section (*Instruction for Simulation Users* (on page 17)) and in the Body Action documentation ([4]).

This body action will also initialize points associated with the MassBody if they are allocated. Allocation can be performed, for example, in the input file right along with the declaration of the values. In the S_define, the body-action itself needs to be instantiated.

**Trick07**

```
dynamics/body_action: MassBodyInit mass_init;
```

**Trick10**

```
MassBodyInit mass_init;
```

#### 4.2.1.c Manipulation

Performing actions on the body (such as attaching to another body, moving, detaching, etc.) is also best handled by the MassBody Attach / Detach sub-model of the Body Action model (see documentation [4]).

Important points regarding attach and detach mechanisms:

- When attaching using mass points, the axes of the mass points will be aligned on the z-axis, and anti-aligned on the x- and y- axes. An assistive visual guide is to imagine two vehicles approaching with the x-axes of the two mass-points pointing at one another and z-axes aligned.
- When attaching using the offset and orientation method, the values specified are:
  - The location of the child structure point with respect to the parent structure point, expressed in the parent structural-axes.
  - The transformation matrix from the parent structural-axes to the child structural-axes.
- When commanding an attach of object A to object B, the mass-tree will actually attach the root of the mass-tree containing object A to object B. Object A will only be attached to B in the mass-tree if A was originally the root of its own tree.
- When using the simple detach, remember that:
  - The mass-tree will be severed immediately above the detached object.
  - The mass-tree does not always represent reality.
  - Consequence is that there are situations where if object A was commanded attached to object B, then object A commanded detached, the mass-tree would not return to its original form. The user must be aware of the effect on the mass-tree of all attach and detach commands.

### 4.2.2 Utilizing the MassBody

The two strengths of the Mass Body Model are in

- Maintaining the properties of compound bodies through attach and detach processes.
- Providing anchor-points between which the relative state (position and orientation only) can be measured.

The maintenance of compound properties is performed automatically, as long as the *update_flag* is set. To set this flag, call *set_update_flag()* on the affected body farthest from the root of the

tree. This method will propagate up the tree, setting the flags for all bodies between the body of interest and the root body (inclusive). Note that this method is automatically called in the following situations:

- At initialization of this MassBody object.
- At destruction of this MassBody object.
- A mass is attached to this body.
- This mass is detached from its parent (call made before links to parent have been canceled)
- A mass is moved (i.e. reattached to this mass).

The relative state (position and orientation only) of "this" mass point can be found with respect to, and expressed in the axes-set associated with, any other mass point in the same tree with a call to *compute_relative_state*. This method takes two arguments:

1. The other mass point (MassBasicPoint),
2. The MassPointState to be populated with the relative state data.

This method is distinct from the one provided in the Reference Frames model, which produces a full relative state (including velocity and angular rate).

## 4.3 Instructions for Model Developers

The Mass Body Model is intended to provide a baseline configuration for objects that have mass. It is quite extensible; an example of that extension is in the Dynamic Body model, discussed briefly in this document and extensively in the Dynamic Body model document ( [3]).

Authors of an extension will have to pay close attention to the limitations of the Mass Body Model, and consider whether the methods provided are suitable for the task in the new environment. For example, when attaching two bodies, the Mass Body Model provides all of the functionality associated with positioning the mass bodies correctly in the mass tree, with the correct orientation and relative positions. It does not provide any capabilities for modeling the changes to the dynamic state, because it knows nothing of any dynamic states. Consequently, the Dynamic Body model extension of the Mass Body Model must provide its own unique methods for handling state; these are called in addition to the generic Mass Body Model methods when two DynBody objects attach.

# Chapter 5

# Inspections, Tests, and Metrics

## 5.1 Inspections

This section describes the inspections of the Mass Body Model.

### 5.1.1 Top-level requirement

*Inspection Mass_1: Top-level inspection*

This document structure, the code, and associated files have been inspected, and together satisfy requirement Mass_1.

### 5.1.2 Extensibility requirement

*Inspection Mass_2: Extensibility inspection*

A DynBody [3] is a successful extension of the MassBody class and satisfies requirement Mass_9.

## 5.2 Tests

This section describes various tests conducted to verify and validate that the Mass Body Model satisfies the requirements levied against it. The tests described in this section are archived in the JEOD directory `models/dynamics/body_action/verif`.

The tests presented in this section are tests of discrete events, not of simulated or propagated values. The output from each test is found in the appropriate run directory's *mass.out* file. This document presents the analytic solution for each of the testing scenarios. The analytic solution can be compared against the data output. Since there were no discrepancies to analyze, those output data have not been duplicated in this document, but are available for the reader to independently verify the claims made herein.

### 5.2.1 Attachment Tests

*Test Mass_1: Two-Object Combination*

**Purpose:**

The purpose of this test is to examine the ability to calculate the composite properties of two adjoined objects. In this test, the MassBody objects are two identical right rectangular prisms.
SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_01 RUN_101

**Requirements:**

Successful completion of this test partially satisfies requirements Mass_2, Mass_3, Mass_4, and Mass_6.

**Procedure:**

In these test, the two objects were conceptualized with uniform density and dimensions (1.0, 1.0, 2.0) $m$. The strucural-axes and body-axes were defined to be equivalent, the mass set to $1.0kg$, and the inertia tensor specified as:

$$\begin{bmatrix} \frac{5}{12} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{6} \end{bmatrix} kg\ m^2$$

(thereby defining the body-axes along the principal axes, and placing the 2.0 $m$ length on the z-axis).

Although the two objects are identical, it is necessary to denote one the *parent* and one the *child*. The *child* will be attached to the *parent*.

In the first test, the attachment of child to parent was performed by the offset-orientation method. The child was attached with an offset of 1.5 $m$ and a relative orientation represented as a rotation of 90 degrees about the parent x-axis.

In the second test, the attachment of child to parent was performed by the mass-point method. A mass point was defined on the child body with position $(0.0,\ 0.0,\ 1.0)m$ and an orientation (with respect to the child structural axes) expressed with the transformation matrix:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The parent body defines a mass point with position $(0.0,\ 0.5,\ 0.0)m$ and an orientation expressed with the transformation matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

In both tests, the resulting attachment should be as illustrated in Figure 5.1, with the parent body beneath the child body. The parent structural axes have origin at the center of the parent body, and are oriented as follows:

- x-axis into the page
- y-axis bottom-to-top
- z-axis left-to-right

.

