ニューラルソフト有限会社

（公開資料）	~~草案・企画書~~ 計画~~・規格書~~ ~~検査計画・規格書~~ ~~検査成績書~~	検認		照査	作成
		－	－	－	市来 博記

表　　題

リアルノイド オリジン 運動制御原始モジュール
順動力学演算ブロック 開発計画書

副　　題

RealNoids-Origin Motion Control Primitive Module
Forward dynamics Arithmetic Block Development Plan

キ ー ワ ー ド

リアルノイド/オリジン/運動制御/原始モジュール/順動力学/
RealNoids/Origin/Motion Control/Primitive Module/Forward dynamics

参 照 / 添 付 資 料

三角錐(四面体)の重心廻りの慣性モーメントについての考察
楕円錐台の慣性モーメントについての考察

管 理 番 号	件　　名	改 定 履 歴
000-10-05-0102	ヒューマノイド研究開発	~~A B C~~ D ~~E F G H I J K L M N O P Q R S T U V W X Y Z~~

目次

1. はじめに

本書はロボバイオ リアルノイド オリジン（等身大ヒューマノイド）の全身の運動制御を実現するための運動制御原始モジュールの順動力学演算ブロックの開発範囲とスケジュールを定義するものである。但し、物理演算の中核（エンジン部）に関する開発範囲とスケジュールは『ロボバイオ リアルノイド オリジン運動制御原始モジュール 順動力学演算ブロック 物理演算エンジン開発計画書』に記載する。

2. 用語と略語の定義

本書で使用する用語と略語を表 2-1 に示す。

表 2-1 用語と略語の一覧

用語・略語	説明
原始モジュール	論理的・感情的解釈による制御のパラメータと経路の最適化を行わずに特定の機能を実現するハードウェアー又はソフトウェアー
フレーム	ヒューマノイドの骨組み
リンク	フレームを構成する節
リンク チェーン	フレーム全体又は特定部位を構成する節の繋がり
ジョイント	リンクを連結する部分
モデル	3D 空間内の固有の座標系を定義する原点と XYZ 直交軸のベクトル、及び形状の情報
軸モデル	空形状のモデル
ジョイント モデル	XYZ 直交軸の角度制限の情報を持つモデル
2D モデル	開いた形状、又は内部が空洞の閉じた形状のモデル
3D モデル	内部が空洞でない閉じた形状のモデル
プリミティブ モデル	数個のパラメータ(原点や半径等)で形状を特定するモデル
2D プリミティブ モデル	開いた形状、又は内部が空洞の閉じた形状のプリミティブ モデル
3D プリミティブ モデル	内部が空洞でない閉じた形状のプリミティブ モデル
空間管理	3D 空間内のモデルを管理する運動制御原始モジュール内のブロック（本書の記載範囲外のブロック）
思考ブロック(T-BLK)	意志と行動を対応付ける最小単位のデータとプログラム(Thinking Block)
思考ブロック チェーン(T-BLKC)	複数の思考ブロックを連結したもの
順動力学演算	リンク チェーンのジョイントに発生するトルクに対する終端の加速度を求める数学的プロセス。
逆動力学演算	リンク チェーンの終端の加速度を実現するために必要となるジョイントのトルクを求める数学的プロセス。
CCD	連続的衝突判定(Continuous Collision Detection)
物理演算要素／物理ワールド要素	順動力学演算と逆動力学演算の対象となるモデル

3. 運動制御原始モジュール 順動力学演算ブロックの概要

順動力学演算ブロックはヒューマノイドの開発段階で、運動における動力計算の適合性を確認する事と、ヒューマノイドが稼働時に運動を学習する過程で、運動に対応する思考ブロック チェーンの妥当性の確認を可能にすることを目的とするソフトウェアーである。

4. 基本方針

順動力学演算ブロックの開発における基本方針を示す。

- 実機環境（ARM 64BIT 4Core 以上 Linux 系 OS）及びシミュレーション環境（Windows 10 で動作する CINEMA 4D のプラグイン）で動作するネイティブ プログラムとし、C++言語で記述する。
- 空間座標系は Y UP/左手系とする。（座標系のスケールは使用しない。）
- ~~リンク チェーンの根本以外のジョイントは親ジョイントの Z 軸上に配置されるものとする。~~
- ジョイントの動作は回転（HPB 順）のみとする。（リンクの伸縮は不可とする。）
- ジョイントは可動範囲の制限を設けることができるものとする。（-180.0 と+180.0 を跨ぐ範囲制限は不可とする。）
- ジョイントの座標変換は固定できるものとする。（任意の角度を 0 度とすることができる。）
- 物理演算エンジンは Bullet、PhysX、RoboBio-X（オリジナル）の何れかを選択できるものとする。但し、PhysX の組み込みは実験的（複数の物理演算エンジンの組み込みが可能であることを実証する為。）なものとし、拘束を伴う運動に関する実装は行わない。（PhysX の実機への組み込みは行わない。）

5. 構成

順動力学演算ブロックは実機環境及びシミュレーション環境下において、上位機能モジュールからの要求と指令に従って動作する。実機環境とシミュレーション環境における関連モジュールの構成を図 5-1 と図 5-2 に示す。

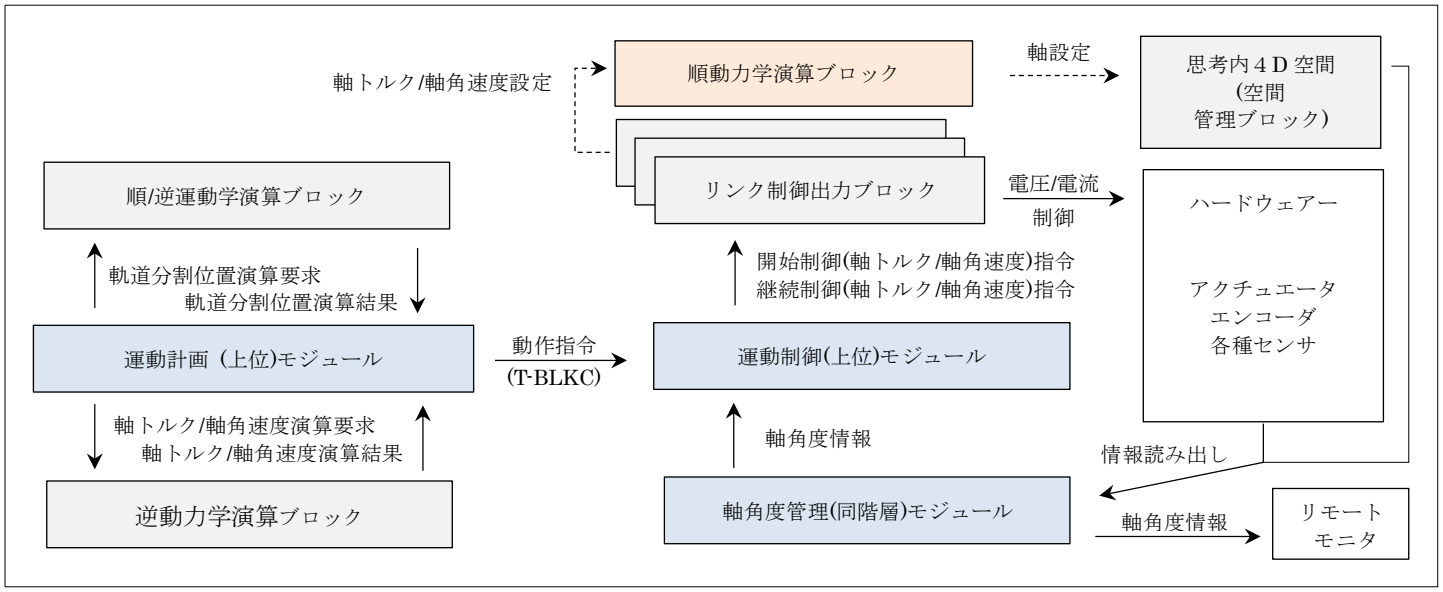

図 5-1 実機環境における関連モジュールの構成

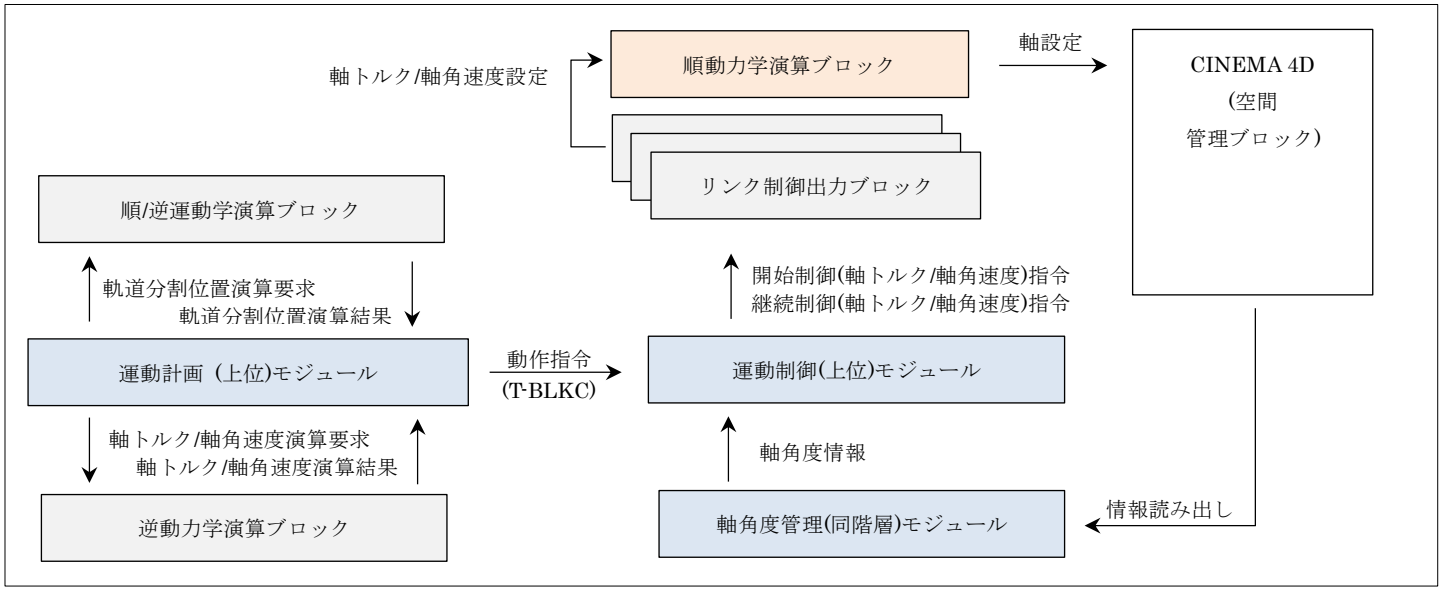

図 5-2 シミュレーション環境における関連モジュールの構成

5.1 順動力学演算ブロック内の詳細ブロックの構成

順動力学演算ブロックの詳細ブロックを図 5-3 に示す。

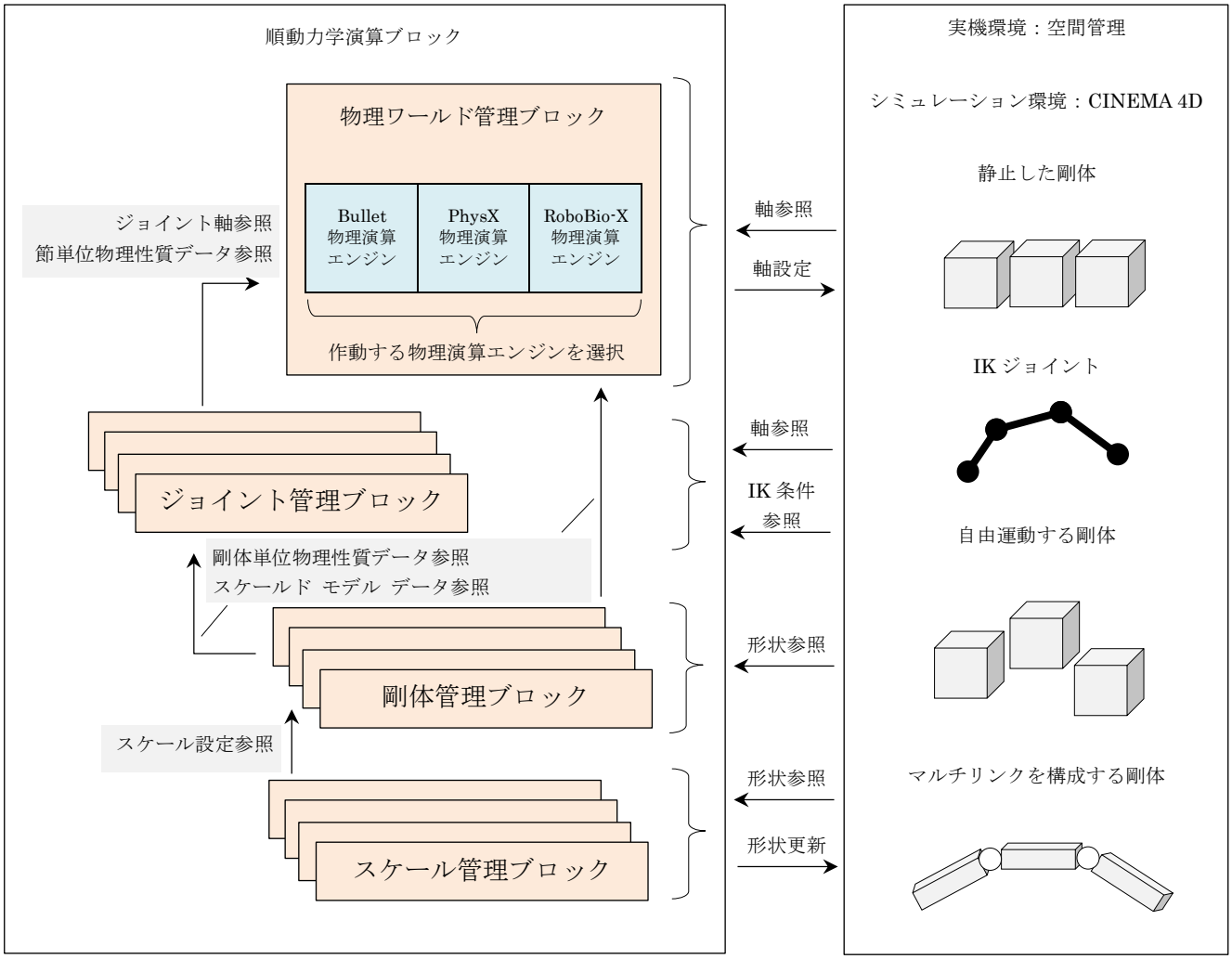

図 5-3 順動力学演算ブロック内の詳細ブロックの構成

各物理演算エンジンは運動制御原始モジュールに静的にリンクされたプログラムとし、シミュレーション環境においても CINEMA 4D 内蔵の物理演算エンジンは使用しない。（プラグイン モジュールからの直接利用が不可であり、質量/慣性モーメントと、ジョイントに対するトルク設定の仕様が不明である為。）

6. 機能

順動力学演算ブロックは単一の剛体及び拘束された複数の剛体の衝突を含む運動のシミュレーションを行うことができる。運動のシミュレーションには剛体の物理性質、拘束条件、駆動条件の定義が必要となるため、5.1 章に示した詳細ブロックに機能を分割する。

以下に詳細ブロックの大機能を示す。

(1) スケール管理ブロック
  (A) スケール変更の対象となるモデルの指定
  (B) プリミティブ モデルのスケール変更（座標系のスケールは変更しない）
(2) 剛体管理ブロック
  (A) 剛体の実体（実形状）となるモデルの指定
  (B) 物理性質データの算出（剛体内で均一の比重と形状から算出）
  (C) 物理性質データの提示（ジョイント管理及び物理ワールド管理ブロックとの連携）
  (D) ジョイント配下の節への所属状態の保持（ジョイント管理ブロックとの連携）
  (E) ジョイント配下の節への所属状態の提示（物理ワールド管理ブロックとの連携）
  (F) 運動状態のモニター（物理ワールド管理ブロックとの連携）
  (G) スケールド モデル データ ファクトリの提示
(3) ジョイント管理ブロック
  (A) ジョイントの実体となるモデルの指定
  (B) ジョイント タイプに応じた軸構成の確立（位置/向き/制限）
  (C) ジョイント タイプに応じた節の確立
  (D) 節単位物理性質データの算出（剛体管理ブロックとの連携）
  (E) 駆動条件の管理
  (F) 軸及び節の構成と、物理性質データ及び駆動条件の提示（物理ワールド管理ブロックとの連携）
  (G) 運動状態のモニター（物理ワールド管理ブロックとの連携）
(4) 物理ワールド管理ブロック
  (A) 物理ワールドの中心となるモデルの指定
  (B) 物理演算エンジンの選択と演算条件の設定
  (C) 物理演算要素の抽出（ジョイント管理及び剛体管理ブロックとの連携）
  (D) 物理演算エンジンへの物理演算要素の登録と登録抹消（物理演算エンジンとの連携）
  (E) 物理演算の実行（物理演算エンジンとの連携）
  (F) 物理ワールド内の物理演算要素への物理演算結果の反映（物理演算エンジンとの連携）
  (G) 物理演算要素の運動状態の提示（剛体管理ブロック及びジョイント管理ブロックとの連携）

図 6-1 に 3D 空間における簡易な脚のフレームとボールを例に各機能の連携の概要を示す。

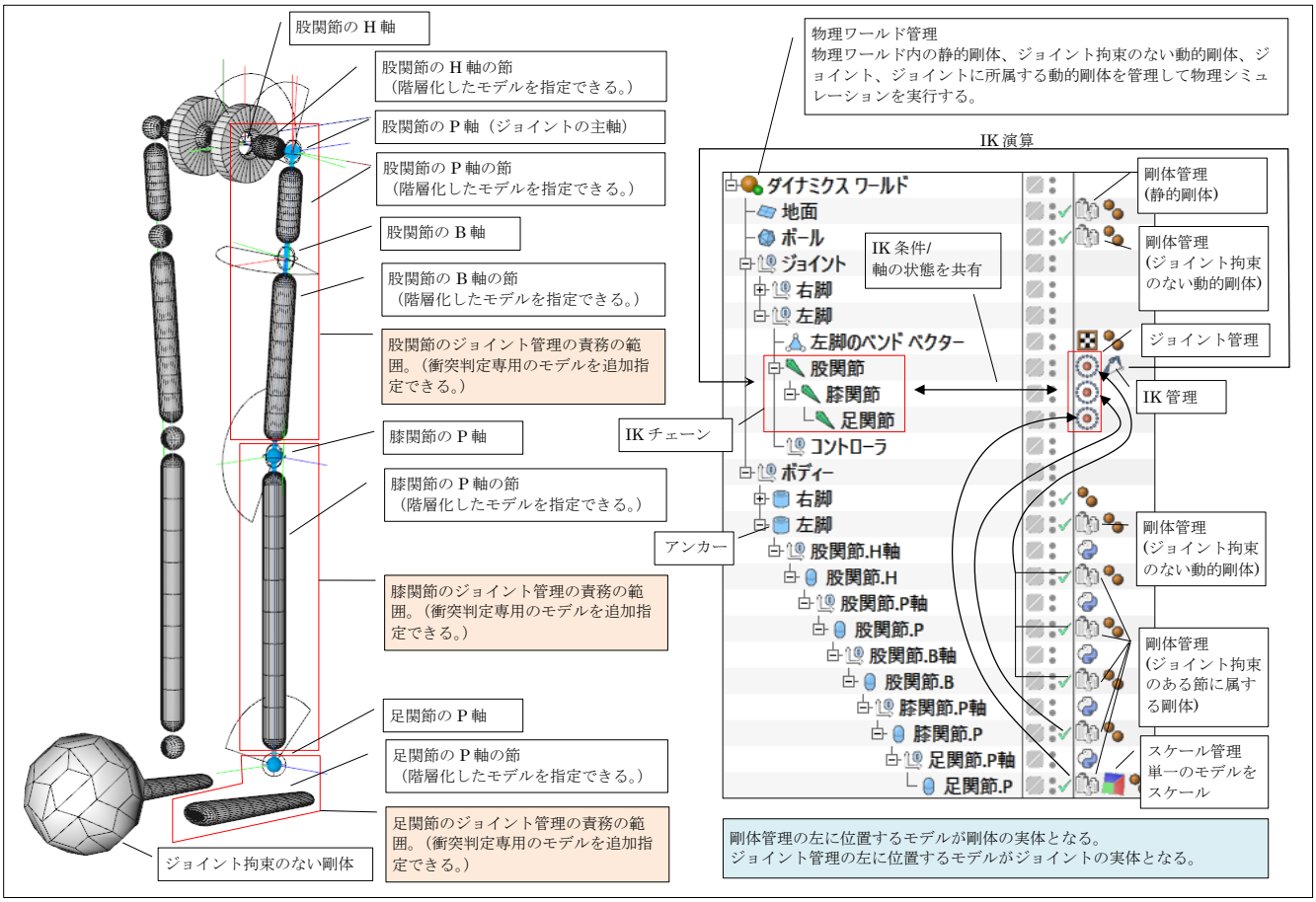

図 6-1 各機能の連携の概要

次章以降に各詳細ブロックの機能の詳細を示す。

6.1 スケール管理ブロックの機能

(1) 表 6-1 の条件に適合するモデルをスケール対象として指定できる。

表 6-1 スケール設定可能なプリミティブ モデルの条件

立方体	球(半球を含む)	円柱	カプセル	円錐
四角錐	正多面体	チューブ	円環	オイルタンク

(2) 機能の無効化

スケール設定の対象となるモデルが表 6-1 の条件に適合しない場合、スケール機能を無効化する。

(3) 軸別スケール設定

X/Y/Z 軸毎にスケール値を指定できる。(スケール値の上下限はなし)

(4) 不均一なカプセルのスケール設定

スケール設定の対象となるモデルがカプセルの場合、上下端の半径のスケール値(0.0 以上)を指定できる。

(5) モデルの変形

(3)と(4)項の設定に従ってプリミティブ モデルの形状を変形する。

6.2 剛体管理ブロックの機能

(1) 剛体の実体の指定

モデルを剛体の実体として指定できる。

(2) 比重指定

剛体の比重（剛体内で均一）を指定することができる。比重＞0 の場合は動的剛体、比重＝0 の場合は無限大の質量を持つ静的剛体とする。

(3) 次元(2D/3D)指定

剛体の内部が詰まった状態(3D)か、空洞の状態(2D)かを指定できる。

内部が空洞の場合は表層の厚みを指定できる。

(4) ポリゴン化指定

剛体が表 6-2 の条件に適合したプリミティブ モデルの場合、ポリゴン化した形状を剛体として扱うように指定することができる。（例：円柱プリミティブ モデル → 六角柱ポリゴン モデル）

表 6-2 プリミティブ モデルの条件

モデルの種類	属性
立方体	フィレットが無効
球	常にポリゴン化が可能
カプセル	スライスが無効
円柱	スライスとフィレットが無効
チューブ	スライスとフィレットが無効
円錐	スライスとフィレットが無効（円錐台の場合は上端のフィレットも無効）

(5) 分割数を制限

プリミティブ モデルの面の分割数を最小値に抑制することができる。（論理形状による物理演算ができない場合、演算負荷を低減できる。）

表 6-3 にプリミティブ モデルの分割数を抑制する部位と制限値を示す。

表 6-3 プリミティブ モデルの分割数を抑制する部位と制限値

モデル	分割数を抑制する部位と制限値
立方体（直方体）	X/Y/Z 方向の分割数=1
カプセル	高さ方向の分割数=1
円柱（楕円柱）	高さ方向の分割数=1
円錐（楕円錐） 円錐台（楕円錐台）	高さ方向の分割数=1 キャップの分割数=1（キャップありの場合）
ディスク（楕円）	放射方向の分割数=1
平面（長方形）	幅方向の分割数=1 高さ方向の分割数=1
ポリゴン（三角形/四角形）	分割数=1

(6) 衝突判定用形状の指定

物理性質データを決定する形状とは別に表 6-4 の条件に適合するモデルを衝突判定で用いる形状として指定できる。（衝突判定で用いる形状を単純化することで物理シミュレーション時の演算負荷を低減できる。）

表 6-4 衝突判定形状モデルの条件

モデルの種類（網掛されていないモデルはプリミティブ モデル）			
立方体	球	カプセル	円柱
チューブ	円錐	正多面体	ディスク
平面	シンプル ポリゴン	地形	ポリゴン

(7) ジョイントで拘束された節への所属管理

剛体がジョイント管理によって構成された節に属する場合、所属先ジョイントのモデルの情報を保持する。

(8) 運動条件の設定

以下の項目を設定できる。

(A) 衝突マージン（範囲は 0.0 以上。単位はシステム設定に従う。）

(B) 初期線形速度(3 次元ベクトル指定。単位はシステム設定に従う。)

(C) 初期回転速度（3 次元ベクトル指定。単位はラジアン（表示は度)）

(D) ローカル/グローバル座標系指定

（初期線形速度と初期回転速度の 3 次元ベクトルをグローバル座標系のベクトルとするか、剛体のローカル座標系のベクトルとするかを指定できる。）

(E) 反発（$= -\frac{\text{衝突後の速度}}{\text{衝突前の速度}}$）（設定範囲は 0 以上）

(F) 摩擦（設定範囲は 0 以上）

以下の計算式で接触する 2 剛体の滑り摩擦係数を決定する。

滑り摩擦係数 = 剛体 A の摩擦 × 剛体 B の摩擦（静止摩擦と動摩擦の区別はない。）

(G) 転がり摩擦（設定範囲は 0 以上）

接触法線に直行する方向に対する摩擦で、回転の停止に寄与する摩擦。

以下の計算式で接触する 2 剛体の転がり摩擦係数を決定する。

転がり摩擦係数 = 剛体 A の転がり摩擦 × 剛体 B の摩擦＋剛体 B の転がり摩擦 × 剛体 A の摩擦

(H) 回転摩擦（設定範囲は 0 以上）

接触法線周りの摩擦で、剛体の把持に寄与する摩擦。

以下の計算式で接触する 2 剛体の回転摩擦係数を決定する。

回転摩擦係数 = 剛体 A の回転摩擦 × 剛体 B の摩擦＋剛体 B の回転摩擦 × 剛体 A の摩擦

(I) 線形ダンピング

以下の計算式で線形速度を減衰する。

$$\text{線形速度} = \text{線形速度} \times \left(1.0 - \text{線形ダンピング}\right)^{\text{シミュレーション ステップ時間(秒単位)}}$$

(J) 回転ダンピング

以下の計算式で回転速度を減衰する。

$$\text{回転速度} = \text{回転速度} \times \left(1.0 - \text{回転ダンピング}\right)^{\text{シミュレーション ステップ時間(秒単位)}}$$

(9) 物理演算不活性化設定

以下の項目を設定できる。

(A) 線形速度の閾値

(B) 回転速度の閾値

(C) 不活性化遅延時間

剛体の運動の速度が線形速度の閾値と回転速度の閾値の両方を下回ってから不活性化遅延時間が経過した時点で、物理演算の対象から剛体を除外する。

(10) 物理性質データの算出

モデルの種類と属性、及び(2)~(4)の設定に応じて以下の項目を自動算出できる。

(A) 体積（単位はm^3とシステム設定の単位^3）

(B) 質量（単位は kg と g）

(C) 重心（モデルのローカル座標系における重心の位置。単位は m とシステム設定の単位）

(D) モデルのローカル座標系の原点における慣性モーメント テンソル（単位は$\mathrm{kg}\cdot m^2$）

(E) モデルの慣性主軸系における慣性モーメント テンソルの対角成分（単位は$\mathrm{kg}\cdot m^2$）

(D)(E)の慣性モーメント テンソルの要素値は 0 値に丸める閾値（指数）を指定できる。

表 6-5 にプリミティブ モデルと演算の対象となる形状の関係を示す。

表 6-5 3D プリミティブ モデルと演算の対象となる形状の関係

モデル	モデルの属性					演算対象
	キャップ	3/2D	スライス	フィレット	ポリゴン化	
立方体（直方体）	－	3	－	×	×	直方体（論理形状）
	－	3	－	×	○	ポリゴン形状
	－	3	－	○	－	ポリゴン形状
	－	2	－	・	－	ポリゴン形状（内部は空洞）
球（楕円体）	－	3	－	－	×	楕円体（論理形状）
	－	3	－	－	○	ポリゴン形状
	－	2	－	－	－	ポリゴン形状（内部は空洞）
半球（半楕円体）	－	2!	－	－	－	ポリゴン 2D 形状
カプセル（上下端均一）	－	3	×	－	×	上下端均一カプセル（論理形状） 楕円柱+半楕円体×2
	－	3	×	－	○	ポリゴン形状
	－	3	○	－	－	ポリゴン形状
	－	2	・	－	－	ポリゴン形状（内部は空洞）
カプセル（上下端不均一）	－	3	×		×	上下端不均一カプセル（論理形状） 楕円錐台+半楕円体×2
	－	3	×		○	ポリゴン形状
	－	3	○		－	ポリゴン形状
	－	2	・	－	－	ポリゴン形状（内部は空洞）