The computed positions and composite inertia tensors were then compared against separate analytical computations.

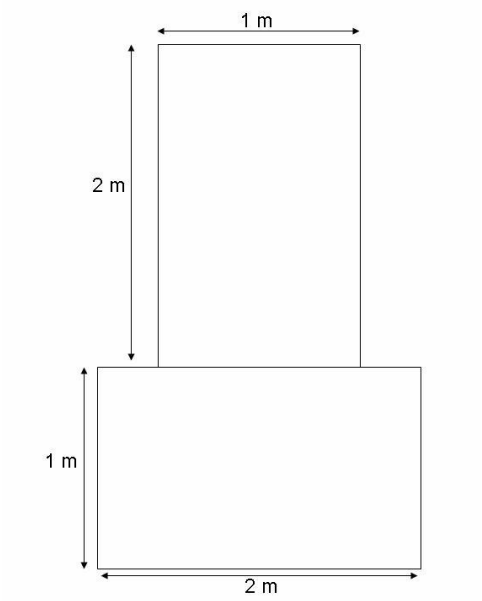


Figure 5.1: Diagram of composite Mass-body for Test 1

**Results:**

The inertia for each individual body in Figure 5.1 is calculated using standard equations for a uniform density body (see, for example, Bedford [1]) and verified against the input values.

For the composite-body, in both tests, the position of the center of mass agrees with the analytical result of $(0.0,\ 0.75,\ 0.0)m$. The composite-body inertia tensors also agree with the analytical result of $\begin{bmatrix} \frac{47}{24} & 0 & 0 \\ 0 & \frac{7}{12} & 0 \\ 0 & 0 & \frac{41}{24} \end{bmatrix} kg\ m^2$

*Test Mass_2: Complex Tree, Three-object Combinations*

**Purpose:**

We have already demonstrated that two objects can be attached (in Test Mass_1). The purpose of this test is to investigate more complex tree structure by attaching three objects together. Two different configurations, one in which a parent has two children attached, and one in which a parent has a child, which has an additional child attached to it, are used. These configurations verify the ability to handle multiple children, and multi-generational trees.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass

Run directory: RUN_02 RUN_03 RUN_04 RUN_102 RUN_103 RUN_104

**Requirements:**

Successful completion of this test partially satisfies requirements Mass_2, Mass_3, Mass_4, and Mass_6.

**Procedure:**

Three identical objects, equivalent to those used in test Mass_1 were stacked along each of the three axes. As with test Mass_1, both the position-orientation and mass-point specifications were tested. The setup of the tests is as follows:

- RUN_02 Stacked on y-axis using the position-orientation specification. Parent has 2 children with identical orientations, located at $(0,\ \pm 1.0,\ 0.0)m$
- RUN_03 Stacked on x-axis, using the position-orientation specification. Parent has 1 child (Child1) with identical orientation to Parent, and relative position at $(1.0,\ 0.0,\ 0.0)\ m$. Child1 has 1 child (Child2), with identical orientation to Child1 (so also identical orientation to Parent), and relative position at $(1.0,\ 0.0,\ 0.0)\ m$ (so Child2 is at $(2.0,\ 0.0,\ 0.0)\ m$ relative to Parent).
- RUN_04 Stacked on z-axis, using the position-orientation specification. Parent has 1 child (Child1) with identical orientation to Parent and relative position at $(0.0,\ 0.0,\ -2.0)\ m$. Child1 has 1 child, also with identical orientation and relative position at $(0.0,\ 0.0, -2.0)\ m$.
- RUN_102 Stacked on y-axis, using the mass-point specification. Parent has 2 children and 2 mass-points located at the center of its $\pm y-$faces. The two children both have mass points on their respective $y-$faces. The orientations of all four mass points have a $\pm 90^o$ yaw (such that each matching pair has one point with a positive yaw and one with a negative yaw).
- RUN_103 Stacked on x-axis, using the mass-point specification. The parent has one mass point, at the center of its $+x-$face and oriented with the body that is used to attach to the first child. The first child has two mass points, one at the center of its $-x-$face with a $180^o$ yaw that is used to attach it to the parent, and one at the center of its $+x-$face and oriented with the body that is used to attach it to the second child. The second child has one mass point at its $-x-$face with a $180^o$ yaw that is used to attach it to the first child.
- RUN_104 Stacked on z-axis, using the mass-point specification. The parent has one mass point, at the center of its $-z-$face and oriented with a transformation matrix

$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ that is used to attach to the first child. The first child has two mass points, one at the center of its $+z-$face with a transformation matrix $\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ that is used to attach it to the parent, and one at the center of its $-z-$face with a transformation matrix $\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ that is used to attach it to the second child. The second child has one mass point at its $+z-$face with a with a transformation matrix $\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ that is used to attach it to the first child.

In all 6 runs, the resulting orientation of all three bodies is identical. In runs 02 and 102, the parent provides the central body with the children attached symmetrically, one on each side. In the other 4 runs, the parent is located at one end of the stack.

Figure 5.2 illustrates the configuration for runs 04 and 104, with the parent to the right, the first child in the center, and the second child at the left. The configurations for the other runs are similarly simple, except on other axes.

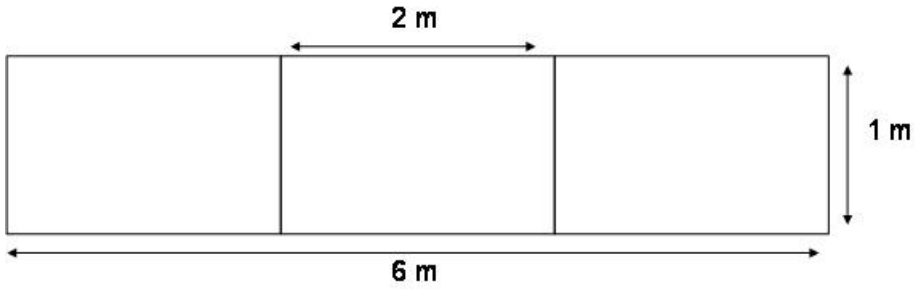


Figure 5.2: Diagram of stacked body for test

**Results:**

For all six runs, the mass of the composite body agrees with the predicted 3 $kg$.