表 6-5 3D プリミティブ モデルと演算の対象となる形状の関係

モデル	モデルの属性					演算対象
	キャップ	3/2D	スライス	フィレット	ポリゴン化	
円柱（楕円柱）	○	3	×	×	×	楕円柱（論理形状）
	○	3	×	×	○	ポリゴン形状
	○	3	×	○	－	ポリゴン形状
	○	3	○	×	－	ポリゴン形状
	○	3	○	○	－	ポリゴン形状
	○	2	・	・	－	ポリゴン形状（内部は空洞）
	×	2!	・	－	－	ポリゴン 2D 形状
チューブ	－	3	×	×	×	楕円柱－楕円柱（論理形状）
	－	3	×	×	○	ポリゴン形状
	－	3	×	○	－	ポリゴン形状
	－	3	○	×	－	ポリゴン形状
	－	3	○	○	－	ポリゴン形状
	－	2	・	・	－	ポリゴン形状（内部は空洞）
円錐（楕円錐）	○	3	×	×	×	楕円錐（論理形状）
	○	3	×	×	○	ポリゴン形状
	○	3	×	○	－	ポリゴン形状
	○	3	○	×	－	ポリゴン形状
	○	3	○	○	－	ポリゴン形状
	○	2	・	・	－	ポリゴン形状（内部は空洞）
	×	2!	・	・	－	ポリゴン 2D 形状
円錐台（楕円錐台）	○	3	×	×	×	楕円錐台（論理形状）
	○	3	×	×	○	ポリゴン形状
	○	3	×	○	－	ポリゴン形状
	○	3	○	×	－	ポリゴン形状
	○	3	○	○	－	ポリゴン形状
	○	2	・	・	－	ポリゴン形状（内部は空洞）
	×	2!	・	・	－	ポリゴン 2D 形状
地形（球状）	－	3	－	－	－	ポリゴン形状
	－	2	－	－	－	ポリゴン形状（内部は空洞）
地形	－	2!	－	－	－	ポリゴン 2D 形状
ポリゴン形状	－	3	－	－	－	ポリゴン形状
	－	2	－	－	－	ポリゴン形状（内部は空洞）
ディスク（楕円）	－	2!	・	－	－	楕円の薄板（論理形状）
平面（長方形）	－	2!	－	－	－	長方形の薄板（論理形状）
ポリゴン（三角形/四角形）	－	2!	－	－	－	三角形の薄板（論理形状） 四角形の薄板（論理形状）

モデルの属性の記号：「－」は選択不可、「○」は有効、「×」は無効、「・」は有効/無効の両方、「2!」は強制的に 2D が設定されることを意味する

表 6-6 にポリゴン モデルと演算の対象となる形状の関係を示す。

表 6-6 3D ポリゴン モデルと演算の対象となる形状の関係

モデル	モデル属性	演算対象
	3/2D	
ポリゴン	3	ポリゴン形状
	2	ポリゴン 2D 形状（内部は空洞）

表 6-7 に形状別に物理性質値の算出方法を示す。

表 6-7 形状別の物理性質値の算出方法

形状	算出方法（計算式では単位を適切に合わせる。）
直方体	2a = len.x, 2b = len.y, 2c = len.z とする。 重心=(0, 0, 0) 体積=2a×2b×2c 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = 1/3 \times (b^2 + c^2) \times 質量$ $Iy = 1/3 \times (c^2 + a^2) \times 質量$ $Iz = 1/3 \times (a^2 + b^2) \times 質量$
楕円体	a = r.x, b = r.y, c = r.z とする。 重心=(0, 0, 0) 体積= 4/3×π×a×b×c 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = 1/5 \times (b^2 + c^2) \times 質量$ $Iy = 1/5 \times (c^2 + a^2) \times 質量$ $Iz = 1/5 \times (a^2 + b^2) \times 質量$
楕円柱	a = r.x, b = r.z, h=高さ(Y 軸方向)とする。 重心=(0, 0, 0) 体積=π×a×b×h 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Iy = 1/4 \times (a^2 + b^2) \times 質量$ $Ix = (b^2/4 + h^2/12) \times 質量$ $Iz = (a^2/4 + h^2/12) \times 質量$
楕円錐	a = r.x, b = r.y, h=高さ(Y 軸方向)とする。 重心=(0, -h/4, 0) 体積= 1/3×π×a×b×h 質量=比重×体積 慣性モーメント 頂点における慣性モーメントから平行軸の定理によって求める。 $I_y = 3/20 \times (a^2 + b^2) \times 質量$ $I_{x(頂点)} = 3/20 \times b^2 \times 質量 + 3/5 \times h^2 \times 質量$ $I_{z(頂点)} = 3/20 \times a^2 \times 質量 + 3/5 \times h^2 \times 質量$ 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = I_{x(頂点)} - (3/4 \times h)^2 \times 質量$ $Iz = I_{z(頂点)} - (3/4 \times h)^2 \times 質量$
楕円錐台	楕円錐台の物理性質値の算出方法は『楕円錐台の慣性モーメントについての考察』を参照のこと。
上下端均一カプセル	上端の半楕円体＋楕円柱＋下端の半楕円体として算出する。 半楕円体 a = r.x, b = r.z, h=高さ(Y 軸方向)とする。 重心= (0, 3/8×h, 0) 体積=1/2×楕円体の体積 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Iy = 1/5 \times (a^2 + b^2) \times 質量$ $Ix = 83/320 \times z \times h \times 質量$ $Iz = 83/320 \times a \times h \times 質量$ 楕円柱の物理性質値の算出方法は本表の楕円柱形状の行を参照のこと。
上下端不均一カプセル	上端の半楕円体＋楕円錐台＋下端の半楕円体として算出する。 半楕円体の物理性質値の算出方法は上下端均一カプセルの半楕円体と同じ。 楕円錐台の物理性質値の算出方法は『楕円錐台の慣性モーメントについての考察』を参照のこと。
チューブ	外径の楕円柱－内径の楕円柱として算出する。 楕円柱の物理性質値の算出方法は本表の楕円柱形状の行を参照のこと。

表 6-7 形状別の物理性質値の算出方法

形状	算出方法（計算式では単位を適切に合わせる。）
楕円の薄板	外径の楕円の薄板－内径の楕円の薄板として算出する。 楕円の薄板の物理性質値の算出方法 XZ 平面 2a = len.x, 2b = len.z とする。 重心= (0, 0, 0) 体積= a×b×π×厚み 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = 1/4 \times b^2 \times$ 質量 $Iy = 1/4 \times (a^2 + b^2) \times$ 質量 $Iz = 1/4 \times a^2 \times$ 質量
長方形の薄板	XZ 平面 2a = len.x, 2b = len.z とする。 重心= (0, 0, 0) 体積= 2a×2b×厚み 質量=比重×体積 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = 1/3 \times b^2 \times$ 質量 $Iy = 1/3 \times (a^2 + b^2) \times$ 質量 $Iz = 1/3 \times a^2 \times$ 質量
三角形の薄板	三角形の面が XZ 平面に平行、三角形の一辺が X 軸に平行になるように座標を回転変換してから算出する。 XZ 平面上の三点を a,b,c とする。bc は X 軸と平行、 l = bc.len, h = a.z - b.z 又は a.z - c.z、 G = 重心、 L1=(a-G).len、L2=(b-G).len、L3=(c-G).len とする。 重心= (0, 0, -h×0.5 + h×(1.0 / 3.0)) 体積= l×h×0.5×厚み 慣性モーメント テンソル=$\begin{pmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{pmatrix}$ $Ix = 1/18 \times h^2 \times$ 質量 $Iy = 1/12 \times (L1^2 + L2^2 + L3^2) \times$ 質量 $Iz = Iy - Ix$
ポリゴン形状	各三角形ポリゴンを底面、原点を頂点とする三角錐の物理性質値を合算（平行軸の定理を使用）して求める。 体積 = ∑三角錐の体積　　（三角錐の体積＝$\frac{底面積 \times 高さ}{3}$） 質量=比重×体積 重心 = $\frac{\sum(三角錐の重心 \times 三角錐の体積)}{ポリゴン形状全体の体積}$ 慣性モーメント テンソル I = 三角錐の重心周りの慣性モーメント テンソル a = $(a_x \quad a_y \quad a_z)$　ポリゴン形状の原点から三角錐の重心へのベクトル I′ = 三角錐の質量 × $\begin{pmatrix} a_y^2 + a_z^2 & -a_y a_x & -a_z a_x \\ -a_x a_y & a_x^2 + a_z^2 & -a_z a_y \\ -a_x a_z & -a_y a_z & a_x^2 + a_y^2 \end{pmatrix}$ ポリゴン形状の原点周りの慣性モーメント テンソル = ∑(I + I′) 三角錐の重心周りの慣性モーメント テンソルの算出方法は『三角錐(四面体)の重心廻りの慣性モーメントについての考察』を参照のこと。 三角錐の慣性モーメント テンソル算出の手順 三角錐の底面の法線が Z 軸と平行になるように回転(=R1) 三角錐の底面の辺 ab が X 軸と平行になるように回転(=R2) 三角錐の重心周りの慣性モーメント テンソルを求める R2 回転を実施した場合は慣性モーメント テンソルを R1 逆回転 R1 回転を実施した場合は慣性モーメント テンソルを R2 逆回転 ポリゴン形状の主慣性モーメントはポリゴン形状の原点周りの慣性モーメント テンソルをヤコビ法で対角化して求める。（対角化に必要な回転角度＝ポリゴン モデルのローカル座標軸と慣性主軸との角度）

表 6-7 形状別の物理性質値の算出方法

形状	算出方法（計算式では単位を適切に合わせる。）
ポリゴン 2D 形状	各三角形ポリゴンの物理性質値を合算（平行軸の定理を使用）して求める。 面積 = ∑三角形の面積 質量=比重×面積×厚み 重心 = $\frac{\sum(\text{三角形の重心}\times\text{三角形の面積})}{\text{ポリゴン 2D 形状全体の面積}}$ 慣性モーメント テンソル I = 三角形の重心周りの慣性モーメント テンソル a = $(a_x \quad a_y \quad a_z)$　ポリゴン 2D 形状の原点から三角形の重心へのベクトル I′ = 三角形の質量（厚み分）× $\begin{pmatrix} a_y^2+a_z^2 & -a_y a_x & -a_z a_x \\ -a_x a_y & a_x^2+a_z^2 & -a_z a_y \\ -a_x a_z & -a_y a_z & a_x^2+a_y^2 \end{pmatrix}$ ポリゴン 2D 形状の原点周りの慣性モーメント テンソル = ∑(I + I′) 三角形の重心周りの慣性モーメント テンソルの算出方法は本表の三角形の薄板形状の行を参照のこと。（但し、XZ 平面は XY 平面に読みかえること。） 三角形の慣性モーメント テンソル算出の手順 三角形の法線が Z 軸と平行になるように回転(=R1) 三角形の辺 ab が X 軸と平行になるように回転(=R2) 三角形の重心周りの慣性モーメント テンソルを求める R2 回転を実施した場合は慣性モーメント テンソルを R1 逆回転 R1 回転を実施した場合は慣性モーメント テンソルを R2 逆回転 ポリゴン 2D 形状の主慣性モーメントはポリゴン 2D 形状の原点周りの慣性モーメント テンソルをヤコビ法で対角化して求める。（対角化に必要な回転角度＝ポリゴン モデルのローカル座標軸と慣性主軸との角度）

(11) 剛体の初期の位置と角度を記憶

現在の剛体の位置と角度を初期の位置と角度として記憶することができる。

(12) 剛体の位置と角度をリセット

剛体の位置と角度を初期の位置と角度に復帰することができる。

(13) 運動状況モニターの有効/無効設定

剛体の運動状況をモニターするかどうかを選択できる。

(14) 運動状況のモニター

剛体の運動に対し、以下の項目をモニターすることができる。

(A) 線形速度（単位は m/s）

(B) 回転速度（単位はラジアン/s。表示単位は度/s）

上記の項目をグローバル座標系の値とするか、剛体のローカル座標系の値とするかを選択できる。

(15) 剛体の物理性質と運動状況の可視化（シミュレーション環境のみ）

以下に示す項目を可視化することができる。

(A) 剛体の重心位置と慣性主軸

重心の描画サイズと描画色を指定できる。（サイズ指定が 0 の場合はモデルのバウンダリ半径の最小値とする。）

慣性主軸の描画長は重心の描画サイズの 40 倍とする。

慣性主軸の表示色は CINEMA 4D の一般設定の軸の表示色に従う。

~~(B) 線形速度ベクトル~~

~~(C) 回転速度ベクトル~~

上記の項目の表示サイズ又はスケールと表示色を設定できる。

(16) 慣性モーメント L スケール設定

慣性モーメントを算出する際の重心からの距離のスケールを指定できる。（ジョイントで拘束されている複数の剛体中の剛体単体の角加速度を小さくすることで単位時間後の姿勢（拘束による位置と姿勢の修正プロセスの前段階）の変位を小さくして物理演算を安定化させることができる。）

(17) スケールド モデル データ ファクトリの提示

他の機能ブロック（スケール管理を除く）にスケールド モデル データを生成するファクトリを提示することができる。

スケールド モデル データ ファクトリは以下の機能を有する。

- モデル、ポリゴン化指定、次元(2D/3D)指定の情報からスケールド モデル データを生成する。
- モデル、剛体管理の情報からスケールド モデル データを生成する。
- モデルの情報から実形状のスケールド モデル データを生成する。
- モデルの情報から衝突判定形状のスケールド モデル データを生成する。
- モデルと実形状/衝突判定形状指定の情報からスケールド モデル データを生成する。

6.3 ジョイント管理ブロックの機能

(1) ジョイントの実体の指定
　モデルをジョイントの実体として指定できる。
(2) ジョイント タイプの選択
　以下のタイプの何れかを選択できる。

　(A) 組み合わせヒンジ
　　H/P/B 順に階層化され、全ての軸と節が独立しているジョイント
　(B) カルダン＋ヒンジ
　　HP/B 順に階層化され、H 軸と P 軸の原点の位置と節の構成が等しく、HP 軸と B 軸が独立しているジョイント
　(C) 3DOF
　　H 軸と P 軸と B 軸の原点の位置と節の構成が等しいジョイント。（P 軸の最大可動範囲は-90 度から 90 度に限定される。）

　上記の全ての形式のジョイントは B 軸の延長上に直動する E 軸を持つことができる。但し、これは直動駆動動力源用の軸とし、ヒューマノイドのフレームのリンクの伸縮を実現するものではない。
　「カルダン＋ヒンジ」と「3DOF」は、物理ワールド管理ブロックの物理演算エンジンの選択が Bullet の場合のみ有効とする。
(3) ルート指定
　ジョイントがリンク チェーンのルートかどうかを指定できる。
(4) 分離リグ指定
　ジョイントの実体がモデルの階層から分離された IK チェーン内の IK ジョイントであることを指定することができる。
(5) ジョイントの状態管理
　外部機能に対して、ジョイントの設定の有効性を提示することができる。
(6) 節の実構成の設定
　ジョイント タイプに応じて節を構成することができる。

　(A) 組み合わせヒンジ
　　（ア）H/P/B/E 軸の節を構成するモデルのルート（以降、ルート モデルと記す。）を指定できる。
　　（イ）ジョイントの最後の軸の節の終端となるモデルを指定できる。（B 軸と E 軸を物理固定軸とした場合、最後の軸の節は、P 軸の節となる。）
　(B) カルダン＋ヒンジ
　　（ア）HP/B/E 軸の節のルート モデルを指定できる。
　　（イ）ジョイントの最後の軸の節の終端となるモデルを指定できる。（B 軸と E 軸を物理固定軸とした場合、最後の軸の節は、HP 軸の節となる。）

(C) 3DOF

(ア) HPB/E 軸の節のルート モデルを指定できる。

(イ) ジョイントの最後の軸の節の終端となるモデルを指定できる。(E 軸を物理固定軸とした場合、最後の軸の節は、HPB 軸の節となる。)

ルート モデルの指定に従って以下の項目を決定する。

- ルート モデルの指定がない軸は物理的に固定された軸とする。
- 各軸の節のローカル座標系はルート モデルのローカル座標系とする。
- 節の形状はルート モデルとその子孫(終端モデルまで)となる動的剛体を合体した形状とする。
- 節の物理性質値は合体した動的剛体の物理性質値を適切に合算した値とする。(平行軸の定理と慣性モーメント テンソルの回転を使用して算出する。)
- 節の運動条件はルート モデルに最も近い剛体の設定値とする。

(7) アンカーの設定

ルート指定と分離リグ指定の組み合わせに応じて固定先となるアンカーを指定することができる。

表 6-8 にルート指定と分離リグ指定の組み合わせと指定できるアンカーの関係を示す。

表 6-8 ルート指定と分離リグ指定の組み合わせと指定できるアンカーの関係

ルート指定	分離リグ指定	指定できるアンカー
×	×	指定不可。
○	×	リンク チェーンの先頭の軸の固定先となるアンカー(静的/動的剛体)を指定できる。
×	○	指定不可。
○	○	リンク チェーンの先頭の軸の固定先となるアンカー(静的/動的剛体)と、分離リグのルート ジョイントを固定するアンカー(モデル)を指定できる。

(8) 節の衝突判定用の形状の設定

ジョイント タイプに応じて節の衝突判定用の形状を設定することができる。

(A) 組み合わせヒンジ

H/P/B/E 軸の節の衝突判定用形状のルート モデル(空形状のモデルを除く。)を指定できる。

(B) カルダン+ヒンジ

HP/B/E 軸の節の衝突判定用形状のルート モデル(空形状のモデルを除く。)を指定できる。

(C) 3DOF

HPB/E 軸の節の衝突判定用形状のルート モデル(空形状のモデルを除く。)を指定できる。

衝突判定用の節の形状はルート モデルとその子孫であるモデルを合体した形状とし、そのローカル座標系は節の実構成のルート モデルのローカル座標系とする。

衝突判定用の節の形状に指定できるモデルは表 6-4 の条件に適合するモデルとする。

(9) リンク チェーン内部の衝突判定の有無設定

節単位でリンク チェーン内部の衝突判定の有無を設定できる。

(10) ジョイントの初期の位置と角度を記憶

現在のジョイントの位置と角度を初期の位置と角度として記憶することができる。

(11) ジョイントの位置と角度をリセット

ジョイントの位置と角度を初期の位置と角度に復帰することができる。

(12) ジョイントの角度変更

ジョイントの角度を指定の角度に変更することができる。その際、ルート側のリンクの角度を変更するか、先端側のリンクの角度を変更するかを指定できる。

(13) 軸の駆動設定（H/P/B 軸）

軸毎に以下の項目を設定できる。

(A) 回転タイプ

固定（論理的に固定された軸)、制限なし、角度制限の何れかを選択できる。角度制限を選択した場合は、最小角度と最大角度（単位はラジアン、表示単位は度）を指定できる。但し、ジョイントの実体が IK ジョイントの場合は、IK ジョイントの最小角度と最大角度に従う。

(B) 駆動タイプ

前項の回転タイプが固定以外の場合、アイドル、トルク、サーボの何れかを選択できる。但し、サーボは実験的実装とする。（リンク制御出力ブロックがサーボの機能を有する為。）

トルクを選択した場合は以下の項目を設定できる。

(ア) 最大トルク（単位 Nm（符号なし））

最大電流で動力源を駆動した場合に回転軸に発生する最大のトルク。

(イ) 最高角速度（単位 ラジアン/s、表示単位 度/s（符号なし））

最大電圧で動力源を駆動した場合に無負荷状態の回転軸が回転する最速の回転速度。

(ウ) 印加トルク（単位 Nm、符号は回転方向を表す）

最大電流制限下で動力源を駆動した場合に回転軸に発生する最大のトルク。

(エ) 印加角速度（単位 ラジアン/s、表示単位 度/s、符号は回転方向を表す）

印加電圧で動力源を駆動した場合に無負荷状態の回転軸が回転する最速の回転速度。

(オ) 回転軸の静的抵抗力（単位 Nm（符号なし））

停止状態の回転軸を回転させるために必要な最小のトルク。（回転方向の区別はなし。）

(カ) 回転軸の動的抵抗力（単位 Nm（符号なし））

回転状態の回転軸が摩擦によって損失するトルク。（回転方向の区別はなし。）

(キ) 短絡ブレーキの有無

印加トルクを 0 とした時に短絡ブレーキを使用するかどうかを決定する。

サーボを選択した場合は上記の項目に加えて以下の項目を設定できる。

(ク) サーボ目標角度（単位 ラジアン、表示単位 度）

回転軸の現在の角度とサーボ目標の角度から回転方向を決定する。（印加トルクと印加角速度の符号は無視する。）

(14) 軸の駆動設定（E 軸）

以下の項目を設定できる。

(A) 直動タイプ

固定（論理的に固定された軸)、長さ制限の何れかを選択できる。長さ制限を選択した場合は、最短と最長（単位は m）を指定できる。

(B) 駆動タイプ

前項の直動タイプが固定以外の場合、アイドル、フォース、サーボの何れかを選択できる。但し、サーボは実験的実装とする。（リンク制御出力ブロックがサーボの機能を有する為。）

フォースを選択した場合は以下の項目を設定できる。

(ア) 最大フォース（単位 N（符号なし））

最大電流で動力源を駆動した場合に直動軸に発生する最大のフォース。

(イ) 最高速度（単位 m/s（符号なし））

最大電圧で動力源を駆動した場合に無負荷状態の直動軸が移動する最速の速度。

(ウ) 印加フォース（単位 N、符号は移動方向を表す）

最大電流制限下で動力源を駆動した場合に直動軸に発生する最大のフォース。

(エ) 印加速度（単位 m/s、符号は移動方向を表す）

印加電圧で動力源を駆動した場合に無負荷状態の直動軸が移動する最速の速度。

(オ) 直動軸の静的抵抗力（単位 N（符号なし））

停止状態の直動軸を移動させるために必要な最小のフォース。（移動方向の区別はなし。）

(カ) 直動軸の動的抵抗力（単位 N（符号なし））

移動状態の直動軸が摩擦によって損失するフォース。（移動方向の区別はなし。）

(キ) 短絡ブレーキの有無

印加フォースを 0 とした時に短絡ブレーキを使用するかどうかを決定する。

サーボを選択した場合は上記の項目に加えて以下の項目を設定できる。

(ク) サーボ目標位置（単位 m）

直動軸の現在の位置とサーボ目標位置から移動方向を決定する。（印加フォースと印加速度の符号は無視する。）

(15) 軸の駆動状況のモニター（H/P/B 軸）

以下の項目の最新値をモニターできる。

(A) カレント トルク（単位 Nm）

回転軸に作用している駆動トルクの現在値

(B) カレント角速度（単位 ラジアン/s、表示単位 度/s）

回転軸の回転速度の現在値

上記項目のモニターを停止/再開することができる。

(16) 軸の駆動状況のモニター（E 軸）

以下の項目の最新値をモニターできる。

(A) カレント フォース（単位 N）

直動軸に作用しているフォースの現在値

(B) カレント速度（単位 m/s）

直動軸の移動速度の現在値

上記項目のモニターを停止/再開することができる。

(17) 物理性質の可視化（シミュレーション環境のみ）

以下の項目を表示することができる。

(A) 節の重心と慣性主軸

重心の描画サイズと描画色を指定できる。

慣性主軸の描画長は重心の描画サイズの 20 倍とする。

慣性主軸の表示色は CINEMA 4D の一般設定の軸の表示色に従う。

(B) 回転軸と回転範囲

表示するサイズ（単位はシステム設定に従う。）を指定できる。

(18) 物理演算誤差に関するパラメータ設定

以下の項目を設定できる。

(A) ERP（Error Reduction Parameter）

$$0 \leq \mathrm{ERP} = \frac{\text{タイム ステップ幅・バネ係数}}{\text{タイム ステップ幅・バネ係数＋減衰係数}} \leq 1.0$$

0 に近いと修正力が弱いが反発力はより正しく反映される。1 に近いと逆になる。

(B) STOP ERP（Error Reduction Parameter At Limit）

回転軸が可動範囲の限界にある時の ERP 値

(C) CFM（Constraint Force Mixing）

ジョイントの結合力を調整する係数

$$0 \leq \mathrm{CFM} = \frac{1}{\text{タイム ステップ幅・バネ係数＋減衰係数}}$$

0 は完全にハードなジョイント、0 より大きな値はソフトなジョイントとなる。

(D) STOP CFM（Constraint Force Mixing At Limit）

回転軸が可動範囲の限界にある時の CFM 値

(E) 破断閾値（単位 N）

ジョイントが破断するフォースの閾値

(19) デバッグ情報の表示（シミュレーション環境のみ）

表 6-9 に示す項目を表示できる。

表 6-9 H/P/B/E 軸のデバッグ情報

項目名	表示形式	単位
体積	指数形式(x.以下 15 桁 E±x)	m と cm
質量	同上	kg と g
重心	ローカル座標系で X/Y/Z 座標を 10 進数表記	システム設定に従う
慣性モーメント テンソル	3 行 3 列（指数形式：x.以下 15 桁 E±x）	$kg \cdot m^2$
主慣性モーメント	1 行 3 列（指数形式：x.以下 15 桁 E±x）	$kg \cdot m^2$
慣性モーメント L スケール	10 進数表記	－

6.4 物理ワールド管理ブロックの機能

(1) 物理演算エンジンの選択

物理演算に使用するエンジンを選択できる。選択できる物理演算エンジンを以下に示す。

(A) Bullet（オープンソースの物理演算エンジン）

(B) PhysX（NVIDIA が開発供給しているオープンソースの物理演算エンジン）

(C) RoboBio-X（オリジナル物理演算エンジン）

(2) 物理ワールドの状態の管理

物理ワールドの状態を管理して外部機能に提示することができる。

物理ワールドの状態の種類を以下に示す。

(A) ロード（詳細は実装の指針の章に記載する。）

(B) コピー（詳細は実装の指針の章に記載する。）

(C) 正常

物理ワールド内の物理演算要素に異常がない状態。

(D) ジョイント構成異常検知

物理ワールド内に物理演算できないリンク チェーンが存在する状態。

(3) 時間スケールの設定

物理シミュレーションのステップ時間のスケール値を 0.0 から 100.0 の範囲で設定できる。（0.0 を設定した場合、物理演算エンジン内の時間が停止する。）

(4) 重力の設定

物理ワールド内の全ての物理演算要素に働く重力の加速度（単位はm/s^2）を設定することができる。

(5) 制御間隔（シミュレーション環境ではフレーム時間）あたりの物理演算ステップ数の設定

制御間隔あたりの物理演算ステップ数を 1 から 1000 の範囲で設定できる。（制御間隔は動力源の制御を更新する間隔の時間）

この設定値を大きくした場合のメリット

- CCD（Continuous Collision Detection）未実施の場合、剛体のすり抜けが発生しにくくなる。
- ジョイントの拘束がハードになる。

この設定値を大きくした場合のデメリット

シミュレーションを進めるための物理演算回数が増える為、CPU 負荷が大きくなる。

(6) ソルバーの最大演算回数

1 ステップ当たりの最大物理演算回数を 1 から 10000 の範囲で設定できる。

この設定値を大きくした場合のメリット

- 拘束による位置と姿勢の修正が完了する前に物理演算が終了してしまうことでシミュレーションが不安定になることを防ぐことができる。
- ジョイントの拘束がハードになる。

この設定値を大きくした場合のデメリット

シミュレーションを進めるための物理演算回数が増える為、CPU 負荷が大きくなる。

(7) ジョイント初期化の選択

物理ワールドの初期化時に物理ワールド内のジョイントを初期化するかどうかを選択できる。

(8) シミュレーション開始時のリセットの選択

シミュレーション開始時に物理ワールド内の全ての剛体とジョイントの位置と角度を初期状態にリセットするかどうかを選択できる。

(9) 物理ワールドの構築

以下の機能を有する。

(A) 思考内 4 D 空間又は CINEMA 4D 内のモデルから物理演算対象のモデルを抽出して物理演算エンジン内に対応する要素を生成する。

(ア) 物理演算エンジンの選択が Bullet の場合

表 6-10 に剛体の抽出条件を示す。

表 6-10 剛体の抽出条件

剛体	抽出条件
静的剛体	以下の条件を全て満たすモデル ・剛体管理ブロックの機能によって剛体の実体に指定されている ・質量＝0 ・ジョイントの節に属していない
動的剛体	以下の条件を全て満たすモデル ・剛体管理ブロックの機能によって剛体の実体に指定されている ・質量＞0 ・ジョイントの節に属していない
節(動的剛体)	以下の条件を全て満たすモデル ・剛体管理ブロックの機能によって剛体の実体に指定されている ・質量＞0 ・ジョイントの節に属している

表 6-11 に剛体と物理演算エンジン内の要素の関係を示す。

表 6-11 剛体と物理演算エンジン内の要素の関係

剛体	物理演算エンジンの要素の型
静的剛体	btRigidBody
動的剛体	btRigidBody
節(動的剛体)	btRigidBody（節に属するモデルを全て結合した物理性質となる。）

表 6-12 に剛体の形状と物理演算エンジン内の要素の関係を示す。

表 6-12 剛体の形状と物理演算エンジン内の要素の関係

モデル (Ⓟ＝プリミティブ)	モデルの属性					物理演算エンジン内の要素の型
	キャップ	3/2D	スライス	フィレット	ポリゴン化	
Ⓟ立方体(直方体)	―	3	―	×	×	btBoxShape
	―	3	―	×	○	btGImpactMeshShape
	―	3	―	○	―	btGImpactMeshShape
	―	2	―	・	―	btBvhTriangleMeshShape
Ⓟ球(楕円体)	―	3	―	―	×	btMultiSphereShape
	―	3	―	―	○	btGImpactMeshShape
	―	2	―	―	―	btBvhTriangleMeshShape
Ⓟ半球(半楕円体)	―	2!	―	―	―	btBvhTriangleMeshShape
Ⓟカプセル (上下端均一/上下端不均一)	―	3	×	―	×	btMultiSphereShape
	―	3	×	―	○	btGImpactMeshShape
	―	3	○	―	―	btGImpactMeshShape
	―	2	・	―	―	btBvhTriangleMeshShape
Ⓟ円柱(楕円柱)	○	3	×	×	×	btCylinderShape
	○	3	×	×	○	btGImpactMeshShape
	○	3	×	○	―	〃
	○	3	○	×	―	〃
	○	3	○	○	―	〃
	○	2	・	・	―	btBvhTriangleMeshShape
	×	2!	・	―	―	btBvhTriangleMeshShape
Ⓟチューブ	―	3	×	×	×	btGImpactMeshShape
	―	3	×	×	○	〃
	―	3	×	○	―	〃
	―	3	○	×	―	〃
	―	3	○	○	―	〃
	―	2	・	・	―	btBvhTriangleMeshShape
Ⓟ円錐(楕円錐)	○	3	×	×	×	btConeShape
	○	3	×	×	○	btGImpactMeshShape
	○	3	×	○	―	〃
	○	3	○	×	―	〃
	○	3	○	○	―	〃
	○	2	・	・	―	btBvhTriangleMeshShape
	×	2!	・	・	―	btBvhTriangleMeshShape
Ⓟ円錐台(楕円錐台)	○	3	×	×	×	btGImpactMeshShape
	○	3	×	×	○	〃
	○	3	×	○	―	〃
	○	3	○	×	―	〃
	○	3	○	○	―	〃
	○	2	・	・	―	btBvhTriangleMeshShape
	×	2!	・	・	―	btBvhTriangleMeshShape
Ⓟ地形(球状)	―	3	―	―	―	btGImpactMeshShape
	―	2	―	―	―	btBvhTriangleMeshShape
Ⓟ地形	―	2!	―	―	―	btBvhTriangleMeshShape
Ⓟディスク(楕円)	―	2!	・	―	―	btBvhTriangleMeshShape
Ⓟ平面(長方形)	―	2!	―	―	―	btBvhTriangleMeshShape
Ⓟシンプル ポリゴン (三角形/四角形)	―	2!	―	―	―	btBvhTriangleMeshShape
ポリゴン形状	―	3	―	―	―	btGImpactMeshShape
	―	2	―	―	―	btBvhTriangleMeshShape

モデルの属性の記号：「ー」は選択不可、「○」は有効、「×」は無効、「・」は有効/無効の両方、

「2!」は強制的に 2D が設定されることを意味する

表 6-13 に節に属する剛体の形状と物理演算エンジン内の要素の関係を示す。

表 6-13 節に属する剛体の形状と物理演算エンジン内の要素の関係

モデル	物理演算エンジン内の要素の型
同一の節に属する表 6-12 の複数のモデル	btCompoundShape（複数のモデルを結合する）

ジョイントの抽出条件は、モデルがジョイント管理ブロックの機能によってジョイントの実体に指定されていることとする。

表 6-14 にジョイントと物理演算エンジン内の要素の関係を示す。

表 6-14 ジョイントと物理演算エンジン内の要素の関係

ショイント		物理演算エンジン内の要素の型
組合せヒンジ ジョイント	H 軸	btGeneric6DofSpring2Constraint（可動軸=Z 軸） 生成時、軸の向きを変更する。(Y -> Z, X ->Y, Z -> X)
	P 軸	btGeneric6DofSpring2Constraint（可動軸=Z 軸） 生成時、軸の向きを変更する。(X -> Z, -Y -> Y, Z -> X)
	B 軸	btGeneric6DofSpring2Constraint（可動軸=Z 軸）
	E 軸	btGeneric6DofSpring2Constraint（可動軸=X 軸） 生成時、軸の向きを変更する。(-X -> Z, Y -> Y, Z -> X)
カルダン＋ヒンジ ジョイント	H/P 軸	btGeneric6DofSpring2Constraint（可動軸=Z/X 軸） 生成時、軸の向きを変更する。(Y -> Z, X -> X, -Z -> Y)
	B 軸	btGeneric6DofSpring2Constraint（可動軸=Z 軸）
	E 軸	btGeneric6DofSpring2Constraint（可動軸=X 軸） 生成時、軸の向きを変更する。(-X -> Z, Y -> Y, Z -> X)
3DOF	H/P/B 軸	btGeneric6DofSpring2Constraint（可動軸=Z/Y/X 軸） 生成時、軸の向きを変更する。(Y -> Z, X -> Y, Z -> X)
	E 軸	btGeneric6DofSpring2Constraint（可動軸=X 軸） 生成時、軸の向きを変更する。(-X -> Z, Y -> Y, Z -> X)

（イ）物理演算エンジンの選択が PhysX の場合

剛体の抽出条件は、Bullet の場合と同じとする。但し、リンク チェーンを構成する節(動的剛体)は物理演算対象から除外する。

表 6-15 に剛体と物理演算エンジン内の要素の関係を示す。

表 6-15 剛体と物理演算エンジン内の要素の関係

剛体	物理演算エンジン内の要素の型
静的剛体	PxRigidStatic
動的剛体	PxRigidDynamic

表 6-16 に剛体の形状と物理演算エンジン内の要素の関係を示す。

表 6-16 剛体の形状と物理演算エンジン内の要素の関係

モデル (Ⓟ＝プリミティブ)	モデルの属性					物理演算エンジンのオブジェクトの型
	キャップ	3/2D	スライス	フィレット	ポリゴン化	
Ⓟ立方体(直方体) スケール設定=(1,1,1)	ー	3	ー	×	×	PxBoxGeometry
	ー	3	ー	×	○	PxTriangleMeshGeometry
	ー	3	ー	○	ー	PxTriangleMeshGeometry
	ー	2	ー	・	ー	PxTriangleMeshGeometry
Ⓟ球 スケール設定=(1,1,1)	ー	3	ー	ー	×	PxSphereGeometry
	ー	3	ー	ー	○	PxTriangleMeshGeometry
	ー	2	ー	ー	ー	PxTriangleMeshGeometry
上記以外のモデルと属性組み合わせ						PxTriangleMeshGeometry

モデルの属性の記号：「ー」は選択不可、「○」は有効、「×」は無効、「・」は有効/無効の両方、
「2!」は強制的に 2D が設定されることを意味する

（ウ）物理演算エンジンの選択が RoboBio-X の場合

剛体の抽出条件は、Bullet の場合と同じとする。

表 6-17 に剛体と物理演算エンジン内の要素の関係を示す。

表 6-17 剛体と物理演算エンジン内の要素の関係

剛体	物理演算エンジン内の要素の型
静的剛体	rbxRigidBody
動的剛体	rbxRigidBody
節(動的剛体)	rbxRigidBody（節に属するモデルを全て結合した物理性質となる。）

表 6-18 に剛体の形状と物理演算エンジン内の要素の関係を示す。

表 6-18 剛体の形状と物理演算エンジン内の要素の関係

モデル (Ⓟ＝プリミティブ)	モデルの属性					物理演算エンジン内の要素の型
	キャップ	3/2D	スライス	フィレット	ポリゴン化	
Ⓟ立方体(直方体)	ー	3	ー	×	×	rbxBoxShape
Ⓟ球(楕円体)	ー	3	ー	ー	×	rbxSphereShape
Ⓟカプセル(上下端均一)	ー	3	×	ー	×	rbxMultiSphereShape
Ⓟカプセル(上下端不均一)	ー	3	×	ー	×	rbxMultiSphereShape
Ⓟ円柱(楕円柱)	○	3	×	×	×	rbxCylinderShape
Ⓟチューブ	ー	3	×	×	×	rbxTubeShape
Ⓟ円錐(楕円錐)	○	3	×	×	×	rbxConeShape
Ⓟ円錐台(楕円錐台)	○	3	×	×	×	rbxTruncatedConeShape
Ⓟディスク(楕円)	ー	2!	・	ー	ー	rbxDiskShape
Ⓟ平面(長方形)	ー	2!	ー	ー	ー	rbxRectangleShape
Ⓟシンプル ポリゴン(四角形)	ー	2!	ー	ー	ー	rbxRectangleShape
Ⓟシンプル ポリゴン(三角形)	ー	2!	ー	ー	ー	rbxTriangleShape
上記以外のモデルと属性組み合わせ						rbxNullShape（衝突判定対象外）

モデルの属性の記号：「ー」は選択不可、「○」は有効、「×」は無効、「・」は有効/無効の両方、
「2!」は強制的に 2D が設定されることを意味する

表 6-19 に節に属する剛体の形状と物理演算エンジン内の要素の関係を示す。

表 6-19 節に属する剛体の形状と物理演算エンジン内の要素の関係

モデル	物理演算エンジン内の要素の型
同一の節に属する表 6-18 の複数のモデル	rbxCompoundShape（複数のモデルを結合する）

ジョイントの抽出条件は、モデルがジョイント管理ブロックの機能によってジョイントの実体に指定されていることとする。

表 6-20 にジョイントと物理演算エンジン内の要素の関係を示す。

表 6-20 ジョイントと物理演算エンジン内の要素の関係

ショイント		物理演算エンジンの要素の型
組合せヒンジ ジョイント	H 軸	rbxRotConstraint（可動軸=Z 軸） 生成時、軸の向きを変更する。(Y -> Z, X ->Y, Z -> X)
	P 軸	rbxRotConstraint（可動軸=Z 軸） 生成時、軸の向きを変更する。(X -> Z, -Y -> Y, Z -> X)
	B 軸	rbxRotConstraint（可動軸=Z 軸）
	E 軸	rbxLinConstraint（可動軸=X 軸） 生成時、軸の向きを変更する。(-X -> Z, Y -> Y, Z -> X)
カルダン＋ヒンジ ジョイント		rbxNullConstraint（拘束に関する物理演算を実施しない。）
3DOF		rbxNullConstraint（拘束に関する物理演算を実施しない。）

(B) 物理演算対象になるモデルのリストと、モデルと物理演算エンジン内の要素の対応リストを生成する。

(C) 物理演算対象になるモデルの位置と角度を初期値として記憶する。（剛体管理ブロックとジョイント管理ブロックの機能を利用して記憶する。）

(10) 物理ワールドの破棄

物理ワールドで消費されているリソースを全て開放する。

(11) 物理ワールドの再構築（シミュレーション環境のみ）

物理ワールド構築後の CINEMA 4D 内のモデルの構成の変化に応じて物理ワールドを再構築する。但し、ジョイントが消失している場合は、物理ワールドの状態を「ジョイント構成異常検知」に遷移させる。（物理ワールドの初期化で復旧する。）

(12) 物理ワールドのリセット

物理演算対象のモデルの位置と角度を初期化時に記憶した位置と角度に設定する。（剛体管理ブロックとジョイント管理ブロックの機能を利用して設定する。）

(13) 物理演算要素の表示（シミュレーション環境のみ）

物理演算要素の動的剛体、静的剛体、ジョイントをリスト表示することができる。

リストの表示項目はモデルの名称とモデルの種類を識別できるアイコンとする。

(14) 物理演算エンジン内の物理演算要素の可視化（シミュレーション環境のみ）

物理演算エンジンの選択が Bullet 又は RoboBio-X の場合、物理演算エンジン内の物理演算要素の状態を確認できる。（PhysX は実験的実装である為、対象外とする。）

可視化できる項目を以下に示す。

(A) 剛体のワイヤー フレーム
(B) 剛体の軸平行境界ボックス（AABB：Axis-Aligned Bounding Box）
(C) 接点
(D) 拘束の軸
(E) 拘束の範囲
(F) 剛体（メッシュ形状のみ）のポリゴンの法線

(15) 物理シミュレーション

以下の機能を有する。

(A) 物理演算エンジン内の時間を一定時間進めることで、物理演算要素の一定時間後の位置と角度を思考内 4 D 空間又は CINEMA 4D 内のモデルに反映する。
(B) 外部機能に対して物理演算対象の剛体の線形速度と回転速度を提示する。
(C) 外部機能に対して物理演算対象のジョイントの各節のカレント トルクと角速度を提示する。

(16) デバッグ情報の表示（シミュレーション環境のみ）

物理ワールドの初期化、及び物理シミュレーションにおいて検知した警告の履歴を表示することができる。

検知可能な警告の種類を以下に示す。

(A) ジョイントの座標変換の固定に関する警告（可動しない軸の角度が 0.0 でない。）
(B) ジョイントの参照に関する警告（分離リグのジョイントの位置参照モデルが存在しない。）

表 6-21 にジョイントの節へのモデルの割り当て状態と位置参照モデルの関係を示す。

表 6-21 ジョイントの節へのモデルの割り当て状態と位置参照モデルの関係

ショイントの節の割り当て状態			位置参照モデル
H 軸	P 軸	B 軸	
○	×	×	H 軸のルート モデル
×	○	×	P 軸のルート モデル
○	○	×	P 軸のルート モデル
×	○	○	P 軸のルート モデル
○	○	○	P 軸のルート モデル
×	×	○	B 軸のルート モデル
○	×	○	B 軸のルート モデル

警告の履歴リストの項目は、検知した日時、モデル名、警告種別とする。

7. 実装の指針

(1) 順動力学演算ブロックの詳細ブロック

順動力学演算ブロックは四つの詳細ブロックで構成する。各詳細ブロックは実行環境に依存する層と依存しない層から構成し、それぞれを C++言語のクラスで実装する。

表 7-1 に詳細ブロックと C++言語のクラスとその実体であるオブジェクトの名称を示す。

表 7-1 詳細ブロックと C++言語のクラスとその実体であるオブジェクトの名称

詳細ブロック	C++言語のクラス	オブジェクト
スケール管理ブロック	動作環境別基本クラス(空間管理の一部) スケール管理コア クラス(環境非依存) └ スケール管理クラス(環境依存)	スケール管理オブジェクト
剛体管理ブロック	動作環境別基本クラス(空間管理の一部) 剛体管理コア クラス(環境非依存) └ 剛体管理クラス(環境依存)	剛体管理オブジェクト
ジョイント管理ブロック	動作環境別基本クラス(空間管理の一部) ジョイント管理コア クラス(環境非依存) └ ジョイント管理クラス(環境依存)	ジョイント管理オブジェクト
物理ワールド管理ブロック	環境非依存	環境非依存
	物理ワールド管理コア クラス ├ Bullet 物理ワールド管理コア クラス ├ RoboBio-X 物理ワールド管理コア クラス └ PhysX 物理ワールド管理コア クラス	物理ワールド管理コア オブジェクト
	環境依存	環境依存
	動作環境別基本クラス(空間管理の一部) └ 物理ワールド管理クラス(環境依存)	物理ワールド管理オブジェクト

(A) スケール/剛体/ジョイント管理ブロックのクラスと管理オブジェクトの関連

本関連の概要を図 7-1 に示す。

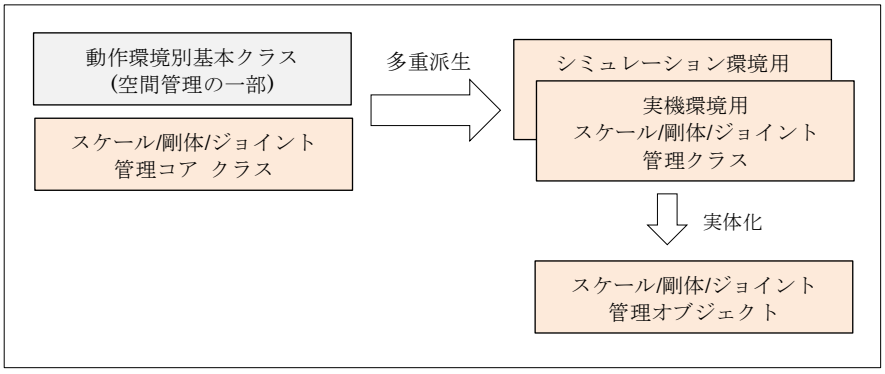

図 7-1 スケール/剛体/ジョイント管理ブロックのクラスとオブジェクトの関連

(B) 物理ワールド管理ブロックのクラスと管理オブジェクトの関連

本関連の概要を図 7-2 に示す。

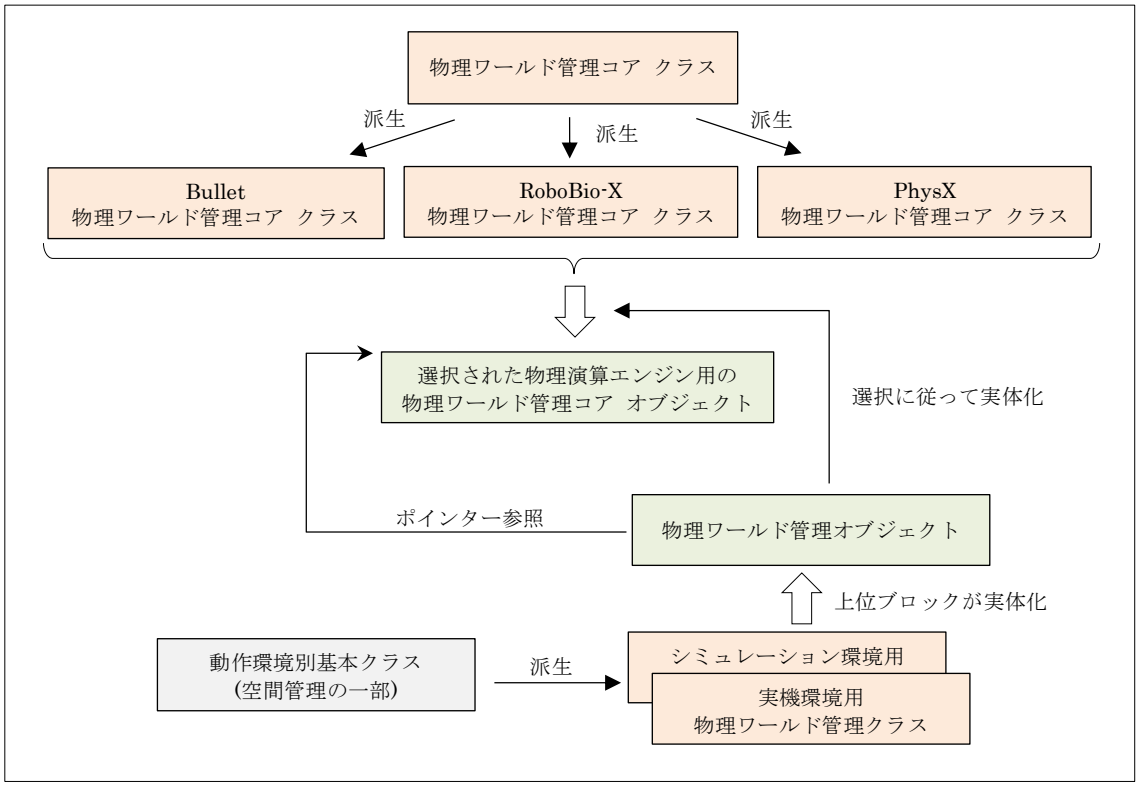

図 7-2 物理ワールド管理ブロックのクラスとオブジェクトの関連

(2) 各管理オブジェクトと管理対象のモデルの関連

(A) スケール管理オブジェクトと管理対象（スケール変更の対象）となるモデルの関連

本関連の概要を図 7-3 に示す。

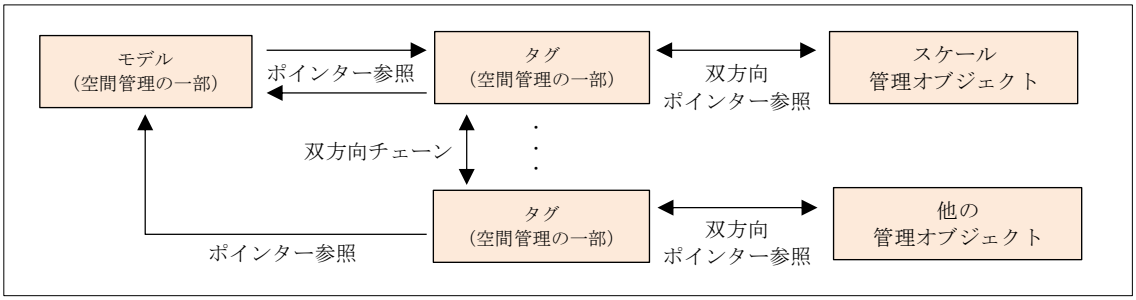

図 7-3 スケール管理オブジェクトと管理対象のモデルの関連

以降、この形式の関連をモデルへの管理オブジェクトの間接アタッチと記す。

(B) 剛体管理オブジェクトに関する関連

(ア) 管理対象となるモデル（剛体の実体）との関連

本関連は(A)項と同じ形式の関連とする。（但し、スケール管理を剛体管理に読み替えること。）

(イ) ジョイント管理オブジェクトがアタッチされたモデルとの関連（管理対象のモデルが節の一部となる場合）

本関連の概要を図 7-4 に示す。

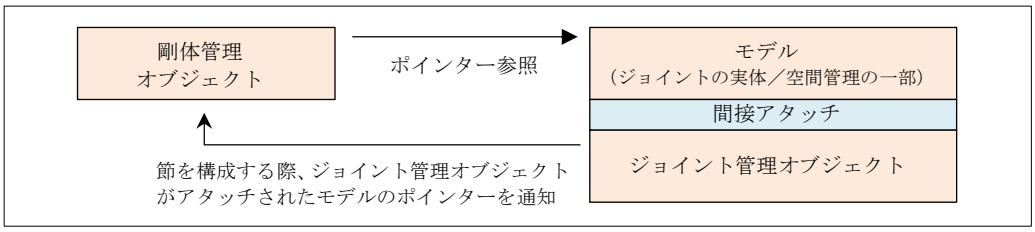

図 7-4 剛体管理オブジェクトとジョイント管理オブジェクトがアタッチされたモデルとの関連

(ウ) スケールド モデル データ ファクトリに関する関連

本関連の概要を図 7-5 に示す。

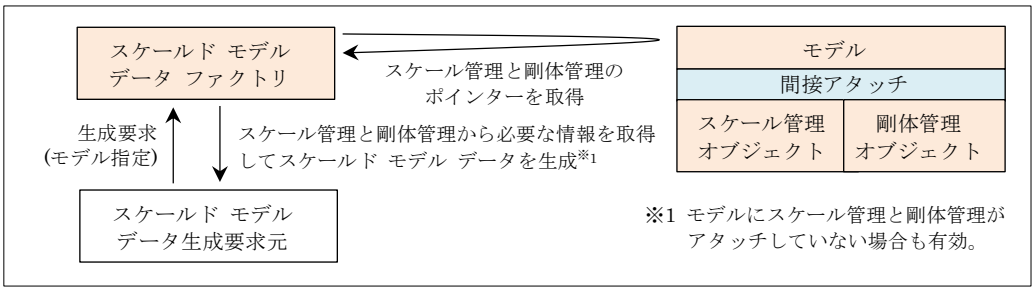

図 7-5 スケールド モデル データ ファクトリに関する関連

(C) ジョイント管理オブジェクトに関する関連

(ア) 管理対象となるモデル（ジョイントの実体）との関連

本関連は(A)項と同じ形式の関連とする。（但し、スケール管理をジョイント管理に読み替えること。）

ジョイント管理オブジェクトは以下に示すモデルの何れかにアタッチされる。

- リンク チェーンに挿入されたモデル
- リンク チェーンから分離されたモデル
- リンク チェーンから分離された IK チェーン内の IK ジョイント モデル