The position of the center of mass should be as follows:

- Runs 02 and 102: $(0.0,\ 0.0,\ 0.0)$ (parent is at the center)
- Runs 03 and 103: $(1.0,\ 0.0,\ 0.0)$ (parent is at one end)
- Runs 04 and 104: $(0.0,\ 0.0, -2.0)$ (parent is at one end)

The moments of inertia should be:

- Runs 02 and 102: $3 \begin{bmatrix} \frac{5}{12} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{6} \end{bmatrix} kg\ m^2 + 2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} kg\ m^2 = \begin{bmatrix} \frac{13}{4} & 0 & 0 \\ 0 & \frac{5}{4} & 0 \\ 0 & 0 & \frac{5}{2} \end{bmatrix} kg\ m^2$

- Runs 03 and 103: $\begin{bmatrix} \frac{5}{4} & 0 & 0 \\ 0 & \frac{13}{4} & 0 \\ 0 & 0 & \frac{5}{2} \end{bmatrix} kg\ m^2$
- Runs 04 and 104: $\begin{bmatrix} \frac{37}{4} & 0 & 0 \\ 0 & \frac{37}{4} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} kg\ m^2$

All results agreed with predictions.

*Test Mass_3: Compound, non-Root Attachment*

**Purpose:**

The purpose of this test is to confirm that the Mass Body Model correctly manages the mass-properties when a composite body is attached to a parent. The test further verifies the legitimacy of commanding a non-root component of the mass tree to attach to the parent.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_08 RUN_108

**Requirements:**

Successful completion of this test completes the satisfaction of requirements Mass_2, Mass_3, Mass_4, and Mass_6.

**Procedure:**

This test utilizes four flat plates similar to those used in previous tests. The four plates are attached to one another in the following sequence

1. child2 is attached to child1
2. child3 is attached to child2
3. child3 is attached to parent.

Following the system established in previous tests, RUN_08 has the attachment performed using offset-orientation specification, while RUN_108 uses the mass-point specification to perform the same task.

Figure 5.3 illustrates the relative orientation and the physical connections between the four plates.

Notice that child1 and child2 are not actually in contact. This is entirely legitimate, although to perform the attachment using mass-points (in RUN_108), at least one of the mass-points must be dislocated from the actual mass body (we arbitrarily chose to do this to child2).

In contrast, Figure 5.4 illustrates the progression of the development of the mass trees as each attachment is performed. Notice that in the mass-tree, *child3* is not attached to *parent*. In the mass tree, it is the root of the tree containing *child3* that is attached to *parent*.

**Results:**

The resulting conter of mass location relative to the parent structure-point (center of the parent plate) is $(-0.5,\ 0.0,\ 1.0)\ m$ (the point where plates *parent*, *child1* and *child3* meet).

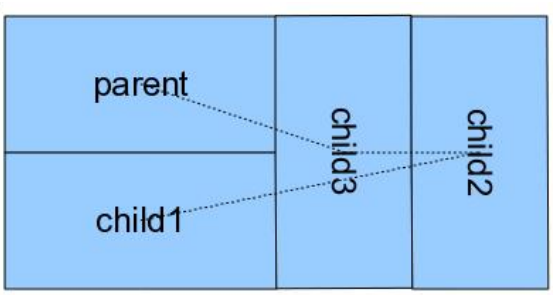


Figure 5.3: Plate arrangement for test

The inertia tensor should be that of a uniform rectangular plate of mass 4.0 $kg$ and sides (2.0, 0.0, 4.0) $m$.

$$I_{comp,CM} = \begin{bmatrix} \frac{16}{3} & 0.0 & 0.0 \\ 0.0 & \frac{20}{3} & 0.0 \\ 0.0 & 0.0 & \frac{4}{3} \end{bmatrix}$$

For Child1, the composite-body comprises the three plates *child1*, *child2*, and *child3*, positioned with respect to the child1 structure-point (and center of mass) at (0, 0, 0) $m$, $(\frac{1}{2},\ 0,\ \frac{3}{2})$ $m$, and $(\frac{1}{2},\ 0,\ \frac{5}{2})$ $m$ respectively. Thus, the center of mass of the composite body should be at $(\frac{1}{3},\ 0, \frac{4}{3})$ $m$. When accumulating the inertia tensor, remember that *child2* and *child3* are oriented so that, oriented to the body-axes of *child1*, their inertia tensors are

$$\begin{bmatrix} \frac{1}{12} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix}$$

The *child1*-based composite body inertia tensor should then be

$$\begin{aligned} I_{child1-composite,body} = &\begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix} + 2 \begin{bmatrix} \frac{1}{12} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} + \\ &\begin{bmatrix} \frac{16}{9} & 0 & -\frac{4}{9} \\ 0 & \frac{17}{9} & 0 \\ -\frac{4}{9} & 0 & \frac{1}{9} \end{bmatrix} + \begin{bmatrix} \frac{1}{36} & 0 & -\frac{1}{36} \\ 0 & \frac{2}{36} & 0 \\ -\frac{1}{36} & 0 & \frac{1}{36} \end{bmatrix} + \begin{bmatrix} \frac{49}{36} & 0 & -\frac{7}{36} \\ 0 & \frac{50}{36} & 0 \\ -\frac{7}{36} & 0 & \frac{1}{36} \end{bmatrix} \\ = &\begin{bmatrix} \frac{11}{3} & 0 & -\frac{2}{3} \\ 0 & \frac{55}{12} & 0 \\ -\frac{2}{3} & 0 & \frac{11}{12} \end{bmatrix} \end{aligned}$$

When considering the composite-body properties of *child2*, notice that the body-axes have been rotated. The composite-body now comprises *child2* and *child3*, which makes a square plate of side 2.0 $m$ and mass 2.0 $kg$, with center of mass at $(\frac{1}{2},\ 0,\ 0)$ $m$ from the structure-point of *child2*. The corresponding inertia tensor is, as we have seen in previous tests:

$$\begin{bmatrix} \frac{2}{3} & 0 & 0 \\ 0 & \frac{4}{3} & 0 \\ 0 & 0 & \frac{2}{3} \end{bmatrix}$$