(イ) 各節の実形状のルート モデルとの関連

本関連の概要を図 7-6 に示す。

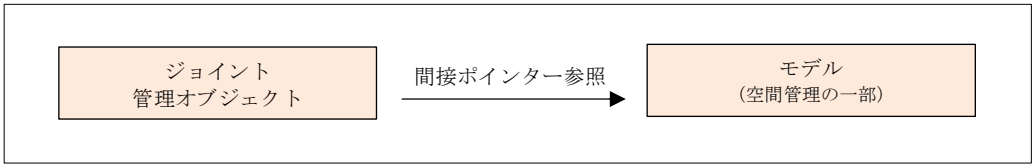

図 7-6 ジョイント管理オブジェクトと各節の実形状のルートとなるモデルとの関連

間接ポインターはモデルが存在する空間を指定したポインターとする。

モデルが存在する空間については(3)の項を参照のこと。

(ウ) 各節の衝突判定用の形状のルート モデルとの関連

本関連は前項と同じ形式とする。

(エ) 各節の実形状を構成するモデルとの関連

本関連の形式を図 7-7 に示す。

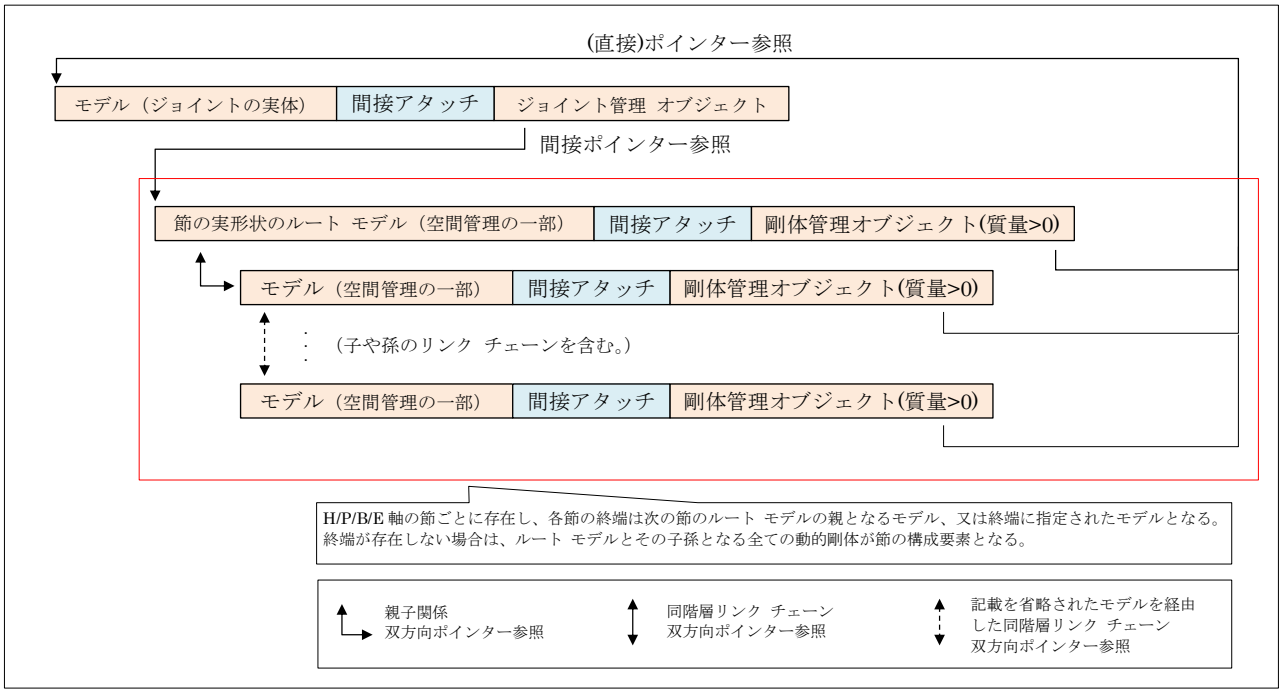

図 7-7 物理ワールド管理オブジェクトと実形状を構成するモデルとの関連

(オ) 各節の衝突判定用の形状を構成するモデルとの関連

本関連の形式を図 7-8 に示す。

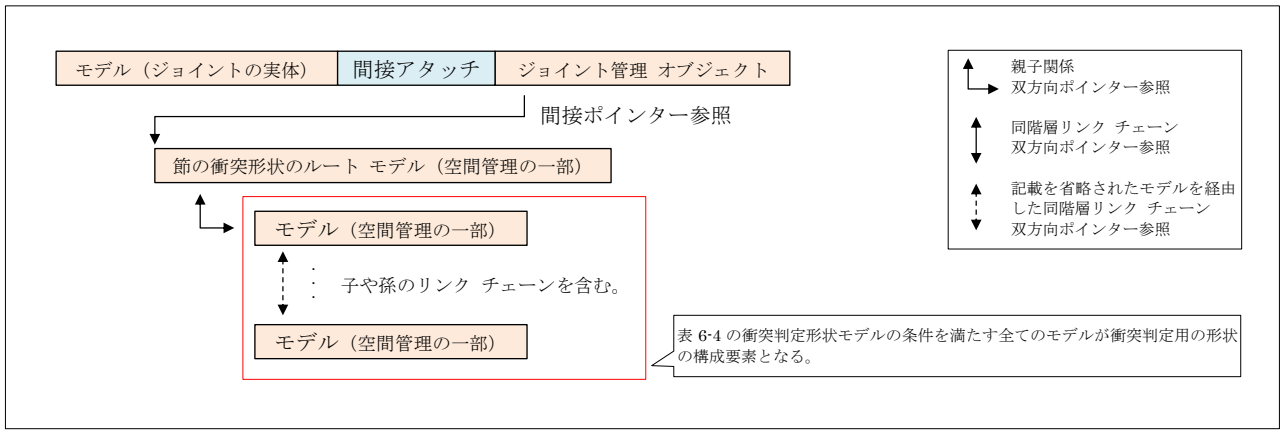

図 7-8 物理ワールド管理オブジェクトと実形状を構成するモデルとの関連

(D) 物理ワールド管理オブジェクトに関する関連

先ず、本項の記載で使用する語句を定義する。

- 物理ワールド：物理ワールド管理オブジェクトが管理する物理ワールド
- 物理エンジン ワールド：物理演算エンジンが管理する物理ワールド

(ア) 物理ワールドのルートとなるモデルとの関連

本関連の概要を図 7-9 に示す。

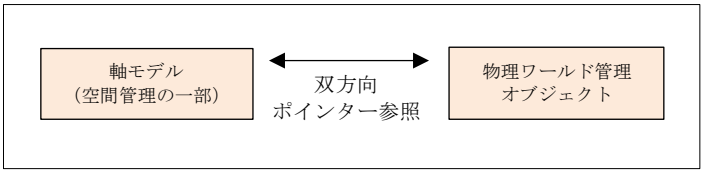

図 7-9 物理ワールド管理オブジェクトと物理ワールドの
ルートとなるモデルとの関連

以降、この形式の関連をモデルへの管理オブジェクトの直接アタッチと記す。

(イ) 物理演算エンジンとの関連

本関連の形式は図 7-2 を参照のこと。

(ウ) 物理ワールドの静的剛体との関連

本関連の形式を図 7-10 に示す。

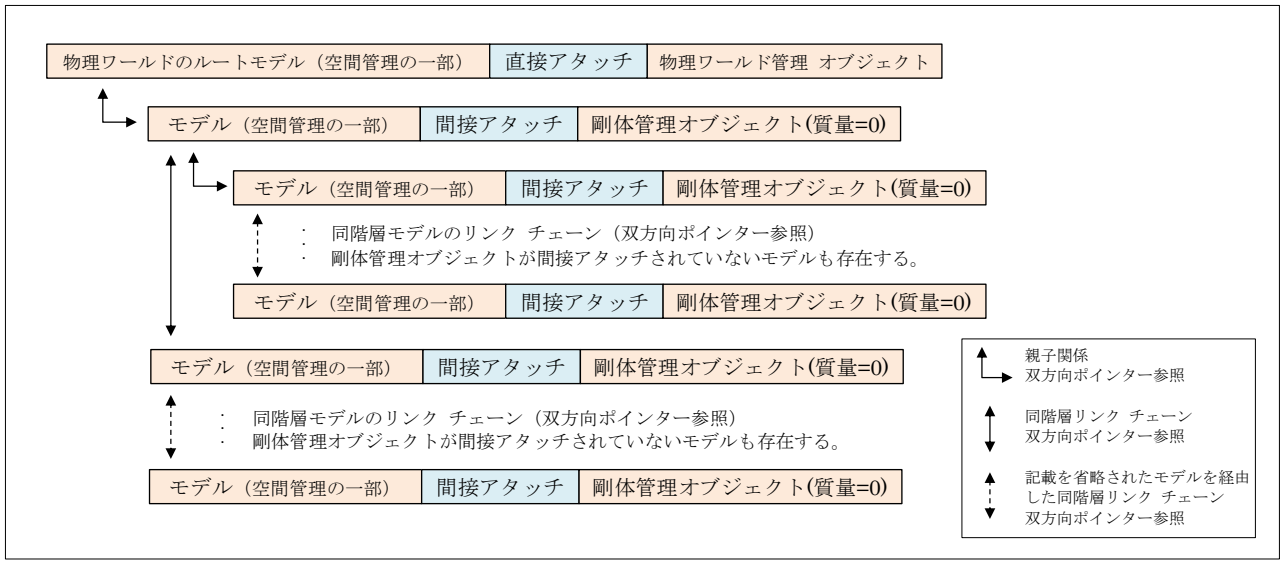

図 7-10 物理ワールド管理オブジェクトと静的剛体との関連

(エ) 物理ワールドの動的剛体との関連

本関連の形式は(ウ)の項と同じとする。但し、図 7-10 中の「剛体管理オブジェクト(質量=0)」を「剛体管理オブジェクト(質量＞0)」に読み替えること。

(オ) 物理ワールドのジョイントとの関連

本関連の形式は(ウ)の項と同じとする。但し、図 7-10 中の「剛体管理オブジェクト(質量=0)」を「ジョイント管理オブジェクト」に読み替えること。

（カ）物理ワールドの動的剛体と物理エンジン ワールドの動的剛体の関連

本関連の形式を図 7-11 に示す。

図 7-11 物理ワールドの動的剛体と物理エンジン ワールドの動的剛体の関連

(キ) 物理ワールドのジョイントと物理エンジン ワールドのジョイントの関連

本関連の形式を図 7-12 に示す。

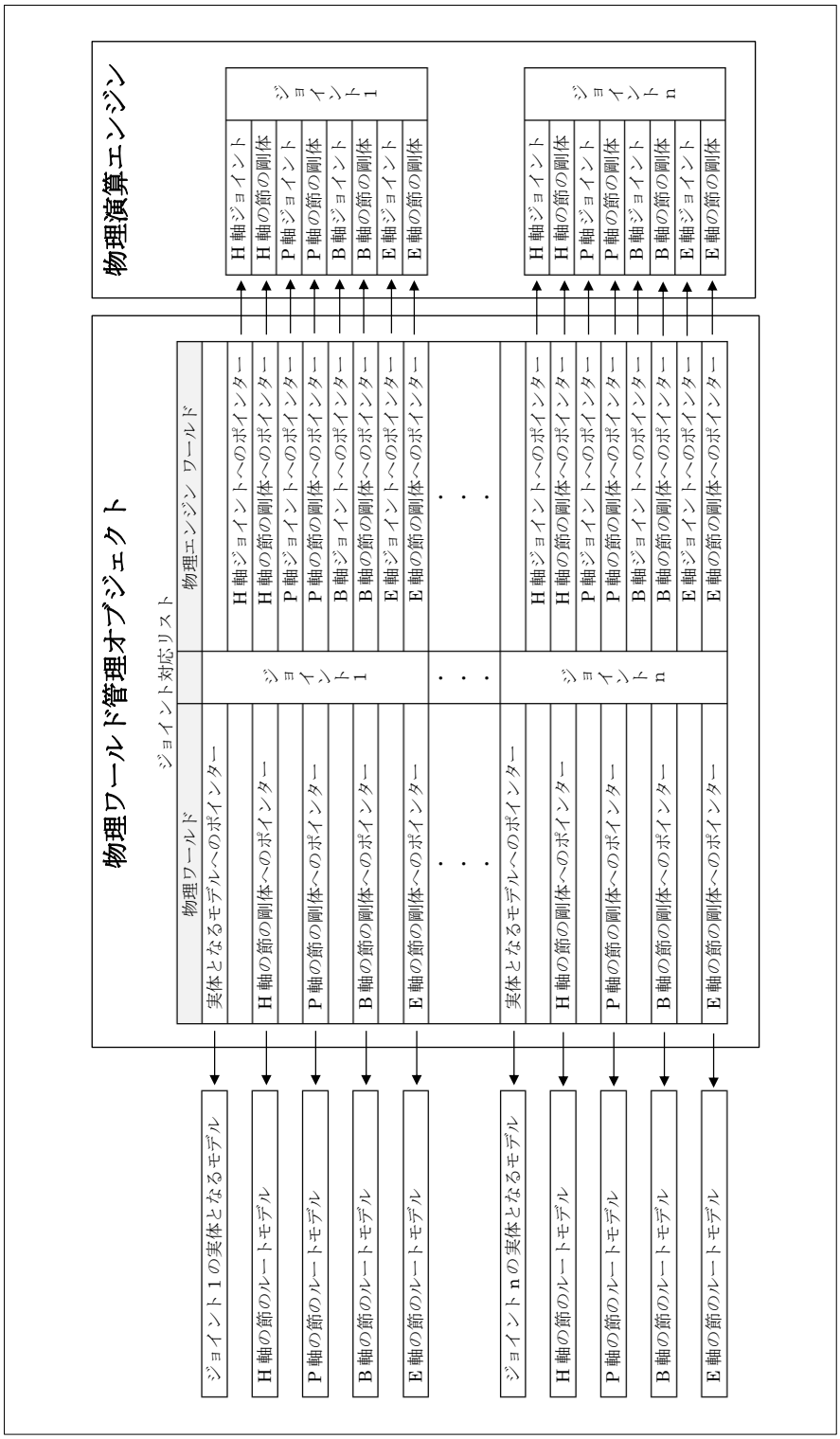

図 7-12 物理ワールド内のジョイントと物理演算エンジン内のジョイントの関連

(3) 実機環境の 4D 空間とシミュレーション環境(CINEMA 4D)のドキュメント空間）の多元化

順動力学演算ブロックの各詳細ブロックのオブジェクトと物理演算対象のモデルは、複数の空間に複製されることを前提に動作するように実装する。これは実機環境において、条件の異なる物理シミュレーションの並行実行を可能することが目的であるが、シミュレーション環境においても編集用の空間とレンダリング用の空間が異なるため同じ対応が必要となる。但し、複製が発生するタイミングは物理シミュレーション未実施時のみとする。

(A) オブジェクトの複製で生じる問題

図 7-13 にオブジェクトの複製で生じる問題を示す。

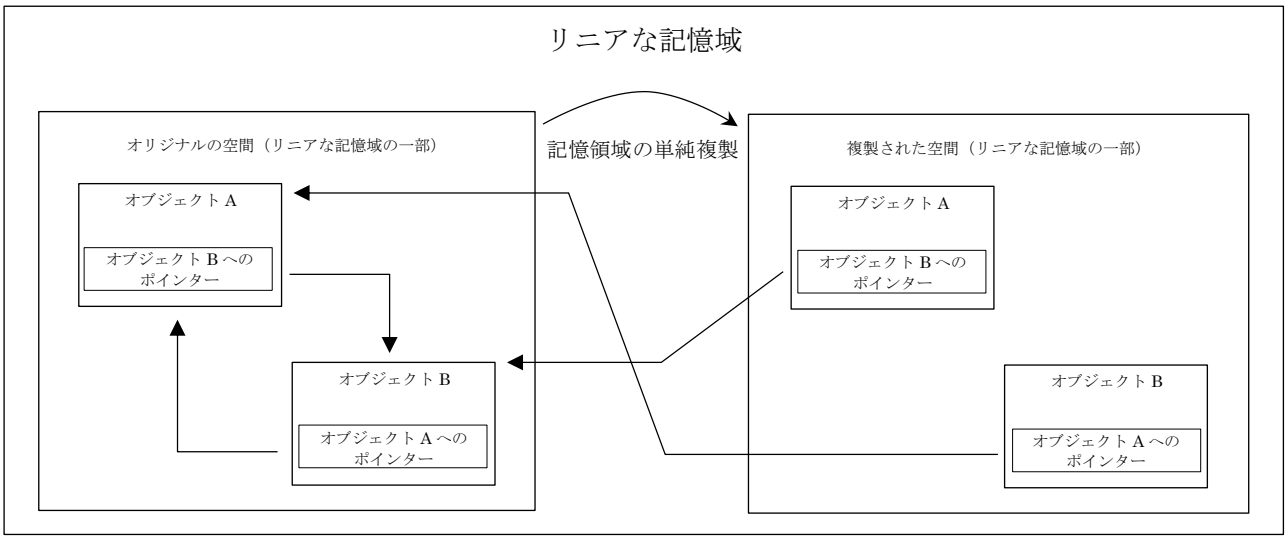

図 7-13 各詳細ブロックのオブジェクトのコピーで生じる問題

(B) オブジェクトの複製で生じる問題の対応策

- 管理オブジェクトの複製は専用の複製関数で実施されるようにする。
- 管理オブジェクトの複製時にポインターが無効になったことを記憶する。（状態を管理する。）
- 管理オブジェクトの実行イベントが発生した際、ポインターが無効な状態であれば、初期化処理でポインターを有効化してから処理を行う。
- 各管理オブジェクトの実行優先順位は以下の通りとする。
  優先度低 物理ワールド管理<ジョイント管理<剛体管理<スケール管理<3D モデル 優先度高

(4) ネイティブ座標系と物理演算エンジン（RoboBio-X 物理演算エンジンを除く）の座標系の差異の解決

以下に座標系の差異を示す。

- ネイティブ座標系は左手系（行ベクトル演算）
- Bullet 物理演算エンジンは右手系（列ベクトル演算）
- PhysX 物理演算エンジンは右手系（行ベクトル演算）

上記の差異を解決するために以下の処理を実装する。

(A) Bullet 物理演算エンジン用の処理（ネイティブ⇔Bullet 間双方向の変換処理）

- ベクトル型変換
- 位置変換
- Z 軸ベクトル変換
- XYZ 回転角変換
- HPB 回転角変換
- 3×3 行列変換（XYZ 軸ベクトル変換）

(B) PhysX 物理演算エンジン用の処理（ネイティブ⇔PhysX 間双方向の変換処理）

前項の処理に相当する処理とする。

(5) 詳細ブロック共通データ

以下のデータは詳細ブロックで共通認識できるデータとする。

- 物理演算エンジン種別
- システム単位
- 単位変換
- 力学サポート
- スケールド モデル
- 並進性質
- 並進･回転性質
- 運動条件
- 物理演算不活性化
- 剛体モニター
- 動力学
- 剛体
- ジョイント種別
- 抵抗トルク
- 抵抗フォース
- トルクと角速度
- フォースと速度
- 回転
- 直動
- 節
- 回転/直動モニター更新指定
- ジョイント
- インデックスド モデル

次章以降に詳細ブロック共通データの構成と、詳細ブロックに対応するクラスの実装の指針を示す。

7.1 実装の指針の記載に関する規約

実装の指針の記載に関する規約と、使用する記号及び略語を以下に示す。

データのアクセス保護レベルに関する文言の定義

- 外部データ：アクセス保護のないデータ
- 保護データ：当該データと当該データを継承するデータの付帯関数のみアクセス可能なデータ
- 内部データ：当該データに付帯する関数のみアクセス可能なデータ

データに付帯する関数の実行保護レベルに関する文言の定義

- 外部関数：実行保護のない関数
- 保護関数：当該データと当該データを継承するデータの付帯関数からのみ実行可能な関数
- 内部関数：当該データに付帯する関数からのみ実行可能な関数

ブロック（データと付帯する関数の組み合わせ）の概念の名称に関する定義

- データの全項目が外部データの場合、名称の最後に「構造」又は「データ」を付加する。
- データの全項目が外部データでない場合、名称の最後に「クラス」を付加する。
- ブロックの名称の最後が「クラス」の場合、データの全項目は、特に指定がない限り内部データとする。

データの構造に関する記載で使用する記号を表 7-2 に示す。

表 7-2 データの構造に関する記載で使用する記号

記号	意味
＜－＞	データの型で定義される初期値
＜・＞	コンストラクタの入力で定義される初期値
Ⓔ	外部データ（public）
Ⓟ	保護データ（protected）
Ⓘ	内部データ（private）

データの構造と付帯する機能に関する記載で使用する略語を表 7-3 に示す。

表 7-3 データの構造と付帯する機能に関する記載で使用する略語

略語	意味
M	マトリクス
Ml	ローカル座標系変換マトリクス
Mg	グローバル座標系変換マトリクス
L 座標／L 原点	ローカル座標系／ローカル座標系の原点
G 座標／G 原点	グローバル座標系／グローバル座標系の原点
ＳＣモデルD	スケールド モデル データ
SMDF	スケールド モデル データ ファクトリ

データに付帯する機能に関する記載で使用する記法を表 7-4 に示す。

表 7-4 データに付帯する機能に関する記載で使用する記法

<table>
<tr><th>記法</th><th>説明／例</th></tr>
<tr><td>I.項目名又は項目番号</td><td>当該機能の入力項目</td></tr>
<tr><td>#.項目名又は項目番号</td><td>当該データの項目</td></tr>
<tr><td>UI</td><td>UI 項目の値を格納したコンテナ</td></tr>
<tr><td>UI.項目名</td><td>UI 項目</td></tr>
<tr><td>~.*</td><td>当該データの全項目</td></tr>
<tr><td>~.$</td><td>当該データの特定の項目（対する辺の項目に対応する項目）</td></tr>
<tr><td>~.[A,B,･･･]／~.[A~B]</td><td>当該データの A 項目と B 項目／当該データの A 項目から B 項目</td></tr>
<tr><td>@、@1、･･･、@n</td><td>可変データの定義</td></tr>
<tr><td>A.項目名</td><td>当該機能内で定義される項目値</td></tr>
<tr><td>代入先 ＝ 条件 ？ 値 1： 値 2</td><td>C 言語の三項演算子と同じ</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>外部機能の呼び出し（括弧内は呼び出す機能への入力）</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>保護機能の呼び出し</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>内部機能の呼び出し</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>本書の記載範囲外のブロックの外部機能の呼び出し</td></tr>
<tr><td>ⓢ機能名</td><td>静的関数</td></tr>
<tr><td>ⓥ機能名／ⓥ機能名</td><td>仮想関数／純粋仮想関数</td></tr>
<tr><td>機能名ⓒ</td><td>コンスト属性付き関数</td></tr>
<tr><td>〔○○○○○〕</td><td>略語の定義、コメント等</td></tr>
<tr><td>~「オブジェクト」の省略</td><td>例 剛体管理オブジェクトは剛体管理と記載する。</td></tr>
<tr><td>場合分け</td><td>二つの場合の例
1 行で条件判定処理と条件を満たした場合の処理を記載する場合の例
“条件判定”、“処理”
“そうでない”、“処理”
条件判定処理と条件を満たした場合の処理を改行して記載する場合の例
“条件判定の処理”
処理（字下げする）
“そうではない”
処理（字下げする）

三つ以上の場合の例
<table>
<tr><th>条件判定</th><th>処理</th></tr>
<tr><td>1 の条件判定</td><td>1 の条件が満たされた場合の処理</td></tr>
<tr><td>・
・
・</td><td>・
・
・</td></tr>
<tr><td>n の条件判定</td><td>n の条件が満たされた場合の処理</td></tr>
</table></td></tr>
</table>

7.2 共通データ

順動力学演算ブロックの詳細ブロック（スケール管理ブロックを除く）の共通データの構造と付帯する機能を示す。

(1) 物理演算エンジン種別型(PE)

表 7-5 物理演算エンジン種別型

番号	項目	初期値	型
1	物理演算エンジン種別 0：RBX(RoboBio-X) 1：BT(Bullet) 2：PX(PhysX)	―	enum class PE

(2) システム単位型(SysUnit)

表 7-6 システム単位

番号	項目	初期値	型
1	システム単位 0：不定 2：メートル 3：センチ メートル 4：ミリ メートル	―	enum class SysUnit

(3) 単位変換データ(DynUnitCnv)

表 7-7 にデータの構成を示す。

表 7-7 単位変換データの構造

番号	項目	初期値	型
1	システム単位	0	SysUnit
2	システム単位 → メートル単位変換係数	1	Float
3	メートル単位 → システム単位変換係数	1	Float
4	システム単位の 2 乗 → メートル単位の 2 乗変換係数	1	Float
5	メートル単位の 2 乗 → システム単位 2 乗変換係数	1	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-7 の初期値から単位変換データを生成する。

機能	初期化
処理	システム設定の単位に応じて#.*を設定する。

機能	読み出し専用単位変換データ取得	
出力	読み出し専用単位変換データ	const DynUnitCnv&

機能	単位変換データ設定©	
入力	単位変換データ	const DynUnitCnv&
処理	#.* = I.単位変換データ.*	

機能	システム単位 → メートル単位変換©	
入力	システム単位の数値	Float
出力	メートル単位の数値	Float

機能	メートル単位 → システム単位変換©	
入力	メートル単位の数値	Float
出力	システム単位の数値	Float

機能	システム単位の 2 乗 → メートル単位の 2 乗変換©	
入力	システム単位の数値	Float
出力	メートル単位の数値	Float

機能	メートル単位の 2 乗 → システム単位 2 乗変換©	
入力	メートル単位の数値	Float
出力	システム単位の数値	Float

(4) 力学サポート(DynSuport)

本データは空データとし、付帯機能のみ定義する。

機能	Ⓢプリミティブ モデル確認	
入力	モデル	const BaseObject*
処理	出力値=I.モデルが表 6-2 に示す条件に適合 ? true : false	
出力	確認結果	Bool

機能	Ⓢ2 次元プリミティブ モデル確認	
入力	モデル	const BaseObject*
処理	出力値=I.モデルが表 7-8 に示す条件に適合 ? true : false	
出力	確認結果	Bool

表 7-8 2 次元プリミティブ モデルの条件

モデルの種類	属性の条件
球	タイプが半球
円柱	キャップなし
円錐	キャップなし
ディスク	なし
平面	なし
シンプル ポリゴン	なし
地形	球状が無効

機能	Ⓢ2/3 次元プリミティブ モデル	
入力	モデル	const BaseObject*
処理	出力値=!<2 次元プリミティブ モデル確認>(I.モデル)	
出力	確認結果	Bool

機能	Ⓢ衝突判定形状モデル確認	
入力	モデル	const BaseObject*
処理	出力値=I.モデルが表 7-9 に示す条件に適合 ? true : false	
出力	確認結果	Bool

表 7-9 衝突判定形状モデルの条件

モデルの種類（網掛されていないモデルはプリミティブ モデル）			
立方体	球	カプセル	円柱
チューブ	円錐	正多面体	ディスク
平面	シンプル ポリゴン	地形	ポリゴン

(5) スケールド モデル データ(ScaledObject)

表 7-10 にデータの構成を示す。

表 7-10 スケールド モデル データの構造

番号	項目	初期値	型
1	モデル	nullptr	BaseObject*
2	全体のスケール（X/Y/Z 軸方向のスケール値）	(1　1　1)	Vector
3	上端のスケール（モデルがカプセルの場合のみ有効）	1	Float
4	下端のスケール（モデルがカプセルの場合のみ有効）	1	Float
5	ポリゴン化指定（true：ポリゴン化する / false：ポリゴン化しない）	false	Bool
6	3D/2D 指定（true：3D / false：2D）	true	Bool

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-10 の初期値からスケールド モデル データを生成する。

機能	コンストラクタ	
入力	モデル	BaseObject*
	全体スケール	const Vector&
	ポリゴン化指定	Bool
	3D/2D 指定(true：3D / false：2D)	Bool
処理	入力値と固定の上下端スケール値(1)からスケールド モデル データを生成する。	

機能	コンストラクタ	
入力	モデル	BaseObject*
	全体スケール	const Vector&
	上端のスケール	Float
	下端のスケール	Float
	ポリゴン化指定	Bool
	3D/2D 指定(true：3D / false：2D)	Bool
処理	入力値からスケールド モデル データを生成する。	

機能	スケールなし確認©	
処理	出力値=#.[2,3,4] == 1 ? true : false	
出力	確認結果	Bool

(6) 並進性質データ(DynTdata)

表 7-11 にデータの構成を示す。

表 7-11 並進性質データの構造

番号	項目	初期値	型
1	比重	1	Float
2	厚み（3D 要素の場合は 0.0）	0	Float
3	体積（単位：システム）	0	Float
4	体積（単位：メートル）	0	Float
5	質量（単位：kg）	0	Float
6	重心（単位：システム／剛体のローカル座標）	(0 0 0)	Float
7	重心（単位：メートル／剛体のローカル座標）	(0 0 0)	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-11 の初期値から並進性質データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定（DONT_INITIALIZE）	ENUM_DONT_INITIALIZE
処理	無処理	

機能	体積と比重から質量を算出©	
入力	体積（システム単位 ^{3}）	Float
	システム単位	SysUnit
処理	出力値=0.001×({システム単位 ^{3}→cm^3 変換}←I.体積)×#.比重	
出力	質量(kg)	Float

機能	体積と比重から質量を算出	
入力	システム単位	SysUnit
処理	#.質量=＜体積と比重から質量を算出＞(#.体積, I.システム単位)	

機能	厚み設定	
入力	厚み	Float
	3D/2D 指定(true：3D / false：2D)	Bool
処理	I.3D/2D 指定 == 3D、#.厚み=0 そうではない、#.厚み=Max(I.厚み, 0.001)	

機能	3D 性質確認©	
処理	出力値=#.厚み == 0 ? true：false	
出力	確認結果（true：3D 性質／false：2D 性質）	Bool

機能	2D 性質確認©	
処理	出力値=#.厚み == 0 ? false：true	
出力	確認結果	Bool

機能	動的性質確認©	
処理	出力値=#.質量 > 0 ? true：false	
出力	確認結果	Bool

機能	静的性質確認©	
処理	出力値=質量 == 0 ? true：false	
出力	確認結果	Bool

(7) 並進･回転性質データ(DynTRdata：DynTdata)

表 7-12 にデータの構成を示す。

表 7-12 並進･回転性質データの構造

番号	項目	初期値	型
1	並進性質データ	＜－＞	DynTdata
2	慣性モーメント テンソル （単位：kg･m²／座標系の原点は重心、軸の向きは L 座標系に一致）	$\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}$	SqMat3<Vector>
3	主慣性モーメント（単位：kg･m²）	(0 0 0)	Vector
4	慣性モーメント L スケール	1	Float
5	慣性主軸 Ml（L 座標系→慣性主軸座標系）	$\begin{pmatrix}0&0&0\\1&0&0\\0&1&0\\0&0&1\end{pmatrix}$	Matrix
6	L 原点 Ml（慣性主軸座標系→L 座標系）	$\begin{pmatrix}0&0&0\\1&0&0\\0&1&0\\0&0&1\end{pmatrix}$	Matrix
7	Ml 有効・無効（true： 有効 / false： 無効）	false	Bool

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-12 の初期値から並進･回転性質データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定(DONT_INITIALIZE)	ENUM_DONT_INITIALIZE
処理	無処理	

機能	慣性モーメントの対角成分取得©	
処理	出力値=#.慣性モーメント テンソルの対角成分	
出力	慣性モーメントの対角成分	Vector

機能	慣性モーメントの対角成分設定	
入力	対角成分値	const Vector&
処理	#.慣性モーメント テンソルの全成分=0 #.慣性モーメント テンソルの対角成分=I.対角成分値	

機能	主慣性モーメントと座標系変換マトリクス算出
処理	A.#対角化慣性モーメント テンソルと A.回転角={ヤコビ法で対角化}← #.慣性モーメント テンソル 〔A.回転角は#.対角化慣性モーメント テンソルから#.慣性モーメント テンソルへの回転角〕 #.主慣性モーメント= A.対角化慣性モーメント テンソルの対角成分 #.重心（単位：メートル）と A.回転角から#.5~7 を設定

(8) 運動条件データ(DynVcdData)

表 7-13 にデータの構成を示す。

表 7-13 運動条件データの構造

番号	項目	初期値	型
1	衝突マージン（単位 m）	0	Float
2	初期線形速度（動的剛体のみ有効、グローバル座標系、単位 m/s）	(0 0 0)	Vector
3	初期回転速度（動的剛体のみ有効、グローバル座標系、単位 Rad/s）	(0 0 0)	Vector
4	反発（範囲は 0 以上）	0	Float
5	摩擦（範囲は 0 以上）	0	Float
6	転がり摩擦（範囲は 0 以上）	0	Float
7	回転摩擦（範囲は 0 以上）	0	Float
8	線形ダンピング（動的剛体のみ有効、範囲は 0 以上）	0	Float
9	回転ダンピング（動的剛体のみ有効、範囲は 0 以上）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-13 の初期値から運動条件データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定（DONT_INITIALIZE）	ENUM_DONT_INITIALIZE
処理	無処理	

機能	クリアー
処理	#.*=初期値

(9) 物理演算不活性化データ(DynDeactData)

表 7-14 にデータの構成を示す。

表 7-14 物理演算不活性化データの構造

番号	項目	初期値	型
1	線形速度の閾値（動的剛体のみ有効、単位 m/s）	0	Float
2	回転速度の閾値（動的剛体のみ有効、単位 Rad/s）	0	Float
3	非アクティブ化までの時間（動的剛体のみ有効、単位 秒）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-14 の初期値から物理演算不活性化データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定（DONT_INITIALIZE）	ENUM_DONT_INITIALIZE
処理	無処理	

機能	クリアー
処理	#.*=初期値

機能	物理演算不活性化無効確認©	
処理	出力値=#.* == 0 ? true : false	
出力	確認結果	Bool

(10) 剛体モニター データ(DynBodyMonData)

表 7-15 にデータの構成を示す。

表 7-15 剛体モニター データの構造

番号	項目	初期値	型
1	プリアンブルワード値（固定値 0x426f64794d6f6e44=PAW）	PAW	static const UInt64
2	ターミネートワード値（固定値 0x446e6f4d79646f42=TW）	TW	static const UInt64
3	領域プリアンブルワード	PAW	UInt64
4	モニター更新指定（true： 更新する / false： 更新しない）	false	Bool
5	線形速度(グローバル座標系、単位 m/s)	(0 0 0)	Vector
6	回転速度(グローバル座標系、単位 rad/s)	(0 0 0)	Vector
~~7~~	~~線形フォース(グローバル座標系、単位 N)~~	~~(0 0 0)~~	~~Vector~~
~~8~~	~~回転トルク(グローバル座標系、単位 Nm)~~	~~(0 0 0)~~	~~Vector~~
9	領域ターミネートワード	TW	UInt64

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-15 の初期値から剛体モニター データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定（DONT_INITIALIZE）	ENUM_DONT_INITIALIZE
処理	無処理	

機能	デストラクタ
処理	#.プリアンブルワード=#.領域ターミネートワード=0

機能	書き込み可能確認©	
処理	出力値=#.領域プリアンブルワード == PAW && #.領域ターミネートワード == TW ? true : false	
出力	確認結果	Bool

(11) 動力学データ(DynData：DynTRdata, DynVcdData, DynDeactData)

表 7-16 にデータの構成を示す。

表 7-16 動力学データの構造

番号	項目	初期値	型
1	並進･回転性質データ	＜－＞	DynTRdata
2	運動条件データ	＜－＞	DynVcdData
3	物理演算不活性化データ	＜－＞	DynDeactData

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-16 の初期値から動力学データを生成する。

機能	コンストラクタ	
入力	初期化未実施指定	ENUM_DONT_INITIALIZE
処理	無処理	

機能	クリアー
処理	#.*=初期値

機能	動的性質項目のクリアー
処理	@=[初期線形速度、初期回転速度、線形ダンピング、回転ダンピング、線形速度の閾値、回転速度の閾値、非アクティブ化までの時間]の処理 #.@=0

(12) 剛体データ(DynBodyData)

表 7-17 にデータの構成を示す。

表 7-17 剛体データの構造

番号	項目	初期値	型
1	実形状のＳＣモデルD	＜・＞	const ScaledObject
2	衝突判定形状のＳＣモデルD	＜・＞	const ScaledObject
3	動力学データ	＜・＞	const DynData*
4	剛体モニター データ	＜・＞	const DynBodyMonData*

以下に付帯機能の構成を示す。

機能	コンストラクタ	
入力	実形状のＳＣモデルD	const ScaledObject&
	衝突判定形状のＳＣモデルD	const ScaledObject&
	動力学データ	const DynData&
	剛体モニター データ	const DynBodyMonData&
処理	入力値から剛体データを生成する。	

機能	コンストラクタ	
入力	実形状のＳＣモデルD	const ScaledObject&
	動力学データ	const DynData&
	剛体モニター データ	const DynBodyMonData&
処理	入力値から剛体データを生成する。〔衝突判定形状は実形状と同一とする。〕	

(13) ジョイント種別型(DynJointType)

表 7-18 にデータの構成を示す。

表 7-18 ジョイント種別型

番号	項目	初期値	型
1	ジョイント種別 -1：未定義 0：組み合わせヒンジ 1：カルダン＋ヒンジ 2：3DOF	―	enum class DynJointType

(14) 抵抗トルク データ(DynResistTrq)

表 7-19 にデータの構成を示す。

表 7-19 抵抗トルク データの構造

番号	項目	初期値	型
1	静止抵抗トルク（単位 Nm）	0	Float
2	動抵抗力トルク（単位 Nm）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-19 の初期値から抵抗トルク データを生成する。

(15) 抵抗フォース データ(DynResistForce)

表 7-20 にデータの構成を示す。

表 7-20 抵抗フォース データの構造

番号	項目	初期値	型
1	静止抵抗力（単位 N）	0	Float
2	動抵抗力（単位 N）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-20 の初期値から抵抗フォース データを生成する。

(16) トルクと角速度データ(DynTrqVel)

表 7-21 にデータの構成を示す。

表 7-21 トルクと角速度データの構造

番号	項目	初期値	型
1	トルク（単位 Nm）	0	Float
2	角速度（単位 Rad/s）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-21 の初期値からトルクと角速度データを生成する。

機能	コンストラクタ	
入力	トルク（単位 Nm）	Float
	角速度（単位 Rad/s）	Float
処理	入力値からトルクと角速度データを生成する。	

機能	クリアー
処理	#.*=0

(17) フォースと速度データ(DynFrcVel)

表 7-22 にデータの構成を示す。

表 7-22 フォースと速度データの構造

番号	項目	初期値	型
1	フォース（単位 N）	0	Float
2	速度（単位 m/s）	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-22 の初期値からフォースと速度データを生成する。

機能	コンストラクタ	
入力	フォース（単位 N）	Float
	速度（単位 m/s）	Float
処理	入力値からフォースと速度データを生成する。	

機能	クリアー
処理	#.*=0

(18) 回転データ(DynRot)

表 7-23 にデータの構成を示す。

表 7-23 回転データの構造

番号	項目	初期値	型
1	プリアンブルワード値（固定値 0x44796e526f74577b=PAW）	PAW	static const UInt64
2	ターミネートワード値（固定値 0x7d44796e526f7457=TW）	TW	static const UInt64
3	回転タイプ（固定/制限なし/角度制限）	固定	enum class RotType
4	駆動タイプ（アイドル/トルク/サーボ）	アイドル	enum class DriveType
5	ショート ブレーキ指定（true：使用する / false：使用しない）	false	Bool
6	制限の範囲（範囲 −π～π、単位 Rad）	(0, 0)	rbxSys::Span<Float>
7	サーボ目標（範囲 −π～π、単位 Rad）	0	Float
8	抵抗トルク	(0, 0)	DynResistTrq
9	最大トルクと最高角速度	(0, 0)	DynTrqVel
10	印加トルクと印加角速度	(0, 0)	DynTrqVel
11	領域プリアンブルワード	PAW	UInt64
12	カレント データ（トルクと角速度）	(0, 0)	DynTrqVel(mutable)
13	領域ターミネートワード	TW	UInt64

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-23 の初期値から回転データを生成する。

機能	デストラクタ
処理	#.領域プリアンブルワード=#.領域ターミネートワード=0

機能	カレント データ クリアー
処理	#.カレント データ（トルクと角速度）=(0, 0)

機能	カレント データ領域書き込み可能確認©	
処理	出力値=#.領域プリアンブルワード == PAW && 領域#.ターミネートワード == TW ? true : false	
出力	確認結果	Bool

機能	カレント データ設定	
入力	カレント データ（トルクと角速度）	const DynTrqVel&
処理	#.カレント データ=I.カレント データ	

機能	トルク制御駆動確認©	
処理	出力値=#.駆動タイプ == トルク ? true : false	
出力	確認結果	Bool

(19) 直動データ(DynLin)

表 7-24 にデータの構成を示す。

表 7-24 直動データの構造

番号	項目	初期値	型
1	プリアンブルワード値（固定値 0x44796e4c696e577b =PAW）	PAW	static const UInt64
2	ターミネートワード値（固定値 0x7d44796e4c696e57 =TW）	TW	static const UInt64
3	直動タイプ（固定/長さ制限）	固定	enum class LinType
4	駆動タイプ（アイドル/フォース/サーボ）	アイドル	enum class DriveType
5	ショート ブレーキ指定（true：使用する / false：使用しない）	false	Bool
6	制限の範囲（単位：メートル）	(0, 0)	rbxSys::Span<Float>
7	サーボ目標（単位：メートル）	0	Float
8	抵抗フォース	(0, 0)	DynResistForce
9	最大フォースと最高速度	(0, 0)	DynFrcVel
10	印加フォースと印加速度	(0, 0)	DynFrcVel
11	領域プリアンブルワード	PAW	UInt64
12	カレント データ（フォースと速度）	(0, 0)	DynFrcVel(mutable)
13	領域ターミネートワード	TW	UInt64

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-24 の初期値から直動データを生成する。

機能	デストラクタ
処理	#.領域プリアンブルワード=#.領域ターミネートワード=0

機能	カレント データ クリアー
処理	#.カレント データ（フォースと速度）=(0, 0)

機能	カレント データ領域書き込み可能確認©	
処理	出力値=#.領域プリアンブルワード == PAW && #.領域ターミネートワード == TW ? true : false	
出力	確認結果	Bool

機能	カレント データ設定	
入力	カレント データ（フォースと速度）	const DynFrcVel&
処理	#.カレント データ=I.カレント データ	

機能	フォース制御駆動確認©	
処理	出力値=#.駆動タイプ == フォース ? true : false	
出力	確認結果	Bool

(20) 節データ(DynJointBodyData：DynData)

表 7-25 にデータの構成を示す。

表 7-25 節データの構造

番号	項目	初期値	型
1	動力学データ	＜－＞	DynData
2	実形状のルート モデル（節の L 座標系を保有）	nullptr	BaseObject*
3	衝突判定形状のルート モデル	nullptr	BaseObject*
4	実形状モデル配列	empty	BaseArray<ScaledObject>
5	衝突判定形状モデル配列	empty	BaseArray<ScaledObject>
6	モニター データ	nullptr	DynBodyMonData*
7	衝突判定無効指定	false	Bool

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-25 の初期値から節データを生成する。

機能	コピー コンストラクタ	
入力	節データ	const DynJointBodyData&
処理	I.節データを初期値とする節データを生成する。	

機能	コンストラクタ	
入力	初期化未実施指定（DONT_INITIALIZE）	ENUM_DONT_INITIALIZE
処理	無処理	

機能	クリアー
処理	#.*=初期値

機能	複製	
入力	複製先	DynJointBodyData&
処理	I.複製先.[1,7]=#.[1,7] I.複製先.[2,3,6]=nullptr I.複製先.[4,5]= empty	

機能	代入演算子	
入力	節データ	const DynJointBodyData&
処理	#.*=I.節データ.*	
出力	節データ	const DynJointBodyData&

(21) 回転/直動モニター更新指定データ(DynRotLinMonUpdate)

表 7-26 にデータの構成を示す。

表 7-26 回転/直動モニター更新指定データの構造

番号	項目	初期値	型
1	H 軸モニター更新指定（true：モニターする/false：モニターしない）	false	Bool
2	P 軸モニター更新指定（true：モニターする/false：モニターしない）	false	Bool
3	B 軸モニター更新指定（true：モニターする/false：モニターしない）	false	Bool
4	E 軸モニター更新指定（true：モニターする/false：モニターしない）	false	Bool

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-26 の初期値から回転/直動モニター更新指定データを生成する。

(22) ジョイント データ(DynJointData)

表 7-27 にデータの構成を示す。

表 7-27 ジョイント データの構造

番号	項目	初期値	型
1	ジョイント モデル（アタッチ先のモデル）	nullptr	BaseObject*
2	アンカー モデル（ルート ジョイントの固定先モデル。）	nullptr	BaseObject*
3	ジョイント タイプ （0：組み合わせヒンジ/1：カルダン＋ヒンジ/2：3DOF）	0	DynJointType
4	ルート指定（true：ルート/false：非ルート）	false	Bool
5	分離リグ指定（true：分離リグ/false：分離リグでない）	false	Bool
6	分離リグのアンカー モデル	nullptr	BaseObject*
7	H 節有効/無効（true：有効/false：無効）	true	Bool
8	P 節有効/無効（true：有効/false：無効）	true	Bool
9	B 節有効/無効（true：有効/false：無効）	true	Bool
10	H 節データ	＜－＞	DynJointBodyData
11	P 節データ	＜－＞	DynJointBodyData
12	B 節データ	＜－＞	DynJointBodyData
13	E 節データ	＜－＞	DynJointBodyData
14	位置参照モデル（ルート指定が true の場合のみ有効）	nullptr	BaseObject*
15	回転/直動モニター更新指定データ	＜－＞	DynRotLinMonUpdate
16	H 軸回転データ	＜－＞	DynRot
17	P 軸回転データ	＜－＞	DynRot
18	B 軸回転データ	＜－＞	DynRot
19	E 軸直動データ	＜－＞	DynLin
20	デバッグ可視化データ	－	DebugParams
20.1	HPB 軸表示サイズ	(0　0　0)	Vector
20.2	E 軸表示サイズ	0	Float
21	演算誤差データ	－	SpecialParams
21.1	ERP 有効指定（true：有効/false：無効）	false	Bool
21.2	STOP ERP 有効指定（true：有効/false：無効）	false	Bool
21.3	CFM 有効指定（true：有効/false：無効）	false	Bool
21.4	STOP CFM 有効指定（true：有効/false：無効）	false	Bool
21.5	破断閾値有効指定（true：有効/false：無効）	false	Bool
21.6	HPB 軸 ERP	(0　0　0)	Vector
21.7	E 軸 ERP	0	Float
21.8	HPB 軸 STOP ERP	(0　0　0)	Vector
21.9	E 軸 STOP ERP	0	Float
21.10	HPB 軸 CFM	(0　0　0)	Vector
21.11	E 軸 CFM	0	Float
21.12	HPB 軸 STOP CFM	(0　0　0)	Vector
21.13	E 軸 STOP CFM	0	Float
21.14	HPB 軸破断閾値	(0　0　0)	Vector
21.15	E 軸破断閾値	0	Float

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-27 の初期値からジョイント データを生成する。

機能	H/P/B 節存在確認©（H/P/B 別の実装とする）	
処理	出力値=#.[7,8,9] == 有効 && #.[10,11,12].実形状モデル配列がモデルを保持している ？true：false	
出力	確認結果	Bool

機能	E 節存在確認©	
処理	出力値=#.13 実形状モデル配列がモデルを保持している ？true：false	
出力	確認結果	Bool

機能	H/P/B 軸存在確認©（H/P/B 別の実装とする）	
処理	出力値=#.[7/8/9] == 有効 && #.[10/11/12].実形状のルート モデル != nullptr ? true : false	
出力	確認結果	Bool

機能	E 軸存在確認©	
処理	出力値=#.13.実形状のルート モデル != nullptr ? true : false	
出力	確認結果	Bool

機能	デバッグ可視化データのデフォルト コンストラクタ
処理	表 7-27.20 の初期値からデバッグ可視化データを生成する。

機能	演算誤差データのデフォルト コンストラクタ
処理	表 7-27.21 の初期値から演算誤差データを生成する。

機能	先頭の回転軸取得©	
処理	＜H 軸存在確認＞() == true の場合、終了(出力値=#.H 軸節データ.実形状のルート モデル) ＜P 軸存在確認＞() == true の場合、終了(出力値=#.P 軸節データ.実形状のルート モデル) ＜B 軸存在確認＞() == true の場合、終了(出力値=#.B 軸節データ.実形状のルート モデル) 出力値= nullptr	
出力	先頭の回転軸のモデル（存在しない場合は nullptr）	BaseObject*

機能	末尾の回転軸取得©	
処理	＜B 軸存在確認＞() == true の場合、終了(出力値=#.B 軸節データ.実形状のルート モデル) ＜P 軸存在確認＞() == true の場合、終了(出力値=#.P 軸節データ.実形状のルート モデル) ＜H 軸存在確認＞() == true の場合、終了(出力値=#.H 軸節データ.実形状のルート モデル) 出力値= nullptr	
出力	末尾の回転軸のモデル（存在しない場合は nullptr）	BaseObject*

機能	位置参照モデル検知
処理	#.位置参照モデル=nullptr #.分離リグ == true 　#.位置参照モデル=表 6-21 の当該ジョイントの存在する回転軸の状況に従った位置参照モデル

機能	HPB 角度取得©	
処理	出力値=HPB 回転軸の角度〔存在しない回転軸の角度は 0 とする。〕	
出力	HPB 回転軸の角度	Vector

(23) インデックスド モデル(IndexedModel)

表 7-28 にデータの構成を示す。

表 7-28 インデックスド モデルの構造

番号	項目	初期値	型
1	モデル	nullptr	BaseObject*
2	インデックス番号	-1	Int32

以下に付帯機能の構成を示す。

機能	デフォルト コンストラクタ
処理	表 7-28 の初期値からインデックスド モデルを生成する。

機能	コンストラクタ	
入力	モデル	BaseObject*
	インデックス番号	Int32
処理	入力値からインデックスド モデルを生成する。	

7.3 スケール管理

スケール管理（形状の変更を含む）を実現する図 7-1 に示した階層のクラスに実装する管理情報と関数を次章以降に示す。

7.3.1 スケール管理コア クラス

動作環境に依存しないスケール管理を実現するスケール管理コア クラス(Scale)の構成を示す。

(1) 管理データ

表 7-29 スケール管理コア クラスの管理データ

番号	項目	初期値	型
1	形状変更結果型Ⓟ （0：変更なし/1：軸スケール変更/2：不均一カプセル スケール変更）	―	enum class Modify
2	軸スケールⓅ	(1 1 1)	Vector
3	不均一なカプセルのスケール	―	―
3.1	上端スケール（範囲：0 以上）Ⓟ	1	Float
3.2	下端スケール（範囲：0 以上）Ⓟ	1	Float

(2) 保護関数

機能	デフォルト コンストラクタ	
処理	表 7-29 の初期値からスケール管理コアを生成する。	

機能	不均一カプセル形状変更	
入力	ポリゴン モデル（ジェネレータのキャッシュ）	BaseObject*
	カプセル属性値	―
	均一カプセルの半径	Float
	均一カプセルの高さ	Float
	カプセルの方向(X+/X-/Y+/Y-/Z+/Z-)	Int32
処理	#.2~3 と I.カプセル属性値に従って、I.ポリゴン モデルの不均一カプセル スケールを変更する。 出力値=不均一カプセルのスケール変更を実施した ？ 不均一カプセル変更 ： 変更なし	
出力	形状変更結果	Modify

機能	軸スケール形状変更	
入力	ポリゴン 3D モデル（ジェネレータのキャッシュ）	BaseObject*
処理	#.2~3 に従って、I.ポリゴン モデルの軸スケールを変更する。 出力値=軸スケール変更を実施した ？ 軸スケール変更 ： 変更なし	
出力	形状変更結果	Modify

7.3.2 スケール管理クラス

動作環境に応じたスケール管理を実現するスケール管理クラス(ScaleNode：public TagData, protected Scale)の構成を示す。

(1) 管理データ

表 7-30 スケール管理クラス管理データ

番号	項目	初期値	型
1	スケール管理コア クラスの管理情報Ⓟ	＜－＞	－
2	最後にスケールを適応したモデル	nullptr	BaseObject*

(2) UI コンテナ（シミュレーション環境のみ）

図 7-14 にスケール管理クラスの UI コンテナを示す。

図 7-14 スケール管理の UI コンテナ

UI 項目の入力範囲は対応する管理情報と同じとする。

表 7-31 に入力可能状態が変化する UI 項目とその条件を示す。

表 7-31 UI 項目と入力不可条件

UI 項目	入力不可となる条件
スケール	アタッチ先のモデルが表 6-1 の条件を満たすプリミティブ モデルでない。
上端のスケール	アタッチ先のモデルがカプセルでない。
下端のスケール	

(3) 外部関数

機能	Ⓢスケール管理生成	
処理	新規メモリ領域にスケール管理を生成する。	
出力	スケール管理	NodeData*

機能	Ⓢサポート プリミティブ形状確認（シミュレーション環境のみ）	
入力	モデル	const BaseObject*
処理	I.モデルが表 6-1 の条件を満たすプリミティブ形状 ? true : false	
出力	確認結果	Bool

機能	Ⓢスケール値取得	
入力	モデル	const BaseObject*
	不均一なカプセルのスケール	－
	上端スケール	Float&
	下端スケール	Float&
処理	不均一なカプセルのスケール=(1,1) I.モデル.スケール管理 == nullptr、終了(出力値=Vector(1,1,1)) I.不均一なカプセルのスケール=モデル.スケール管理.(1).不均一なカプセルのスケール 出力値=モデル.スケール管理.軸スケール	
出力	軸スケール値	Vector

機能	ⓥUI コンテナの初期化（シミュレーション環境のみ）	
入力	タグ	GeListNode*
処理	I.タグ.UI コンテナ.* = 初期値	
出力	処理結果（常に true）	Bool

機能	ⓥUI 項目の有効性取得（シミュレーション環境のみ）	
入力	タグ	GeListNode*
	UI 項目 ID	const DescID&
	パラメータ データ（未使用）	const GeData&
	フラグ（未使用）	DESCFLAGS_ENABLE
	ディスクリプション コンテナ（未使用）	const BaseContainer*
処理	表 7-31 の内容に従って、UI 項目の編集の不可設定値を決定する。	
出力	編集可否設定値（true：編集可能／false：編集不可）	Bool

機能	ⓥ実行タイミング追加（シミュレーション環境のみ）	
入力	タグ	BaseTag*
	優先度リスト	PriorityList*
処理	I.優先度リスト.＜追加＞(I.タグ，EXECUTIONPRIORITY_GENERATOR + 950，起動フラグ=0)	
出力	処理結果（常に true）	Bool

機能	ⓥ実行ハンドラ	
入力	タグ	BaseTag*
	空間管理	BaseDocument*
	モデル（アタッチ先のモデル）	BaseObject*
	スレッド（未使用）	BaseThread*
	優先度（未使用）	Int32
	起動フラグ（未使用）	EXECUTIONFLAGS
処理	A.モデルのサポート状況=＜サポート プリミティブ形状確認＞(I.モデル) A.モデルのサポート状況 != true #.最後にスケールを適応したモデル=nullptr 終了(出力値=EXECUTIONRESULT::OK) A.キャシュ モデル=I.モデル.キャッシュ A.キャシュ モデル == nullptr #.最後にスケールを適応したモデル=nullptr 終了(出力値=EXECUTIONRESULT::OK) A.キャシュ モデル != #.最後にスケールを適応したモデル ＜データ キャシュ更新＞(I.タグ.UI コンテナ，I.モデル.タイプ == カプセル) #.(1).[2~3]に従って A.キャシュ モデルのスケールを変更する。〔スケール管理コアの保護関数を使用〕 A.キャシュ モデルの形状が変化した場合 #.最後にスケールを適応したモデル=A.キャシュ モデル A.キャシュ モデル.＜メッセージ＞(更新) 出力値=EXECUTIONRESULT::OK	
出力	実行結果	EXECUTIONRESULT

機能	ⓥ複製	
入力	複製先スケール管理	NodeData*
	複製元タグ（未使用）	GeListNode*
	複製先タグ（未使用）	GeListNode*
	複製フラグ（未使用）	COPYFLAGS
	エイリアス トランスレータ（未使用）	AliasTrans*
処理	出力値＝基本クラス定義の＜複製＞(I.*) 出力値 == true 複製先スケール管理.(1).[2~3]＝#.(1).[2~3]	
出力	処理結果（true：正常終了/false：異常終了）	Bool

(4) 内部関数

機能	デフォルト コンストラクタ
処理	表 7-30 の初期値からスケール管理を生成する。

機能	データ キャシュ更新	
入力	UI コンテナ	GeListNode*
	カプセル指定（true：カプセル形状 / false：カプセル形状以外）	Bool
処理	#.(1).軸スケール=UI コンテナ.軸スケール I.カプセル指定 == true #.(1).不均一なカプセルのスケール=UI コンテナ.不均一なカプセル/* そうではない #.(1).不均一なカプセルのスケール=(1,1)	

7.4 剛体管理

剛体管理を実現する図 7-1 に示した階層のクラスに実装する管理情報と関数を次章以降に示す。

7.4.1 剛体管理コア クラス

動作環境に依存しない剛体管理を実現する剛体管理コア クラス(DynBody：protected DynUnitCnv)の構成を示す。

(1) 管理データ

表 7-32 剛体管理コア クラスの管理データ

番号	項目	初期値	型
1	単位変換データⓅ	＜－＞	DynUnitCnv
2	並進・回転性質データ算出結果Ⓟ -7：未知の異常　-6：事前処理異常　-5：非サポート形状異常 -4：リソース確保異常　~~-3：非同期処理起動異常~~　~~-2：分解能異常~~ ~~-1：取り消し~~　0：正常完了　~~1：非同期処理中~~	－	enum class Result
3	慣性モーメント算出方式Ⓟ -1：プリミティブ　0：テトラ積分（3D 用）　~~1：テトラ放射（3D 用）~~ ~~2：グリッド（3D 用）~~　3：三角形（2D 用）	-1	enum class IMethod
4	並進･回転性質データⓅ	＜－＞	DynTRdata

(2) 保護関数

機能	デフォルト コンストラクタ
処理	処理表 7-32 の初期値から剛体管理コアを生成する。

機能	デストラクタ
処理	剛体管理コアを破棄する。

機能	慣性モーメント算出方式取得Ⓒ	
処理	出力値=#.慣性モーメント算出方式	
出力	慣性モーメント算出方式	IMethod

機能	並進・回転性質データ取得Ⓒ	
処理	出力値=#.並進･回転性質データ	
出力	#.並進･回転性質データ	const DynTRdata&

機能	並進・回転性質データ算出	
入力	ＳＣモデルD	const ScaledObject&
	システム単位	SysUnit
	比重	Float
	厚み（モデルの内部が空洞でない場合は 0）	Float
処理	入力の項目で指定されるモデルの並進・回転性質データを算出〔算出方法は 6.2 章の(10)項を参照のこと〕 本処理は以下に示す内部関数で構成する。 3D 並進・回転性質データ値算出 ├3D プリミティブ モデルの並進・回転性質データ算出 │├直方体の並進・回転性質データ算出 │├球体(楕円体)の並進・回転性質データ算出 │├均一スケール カプセルの並進・回転性質データ算出 │├不均一スケール カプセルの並進・回転性質データ算出 │├円柱(楕円柱)の並進・回転性質データ算出 │├チューブの並進・回転性質データ算出 │├円錐(楕円錐)の並進・回転性質データ算出 │└円錐台(楕円錐台)の並進・回転性質データ算出 └3D ポリゴン モデルの並進・回転性質データ算出 2D 並進・回転性質データ算出 ├2D プリミティブ モデルの並進・回転性質データ算出 │├ディスク(楕円盤)の並進・回転性質データ算出 │├平面の並進・回転性質データ算出 │└ポリゴン(四角形/三角形)の並進・回転性質データ算出 └2D ポリゴン モデルの並進・回転性質データ算出	
出力	処理結果(RESULT)	

(3) 内部関数

設計・製作時に決定する。（少なくとも並進・回転性質データ算出保護関数に記載した内部関数構成に含まれる内部関数は実装すること。）

7.4.2 剛体管理クラス

動作環境に応じた剛体管理を実現する剛体管理クラス(DynBodyNode：public TagData, protected DynBody)の構成を示す。

(1) 管理データ

表 7-33 剛体管理クラス管理データ

番号	項目	初期値	型
1	剛体管理コア クラスの管理情報Ⓟ	＜－＞	－
2	スケールド モデル データ ファクトリⒺ （構成は 7.4.2.1 章を参照のこと。）	－	ScModelFactory
3	状態（0：初期/1：ロード/2：コピー/3：通常）	0	enum class State
4	動的剛体識別（true：動的剛体/false：静的剛体）	false	Bool
5	ポリゴン化指定（true：ポリゴンする/false：ポリゴンしない）	false	Bool
6	衝突判定形状モデル	nullptr	BaseObject*
7	所属先ジョイントのモデル	nullptr	BaseObject*
8	所属ルート識別（true：ルート /false：ルートでない）	false	Bool
9	前回検出したアタッチ先のモデル	nullptr	BaseObject*
10	動力学データ	＜－＞	DynData
11	剛体モニター データ	＜－＞	DynBodyMonData

(2) UI コンテナ（シミュレーション環境のみ）

図 7-15 から図 7-19 に剛体管理クラスの UI コンテナを示す。（青枠で囲んだ項目は入力変更不可。）

図 7-15 剛体管理の UI コンテナーメイン

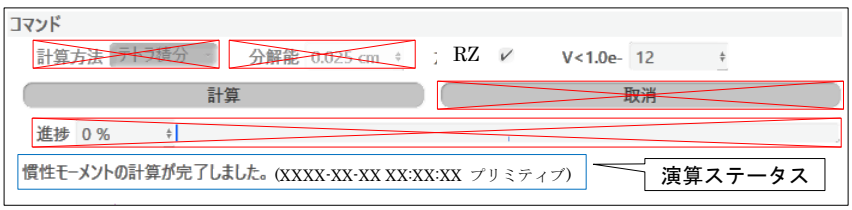

図 7-16 剛体管理の UI コンテナーコマンド

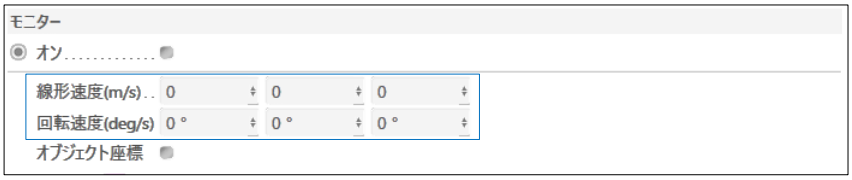

図 7-17 剛体管理の UI コンテナーモニター

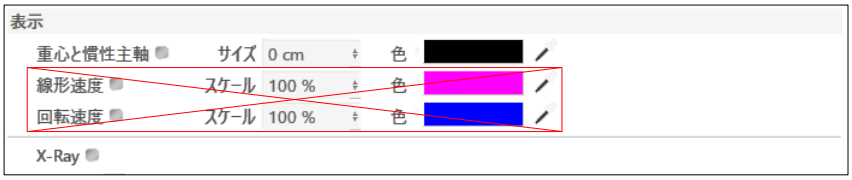

図 7-18 剛体管理の UI コンテナー表示

図 7-19 剛体管理の UI コンテナースペシャル

UI コンテナの非表示項目を表 7-34 に示す。

表 7-34　UI コンテナの非表示項目

項目名	型
所属ルート識別	BOOL
体積	REAL
質量	REAL
慣性モーメント テンソル.V1	VECTOR
慣性モーメント テンソル.V2	VECTOR
慣性モーメント テンソル.V3	VECTOR
主慣性モーメント	VECTOR
変換マトリクス有効/無効	BOOL
慣性主軸 Ml（L 座標系→慣性主軸座標系）	MATRIX
L 原点 Ml（慣性主軸座標系→L 座標系）	MATRIX
剛体の初期位置	VECTOR
剛体の初期角度	VECTOR