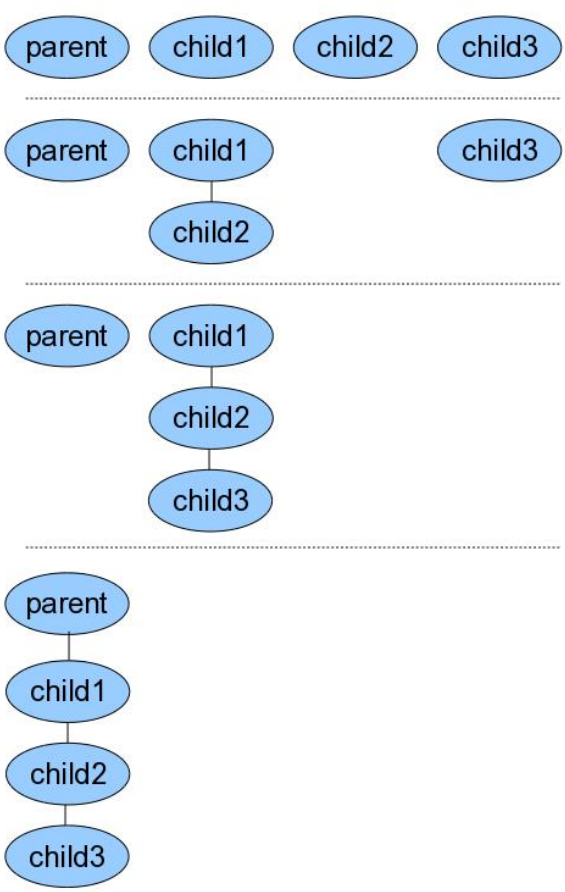


Figure 5.4: Four frames (top to bottom) showing the sequential evolution of the mass tree as the objects are attached.

Finally, the composite-body properties of *child3* should be the same as its core-properties, since it has no children.

All results match predictions.

### 5.2.2 Detach Process

*Test Mass_4: Detach One of Four Plates*

**Purpose:**

The purpose of this test is to confirm that the Mass Body Model correctly monitors mass properties through a detach process.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_10 RUN_110

**Requirements:**

Successful completion of this test satisfies requirement Mass_7.

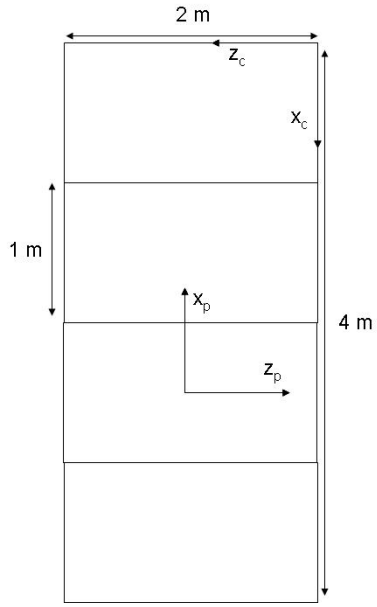


Figure 5.5: Diagram of Test Setup and Reference Frames

**Procedure:**

In this test, we use four flat plates again, stacked along the x-axis as shown in Figure 5.5. All four plates are attached directly as children of the parent, and appear in order of decreasing x (top to bottom):

1. child2
2. child3
3. parent
4. child1

To add to the complexity, the inertia tensor of *child2* is defined with respect to its structure-point, which is defined in one corner (the top right corner in Figure 5.5); the appropriate transformation of such a specification into the body-axes has already been verified in test Mass_6,

and the attach process verified in test Mass_8. It remains only to demonstrate that the mass properties are correct following detachment.

At some time after initialization, *child2* is detached from the mass tree, leaving only three uniform plates attached together. These three plates can be represented as a single uniform plate, of dimensions (3, 0, 2) $m$ and mass 3 $kg$. The parent structure-point, at the center of the parent plate, is now at the geometric center of the composite plate, so the composite body position should be (0, 0, 0) $m$.

The composite-body inertia tensor should be that of a uniform plate:

$$I_{parent-comp,body} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{13}{4} & 0 \\ 0 & 0 & \frac{9}{4} \end{bmatrix}$$

*Child2* properties should remain those of the atomic body.

**Results:**

The results are as expected.

### 5.2.3 Reattach Process

*Test Mass_5: Four Plates with Reattachment*

**Purpose:**

The purpose of this test is to confirm that the Mass Body Model can move an attached body from one location to another (with respect to its parent). It also further confirms the mass model's capability to calculate the inertia of the composite body in the child's reference frame.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_11 RUN_111

**Requirements:**

Successful completion of this test satisfies requirement Mass_8.

**Procedure:**

This test uses three plates similar to those used in previous tests. Initially, they are attached in a stack on the x-axis, similar to test Mass_4:

- *child2*
- *child1*
- *parent*

*child1* and *child2* are both attached to *parent*.

At some time after initialization, *child2* is moved (reattached) such that it is rotated through 90 degrees and attached to the left (-z) of the other two plates. A diagram of the resultant bodies is shown in figure 5.6.

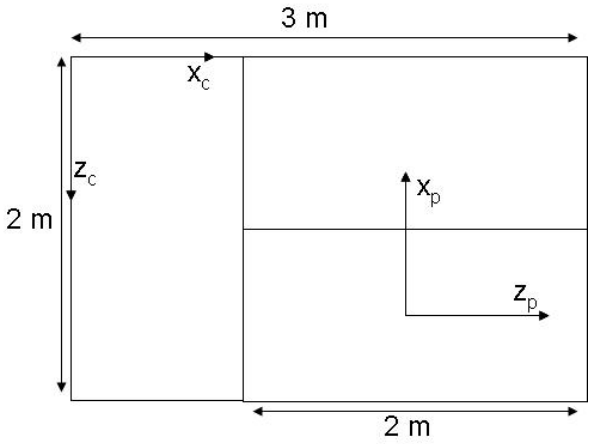


Figure 5.6: Diagram of Reattached Body

**Results:**

The position of the new center of mass should be up and to the left of the center of the parent plate, at a location $(\frac{1}{2},\ 0,\ -\frac{1}{2})\ m$ relative to its structure-point.

The inertia tensor of the new composite plate should be that of a rectangular plate of dimensions $(2,\ 0,\ 3)\ m$ and mass of 3 $kg$.

$$I_{parent-comp,body} = \begin{bmatrix} \frac{9}{4} & 0 & 0 \\ 0 & \frac{13}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since neither *child1* nor *child2* have children, their composite properties should duplicate their core properties.

All results are as expected.

### 5.2.4 Inertia Specification Options

*Test Mass_6: Inertia Specification Option Struct*

**Purpose:**

The purpose of this test is to confirm that the Mass Body Model can correctly compute the inertia tensor referenced to the body-axes when the inertia tensor is provided referenced to an axes-set aligned with the body-axes, but offset from the center of mass. This test additionally verifies the usage of the *inertia_spec* option *Struct*, setting the structural-axes to be aligned with, and offset from, the body-axes, and using these axes for specifying the inertia tensor.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_05

**Requirements:**

Successful completion of this test partially satisfies requirement Mass_5 (aligned and offset).

**Procedure:**

The object used in this test is conceptualized as a planar object, with dimensions (1.0, 0.0, 2.0) $m$, with the structure point in one corner, structure-axes and body-axes aligned, with the body-axes along the body principal axes, and the inertia tensor specified using the structure-axes.

Note - The specification that we are using the structural-axes is provided with the *Struct* option selected for the *inertia_spec* variable.

The inertia tensor refrenced to the body-axes for such a flat plate is:

$$I_{body} = \begin{bmatrix} \frac{1}{3} & 0.0 & 0.0 \\ 0.0 & \frac{5}{12} & 0.0 \\ 0.0 & 0.0 & \frac{1}{12} \end{bmatrix}$$

Transposing to one corner adjusts the inertia tensor by:

$$I_{struc} = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix} + \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & \frac{5}{4} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{4}{3} & 0 & -\frac{1}{2} \\ 0 & \frac{5}{3} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{3} \end{bmatrix}$$

Using this value as the specified inertia tensor, adding that the inertia is specified in the structural, and offsetting the position of the center of mass with respect to structural to be at $(0.5, 0.0, 1.0)$ $m$, the model should produce the correct inertia tensor about the center of mass.

**Results:**

The mass, position, orientation, and inertia tensor all agree with expected values.

*Test Mass_7: Inertia Specification Option, StructCG*

**Purpose:**

The purpose of this test is to confirm that:

1. The Mass Body Model can correctly compute the inertia tensor referenced to the body-axes when the inertia tensor is provided referenced to an axes-set that has its origin at the center of mass, but is misaligned with the body-axes.
2. The inertia-tensor will be correctly computed referenced to non-principal axes.
3. The *inertia_spec* option *StructCG* functions as expected. For purposes of this verification, the specified inertia tensor is referenced to the structural-axes, which are set to be co-located with, but rotated with respect to, the body-axes.
4. A body so defined can be used as the parent in a subsequent attach process.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_09 RUN_109

**Requirements:**

Successful completion of this test partially satisfies requirement Mass_5 (mis-aligned and co-located).

**Procedure:**

This test utilizes two objects, both flat plates similar to that used in test Mass_6, with dimensions $(1.0,\ 0.0,\ 2.0)\ m$. The two plates are joined edge-to-edge to make one uniform flat plate of dimension $(2.0,\ 0.0,\ 2.0)\ m$.

Following the system established in previous tests, RUN_09 has the attachment performed using offset-orientation specification, while RUN_109 uses the mass-point specification to perform the same task.

Both plates have a structure-point at the center of the plate, and structural-axes aligned with the principle axes of the plate. For the child, the body-axes are aligned with the structural-axes, while for the parent they are rotated.

The inertia tensor - referenced to the structural-axes, and body-axes of the child - is that of a flat plate referenced to its principal axes, the same as for the body-axes from test Mass_6

$$I_{struc} = \begin{bmatrix} \frac{1}{3} & 0.0 & 0.0 \\ 0.0 & \frac{5}{12} & 0.0 \\ 0.0 & 0.0 & \frac{1}{12} \end{bmatrix}$$

For the parent, the body axes are rotated with respect to the structural axes such that the transformation matrix from structural to body is $\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{2\sqrt{2}} & -\frac{\sqrt{3}}{2\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$

Consequently, the core-body inertia tensor (referenced to the core-body-axes) for the parent is

$$I_{parent-core,body} = T_{struc\to body} I_{struc} T_{body\to struc} = \begin{bmatrix} \frac{17}{48} & -\frac{\sqrt{3}}{48\sqrt{2}} & \frac{\sqrt{3}}{48\sqrt{2}} \\ -\frac{\sqrt{3}}{48\sqrt{2}} & \frac{23}{96} & -\frac{5}{32} \\ \frac{\sqrt{3}}{48\sqrt{2}} & -\frac{5}{32} & \frac{23}{96} \end{bmatrix} \tag{5.1}$$

The child is attached to the parent such that it has a $-1.0\ m$ offset on the x-axis, and is aligned with the parent. After attachment, the composite-body should be located at $(-0.5,\ 0.0,\ 0.0)\ m$. Referenced to a set of axes aligned with the structural-axes, and centered at the composite center of mass, the composite structure has an inertia tensor equal to that of a square plate of side $2.0\ m$ and mass $2.0\ kg$

$$I_{comp,struc} = \begin{bmatrix} \frac{2}{3} & 0.0 & 0.0 \\ 0.0 & \frac{4}{3} & 0.0 \\ 0.0 & 0.0 & \frac{2}{3} \end{bmatrix}$$

Since the composite-body axes are aligned with the core-body axes, the same transformation used in equation 5.1 must be applied to obtain

$$I_{composite,body} = T_{struc\to body} I_{struc} T_{body\to struc} = \begin{bmatrix} \frac{5}{6} & -\frac{1}{2\sqrt{6}} & \frac{1}{2\sqrt{6}} \\ -\frac{1}{2\sqrt{6}} & \frac{11}{12} & -\frac{1}{4} \\ \frac{1}{2\sqrt{6}} & -\frac{1}{4} & \frac{11}{12} \end{bmatrix}$$

**Results:**

The results for position and for the inertia tensors for both the core and composite bodies agree with predicted values.

*Test Mass_8: Inertia Specification Option SpecCG*

**Purpose:**

The purpose of this test is to exercise the usage of the *inertia_spec* option *SpecCG*. As with test Mass_7, the specified inertia tensor will be referenced to an axes-set that has its origin at the center of mass, but is misaligned with the body-axes. For this test, neither the body-axes nor structural-axes will be used as the reference axes-set for the inertia-tensor; a third axes-set, the specified-axes, must be defined instead. This test additionally confirms that a body so defined can be correctly used as the child in a subsequent attach process.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_07 RUN_107

**Requirements:**

Successful completion of this test partially satisfies requirement Mass_5 (mis-aligned and co-located, body-independent axes).