UI 項目の入力範囲は対応する管理情報と同じとする。

表 7-35 に入力可能状態が変化する UI 項目とその条件を示す。

表 7-35　UI 項目と入力不可条件

UI 項目	入力不可となる条件
ポリゴン化	UI コンテナ.次元 == 2D || アタッチ先のモデル != サポート プリミティブ （入力不可時はポリゴン化=false と見なす。）
厚み	UI コンテナ.次元 != 2D（入力不可時は厚み=0 と見なす。）
初期線形速度	#.動的剛体識別 == false || (#.所属先ジョイントのモデル != nullptr && #.所属ルート識別 == false)
初期回転速度	
オブジェクト座標	
線形ダンピング	
回転ダンピング	
線形速度の閾値	
回転速度の閾値	
不活性化遅延時間	
慣性モーメント L スケール	
反発	#.所属先ジョイントのモデル != nullptr && #.所属ルート識別 == false
摩擦	
転がり摩擦	
回転摩擦	
モニター	－
オン	#.所属先ジョイントのモデル != nullptr && #.所属ルート識別 == false
線形速度	
回転速度	
オブジェクト座標	
RZ の閾値（指数）	UI コンテナ.RZ == false
X-Ray	#.所属先ジョイントのモデル != nullptr

(3) 外部関数

機能	Ⓢ剛体管理生成	
処理	新規メモリ領域に剛体管理を生成する。	
出力	剛体管理	NodeData*

機能	Ⓢ剛体管理取得	
入力	モデル	const BaseObject*
処理	出力値= I.モデルにアタッチされた剛体管理〔I.モデルに剛体管理がアタッチされていない場合は nullptr〕	
出力	剛体管理	DynBodyNode*

機能	ⓥUI コンテナの初期化（シミュレーション環境のみ）	
入力	タグ	GeListNode*
処理	I.タグ.UI 項目=初期値	
出力	処理結果（常に true）	Bool

機能	ⓥメッセージ ハンドラ	
入力	タグ	GeListNode*
	タイプ	Int32
	データ	void*
処理	入力値から以下のトリガーを検出 トリガー別の処理	

トリガー	処理
ロード	＜ロード＞(I.タグ，I.データ.空間管理)
UI 項目値変更	＜UI 項目値変更＞(I.データ.UI 項目の ID，I.タグ，I.データ.空間管理)
コマンド実行	＜コマンド実行＞(I.タグ，I.データ.コマンド ID)

出力	処理結果（常に true）	Bool

機能	ⓥUI 項目の有効性取得（シミュレーション環境のみ）	
入力	タグ	GeListNode*
	UI 項目 ID	const DescID&
	パラメータ データ（未使用）	const GeData&
	フラグ（未使用）	DESCFLAGS_ENABLE
	ディスクリプション コンテナ（未使用）	const BaseContainer*
処理	表 7-35 の内容に従って、UI 項目の編集の不可設定値を決定する。	
出力	編集可否設定値（true：編集可能／false：編集不可）	Bool

機能	ⓥ実行タイミング追加（シミュレーション環境のみ）	
入力	タグ（未使用）	BaseTag*
	優先度リスト	PriorityList*
処理	I.優先度リスト.＜追加＞(I.タグ，EXECUTIONPRIORITY_ANIMATION，起動フラグ=0)	
出力	処理結果（常に true）	Bool

機能	ⓥ実行ハンドラ	
入力	タグ	BaseTag*
	空間管理	BaseDocument*
	モデル	BaseObject*
	スレッド（未使用）	BaseThread*
	優先度（未使用）	Int32
	起動フラグ（未使用）	EXECUTIONFLAGS
処理	＜状態遷移＞(I.モデル，I.空間管理，I.タグ.UI コンテナ) 力学サポート::＜2 次元プリミティブ モデル確認＞(I.モデル) == true I.タグ.UI コンテナ.次元=2D I.タグ.UI コンテナ.分割数を制限 == true ＜プリミティブの分割数制限設定＞(I.モデル) ＜プリミティブの分割数以外の制限設定＞(I.モデル) ＜データ キャッシュ更新＞(I.タグ.UI コンテナ，I.空間管理，I.モデル) #.剛体モニター データ.モニター更新指定 == true ＜モニター データ更新＞(I.タグ.UI コンテナ) 終了(EXECUTIONRESULT::OK)	
出力	実行結果(EXECUTIONRESULT::OK)	EXECUTIONRESULT

機能	ⓥ複製	
入力	複製先剛体管理	NodeData*
	複製元タグ（本関数では未使用）	GeListNode*
	複製先タグ（本関数では未使用）	GeListNode*
	複製フラグ（本関数では未使用）	COPYFLAGS
	エイリアス トランスレータ（本関数では未使用）	AliasTrans*
処理	出力値＝基本クラス定義の＜複製＞(I.*) 出力値 == true 複製先剛体管理.動的剛体識別=#.動的剛体識別 複製先剛体管理.並進･回転性質データ=#.並進･回転性質データ 複製先剛体管理.状態=コピー	
出力	処理結果（true：正常終了/false：異常終了）	Bool

機能	ⓥ可視エレメント描画（シミュレーション環境のみ）	
入力	タグ	BaseTag*
	モデル	BaseObject*
	描画コンテキスト	BaseDraw*
	描画ヘルパー	BaseDrawHelp*
処理	I.描画コンテキスト.描画パス != オブジェクト、終了(出力値=true) I.タグ.UI コンテナ.表示/重心と慣性主軸 == true ＜重心と慣性主軸を描画＞(I.タグ.UI コンテナ，I.モデル，I.描画コンテキスト，I.描画ヘルパー) #.所属先ジョイントのモデル == nullptr I.タグ.UI コンテナ.表示/X-Ray の設定に従って I.モデル.表示形式〔通常/X-Ray〕を切り替える。 出力値=true	
出力	処理結果（常に true）	Bool

機能	動的剛体確認	
処理	出力値=#.動的剛体識別	
出力	確認結果	Bool

機能	静的剛体確認	
処理	出力値=!#.動的剛体識別	
出力	確認結果	Bool

機能	ポリゴン化確認	
処理	出力値=#.ポリゴン化指定	
出力	確認結果	Bool

機能	3D 性質確認	
処理	出力値=#.動力学データ.＜3D 性質確認＞()	
出力	確認結果（true：3D 性質／false：2D 性質）	Bool

機能	所属確認	
処理	出力値=#.所属先ジョイントのモデルが nullptr ? true : false	
出力	確認結果	Bool

機能	所属ルート確認	
処理	出力値=#.所属先ジョイント != nullptr && #.所属ルート識別 == true ? true : false	
出力	確認結果	Bool

機能	所属非ルート確認	
処理	出力値=#.所属先ジョイント != nullptr && #.所属ルート識別 == false ? true : false	
出力	確認結果	Bool

機能	動力学データ取得	
処理	出力値=#.動力学データ	
出力	動力学データ	const DynTRdata&

機能	動力学データ取得	
入力	衝突判定形状モデル	BaseObject*&
	剛体モニター データ	const DynBodyMonData*&
処理	I.衝突判定形状モデル=#.衝突判定形状モデル I.剛体モニター データ=#.剛体モニター データ 出力値=#.動力学データ	
出力	動力学データ	const DynTRdata&

機能	剛体モニター データ取得	
処理	出力値=#.剛体モニター データ	
出力	剛体モニター データ	const DynBodyMonData*

機能	データ キャッシュ更新	
入力	空間管理	BaseDocument*
処理	＜データ キャッシュ更新＞(#.タグ.UI コンテナ，空間管理，#.タグ.モデル)	

機能	初期の位置と角度を保存
処理	#.タグ.UI コンテナ.非表示項目.剛体の初期位置=#.タグ.モデル.相対位置 #.タグ.UI コンテナ.非表示項目.剛体の初期角度=#.タグ.モデル.相対角度

機能	位置と角度をリセット
処理	#.タグ.モデル.相対位置=#.タグ.UI コンテナ.非表示項目.剛体の初期位置 #.タグ.モデル.相対角度=#.タグ.UI コンテナ.非表示項目.剛体の初期角度

機能	所属設定	
入力	所属先ジョイントのモデル	BaseObject*
	所属ルート識別（true：ルート / false：非ルート / 省略時：非ルート）	Bool
処理	#.タグ.UI コンテナ.所属=I.所属先ジョイントのモデル #.タグ.UI コンテナ.非表示項目.所属ルート識別=I.所属ルート識別 #.所属先ジョイントのモデル=I.所属先ジョイントのモデル #.所属ルート識別=I.所属ルート識別	

(4) 内部関数

機能	デフォルト コンストラクタ
処理	表 7-33 の初期値から剛体管理を生成する。

機能	デストラクタ
処理	#.状態 == 通常 #.所属先ジョイントのモデル != nullptr ＜所属解除＞() アクティブ ドキュメントが存在する場合 アクティブ ドキュメント.＜マルチ メッセージ送信＞(ブロード キャスト，モニター データ無効化，#.モニタ データ) 剛体管理を破棄する。

機能	ロード	
入力	タグ：未使用	BaseTag*
	空間管理：未使用	BaseDocument*
処理	#.状態=ロード	

機能	UI 項目値変更	
入力	UI 項目 ID	Int32
	タグ	GeListNode*
	空間管理：未使用	BaseDocument*
処理	@=[図 7-15 から図 7-19 中の入力変更不可の項目]の処理 I.UI 項目 ID == @.ID I.UI コンテナ.[I.UI 項目 ID の項目]=#.$〔入力変更を取り消す〕	

機能	コマンド実行	
入力	タグ	GeListNode*
	コマンド ID	Int32
処理	コマンド ID 別の処理	

コマンド ID	処理
計算	＜並進・回転性質データ値算出＞(タグ)

機能	並進・回転性質データ算出	
入力	タグ	GeListNode*
処理	#.単位変換データ.＜初期化＞() ＜データ キャシュ更新＞(I.タグ，I.タグ.UI コンテナ，I.タグ.空間管理) A.3D 指定=#.動力学データ.＜3D 性質確認＞() A.ＳＣモデルD=SMDF::＜ＳＣモデルD生成＞(タグ.モデル，#.ポリゴン化指定，A.3D 指定) A.処理結果＝#.(剛体管理コア).＜並進・回転性質データ算出＞(A.ＳＣモデルD，#.動力学データ.比重，#.動力学データ.厚み) A.処理結果 == true #.動力学データ.並進・回転性質データ= {RZ※1} ← #.(剛体管理コア).＜並進・回転性質データ取得＞() I.タグ.UI コンテナ.$=#.動力学データ.並進・回転性質データ.* そうではない、エラーの通知又は蓄積を行う。 ※1 I.タグ.UI コンテナの RZ（Rounding toward zero）設定に応じた 0 への丸め処理	

機能	データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
	モデル	BaseObject*
処理	#.動力学データ.$=I.UI コンテナの表示のみの項目以外の項目値 〔I.UI コンテナ.タイプの更新処理〕 #.動力学データ.質量 ＞0 #.動的剛体識別=true、I.UI コンテナ.タイプを更新 そうではない #.動的剛体識別=false、I.UI コンテナ.タイプを更新 #.動力学データ.＜動的性質項目のクリアー＞() 〔#.ポリゴン化指定の更新処理〕 A.3 次元物質=#.動力学データ.＜3D 性質確認＞() A.プリミティブ モデル=力学サポート::＜プリミティブ モデル確認＞(I.モデル) #.ポリゴン化指定= A.3 次元物質 == true && A.プリミティブ モデル == true ? I.UI コンテナ.ポリゴン化 ：false 〔#.[衝突判定形状，所属，所属ルート識別]の更新処理〕 #.所属先ジョイントのモデル=I.UI コンテナ.所属 #.所属ルート識別= I.UI コンテナ.非表示項目.所属ルート識別 #.衝突判定形状モデル= I.UI コンテナ.フェイク形状	

機能	所属解除
処理	A.ジョイント管理=#.所属先ジョイントのモデルにアタッチされたジョイント管理 A.ジョイント管理.＜剛体のページ＞(#.前回検出したアタッチ先のモデル)

機能	状態遷移	
入力	モデル	BaseObject*
	空間管理	BaseDocument*
	UI コンテナ	BaseContainer*
処理	状態別の処理	

状態	処理
ロード	＜データ キャッシュ更新＞(I.UI コンテナ，I.空間管理，I.モデル) 初期又はコピーの処理へ
初期 又はコピー	#.前回検出したアタッチ先のモデル=I.モデル #.状態=通常
通常	#.前回検出したアタッチ先のモデル != I.モデル #.所属先ジョイントのモデル != nullptr ＜所属解除＞() I.UI コンテナ.所属をクリアー #.前回検出したアタッチ先のモデル=I.モデル

機能	プリミティブの分割数制限設定	
入力	モデル	BaseObject*
処理	表 6-3 の内容に従って、I.モデルの属性値に制限を掛ける。	

機能	プリミティブの分割数以外の制限設定	
入力	モデル	BaseObject*
処理	I.モデルが円錐の場合 円錐.上端の半径 ＞ 円錐.下端の半径、上端の半径と下端の半径を入れ替える	

機能	モニター データ更新	
入力	UI コンテナ	BaseContainer*
処理	I.UI コンテナ.オブジェクト座標 == true I.UI コンテナ.線形速度={#.モデルのローカル座標系に変換}←#.動力学データ.線形速度 I.UI コンテナ.回転速度={#.モデルのローカル座標系に変換}←#.動力学データ.回転速度 そうではない I.UI コンテナ.線形速度=#.動力学データ.線形速度 I.UI コンテナ.回転速度=#.動力学データ.回転速度	

機能	重心と慣性主軸を描画	
入力	UI コンテナ	BaseContainer*
	モデル	BaseObject*
	描画コンテキスト	BaseDraw*
	描画ヘルパー：未使用	BaseDrawHelp*
処理	6.2 章(15)(A)項の内容に従って重心と慣性主軸を描画する。	

7.4.2.1 スケールド モデル データ ファクトリ

スケールド モデル データ ファクトリ クラス(ScModelFactory)の構成を示す。

(1) 管理情報

なし

(2) 外部関数

機能	ⓈＳＣモデルD生成	
入力	モデル	BaseObject*
	ポリゴン化指定	Bool
	3D/2D 指定	Bool
処理	A.[表 7-10.2~4 の項目]＝I.モデルにアタッチされたスケール管理.$ 出力値=[A.表 7-10.2~4 の項目]と I.*からＳＣモデルDを生成	
出力	ＳＣモデルD	ScaledObject

機能	Ⓢ実形状ＳＣモデルD生成	
入力	モデル	BaseObject*
	剛体管理	const TagDynBody*
処理	A.[表 7-10.5~6 の項目]＝I.剛体管理.$ 出力値=＜ＳＣモデルD生成＞(I.モデル，A.[表 7-10.5~6 の項目])	
出力	ＳＣモデルD	ScaledObject

機能	Ⓢ実形状ＳＣモデルD生成	
入力	モデル	BaseObject*
処理	A.剛体管理＝I.モデルにアタッチされた剛体管理 出力値=＜ＳＣモデルD生成＞(I.モデル，A.剛体管理)	
出力	ＳＣモデルD	ScaledObject

機能	Ⓢ衝突判定形状ＳＣモデルD生成	
入力	モデル	BaseObject*
処理	A.3D/2D 指定=力学サポート::＜2/3 次元プリミティブ モデル確認＞(I.モデル) 出力値=＜ＳＣモデルD生成＞(I.モデル，ポリゴン化しない，A.3D/2D 指定)	
出力	ＳＣモデルD	ScaledObject

機能	Ⓢ実形状/衝突判定形状ＳＣモデルD生成	
入力	モデル	BaseObject*
	実形状指定(true：実形状／false：衝突判定形状)	Bool
処理	出力値=I.実形状指定 == true ? ＜実形状ＳＣモデルD生成＞(I.モデル)： ＜衝突判定形状ＳＣモデルD生成＞(I.モデル)	
出力	ＳＣモデルD	ScaledObject

7.5 ジョイント管理

ジョイント管理を実現する図 7-1 に示した階層のクラスに実装する管理情報と関数を次章以降に示す。

7.5.1 ジョイント管理コア クラス

動作環境に依存しないジョイント管理を実現するジョイント管理コア クラス(DynJoint：protected DynJointData, protected DynUnitCnv)の構成を示す。

(1) 管理情報

表 7-36 ジョイント管理コア クラスの管理情報

番号	項目	初期値	型
1	ジョイント データⓅ	＜－＞	DynJointData
2	単位変換データⓅ	＜－＞	DynUnitCnv
3	節要素データ（型）Ⓟ	＜・＞	JtBodyElmData
3.1	スケールド モデル データ	－	ScaledObject
3.2	動力学データ	－	const DynData*
4	終端モデルⓅ	nullptr	BaseObject*

(2) 保護関数

機能	デフォルト コンストラクタ
処理	表 7-36 の初期値からジョイント管理コアを生成する。

機能	デストラクタ
処理	ジョイント管理コアを破棄する。

機能	ジョイント タイプに応じて節データ領域を構成
処理	ジョイント タイプ別の処理

ジョイント タイプ	処理
組み合わせヒンジ	#.ジョイント データ.[H/P/B]軸節有効/無効=true
カルダン＋ヒンジ	#.ジョイント データ.H 軸節有効/無効=false #.ジョイント データ.[P/B]軸節有効/無効=true
3DOF	#.ジョイント データ.[H/P]軸節有効/無効=false #.ジョイント データ.B 軸節有効/無効=true

機能	節の初期化	
入力	1. 節のルート モデルの Mg	const Matrix&
	2. 実形状構成モデルの配列	const BaseArray<BaseObject*>&
	3. 衝突判定形状構成モデルの配列	const BaseArray<BaseObject*>&
	4. 節データ	DynJointBodyData&
	5. 節の構成のみ実施フラグ	Bool
処理	〔実形状構成モデルを構成モデル、慣性モーメント テンソルを慣性 M と記載する〕 ＜節の構成＞(I.[2~4]) I.節の構成のみ実施フラグ == false I.構成モデル.要素数 ＞0 節データ.重心 = ∑(構成モデルの重心 × 構成モデルの質量)/∑構成モデルの質量 節データ.体積 = ∑構成モデルの体積 節データ.質量 = ∑構成モデルの質量 節データ.慣性 M = ∑(R × 構成モデルの慣性 M × R^{-1} + I′) 節データ.(並進･回転性質データ).＜主慣性モーメントと座標系変換マトリクス算出＞() そうではない 節データ.(動力学データ).＜クリアー＞() R と I’ については最後の※1,2 を参照のこと。 ※1 R の求め方 Ml=Inv(I.節のルート モデルの Mg)×(構成モデルの Mg×重心への移動 M) R=Ml.回転 M^T ※2 I'の求め方 a = $(a_x \quad a_y \quad a_z)$ ルート モデルの原点から構成モデルの原点へのベクトル I′ = 節データ.質量 × $\begin{pmatrix} a_y^2+a_z^2 & -a_ya_x & -a_za_x \\ -a_xa_y & a_x^2+a_z^2 & -a_za_y \\ -a_xa_z & -a_ya_z & a_x^2+a_y^2 \end{pmatrix}$	
出力	処理結果（true：正常終了/false：異常終了）	

機能	節の終端モデル取得	
入力	軸指定（H/P/B）	rbxSys::Axis
処理	A.節終端モデル=nullptr I.軸指定別の処理〔#.ジョイント データを JtD と記載する〕	

軸	処理
H 軸	#.JtD.P 節有効/無効 == true && #.JtD.P 軸節データ.実形状のルート モデル != nullptr A.軸節終端モデル=#.ジョイント データ.P 軸節データ.実形状のルート モデル そうではない、P 軸の処理へ
P 軸	#.JtD.B 節有効/無効 == true && #.JtD.B 軸節データ.実形状のルート モデル != nullptr A.軸節終端モデル=#.ジョイント データ.B 軸節データ.実形状のルート モデル そうではない、B 軸の処理へ
B 軸	#.JtD.E 軸節データ.実形状のルート モデル != nullptr A.軸節終端モデル=#.ジョイント データ.E 軸節データ.実形状のルート モデル

処理	A.軸節終端モデル != nullptr 終了(出力値=A.軸節終端モデル.親モデル) 出力値=#.終端モデル	
出力	節の終端モデル（終端モデルが存在しない場合は nullptr）	BaseObject*

機能	節データ パージ
処理	@=H/P/B/E の処理 #.ジョイント データ.[@軸節データ].＜クリアー＞()

(3) 内部関数

機能	ⓢ節の構成	
入力	実形状構成モデルの配列	const BaseArray<JtBodyElmData>&
	衝突判定形状構成モデルの配列	const BaseArray<ScaledObject>&
	節データ	DynJointBodyData&
処理	I.節データ.実形状モデル配列[*]=I.実形状構成モデルの配列[*].ＳＣモデルＤ I.節データ.衝突判定形状モデル配列[*]=衝突判定形状構成モデルの配列[*].ＳＣモデルＤ	
出力	処理結果（true：正常終了／false：異常終了）	Bool

7.5.2 ジョイント管理クラス

動作環境に応じたジョイント管理を実現するジョイント管理クラス(DynJointNode：public TagData, protected DynJoint)の構成を示す。

(1) 管理情報

表 7-37 ジョイント管理クラスの管理情報

番号	項目	初期値	型
1	ジョイント管理コア クラスの管理情報Ⓟ	＜－＞	－
2	状態（0：初期/1：ロード/2：コピー/3：無効/4：有効）	0	enum class State
3	前回イベント処理時の時間フレーム	-1	Int32
4	前回検出したアタッチ先のモデル	nullptr	BaseObject*

(2) UI コンテナ（シミュレーション環境のみ）

図 7-20 から図 7-23 にジョイント管理クラスの UI コンテナを示す。（青枠で囲んだ項目は入力変更不可。）

図 7-20 ジョイント管理の UI コンテナーメイン

図 7-21 ジョイント管理の UI コンテナー軸

図 7-22 ジョイント管理の UI コンテナーコマンドと表示

図 7-23 ジョイント管理の UI コンテナースペシャル

UI コンテナの非表示項目を表 7-38 と表 7-39 に示す。

表 7-38　UI コンテナの非表示項目

項目名	型
初期位置	VECTOR
初期角度	VECTOR
H 節データ	表 7-39 を参照のこと。
P 節データ	〃
B 節データ	〃
E 節データ	〃

表 7-39 UI コンテナの非表示項目ー(H/P/B/E)節データ

項目名		型
モデル系データ	節の実形状構成モデルのリスト	IN_EXCLUDE
	節の衝突判定形状構成モデルのリスト	IN_EXCLUDE
動力学データ	体積	REAL
	質量	REAL
	重心	VECTOR
	慣性モーメント テンソル.V1	VECTOR
	慣性モーメント テンソル.V2	VECTOR
	慣性モーメント テンソル.V3	VECTOR
	主慣性モーメント	VECTOR
	慣性モーメント L スケール	REAL
	変換マトリクス有効/無効	BOOL
	慣性主軸 Ml（L 座標系→慣性主軸座標系）	MATRIX
	L 原点 Ml（慣性主軸座標系→L 座標系）	MATRIX
	初期線形速度	VECTOR
	初期回転速度	VECTOR
	オブジェクト座標	BOOL
	反発	REAL
	摩擦	REAL

表 7-39 UI コンテナの非表示項目－(H/P/B/E)節データ

項目名		型
動力学データ	転がり摩擦	REAL
	回転摩擦	REAL
	線形ダンピング	REAL
	回転ダンピング	REAL
	線形速度の閾値	REAL
	回転速度の閾値	REAL
	不活性化遅延時間	REAL
	剛体の初期位置	VECTOR
	剛体の初期角度	VECTOR

UI 項目の入力範囲は対応する管理情報と同じとする。

表 7-40 に入力可能状態が変化する UI 項目とその条件を示す。

表 7-40 UI 項目と入力不可条件

UI 項目	入力不可となる条件
分離リグ アンカー	ジョイント/ルートが false。
ジョイント/リアル/アンカー	ジョイント/ルートが false。
ジョイント/リアル/H	ジョイント/拘束タイプが組み合わせヒンジでない。
ジョイント/リアル/P(HP)	ジョイント/拘束タイプが 3DOF。
軸/H 軸/全項目	ジョイント/リアル/H が入力不可、又はモデルの指定がない。
軸/P 軸/全項目	ジョイント/リアル/P(HP)が入力不可、又はモデルの指定がない。
軸/B 軸/全項目	ジョイント/リアル/B(HPB)が入力不可、又はモデルの指定がない。
軸/E 軸/全項目	ジョイント/リアル/E のモデルの指定がない。
軸/(H/P/B)軸/最小角度	軸/(H/P/B)軸/回転が角度制限でない。
軸/(H/P/B)軸/最大角度	軸/(H/P/B)軸/回転が角度制限でない。
軸/(H/P/B)軸/サーボ目標	軸/(H/P/B)軸/駆動がサーボでない。
軸/E 軸/サーボ目標	軸/E 軸/駆動がサーボでない。

(3) 外部関数

機能	Ⓢジョイント管理生成	
処理	新規メモリ領域にジョイント管理を生成する。	
出力	ジョイント管理	NodeData*

機能	Ⓢジョイント管理取得	
入力	モデル	const BaseObject*
処理	出力値=I.モデルにアタッチされたジョイント管理〔I.モデルにジョイント管理がアタッチされていない場合は nullptr〕	
出力	ジョイント管理	DynJointNode*

機能	Ⓢルート ジョイント取得	
入力	モデル	BaseObject*
処理	出力値=以下の条件を全て満たすモデル 条件 1：I.モデルとその祖先となるモデル 条件 2：ジョイント管理がアタッチされている、且つルート指定が true 条件 3：階層上、I.モデルに最も近いモデル	
出力	リンク チェーンのルート ジョイント モデル	BaseObject*

機能	ⓥUI コンテナの初期化（シミュレーション環境のみ）	
入力	タグ	GeListNode*
処理	I.タグ.UI コンテナ.* = 初期値	
出力	処理結果（常に true）	Bool

機能	ⓥメッセージ ハンドラ	
入力	タグ	GeListNode*
	タイプ	Int32
	データ	void*
処理	入力値から以下のトリガーを検出 トリガー別の処理 トリガー：処理 ロード：＜ロード＞(I.タグ，I.データ.空間管理) UI 項目値変更：＜UI 項目値変更＞(I.データ.UI 項目の ID，I.タグ，I.データ.空間管理) コマンド実行：＜コマンド実行＞(I.タグ，I.データ.コマンド ID) 初期化：＜初期化＞(I.タグ，I.タグ.空間管理)	
出力	処理結果（常に true）	Bool

機能	ⓥUI 項目の有効性取得（シミュレーション環境のみ）	
入力	タグ	GeListNode*
	UI 項目 ID	const DescID&
	パラメータ データ（未使用）	const GeData&
	フラグ（未使用）	DESCFLAGS_ENABLE
	ディスクリプション コンテナ（未使用）	const BaseContainer*
処理	表 7-40 の内容に従って、UI 項目の編集の不可設定値を決定する。	
出力	編集可否設定値（true：編集可能／false：編集不可）	Bool

機能	ⓥ実行タイミング追加（シミュレーション環境のみ）	
入力	タグ（未使用）	BaseTag*
	優先度リスト	PriorityList*
処理	I.優先度リスト.＜追加＞(I.タグ，EXECUTIONPRIORITY_ANIMATION，起動フラグ=0)	
出力	処理結果（常に true）	Bool

機能	ⓥ実行ハンドラ	
入力	タグ	BaseTag*
	空間管理	BaseDocument*
	モデル	BaseObject*
	スレッド（未使用）	BaseThread*
	優先度（未使用）	Int32
	起動フラグ（未使用）	EXECUTIONFLAGS
処理	＜状態遷移＞(I.モデル，I.空間管理，I.タグ.UI コンテナ) ＜ジョイント タイプ選択＞(I.タグ.UI コンテナ，I.空間管理) I.モデル == IK ジョイント ＜ジョイント回転制限同期＞(I.モデル，I.タグ.UI コンテナ) A.現在の時間フレーム=空間管理.時間フレーム A.現在の時間フレーム != #.前回イベント処理時の時間フレーム ＜回転・直動データ キャッシュ更新＞(I.タグ.UI コンテナ) #.前回イベント処理時の時間フレーム=A.現在の時間フレーム ＜モニター データ更新＞(I.タグ.UI コンテナ) 終了(EXECUTIONRESULT::OK)	
出力	実行結果(EXECUTIONRESULT::OK)	EXECUTIONRESULT

機能	ⓥ複製	
入力	複製先ジョイント管理	NodeData*
	複製元タグ（本関数では未使用）	GeListNode*
	複製先タグ（本関数では未使用）	GeListNode*
	複製フラグ（本関数では未使用）	COPYFLAGS
	エイリアス トランスレータ（本関数では未使用）	AliasTrans*
処理	出力値＝基本クラス定義の＜複製＞(I.*) 出力値 == true I.複製先スケール管理.状態!=初期の場合、アサート I.複製先スケール管理.状態=コピー	
出力	処理結果（true：正常終了/false：異常終了）	Bool