**Procedure:**

The conceptual object used in this test comprises two identical uniform flat plates similar to those used in tests Mass_6 and Mass_7, each with dimension ( 1.0, 0.0, 2.0) $m$. The child plate is attached to the parent with a 1.0 $m$ offset on the x-axis, making an edge-to-edge attachment to form one uniform flat plate of dimension (2.0, 0.0, 2.0) $m$.

Following the system established in previous tests, RUN_07 has the attachment performed using offset-orientation specification, while RUN_107 uses the mass-point specification to perform the same task.

The parent body has body-axes and structural-axes equivalent and aligned with the principal axes of the plate; while the inertia tensor reference option (*inertia_spec* is technically *StructCG*, this is equivalent to the body-axes anyway. Therefore, its inertia tensor specified about its body axes is, as in previous tests,

$$I_{Body} = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix}$$

The child body also has body-axes and structural-axes equivalent and aligned with the principal axes of the plate. however, it has its inertia tensor specified about a set of axes located at the center of mass, and oriented with transformation matrix $\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ (i.e. a 30-degree rotation about the z-axis)

Note 1 - specified values are expressed with respect to structural-axes that are aligned with body-axes

Note 2 - The specification that we are using an oriented axes-set with origin at the center of mass is provided with the *SpecCG* option selected for the *inertia_spec* variable.

The effective inertia tensor referenced to these axes can be found thus:

$$I_{oriented-axes} = T_{body \to orient} I_{body} T_{orient \to body} = \begin{bmatrix} \frac{17}{48} & \frac{\sqrt{3}}{48} & 0 \\ \frac{\sqrt{3}}{48} & \frac{19}{48} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix}$$

**Results:**

With the inputs provided, the composite-body should have position at $(0.5,\ 0.0,\ 0.0)\ m$ and an inertia tensor equal to that of a flat square plate of mass 2.0 $kg$ and side 2.0 $m$.:

$$I_{composite,body} = \begin{bmatrix} \frac{2}{3} & 0 & 0 \\ 0 & \frac{4}{3} & 0 \\ 0 & 0 & \frac{2}{3} \end{bmatrix}$$

The output from the test matches the analytic prediction.

*Test Mass_9: Inertia Specification Option Spec*

**Purpose:**

The purpose of this test is to build upon the results of test Mass_6 and test Mass_8 to confirm that the Mass Body Model can correctly compute the inertia tensor referenced to the body-axes when the inertia tensor is provided referenced to an axes-set that is both mis-aligned with the body-axes, and offset from the center of mass. As with test Mass_8, this test additionally confirms that a body so defined can be correctly attached to another body.

SIM directory: models/dynamics/body_action/verif/SIM_verif_attach_mass
Run directory: RUN_06 RUN_106

**Requirements:**

Successful completion of this test partially satisfies requirement Mass_5 (mis-aligned and offset, body-independent axes).

**Procedure:**

The conceptual object used in this test comprises two identical uniform flat plates with dimension $(\ 1.0,\ 0.0,\ 2.0)\ m$, joined end-to-end to make one uniform flat plate of dimension $(1.0,\ 0.0,\ 4.0)\ m$.

Following the system established in previous tests, RUN_06 has the attachment performed using offset-orientation specification, while RUN_106 uses the mass-point specification to perform the same task.

The parent body has its inertia tensor referenced to its body axes

$$I_{core,body} = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{5}{12} & 0 \\ 0 & 0 & \frac{1}{12} \end{bmatrix}$$

The child body has its inertia tensor referenced to a set of axes located at $(\frac{\sqrt{3}}{4},\ -\frac{1}{4},\ 1)\ m$, and oriented with transformation matrix $\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ (i.e. a 30-degree rotation about the z-axis)

Note 1 - specified values are expressed with respect to structural-axes that are aligned with body-axes

Note 2 - The specification that we are using an oriented axes-set with offset origin is provided with the *Spec* option selected for the *inertia_spec* variable.

The effective inertia tensor referenced to these axes is:

$$I_{core,specified-axes} = \begin{bmatrix} \frac{5}{12} & \frac{\sqrt{3}}{12} & -\frac{\sqrt{3}}{4} \\ \frac{\sqrt{3}}{12} & \frac{19}{12} & \frac{1}{4} \\ -\frac{\sqrt{3}}{4} & \frac{1}{4} & \frac{4}{3} \end{bmatrix}$$

**Results:**

The resulting parent-based composite body should have a center of mass located at $(0.0,\ 0.0,\ 1.0)\ m$ from the structural-point of the parent, and have an inertia tensor referenced to the composite-body-axes equal to that of a uniform rectangular plate of mass $2.0\ kg$ and sides $(1.0,\ 0.0,\ 4.0)\ m$.

$$I_{parent-composite,body} = \begin{bmatrix} \frac{8}{3} & 0 & 0 \\ 0 & \frac{34}{12} & 0 \\ 0 & 0 & \frac{1}{6} \end{bmatrix}$$

Results agree with expected values to within expected numerical precision.

### 5.2.5 Test in Simulation Environment

*Test Mass_10: SIM Apollo*

**Purpose:**

The purpose of this test is to verif the attach and detach functionality of the Mass Body Model during operation of a regular simulation. This is accomplished by creating a column of attached vehicles to simulate the prelaunch stack of an Apollo mission. Then in a very short time frame the vehicles are detached and reattached as appropriate in an Apollo mission.
SIM directory: sims/SIM_Apollo
Run directory: RUN_test

**Requirements:**

Successful completion of this test confirms that the results of the previous verification tests are legitimate in a simulation environment. It partially satisfies requirements Mass_2, Mass_3, Mass_4, Mass_5, Mass_6, Mass_7, and Mass_8.

**Procedure:**

In this Apollo simulation the Satern V launch stack detaches mass bodies in the samee order as a real Apollo mission, but in a condensed time frame (12 seconds).

1. Detach the first stage.
2. Detach the second stage.
3. Detach the third stage.
4. Detach the LEM.
5. Attach LEM to CM.
6. Detach the LEM.
7. Detach the LEM descent stage.
8. Attach LEM to CM.
9. Detach the LEM for last time.

**Results:**

The Mass Body Model utilization in SIM_Apollo successfully demonstrates run-time attach and detach operations.

## 5.3 Requirements Traceability

Table 5.1 summarizes the inspections and tests that demonstrate the satisfaction of the requirements levied on the model.

Table 5.1: Requirements Traceability

| **Requirement** | **Traces to** |
|---|---|
| Mass_1 JEOD Requirements | Insp. Mass_1 Top-level inspection |
| Mass_2 Center of Mass Location | Test Mass_1 Two-Object Combination<br>Test Mass_2 Complex Tree, Three-object Combinations<br>Test Mass_3 Compound, non-Root Attachment<br>Test Mass_10 SIM Apollo |
| Mass_3 Total Body Mass | Test Mass_1 Two-Object Combination<br>Test Mass_2 Complex Tree, Three-object Combinations<br>Test Mass_3 Compound, non-Root Attachment<br>Test Mass_10 SIM Apollo |
| Mass_4 Composite Inertia Properties | Test Mass_1 Two-Object Combination<br>Test Mass_2 Complex Tree, Three-object Combinations<br>Test Mass_3 Compound, non-Root Attachment<br>Test Mass_10 SIM Apollo |
| Mass_5 Non-Centroidal Inertia Properties | Test Mass_6 Inertia Specification Option Struct<br>Test Mass_7 Inertia Specification Option, StructCG<br>Test Mass_8 Inertia Specification Option SpecCG<br>Test Mass_9 Inertia Specification Option Spec<br>Test Mass_10 SIM Apollo |
| Mass_6 Mass Body Attachment | Test Mass_1 Two-Object Combination<br>Test Mass_2 Complex Tree, Three-object Combinations<br>Test Mass_3 Compound, non-Root Attachment<br>Test Mass_10 SIM Apollo |
| Mass_7 Mass Body Detachment | Test Mass_4 Detach One of Four Plates<br>Test Mass_10 SIM Apollo |
| Mass_8 Reattach Child to Different Point | Test Mass_5 Four Plates with Reattachment<br>Test Mass_10 SIM Apollo |

Continued on next page

Table 5.1: Source Files (continued from previous page)

| **Requirement** | **Traces to** |
|---|---|
| Mass_9 Model Extensibility | Insp. Mass_2 Extensibility inspection |

## 5.4 Metrics

Table 5.2 presents coarse metrics on the source files that comprise the model.

Table 5.2: Coarse Metrics

| File Name | Number of Lines | | | |
|---|---|---|---|---|
| | **Blank** | **Comment** | **Code** | **Total** |
| **Total** | 0 | 0 | 0 | 0 |

Table 5.3 presents the extended cyclomatic complexity (ECC) of the methods defined in the model.

Table 5.3: Cyclomatic Complexity

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassBody::get_mass_properties_initialized () | include/mass.hh | 146 | 1 |
| jeod::MassBody::(std::set_name (std::string name_in) | include/mass.hh | 148 | 1 |
| jeod::MassPoint::MassPoint () | include/mass_point.hh | 118 | 1 |
| jeod::MassPoint::(std::set_name (std::string name_in) | include/mass_point.hh | 133 | 1 |
| jeod::MassPoint::get_name () | include/mass_point.hh | 139 | 1 |
| jeod::MassPoint::find_last_common_index (const Mass Point & ref_point) | include/mass_point.hh | 202 | 1 |
| jeod::MassPoint::find_last_common_node (const Mass Point & frame) | include/mass_point.hh | 219 | 2 |
| jeod::MassPoint::attach (Mass Point & parent) | include/mass_point.hh | 243 | 1 |
| jeod::MassPoint::detach () | include/mass_point.hh | 259 | 1 |
| jeod::MassPointInit::(std::set_name (std::string name_in) | include/mass_point_init.hh | 152 | 1 |
| jeod::MassPointState::update_point (const double pt_location[3]) | include/mass_point_state.hh | 161 | 1 |
| jeod::MassPointState::update_orientation (const double transformation[3][3]) | include/mass_point_state.hh | 175 | 1 |
| jeod::MassPointState::update_orientation (const Quaternion & left_quat) | include/mass_point_state.hh | 191 | 1 |
| jeod::MassPointState::compute_quaternion () | include/mass_point_state.hh | 207 | 1 |
| jeod::MassPointState::compute_transformation () | include/mass_point_state.hh | 218 | 1 |
| jeod::MassPointState::copy_state (const MassPointState & source) | include/mass_point_state.hh | 230 | 1 |

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassBody::JEOD_DECLARE_ATTRIBUTES ( MassPoint) | src/mass.cc | 72 | 1 |
| jeod::MassBody::MassBody ( DynBody& owner) | src/mass.cc | 115 | 1 |
| jeod::MassBody::~MassBody (void) | src/mass.cc | 156 | 6 |
| jeod::MassBody::initialize_mass (const MassPropertiesInit & properties, const MassPointInit * points, unsigned int num_points) | src/mass.cc | 206 | 3 |
| jeod::MassBody::get_parent_body (void) | src/mass.cc | 247 | 1 |
| jeod::MassBody::get_parent_body_internal (void) | src/mass.cc | 261 | 1 |
| jeod::MassBody::get_root_body (void) | src/mass.cc | 274 | 1 |
| jeod::MassBody::get_root_body_internal (void) | src/mass.cc | 287 | 1 |
| jeod::MassBody::is_progeny_of (const MassBody& test_body) | src/mass.cc | 299 | 3 |
| jeod::MassBody::set_update_flag (void) | src/mass.cc | 326 | 2 |
| jeod::MassBody::mass_points_size (void) | src/mass.cc | 342 | 1 |
| jeod::MassBody::find_mass_point (const char * pt_name) | src/mass.cc | 355 | 4 |
| jeod::MassBody::add_mass_point (const MassPointInit & mass_point_init) | src/mass.cc | 386 | 4 |
| jeod::MassBody::attach_to (const char * this_point_name, const char * parent_point_name, MassBody & parent) | src/mass_attach.cc | 60 | 4 |