機能	ⓥ可視エレメント描画（シミュレーション環境のみ）	
入力	タグ	BaseTag*
	モデル	BaseObject*
	描画コンテキスト	BaseDraw*
	描画ヘルパー	BaseDrawHelp*
処理	I.描画コンテキスト.描画パス != オブジェクト、終了(true) I.タグ.UI コンテナ.重心と慣性主軸 == true ＜重心と慣性主軸を描画＞(I.タグ.UI コンテナ，I.モデル，I.描画コンテキスト，I.描画ヘルパー) ＜節を構成するモデルの X-Ray 化＞(I.タグ.UI コンテナ)	
出力	処理結果（常に true）	Bool

機能	ジョイント設定の有効確認	
処理	出力値=#.状態 == 有効 ? true : false	
出力	確認結果	Bool

機能	データ キャッシュ更新	
入力	空間管理	BaseDocument*
処理	＜データ キャッシュ更新＞(#.タグ.UI コンテナ，I.空間管理)	
出力	処理結果（常に true）	Bool

機能	ジョイント データ取得	
処理	出力値=#.ジョイント管理コア クラスの管理情報.ジョイント データ	
出力	ジョイント データ	const DynJointData*

機能	データ キャッシュ チェック	
入力	モデル	BaseObject*
	空間管理	BaseDocument*
処理	I.モデルがアタッチ先モデルと異なる場合、終了(出力値=false) @=H/P/B/E の処理 #.@軸節有効/無効 == true #.@軸節データ.実形状のルート モデル != #.タグ.UI コンテナ.ボディー.リアル.@ 終了(出力値=false) 出力値=true	
出力	処理結果	Bool

機能	初期の位置と角度を保存
処理	＜初期の位置と角度を保存＞(#.ジョイント モデル，#.タグ.UI コンテナ)

機能	位置と角度をリセット
処理	＜位置と角度をリセット＞(#.ジョイント モデル，#.タグ.UI コンテナ)

機能	剛体パージ	
入力	モデル（パージ対象の剛体モデル）	BaseObject*
処理	＜全節データ パージ＞(#.タグ.UI コンテナ，#.タグ.空間管理) #.タグ.UI コンテナ.状態=#.状態=無効	

(4) 内部関数

［基礎系］

機能	デフォルト コンストラクタ
処理	表 7-37 の初期値からジョイント管理を生成する。

機能	デストラクタ
処理	#.状態 == 有効 ＜剛体ページ＞(nullptr) ジョイント管理を破棄する。

機能	ロード	
入力	タグ：未使用	BaseTag*
	空間管理：未使用	BaseDocument*
処理	#.状態=ロード	

機能	ジョイント タイプ選択	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
	初期化指定（true：初期化／false：変化検知）	Bool
処理	I.初期化指定 == true && #.ジョイント タイプ != I.UI コンテナ.ジョイント タイプ #.ジョイント タイプ=I.UI コンテナ.ジョイント タイプ ＜全節データ ページ＞(I.UI コンテナ，I.空間管理) #.(ジョイント管理コア).＜ジョイント タイプに応じて節データ領域を構成＞() I.UI コンテナ.状態=#.状態=無効	

機能	全節データ ページ	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	@=H/P/B/E の処理 I.UI コンテナ. @節データ.節の実形状構成モデルのリスト.*.剛体管理.＜所属設定＞(nullptr) I.UI コンテナ. @節データ.節の実形状構成モデルのリストを破棄 #.(ジョイント管理コア).＜節データ ページ＞()	

機能	節構成モデルに所属を設定
処理	@=#.ジョイント データ.H/P/B/E 節データの処理 ＜節構成モデルに所属を設定＞(@，#.ジョイント データ.ジョイント モデル)

機能	節構成モデルに所属を設定	
入力	節データ	BaseContainer*
	モデル（ジョイント モデル）	BaseObject*
処理	@=I.節データ.実形状モデル配列[*]の処理 @.モデル.剛体管理.＜所属設定＞(I.モデル，@ == 配列の先頭要素 ? true : false)	

機能	初期の位置と角度を保存	
入力	ジョイント モデル	BaseObject*
	UI コンテナ	BaseContainer*
処理	@=H/P/B の処理 #.@節有効/無効 == 有効 && #.@節データ.実形状のルート モデル != nullptr I.UI コンテナ.@節データ.[初期位置、初期角度]=#.@節データ.実形状のルート モデルの位置と角度 #.E 節データ.実形状のルート モデル != nullptr I.UI コンテナ.E 節データ.[初期位置，初期角度]=#.E 節データ.実形状のルート モデルの位置と角度 I.UI コンテナ.[初期位置，初期角度]=I.ジョイント モデルの位置と角度	

機能	位置と角度をリセット	
入力	ジョイント モデル	BaseObject*
	UI コンテナ	BaseContainer*
処理	@=H/P/B の処理 #.@節有効/無効 == 有効 && #.@節データ.実形状のルート モデル != nullptr #.@節データ.実形状のルート モデルの位置と角度=I.UI コンテナ.@節データ.[初期位置，初期角度] #.E 節データ.実形状のルート モデル != nullptr #.E 節データ.実形状のルート モデルの位置と角度=I.UI コンテナ.E 節データ.[初期位置，初期角度] I.ジョイント モデルの位置と角度=I.UI コンテナ.[初期位置，初期角度]	

機能	状態遷移	
入力	モデル	BaseObject*
	空間管理	BaseDocument*
	UI コンテナ	BaseContainer
処理	状態別の処理	

状態	処理
初期	I.UI コンテナ.ステート=#.状態=無効 #.前回検出したアタッチ先のモデル=I.モデル
ロード	＜UI コンテナの全データ取得＞(I.UI コンテナ，I.空間管理) A.処理結果=＜節構成モデル配列のロード＞(I.UI コンテナ，I.空間管理) I.UI コンテナ.ステート=#.状態=A.処理結果 == 正常完了？有効：無効 #.前回検出したアタッチ先のモデル=I.モデル
コピー	＜節モデル データ キャッシュ更新＞(I.UI コンテナ，I.空間管理) A.処理結果=＜節構成モデル配列のロード＞(I.UI コンテナ，I.空間管理) ＜節構成モデルに所属を設定＞() I.UI コンテナ.ステート＝#.状態＝A.処理結果 == 正常完了？有効：無効 #.前回検出したアタッチ先のモデル=I.モデル
無効&有効	#.前回検出したアタッチ先のモデル != I.モデル ＜剛体ページ＞(nullptr) #.前回検出したアタッチ先のモデル=I.モデル

機能	重心と慣性主軸を描画	
入力	UI コンテナ	BaseContainer*
	モデル	BaseObject*
	描画コンテキスト	BaseDraw*
	描画ヘルパー：未使用	BaseDrawHelp*
処理	6.3 章(17)(A)項の内容に従って重心と慣性主軸を描画する。	

機能	節を構成するモデルの X-Ray 化	
入力	UI コンテナ	BaseContainer*
処理	A.X-Ray=I.UI コンテナ.X-Ray @=H/P/B の処理 #.@節有効/無効 == true @節データ.[実形状モデル配列，衝突判定形状モデル配列][*].モデル.X-Ray=A.X-Ray E 節データ.[実形状モデル配列，衝突判定形状モデル配列][*].モデル.X-Ray=A.X-Ray	

［データ系］

機能	UI コンテナの全データ取得	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	#.$= I.UI コンテナ.*	

機能	動力学データ保存	
入力	UI コンテナ	BaseContainer*
処理	@=H/P/B/E の処理 I.UI コンテナ.$〔非表示項目〕= #.@節データ.動力学データ.*	

機能	データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	＜モデル データ キャッシュ更新＞(I.UI コンテナ，I.空間管理) ＜動力学データ キャッシュ更新＞(I.UI コンテナ) ＜回転・直動データ キャッシュ更新＞(I.UI コンテナ) ＜可視化パラメータ キャッシュ更新＞(I.UI コンテナ) ＜スペシャル データ キャッシュ更新＞(I.UI コンテナ)	

機能	モデル データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	#.ジョイント モデル=アッタッチ先のモデル @=I.UI コンテナ/ボディー/[ルート，分離リグ，リアル/*，終端，フェイク/*，衝突判定無効]の処理 #.$=@	

機能	動力学データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
処理	@=H/P/B/E の処理 #.@節データ.動力学データ.* = I.UI コンテナ.$〔非表示項目〕	

機能	回転・直動データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
処理	@=H/P/B/E の処理 #.回転/直動モニター更新指定データ.@軸モニター更新指定=I.UI コンテナ.軸.モニター@ @=H/P/B の処理 #.@軸回転データ.[3~10]=I.UI コンテナ.$ #.E 軸直動データ.[3~10]=I.UI コンテナ.$	

機能	可視化パラメータ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
処理	#.デバッグ可視化データ.HPB 軸表示サイズ=I.UI コンテナ.表示/物理エンジン/HPB.サイズ #.デバッグ可視化データ.E 軸表示サイズ=I.UI コンテナ.表示/物理エンジン/E.サイズ	

機能	スペシャル データ キャッシュ更新	
入力	UI コンテナ	BaseContainer*
処理	#.演算誤差データ.*=I.UI コンテナ.スペシャル.*	

機能	モニター データ更新	
入力	UI コンテナ	BaseContainer*
処理	@=H/P/B の処理 #.回転モニター更新指定データ.@軸モニター更新指定 == true I.UI コンテナ.軸.@.[トルク、角速度]=#.@軸回転データ.カレント データ（トルクと角速度） #.直動モニター更新指定データ.@軸モニター更新指定 == true I.UI コンテナ.軸.E.[並進力、速度]=#.E 軸回転データ.カレント データ（フォースと速度）	

機能	節構成モデル配列の保存	
入力	UI コンテナ	BaseContainer*
処理	〔実形状モデルを R、衝突判定形状モデルを F と記載する〕 @=H/P/B/E の処理 I.UI コンテナ.@節データ.節の R リスト[*].モデル=#.@節データ.R 配列[*].モデル※1 I.UI コンテナ.@節データ.節の F リスト[*].モデル=#.@節データ.F 配列[*].モデル※1 ※1 処理異常の場合は、終了(出力値=false)	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	節構成モデル配列のロード	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	〔実形状モデルを R、衝突判定形状モデルを F と記載する〕 @=H/P/B/E の処理〔処理異常発生時は終了(出力値=false)〕 #.@節データ.R 配列[*]={ＳＣモデルＤ化※1}←I.UI コンテナ.@節データ.節の R リスト[*].モデル #.@節データ.F 配列[*]={ＳＣモデルＤ化※2}←I.UI コンテナ.@節データ.節の F リスト[*].モデル #.@節データ.モニター データ=#.@節データ.R 配列[0].モデル.剛体管理.＜剛体モニター データ取得＞() ＜位置参照モデル検知＞() 出力値=true ※1 SMDF::＜実形状ＳＣモデルＤ生成＞(モデル) ※2 SMDF::＜衝突判定形状ＳＣモデルＤ生成＞(モデル)	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	ジョイント回転制限同期	
入力	IK ジョイント	BaseObject*
	UI コンテナ	BaseContainer*
処理	@=H/P/B の処理 I.UI コンテナ.軸.@.[回転、最小角度、最大角度] = I.IK ジョイント.UI コンテナ.キネマティクス.@.[有効、最小角度、最大角度]	

［コマンド系］

機能	Ⓢ実形状を構成する要素の配列生成	
入力	ルート モデル	BaseObject*
	除外モデル（衝突判定形状モデル）	BaseObject*
	実形状構成要素の配列	BaseArray<DynJoint::JtBodyElmData>&
	終端モデル	BaseObject*
処理	I.ルート モデルから終端モデルまでの階層にある動的剛体※1のＳＣモデルD※2と動力学データ※3をI.実形状構成要素の配列に追加する。 ※1 動的剛体である条件 A.剛体管理=剛体管理::＜剛体管理取得＞(I.ルート モデルから終端モデルまでの階層のモデル) nullptr != A.剛体管理 && A.剛体管理.＜動的剛体確認＞() == true の場合、動的剛体とする ※2 ＳＣモデルDの生成方法 A.ＳＣモデルD=SMDF::＜実形状ＳＣモデルD生成＞(動的剛体のモデル) ※3 動力学データの取得方法 A.動力学データ= A.剛体管理.＜動力学データ取得＞()	
出力	処理結果（0：正常終了／0より大きな値：異常発生通知のリソース番号）	Int32

機能	Ⓢ衝突判定形状を構成する要素の配列生成	
入力	ルート モデル	BaseObject*
	衝突判定形状構成要素の配列	BaseArray<ScaledObject>&
処理	I.ルート モデル以降の階層にある衝突判定形状※1のＳＣモデルD※2をI.衝突判定形状構成要素の配列に追加する。 ※1 衝突判定形状モデルである条件 力学サポート::＜衝突判定形状モデル確認＞(I.ルート モデル以降の階層のモデル) == true の場合、衝突判定形状とする ※2 ＳＣモデルDの生成方法 A.ＳＣモデルD=SMDF::＜衝突判定形状ＳＣモデルD生成＞(衝突判定形状のモデル)	
出力	処理結果（0：正常終了／0より大きな値：異常発生通知のリソース番号）	Int32

機能	コマンド実行	
入力	タグ	GeListNode*
	コマンドID	Int32
処理	コマンドID別の処理	

コマンドID	処理
初期化	＜初期化＞(I.タグ、I.タグ.空間管理) ＜現在の角度を変更角度に設定＞(I.タグ)
リセット	＜リセット＞(I.タグ、I.タグ.空間管理) ＜現在の角度を変更角度に設定＞(I.タグ)
(角度の)現在値(取得)	＜現在の角度を変更角度に設定＞(I.タグ)
(角度)設定	＜正回転角度設定＞(I.タグ)
(角度)逆設定	＜逆回転角度設定＞(I.タグ)

機能	初期化	
入力	タグ	GeListNode*
	空間管理	BaseDocument*
	構成のみ指定（true：構成のみ／false：全処理）	Bool
処理	#.(単位変換データ).＜初期化＞() ＜ジョイント タイプ選択＞(I.タグ.UI コンテナ，I.空間管理，true) ＜モデル データ キャッシュ更新＞(I.タグ.UI コンテナ，I.空間管理) @=H/P/B/E の処理 #.@軸節有効/無効 == true A.節の終端モデル=#.(ジョイント管理コア).＜節の終端モデル取得＞(@) ＜節の初期化＞(A.節の終端モデル，#.@軸節データ，I.構成のみ指定)※1 ＜位置参照モデル検知＞() ＜節構成モデル配列の保存＞(I.タグ.UI コンテナ)※1 ＜動力学データ保存＞(I.タグ.UI コンテナ) 構成のみ指定 != true ＜初期の位置と角度を保存＞(#.ジョイント モデル，I.タグ.UI コンテナ) ＜節構成モデルに所属を設定＞(#.ジョイント モデル) 節の質量を確認※1〔各節の質量の合計が 0 の場合は異常とする。〕 I.タグ.UI コンテナ.状態=#.状態=有効 出力値=0 ※1 処理が異常終了した場合 ＜全節データ ページ＞(I.タグ.UI コンテナ，I.タグ.空間管理) 終了(出力値=異常発生通知用リソース ID)	
出力	処理結果（0：正常終了／異常発生通知用リソース ID）	Int32

機能	節の初期化	
入力	終端モデル	BaseObject*
	節データ	DynJointBodyData&
	構成のみ指定（true：構成のみ／false：全処理）	Bool
処理	＜実形状を構成する要素の配列生成＞(I.節データ.実形状のルート モデル、I.節データ.衝突判定形状のルート モデル, A.実形状構成モデルの配列生成、I.終端モデル)※1 ＜衝突判定形状を構成する要素の配列生成＞(I.節データ.衝突判定形状のルート モデル，A.衝突判定形状構成モデルの配列)※1 #.(ジョイント管理コア).＜節の初期化＞(I.節データ.実形状のルート モデルのグローバル マトリクス, A.実形状構成モデルの配列生成，A.衝突判定形状構成モデルの配列, I.節データ，I.構成のみ指定)※1 I.節データ.モニター データ=I.節データ.実形状モデル配列[0].剛体管理.＜剛体モニター データ取得＞() 出力値=0 ※1 処理が異常終了した場合、終了(出力値=異常発生通知用リソース ID)	
出力	0（正常終了）、又は異常発生通知用リソース ID	Int32

機能	リセット	
入力	タグ	GeListNode*
	空間管理	BaseDocument*
処理	＜モデル データ キャッシュ更新＞(I.タグ.UI コンテナ，I.空間管理) ＜位置と角度をリセット＞(#.ジョイント モデル、I.タグ.UI コンテナ)	

機能	現在の角度を変更角度に設定	
入力	タグ	GeListNode*
処理	I.タグ.UI コンテナ.コマンド.角度変更=#.(ジョイント データ).＜HPB 角度取得＞()	

機能	正回転角度設定	
入力	タグ	GeListNode*
処理	A.角度=I.タグ.UI コンテナ.コマンド.角度変更 #.分離リグ == true 　ジョイント モデルの角度=A.角度 そうではない 　＜正回転角度設定＞(A.角度)	

機能	正回転角度設定	
入力	角度	const Vector&
処理	#.ジョイント タイプ別の処理	

ジョイント タイプ	処理
組合せヒンジ	@=H/P/B の処理 #.@節有効/無効 == true && @節データ.実形状のルート モデル != nullptr @節データ.実形状のルート モデルの@軸の角度=I.角度.@軸の角度
カルダン+ヒンジ	#.P 節有効/無効 == true && P 節データ.実形状のルート モデル != nullptr P 節データ.実形状のルート モデルの H/P 軸の角度=I.角度.H/P 軸の角度 #.B 節有効/無効 == true && B 節データ.実形状のルート モデル != nullptr B 節データ.実形状のルート モデルの B 軸の角度=I.角度.B 軸の角度
3DOF	#.B 節有効/無効 == true && B 節データ.実形状のルート モデル != nullptr B 節データ.実形状のルート モデルの角度=I.角度

機能	逆回転角度設定	
入力	タグ	GeListNode*
処理	A.ルート ジョイント モデル=＜ルート ジョイント取得＞(I.タグ.モデル) A.ルート ジョイント モデル == nullptr の場合は処理を終了 〔@=A.ルート ジョイント モデル.ジョイント管理〕 @.分離リグ == true && @.アンカー モデル != nullptr && @.分離リグのアンカー モデル == nullptr && A.ルート ジョイント モデルの親 == nullptr 　異常発生通知後に処理を終了 A.角度=I.タグ.UI コンテナ.コマンド.角度変更 #.ジョイント タイプ別の処理	

ジョイント タイプ	処理
組合せヒンジ	＜組合せヒンジ逆回転角度設定＞(A.ルート ジョイント，A.角度)
カルダン+ヒンジ	＜カルダン＋ヒンジ逆回転角度設定＞(A.ルート ジョイント，A.角度)
3DOF	＜3DOF 逆回転角度設定＞(A.ルート ジョイント，A.角度)

#.分離リグ == true
　＜分離リグ逆回転角設定＞(I.タグ.モデル，A.ルート ジョイント，A.角度)

機能	組合せヒンジ逆回転角度設定	
入力	ルート ジョイント モデル	BaseObject*
	設定角度	const Vector&
処理	〔ルート ジョイント モデル.ジョイント管理を root と記載する〕 A.Root = root.アンカー モデル != nullptr ? root.アンカー モデル ：root.＜先頭の回転軸取得＞() A.Root == nullptr、終了 A.RootMg = A.Root の Mg A.First = ＜先頭の回転軸取得＞() A.First == nullptr、終了 A.Last = ＜末尾の回転軸取得＞() A.RootMl = A.First の Mg の逆行列×A.Root の Mg @= A.Last から A.First の次の軸までの処理〔ジョイントを逆階層にした Ml を求める〕 　A.@の前の軸の Ml = @の Mg の逆行列×@の前の軸の Mg A.Mg = A.Last の Mg A.First の前の軸の Ml = A.RootMl @= A.Last から A.First までの処理 　A.Mg = A.Mg×(I.設定角度.@の軸の角度×－１)の回転行列×A.@の前の軸の Ml A.Root の Mg = A.Mg ＜正回転角度設定＞(I.設定角度)	

機能	カルダン＋ヒンジ逆回転角度設定	
入力	ルート ジョイント モデル	BaseObject*
	設定角度	const Vector&
処理	〔ルート ジョイント モデル.ジョイント管理を root と記載する〕 A.Root = root.アンカー モデル != nullptr ? root.アンカー モデル ：root.＜先頭の回転軸取得＞() A.Root == nullptr、終了 A.RootMg = A.Root の Mg A.First = ＜先頭の回転軸取得＞() A.First == nullptr、終了 A.Last = ＜末尾の回転軸取得＞() A.RootMl = A.First の Mg の逆行列×A.Root の Mg @= A.Last から A.First の次の軸までの処理〔ジョイントを逆階層にした Ml を求める〕 　A.@の前の軸の Ml = @の Mg の逆行列×@の前の軸 Mg A.Mg = A.Last の Mg A.First の前の軸の Ml = A.RootMl @= A.Last から A.First までの処理 　@ == B 軸 　　A.Mg= A.Mg×(I.設定角度.B×－１)の回転行列×A.@の前の軸の Ml 　@ == HP 軸 　　A.Mg= A.Mg×(I.設定角度.HP×－１)の回転行列〔回転順序は P/H〕 ×A.@の前の軸の Ml A.Root の Mg = A.Mg ＜正回転角度設定＞(I.設定角度)	

機能	3DOF 逆回転角度設定	
入力	ルート ジョイント モデル	BaseObject*
	設定角度	const Vector&
処理	＜B 軸存在確認＞() != true、終了 〔ルート ジョイント モデル.ジョイント管理を root と記載する〕 A.Root = root.アンカー モデル != nullptr ? root.アンカー モデル ：root.＜先頭の回転軸取得＞() A.Root == nullptr、終了 A.RootMg = A.Root の Mg A.AxisMg = #.B 節データ.実形状のルート モデルの Mg A.InvAxisMg = A.AxisMg の逆行列 A.RootMl = A.InvAxisMg×A.RootMg A.AxisMg = A.AxisMg×(I.設定角度)の回転行列〔回転順序は BPH〕 A.RootMg = A.AxisMg×A.RootMl A.Root の Mg = A.RootMg ＜正回転角度設定＞(I.設定角度)	

機能	分離リグ逆回転角度設定	
入力	ジョイント モデル	BaseObject*
	ルート ジョイント モデル	BaseObject*
	設定角度	const Vector&
処理	〔I.ルート ジョイント モデル.ジョイント管理を root と記載する〕 root.アンカー モデル != nullptr 　A.分離リグ アンカー モデル= 　　root.分離リグのアンカー モデル != nullptr ? 　　R.分離リグのアンカー モデル : ルート ジョイント モデルの親 　A.分離リグ アンカー モデルの Mg = root.アンカー モデルの Mg root.アンカー モデル == nullptr 　I.ルート ジョイント モデルの Mg.off = root.位置参照モデルの Mg.off 　I.ルート ジョイント モデルの角度 = root.＜HPB 角度取得＞() I.ジョイント モデルの角度=I.設定角度	

［UI 系］

機能	UI 項目値変更	
入力	UI 項目 ID	Int32
	タグ	GeListNode*
	空間管理：未使用	BaseDocument*
処理	@=[図 7-20 から図 7-23 中の入力変更不可の項目]の処理 I.UI 項目 ID == @.ID I.タグ.UI コンテナ.[I.UI 項目 ID の項目]=#.$〔入力変更を取り消す〕	

7.6 物理ワールド管理

ジョイント管理を実現する図 7-2 に示した階層のクラスに実装する管理情報と関数を次章以降に示す。

7.6.1 物理ワールド管理コア クラス

動作環境に依存しない物理ワールド管理を実現する物理ワールド管理コア クラス(DynWorld：protected DynUnitCnv)の構成を示す。

(1) 管理情報

表 7-41 物理ワールド管理コア クラスの管理情報

番号	項目	初期値	型
1	物理シミュレーション パラメータ(型)Ⓔ※1	－	DynSimParams
2	ジョイント候補データ(型)Ⓔ※1	－	DynJCandidateD
3	物理ワールド要素候補データ(型)Ⓔ※1	－	Candidates
4	デバッグ描画指定データ(型)Ⓔ※1	－	DebugDraw
5	警告番号(型)Ⓔ 0：警告なし 1：初期化（座標変換の固定） 2：初期化（分離リグの位置参照モデルなし）	－	enum class WarningId
6	警告データ(型)Ⓔ※1	－	WarningInfo
7	警告配列Ⓟ	＜－＞	BaseArray<WarningInfo>

※1 表 7-42～表 7-46 を参照のこと。

(2) 外部関数

機能	Ⓢ参照を追加	
入力	物理ワールド管理コア	DynWorld*&
処理	I.物理ワールド管理コア != nullptr I.物理ワールド管理コア.＜参照を追加※1＞() ※1 rbxSys::RefSingleThrMng クラスの関数	

機能	Ⓢ物理ワールド管理コア削除	
入力	物理ワールド管理コア	DynWorld*&
処理	I.物理ワールド管理コア != nullptr I.物理ワールド管理コア.＜参照を削除※1＞() I.物理ワールド管理コアを削除 I. 物理ワールド管理コア= nullptr ※1 rbxSys::RefSingleThrMng クラスの関数	

機能	Ⓥ物理エンジン種別取得	
出力	物理エンジン種別	PE

機能	Ⓥ物理ワールド初期化	
入力	物理ワールド モデル	BaseObject*
	単位変換データ	const DynUnitCnv&
	物理ワールド要素候補データ	const Candidates&

機能	Ⓥ動的剛体削除	
入力	削除する動的剛体モデルの配列	const BaseArray<IndexedModel>&

機能	Ⓥ静的剛体削除	
入力	削除する静的剛体モデルの配列	const BaseArray<IndexedModel>&

機能	ⓥモニター データ無効化	
入力	モニター データ	const DynBodyMonData*

機能	ⓥシミュレーション進行	
入力	物理シミュレーション パラメータ	const DynSimParams&

機能	ⓥシミュレーション結果更新
処理	無処理

機能	ⓥクリーン アップ
処理	無処理

機能	ⓥデバッグ描画	
入力	描画コンテキスト	BaseDraw*
	デバッグ描画指定データ	const DebugDraw&
処理	無処理	

機能	警告数取得	
処理	出力値=#.警告配列.要素数	
出力	警告数	Int32

機能	警告配列取得	
処理	出力値=#.警告配列	
出力	警告配列	const BaseArray<WarningInfo>&

(3) 保護関数

機能	コンストラクタ	
処理	無処理	

機能	デストラクタ	
処理	無処理	

7.6.1.1 物理シミュレーション パラメータ

物理ワールド管理コア クラス::物理シミュレーション パラメータ(DynWorld::DynSimParams)の構成を示す。

(1) 管理情報

表 7-42 物理シミュレーション パラメータの管理情報

番号	項目	初期値	型
1	物理シミュレーション パラメータ(型)Ⓔ	－	DynSimParams
1.1	ステップ時間(秒)	＜・＞	Float
1.2	重力加速度(m/s²)	＜・＞	Float
1.3	最大演算回数	10	Int32
1.4	時間フレーム当たりのステップ数	1	Int32

(2) 外部関数

機能	コンストラクタ	
入力	ステップ時間(秒)	Float
	重力加速度(m/s²)	Float
	最大演算回数（デフォルト=10）	Int32
	時間フレーム当たりのステップ数（デフォルト=1）	Int32
処理	#$=I.*	

7.6.1.2 ジョイント候補データ

物理ワールド管理コア クラス::ジョイント候補データ(DynWorld::DynJCandidateD)の構成を示す。

(1) 管理情報

表 7-43 ジョイント候補データの管理情報

番号	項目	初期値	型
1	ジョイント候補データ(型)Ⓔ	－	DynJCandidateD
1.1	ジョイント データ	nullptr	const DynJointData*
1.2	親ジョイント候補データの番号	-1	Int32
1.3	ジョイント関連データ番号	-1	mutable Int32
1.4	親ジョイント候補データ	nullptr	mutable const DynJCandidateD*

(2) 外部関数

機能	コンストラクタ
処理	#.*=表 7-43 のジョイント候補データ型の初期値

7.6.1.3 物理ワールド要素候補データ

物理ワールド管理コア クラス::物理ワールド要素候補データ(DynWorld::Candidates)の構成を示す。

(1) 管理情報

表 7-44 物理ワールド要素候補データの管理情報

番号	項目	初期値	型
1	物理ワールド要素候補データ(型)Ⓔ	－	Candidates
1.1	動的剛体候補配列	＜・＞	BaseArray<DynBodyData>
1.2	静的剛体候補配列	＜・＞	BaseArray<DynBodyData>
1.3	ジョイント候補配列	＜・＞	BaseArray<DynJCandidateD>

(2) 外部関数

機能	コンストラクタ
処理	#.*.配列のサイズ=250

7.6.1.4 デバッグ描画指定データ

物理ワールド管理コア クラス::デバッグ描画指定データ(DynWorld::DebugDraw)の構成を示す。

(1) 管理情報

表 7-45 デバッグ描画指定データの管理情報

番号	項目	初期値	型
1	デバッグ描画指定データ(型)Ⓔ	－	DebugDraw
1.1	ワイヤー フレーム	＜・＞	Bool
1.2	AABB	＜・＞	Bool
1.3	接触点	＜・＞	Bool
1.4	拘束	＜・＞	Bool
1.5	拘束の範囲	＜・＞	Bool
1.6	ノーマル ベクター	＜・＞	Bool