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassBody::attach_to (double offset_pstr_cstr_ pstr[3], double T_pstr_ cstr[3][3], MassBody & parent) | src/mass_attach.cc | 153 | 4 |
| jeod::MassBody::attach_child (const char * this_point_ name, const char * child_ point_name, MassBody & child) | src/mass_attach.cc | 238 | 1 |
| jeod::MassBody::attach_child (double offset_pstr_cstr_ pstr[3], double T_pstr_ cstr[3][3], MassBody & child) | src/mass_attach.cc | 255 | 1 |
| jeod::MassBody::attach_root_ body (double offset_pstr_ cstr_pstr[3], double T_pstr_ cstr[3][3], MassBody & parent) | src/mass_attach.cc | 275 | 3 |
| jeod::MassBody::attach_ validate (const MassBody & parent, bool generate_ message) | src/mass_attach.cc | 333 | 2 |
| jeod::MassBody::attach_ validate_parent (const Mass Body & parent, bool generate_message) | src/mass_attach.cc | 365 | 7 |
| jeod::MassBody::attach_ validate_child (const Mass Body & child, bool generate_message) | src/mass_attach.cc | 422 | 7 |
| jeod::MassBody::attach_ establish_links (MassBody & parent) | src/mass_attach.cc | 477 | 1 |

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassBody::attach_ update_properties (const double offset_pstr_cstr_ pstr[3], const double T_ pstr_cstr[3][3], MassBody & child) | src/mass_attach.cc | 509 | 1 |
| jeod::MessageHandler:: generate_bad_point_message (const char * file, unsigned int line, const char * child_ body_name, const char * child_point_name, const MassPoint * child_point, const char * parent_body_ name, const char * parent_ point_name, const Mass Point * parent_point) | src/mass_attach.cc | 576 | 12 |
| jeod::MassBody::calc_ composite_cm (void) | src/mass_calc_composite_ cm.cc | 40 | 3 |
| jeod::MassBody::calc_ composite_inertia (void) | src/mass_calc_composite_ inertia.cc | 38 | 2 |
| jeod::MassBody::detach ( MassBody & mass_body) | src/mass_detach.cc | 49 | 7 |
| jeod::MassBody::detach (void) | src/mass_detach.cc | 120 | 3 |
| jeod::MassBody::detach_ validate (const MassBody * parent, bool generate_ message) | src/mass_detach.cc | 170 | 2 |
| jeod::MassBody::detach_ validate_parent (const Mass Body * parent, bool generate_message) | src/mass_detach.cc | 203 | 6 |
| jeod::MassBody::detach_ validate_child (const Mass Body & child, bool generate_message) | src/mass_detach.cc | 254 | 4 |
| jeod::MassBody::detach_ sever_links (MassBody & parent JEOD_UNUSED) | src/mass_detach.cc | 290 | 1 |

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassBody::detach_update_properties (Mass Body & child) | src/mass_detach.cc | 316 | 3 |
| jeod::MassPoint::~MassPoint () | src/mass_point.cc | 56 | 2 |
| jeod::MassPoint::initialize_mass_point () | src/mass_point.cc | 74 | 1 |
| jeod::MassPoint::compute_relative_state (const Mass Point & ref_point, Mass PointState & rel_state) | src/mass_point.cc | 85 | 6 |
| jeod::MassPoint::compute_state_wrt_pred (const Mass Point & ref_point, Mass PointState & rel_state) | src/mass_point.cc | 155 | 2 |
| jeod::MassPoint::compute_state_wrt_pred (unsigned int ref_point_index, Mass PointState & rel_state) | src/mass_point.cc | 185 | 3 |
| jeod::MassPoint::compute_pred_rel_state (const Mass Point & ref_point, Mass PointState & rel_state) | src/mass_point.cc | 236 | 2 |
| jeod::MassPoint::compute_pred_rel_state (unsigned int ref_point_index, MassPoint State & rel_state) | src/mass_point.cc | 270 | 3 |
| jeod::MassPointInit::Mass PointInit () | src/mass_point_init.cc | 54 | 1 |
| jeod::MassPointInit::initialize_mass_point (MassPoint & mass_point) | src/mass_point_init.cc | 68 | 3 |
| jeod::MassBody::compute_point_mass_inertia (double mass, const double r_pt[3], double inertia[3][3]) | src/mass_point_mass_inertia.cc | 36 | 1 |
| jeod::MassPointState::Mass PointState (void) | src/mass_point_state.cc | 50 | 1 |

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

| Method | File | Line | ECC |
|---|---|---|---|
| jeod::MassPointState:: initialize_mass_point (void) | src/mass_point_state.cc | 69 | 1 |
| jeod::MassPointState::negate (const MassPointState & source) | src/mass_point_state.cc | 143 | 2 |
| jeod::MassPointState::incr_ left (const MassPointState & s_ab) | src/mass_point_state.cc | 178 | 2 |
| jeod::MassPointState::incr_ right (const MassPointState & s_bc) | src/mass_point_state.cc | 225 | 2 |
| jeod::MassPointState::decr_ left (const MassPointState & s_ab) | src/mass_point_state.cc | 277 | 2 |
| jeod::MassPointState::decr_ right (const MassPointState & s_bc) | src/mass_point_state.cc | 325 | 3 |
| jeod::MassBody::print_body ( FILE *file_ptr, int levels) | src/mass_print_body.cc | 41 | 4 |
| jeod::MassBody::print_tree (const char * file_name, int levels) | src/mass_print_tree.cc | 43 | 2 |
| jeod::MassProperties::Mass Properties (void) | src/mass_properties.cc | 47 | 1 |
| jeod::MassPropertiesInit:: MassPropertiesInit (void) | src/mass_properties_init.cc | 54 | 1 |
| jeod::MassPropertiesInit:: initialize_mass_properties ( MassProperties & properties) | src/mass_properties_init.cc | 72 | 7 |
| jeod::MassBody::reattach (double offset[3], double T_ pstr_cstr[3][3]) | src/mass_reattach.cc | 40 | 2 |
| jeod::MassBody::update_ mass_properties (void) | src/mass_update.cc | 43 | 10 |

# Bibliography


[1] Bedford and Fowler. *Engineering Mechanics*. Prentice Hall, 2002.

[2] Generated by doxygen. *JEOD Mass Body Model API*. NASA, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2023.

[3] Hammen, D. Dynamic Body Model. Technical Report JSC-61777-dynamics/dyn_body, NASA, Johnson Space Center, Houston, Texas, July 2023.

[4] Hammen, D. Body Action Model. Technical Report JSC-61777-dynamics/body_action, NASA, Johnson Space Center, Houston, Texas, July 2023.

[5] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, July 2023.

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

[7] Thompson, B. Orientation Model. Technical Report JSC-61777-utils/orientation, NASA, Johnson Space Center, Houston, Texas, July 2023.