(2) 外部関数

機能	コンストラクタ	
入力	ワイヤー フレーム	Bool
	AABB	Bool
	接触点	Bool
	拘束	Bool
	拘束の範囲	Bool
	ノーマル ベクター	Bool
処理	#.$=I.*	

機能	デバッグ描画確認	
処理	出力値=#.*の何れか == true ? true : false	
出力	確認結果（true：デバッグ描画の必要あり／false：デバッグ描画の必要なし）	Bool

7.6.1.5 警告データ

物理ワールド管理コア クラス::警告データ(DynWorld::WarningInfo)の構成を示す。

(1) 管理情報

表 7-46 警告データの管理情報

番号	項目	初期値	型
1	警告データ(型)Ⓔ	－	WarningInfo
1.1	警告検知日時	＜－＞	LocalDateTime
1.2	警告対象のモデルの名前	＜－＞	U8String
1.3	警告番号	0	WarningId

(2) 外部関数

機能	デフォルト コンストラクタ	
処理	#.*=表 7-46 のジョイント警告データの初期値	

機能	コンストラクタ	
入力	警告対象のモデルの名前	U8String
	警告番号	WarningId
処理	＜警告設定＞(I.*)	

機能	警告設定	
入力	警告対象のモデルの名前	U8String
	警告番号	WarningId
処理	#.警告検知日時=現在の日時 #.$=I.*	

機能	警告確認	
処理	出力値=#.警告番号 != 0 ? true : false	
出力	確認結果（true：警告あり／false：警告なし）	Bool

7.6.2 Bullet 物理ワールド管理コア クラス

Bullet 物理演算エンジン用の物理ワールド管理を実現する Bullet 物理ワールド管理コア クラス(DynWorldBt：public DynWorld)の構成を示す。

(1) 管理情報

表 7-47 Bullet 物理ワールド管理コア クラスの管理情報

番号	項目	初期値	型
1	剛体関連データ(型)※1	－	RbRelation
2	ジョイント関連データ(型)※1	－	JtRelation
3	ヒンジ フレーム データ(型)※1	－	HingeFrame
4	スライダー フレーム データ(型)※1	－	SliderFrame
5	カルダン フレーム データ(型)※1	－	CardanFrame
6	3DOF フレーム データ(型)※1	－	Ge3DofFrame
7	動的剛体関連配列	＜－＞	BaseArray<RbRelation>
8	静的剛体関連配列	＜－＞	BaseArray<RbRelation>
9	ジョイント関連配列	＜－＞	BaseArray<JtRelation>
10	Bullet 基幹オブジェクト	－	－
10.1	衝突検出(広域)	＜－＞	btDbvtBroadphase
10.2	衝突検出の構成	＜－＞	btDefaultCollisionConfiguration
10.3	衝突検出(狭域)ﾃﾞｨｽﾊﾟｯﾁｬｰ	＜－＞	btCollisionDispatcher
10.4	ソルバー	＜－＞	btSequentialImpulseConstraintSolver
10.5	物理ワールド	＜－＞	btDiscreteDynamicsWorld

※1 表 7-48～表 7-53 を参照のこと。

(2) 外部関数

機能	Ⓢ物理ワールド管理コア生成	
処理	新規メモリ領域に物理ワールド管理コアを生成する。	
出力	物理ワールド管理コア	DynWorldBt*

機能	Ⓥ物理エンジン種別取得	
処理	出力値＝PE::Bullet	
出力	物理エンジン種別	PE

機能	Ⓥ物理ワールド初期化	
入力	物理ワールド モデル	BaseObject*
	単位変換データ	const DynUnitCnv&
	物理ワールド要素候補データ	const Candidates&
処理	#.警告配列.サイズ=0 #.(単位変換データ).＜単位変換データ設定＞(I.単位変換データ) ＜クリーン アップ＞() ＜動的剛体追加＞(物理ワールド要素候補データ.動的剛体候補配列)※1 ＜静的剛体追加＞(物理ワールド要素候補データ.静的剛体候補配列) ※1 ＜ジョイントと節追加＞(物理ワールド要素候補データ.ジョイント候補配列) ※1 出力値=true ※1 処理で異常が発生した場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

機能	ⓥ動的剛体削除	
入力	削除対象剛体配列（配列要素のインデックス番号は昇順）	const BaseArray<IndexedModel>&
処理	@=削除対象剛体配列[*]の処理〔降順に処理〕 A.Bullet 剛体=#.動的剛体関連配列[@.インデックス番号].Bullet 剛体 #.Bullet 物理ワールド.＜剛体を除去＞(A.Bullet 剛体) A.Bullet 剛体.動作状況データを削除 A.Bullet 剛体.衝突判定形状を削除 A.Bullet 剛体を削除 #.動的剛体関連配列.＜要素を除去＞(@.インデックス番号)	

機能	ⓥ静的剛体削除	
入力	削除対象剛体配列（配列要素のインデックス番号は昇順）	const BaseArray<IndexedModel>&
処理	@=削除対象剛体配列[*]の処理〔降順に処理〕 A.Bullet 剛体=#.静的剛体関連配列[@.インデックス番号].Bullet 剛体 #.Bullet 物理ワールド.＜剛体を除去＞(A.Bullet 剛体) A.Bullet 剛体.動作状況データを削除 A.Bullet 剛体.衝突判定形状を削除 A.Bullet 剛体を削除 #.静的剛体関連配列.＜要素を除去＞(@.インデックス番号)	

機能	ⓥモニター データ無効化	
入力	モニター データ	const DynBodyMonData*
処理	@=動的剛体関連配列[*]と静的剛体関連配列[*]の処理 @.モニター データ == I.モニター データ、@.モニター データ=nullptr	

機能	ⓥシミュレーション進行	
入力	物理シミュレーション パラメータ	const DynSimParams&
処理	〔物理シミュレーション パラメータを I と記載する〕 A.ソルバー情報 =#.ソルバー.＜ソルバー情報取得＞() A.ソルバー情報.最大演算回数 = I.最大演算回数 #.Bullet 物理ワールド.＜重力設定＞(btVector3(0，#.MtoU(-I. 重力加速度)，0)) ＜ジョイント駆動設定更新＞() I.時間フレーム当たりのステップ数回の繰り返し処理 #.Bullet 物理ワールド.＜シミュレーション進行＞(I.ステップ時間，0，0)	

機能	ⓥシミュレーション結果更新
処理	A.単位変換データ=#.(単位変換データ).＜読み出し専用単位変換データ取得＞() @=#.動的剛体関連配列[*]の処理 @.＜ネイティブ モデルの位置と角度更新＞(A.単位変換データ) @=#.ジョイント関連配列[*]の処理 @.＜ネイティブ モデルの位置と角度更新＞(A.単位変換データ)

機能	ⓥクリーン アップ
処理	#.動的剛体関連配列.サイズ=0 #.静的剛体関連配列.サイズ=0 #.ジョイント関連配列.サイズ=0 @=#.Bullet 物理ワールド.ジョイント[*]の処理 #.Bullet 物理ワールド.＜ジョイントを除去＞(@) @を削除 @=#.Bullet 物理ワールド.剛体 [*]の処理 #.Bullet 物理ワールド.＜剛体を除去＞(@) @.動作状況データを削除 @.衝突判定形状を削除 @を削除

機能	ⓥデバッグ描画	
入力	描画コンテキスト	BaseDraw*
	デバッグ描画指定データ	const DebugDraw&
処理	A.Bullet デバッグ描画指定データ={btIDebugDraw::DebugDrawModes へ変換}←I.デバッグ描画指定データ Bullet デバッグ描画クラス::＜コンストラクタ＞(#.Bullet 物理ワールド，A.Bullet デバッグ描画指定データ，I.描画コンテキスト)	

(3) 内部関数

機能	コンストラクタ
処理	Bullet 基幹オブジェクトを初期化 ポリゴン メッシュ同士の衝突検知を有効化※1 ※1 btGImpactCollisionAlgorithm::registerAlgorithm(衝突検出(狭域)ﾃﾞｨｽﾊﾟｯﾁｬｰ)

機能	デストラクタ
処理	＜クリーン アップ＞()

処理	動的剛体追加	
入力	動的剛体候補配列	const BaseArray<DynBodyData>&
処理	#.動的剛体関連配列.サイズ=I.動的剛体候補配列.サイズ※1 @=0 から動的剛体候補配列数-1 の処理 A.剛体データ=動的剛体候補配列[@] A.Bullet 剛体=＜剛体生成＞(剛体データ)※2 #.Bullet 物理ワールド.＜剛体追加＞(A.Bullet 剛体) A.剛体データ.Ml 有効・無効 == false 動的剛体関連配列[@].＜コンストラクタ＞(A.剛体データ.実形状のＳＣモデルＤ.モデル, A.剛体データ.剛体モニター データ，A.Bullet 剛体) そうではない 動的剛体関連配列[@].＜コンストラクタ＞(A.剛体データ.実形状のＳＣモデルＤ.モデル, A.剛体データ.慣性主軸 Ml，A.剛体データ.L 原点 Ml, A.剛体データ.剛体モニター データ，A.Bullet 剛体) ※1 配列のサイズ変更できない場合は終了(出力値=false) ※2 剛体を生成できない場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

処理	静的剛体追加	
入力	静的剛体候補配列	const BaseArray<DynBodyData>&
処理	#.静的剛体関連配列.サイズ=I.静的剛体候補配列.サイズ※1 @=0 から静的剛体候補配列数-1 の処理 A.剛体データ=静的剛体候補配列[@] A.Bullet 剛体=＜剛体生成＞(剛体データ)※2 #.Bullet 物理ワールド.＜剛体追加＞(A.Bullet 剛体) 静的剛体関連配列[@].＜コンストラクタ＞(A.剛体データ.実形状のＳＣモデルＤ.モデル，nullptr，A.Bullet 剛体) ※1 配列のサイズ変更できない場合は終了(出力値=false) ※2 剛体を生成できない場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

処理	ジョイントと節追加	
入力	ジョイント候補配列	const BaseArray<DynJCandidateD>&
処理	〔ジョイントを Jt と記載する〕 #.Jt 関連配列.サイズ=I.Jt 候補配列.サイズ※1 @1=0 から Jt 候補配列数-1 の処理 A.Jt 候補=I.Jt 候補配列数[@1] A.Jt 候補.親 Jt 候補データの番号 != -1 A.Jt 候補.親 Jt 候補データ=I.Jt 候補配列[A.Jt 候補.親ジョイント候補データの番号] A.Jt 関連=#.ジョイント関連配列[@1] ＜ジョイントと節を生成＞(A.Jt 候補，A.Jt 関連)※2 @2=nullptr 以外の A.Jt 関連.[H/P/B/E 節剛体関連データ].Bullet 剛体の処理 #.Bullet 物理ワールド.＜剛体追加＞(@2) @2=nullptr 以外の A.Jt 関連.[H/P/B/E 軸拘束].Bullet 拘束の処理 #.Bullet 物理ワールド.＜ジョイント追加＞(@2) A.Jt 候補.ジョイント関連データ番号=@1 ※1 配列のサイズ変更できない場合は終了(出力値=false) ※2 ジョイントと節を生成できない場合は終了(出力値=false)	
	処理結果（true： 正常終了 / false： 異常終了）	Bool

機能	剛体生成	
入力	剛体データ	const DynBodyData&
処理	〔I.剛体データ.実形状のＳＣモデルＤを実ＳＣモデルＤ、 I.剛体データ.衝突判定形状のＳＣモデルＤを偽ＳＣモデルＤ、 I.剛体データ.動力学データを動力学データと記載する〕 A.bullet 衝突判定形状=＜衝突判定形状生成＞(I.偽ＳＣモデルD)※1 A.bullet 衝突判定形状.＜マージン設定＞(I.動力学データ.衝突マージン) A.剛体の慣性主軸 Mg=I.実ＳＣモデルD.モデル.Mg I.動力学データ.Ml 有効・無効 == true A.剛体の慣性主軸 Mg＝A.剛体の慣性主軸 Mg×I.動力学データ.慣性主軸 Ml A.bullet 複合形状※2 を生成※1 A.Ml= I.実ＳＣモデルD.モデル == I.偽ＳＣモデルD.モデル ？ 動力学データ.L 原点 Ml： Inv(A.剛体の慣性主軸 Mg)×I.偽ＳＣモデルD.モデル.Mg A. bullet 複合形状.＜形状を追加＞(btTransform({座標系変換※3}←A.Ml.sqmat)，{座標系変換※4}←A.Ml.off)， A.bullet 衝突判定形状) A.bullet 衝突判定形状=A.bullet 複合形状 そうではない、I.実ＳＣモデルD.モデル != I.偽ＳＣモデルD.モデル A.bullet 複合形状※2 を生成※1 A.Ml=Inv(A.剛体の慣性主軸 Mg)×I.偽ＳＣモデルD.モデル.Mg A. bullet 複合形状.＜形状を追加＞(btTransform({座標系変換※3}←A.Ml.sqmat)，{座標系変換※4}←A.Ml.off)， A.bullet 衝突判定形状) A.bullet 衝突判定形状=A.bullet 複合形状 出力値=＜剛体生成＞(A.剛体の慣性主軸 Mg，A.bullet 衝突判定形状，I.動力学データ) 出力値 == nullptr、A.bullet 衝突判定形状を削除 ※1 処理異常の場合は終了(出力値=nullptr) ※2 btCompoundShape ※3 左手・行ベクトル系から右手・列ベクトル系へ変換 ※4 左手系から右手系へ変換	
出力	Bullet 剛体データ（処理異常の場合は nullptr）	btRigidBody*

機能	剛体生成	
入力	慣性主軸の Mg	const Matrix&
	衝突判定形状	btCollisionShape*
	動力学データ	const DynData&
処理	〔#.(単位変換データ).メートル単位 → システム単位変換係数を MtoU、 #.(単位変換データ).メートル単位の 2 乗 → システム単位 2 乗変換係数を MtoUsq と記載する〕 A.運動状況の領域生成※1(btTransform({座標系変換※2}←I.慣性主軸の Mg.sqmat), {座標系変換※3}←I.慣性主軸の Mg.off)) A.剛体生成情報※4 を生成(I.動力学データ.質量, A.運動状況, I.衝突判定形状, I.動力学データ.主慣性モーメント×MtoUsq×I.動力学データ.慣性モーメント L スケール 2) A.Bullet 剛体=Bullet 剛体を生成(A.剛体生成情報)※5 A.Bullet 剛体.$=I.動力学データ.(運動条件データ).[4~9] I.動力学データ.＜動的性質確認＞() == true A.Bullet 剛体.初期線形速度=I.動力学データ.(運動条件データ).初期線形速度×MtoU A.Bullet 剛体.初期回転速度=I.動力学データ.(運動条件データ).初期回転速度 A.Bullet 剛体.$=I.動力学データ.(物理演算不活性化データ).[*] I.動力学データ.＜物理演算不活性化無効確認＞() == true A.Bullet 剛体.＜強制活性化設定＞(不活性化無効) A.Bullet 剛体.＜活性化＞() ※1 new btDefaultMotionState〔処理異常の場合、終了(出力値=nullptr)〕 ※2 左手・行ベクトル系から右手・列ベクトル系へ変換 ※3 左手系から右手系へ変換 ※4 btRigidBody::btRigidBodyConstructionInfo ※5 処理異常の場合は A.運動状況を削除後、終了(出力値=nullptr)	
出力	Bullet 剛体（処理異常の場合は nullptr）	btRigidBody*

機能	衝突判定形状生成	
入力	スケールド モデル データ	const ScaledObject&
処理	出力値=表 6-12 の内容に従って衝突判定用形状を生成	
出力	Bullet 形状（処理異常の場合は nullptr）	btCollisionShape*

<table>
<tr><td>機能</td><td colspan="2">ジョイントと節を生成</td></tr>
<tr><td rowspan="2">入力</td><td>ジョイント候補データ</td><td>const DynJCandidateD&</td></tr>
<tr><td>ジョイント関連データ</td><td>JtRelation&</td></tr>
<tr><td>処理</td><td colspan="2">〔I.ジョイント候補データ.ジョイント データを Jt データ、I.ジョイント関連データを Jt 関連と記載する〕
I.Jt 関連.ジョイント モデル=I.Jt データ.ジョイント モデル
I.Jt 関連.回転/直動モニター更新指定データ=I.Jt データ.回転/直動モニター更新指定データ

@=H/P/B/E の処理〔Jt 関連.[@節剛体関連データ、@軸回転/直動データ]を設定〕
I.Jt データ.@節データ.実形状のルート モデル != nullptr
I.Jt 関連.@節剛体関連データ.モデル=I.Jt データ.@節データ.実形状のルート モデル
I.Jt 関連.@節剛体関連データ.Bullet 剛体=＜節生成＞(I.Jt データ.@節データ)※1
I.Jt 関連.@節剛体関連データ.Ml 有効・無効=I.Jt データ.@節データ.Ml 有効・無効
I.Jt 関連.@節剛体関連データ.慣性主軸 Ml=I.Jt データ.@節データ.慣性主軸 Ml
I.Jt 関連.@節剛体関連データ.L 原点 Ml=I.Jt データ.@節データ.L 原点 Ml
I.Jt 関連.@節剛体関連データ.モニター データ=I.Jt データ.@節データ.モニター データ
I.Jt 関連.@軸[回転/直動]※2 データ=I.Jt データ.@軸[回転/直動]※2 データ
そうではない
@と I.Jt データ.ジョイント タイプの組み合わせ別の処理
<table>
<tr><th>@</th><th>ジョイント タイプ</th><th>処理</th></tr>
<tr><td>H</td><td>組合せヒンジ以外</td><td>I.Jt 関連.H 軸回転データ=I.Jt データ.H 軸回転データ</td></tr>
<tr><td>P</td><td>3DOF</td><td>I.Jt 関連.P 軸回転データ=I.Jt データ.P 軸回転データ</td></tr>
<tr><td colspan="2">上記以外の組み合わせ</td><td>無処理</td></tr>
</table>
連結する剛体を特定する処理
A.RelA = nullptr
I.Jt データ.ルート指定 == true
I.Jt データ.アンカー モデル != nullpt
A.RelA = ＜アンカーの剛体関連データ取得＞(I.Jt データ.アンカー モデル)
そうではない、I.Jt データ.親ジョイント候補データ != nullptr
A.親 Jt 関連番号= I.Jt データ.親ジョイント候補データ.ジョイント関連データ番号
A.親 Jt 関連番号 != -1
I.Jt 関連.前節剛体関連データ= A.RelA =
#.ジョイント関連配列[A.親 Jt 関連番号].＜末尾の節の剛体関連データ取得＞()

I.Jt 関連.ジョイント タイプ=I.Jt データ.ジョイント タイプ
I.Jt 関連.ルート指定=I.Jt データ.ルート指定
I.Jt 関連.分離リグ指定= I.Jt データ.分離リグ指定
I.Jt 関連.位置参照モデル=I.Jt データ.位置参照モデル※3

I.関連.ジョイント タイプ別の処理
<table>
<tr><th>ジョイント タイプ</th><th>処理</th></tr>
<tr><td>組合せヒンジ</td><td>＜組み合わせヒンジ＋スライダー ジョイント生成＞(A.RelA, I.Jt 関連, I.Jt データ)</td></tr>
<tr><td>カルダン＋ヒンジ</td><td>＜カルダン＋ヒンジ＋スライダー ジョイント生成＞(A.RelA, I.Jt 関連, I.Jt データ)</td></tr>
<tr><td>3DOF</td><td>＜3DOF＋スライダー ジョイント生成＞(A.RelA, I.Jt 関連, I.Jt データ)</td></tr>
</table>
I.Jt 関連.＜モニター データ初期化＞()
出力値=true

※1 以下の異常処理を実行した後、終了(出力値=false)
@=H/P/B/E の処理
I.Jt 関連.@節剛体関連データ.Bullet 剛体を削除
I.Jt 関連.@軸拘束を削除
※2 @=E の場合は直動、それ以外の場合は回転
※3 分離リグの位置参照モデルが存在しない場合は #.警告配列 に警告を追加する</td></tr>
<tr><td>出力</td><td>処理結果（true：正常終了／false：異常終了）</td><td>Bool</td></tr>
</table>

機能	節生成	
入力	節データ	const DynJointBodyData&
処理	〔#.(単位変換データ).メートル単位 → システム単位変換係数を MtoU と記載する〕 I.節データ.衝突判定形状モデル配列.サイズ ＞0 　A.bullet 形状=衝突判定形状モデル配列の全モデルで衝突判定用形状を生成※1 そうではない 　A.bullet 形状=実形状モデル配列の全モデルで衝突判定用形状を生成※1 A.bullet 形状.＜マージン設定＞(I.節データ.(動力学データ).衝突マージン×MtoU) 出力値=＜剛体生成＞(　I.節データ.実形状のルート モデル.Mg×I.節データ.(並進・回転性質データ).慣性主軸 Ml, 　A.bullet 衝突判定形状, 　I.節データ.(動力学データ)) 出力値 == nullptr、A.bullet 形状を削除 ※1 表 6-13 と表 6-12 を参照のこと。〔処理異常の場合は、終了(出力値=nullptr)〕	
出力	Bullet 剛体（処理異常の場合は nullptr）	btRigidBody*

機能	組み合わせヒンジ＋スライダー ジョイント生成	
入力	剛体 A の剛体関連データ	const RbRelation*
	ジョイント関連データ	JtRelation&
	ジョイント データ	const DynJointData&
処理	〔I.剛体 A の剛体関連データを A 関連、I.ジョイント関連データを Jt 関連、 　I.ジョイント データを Jt データと記載する〕 @=H/P/B 軸の処理〔@軸のヒンジ拘束を生成する処理〕 　I.Jt 関連.@節剛体関連データ.Bullet 剛体 != nullptr 　　I.A 関連 != nullptr 　　　I.Jt 関連.@軸拘束=＜ヒンジ拘束生成＞(　　　　rbxSys::Axis::@, 　　　　I.A 関連, 　　　　I.Jt 関連.@節剛体関連データ, 　　　　I.Jt データ.@軸回転データ, 　　　　I.Jt データ.演算誤差データ, 　　　　I.Jt データ.デバッグ可視化データ)※1 　　I.A 関連=I.Jt 関連.@節剛体関連データ E 軸の処理〔E 軸のスライダー拘束を生成する処理〕 　I.Jt 関連.E 節剛体関連データ.Bullet 剛体 != nullptr 　　I.A 関連 != nullptr 　　　I.Jt 関連.E 軸拘束=＜スライダー拘束生成＞(　　　　I.A 関連, 　　　　I.Jt 関連.E 節剛体関連データ, 　　　　I.Jt データ.E 軸直動データ, 　　　　I.Jt データ.演算誤差データ, 　　　　I.Jt データ.デバッグ可視化データ)※1 出力値=true ※1 処理異常の場合は、終了(出力値=false)	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	カルダン＋ヒンジ＋スライダー ジョイント生成	
入力	剛体 A の剛体関連データ	const RbRelation*
	ジョイント関連データ	JtRelation&
	ジョイント データ	const DynJointData&
処理	〔I.剛体 A の剛体関連データを A 関連、I.ジョイント関連データを Jt 関連、 I.ジョイント データを Jt データと記載する〕 HP 軸の処理〔HP 軸のカルダン拘束を生成する処理〕 I.Jt 関連.P 節剛体関連データ.Bullet 剛体 != nullptr I.A 関連 != nullptr I.Jt 関連.P 軸拘束=＜カルダン拘束生成＞(I.A 関連, I.Jt 関連.P 節剛体関連データ, I.Jt データ.H 軸回転データ, I.Jt データ.P 軸回転データ, I.Jt データ.演算誤差データ, I.Jt データ.デバッグ可視化データ)※1 I.A 関連=I.Jt 関連.P 節剛体関連データ B 軸の処理〔B 軸のヒンジ拘束を生成する処理〕 I.Jt 関連.B 節剛体関連データ.Bullet 剛体 != nullptr I.A 関連 != nullptr I.Jt 関連.B 軸拘束=＜ヒンジ拘束生成＞(rbxSys::Axis::B, I.A 関連, I.Jt 関連.B 節剛体関連データ, I.Jt データ.B 軸回転データ, I.Jt データ.演算誤差データ, I.Jt データ.デバッグ可視化データ)※1 I.A 関連=I.Jt 関連.B 節剛体関連データ E 軸の処理〔E 軸のスライダー拘束を生成する処理〕 I.Jt 関連.E 節剛体関連データ.Bullet 剛体 != nullptr I.A 関連 != nullptr I.Jt 関連.E 軸拘束=＜スライダー拘束生成＞(I.A 関連, I.Jt 関連.E 節剛体関連データ, I.Jt データ.E 軸直動データ, I.Jt データ.演算誤差データ, I.Jt データ.デバッグ可視化データ)※1 出力値=true ※1 処理異常の場合は、終了(出力値=false)	
出力	処理結果（true： 正常終了／false： 異常終了）	Bool

機能	3DOF＋スライダー ジョイント生成	
入力	剛体 A の剛体関連データ	const RbRelation*
	ジョイント関連データ	JtRelation&
	ジョイント データ	const DynJointData&
処理	〔I.剛体 A の剛体関連データを A 関連、I.ジョイント関連データを Jt 関連、 I.ジョイント データを Jt データと記載する〕 HPB 軸の処理〔HPB 軸の 3DOF 拘束を生成する処理〕 I.Jt 関連.B 節剛体関連データ.Bullet 剛体 != nullptr I.A 関連 != nullptr I.Jt 関連.B 軸拘束=＜3DOF 拘束生成＞(I.A 関連, I.Jt 関連.B 節剛体関連データ, I.Jt データ.H 軸回転データ, I.Jt データ.P 軸回転データ, I.Jt データ.B 軸回転データ, I.Jt データ.演算誤差データ, I.Jt データ.デバッグ可視化データ)※1 I.A 関連=I.Jt 関連.B 節剛体関連データ E 軸の処理〔E 軸のスライダー拘束を生成する処理〕 I.Jt 関連.E 節剛体関連データ.Bullet 剛体 != nullptr I.A 関連 != nullptr I.Jt 関連.E 軸拘束=＜スライダー拘束生成＞(I.A 関連, I.Jt 関連.E 節剛体関連データ, I.Jt データ.E 軸直動データ, I.Jt データ.演算誤差データ, I.Jt データ.デバッグ可視化データ)※1 出力値=true ※1 処理異常の場合は、終了(出力値=false)	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	ヒンジ拘束生成	
入力	軸指定（H/P/B）	rbxSys::Axis
	剛体 A の剛体関連データ	const RbRelation&
	剛体 B の剛体関連データ	const RbRelation&
	回転データ	const DynRot&
	演算誤差データ	const DynJointData::SpecialParams&
	デバッグ可視化データ	const DynJointData::DebugParams&
処理	〔I.剛体 A の剛体関連データを A 関連データ、I.剛体 B の剛体関連データを B 関連データと記載する〕 A.フレーム=ヒンジ フレーム データ::＜コンストラクタ＞(A 関連データ，B 関連データ，軸指定) A.フレーム.(警告データ).＜警告確認＞() == true #.警告配列.＜追加＞(A.フレーム.(警告データ)) A.Bullet ヒンジ拘束= new btGeneric6DofSpring2Constraint(I.剛体 A の剛体関連データ.Bullet 剛体, I.剛体 B の剛体関連データ.Bullet 剛体, btTransform({座標系変換※1}←A.フレーム.A_Fr.sqmat)，{座標系変換※2}←A.A_Fr.off), btTransform({座標系変換※1}←A.フレーム.B_Fr.sqmat)，{座標系変換※2}←A.B_Fr.off)) A.Bullet ヒンジ拘束 != nullptr 〔直動/回転制限設定処理〕 ＜直動固定設定(全軸)＞(A.Bullet ヒンジ拘束) ＜回転制限設定(軸指定)＞(A.Bullet ヒンジ拘束，Z 軸指定値=2，I.回転データ) ＜回転固定設定(軸指定)＞(A.Bullet ヒンジ拘束，Y 軸指定値=1) ＜回転固定設定(軸指定)＞(A.Bullet ヒンジ拘束，X 軸指定値=0) 〔駆動設定設定処理〕 ＜直動駆動 OFF 設定(全軸)＞(A.Bullet ヒンジ拘束) ＜回転駆動設定(軸指定)＞(A.Bullet ヒンジ拘束，Z 軸指定値=2，I.回転データ) ＜回転駆動 OFF 設定(軸指定)＞(A.Bullet ヒンジ拘束，Y 軸指定値=1) ＜回転駆動 OFF 設定(軸指定)＞(A.Bullet ヒンジ拘束，X 軸指定値=0) 〔演算誤差設定処理〕 〔A.Bullet ヒンジ拘束を A.拘束、演算誤差データを$と記載する〕 @=ERP/STOP ERP/CFM/STOP CFM の処理 $.@有効指定 == true、＜演算誤差設定＞(A.拘束，@の識別子，$.HPB 軸の@.[軸指定]) $.破断閾値有効指定 == true、A.拘束.＜破断閾値設定※3＞($.HPB 軸の破断閾値.[軸指定]) 〔デバッグ設定処理〕 A.Bullet ヒンジ拘束.＜可視化サイズ設定※4＞(I.デバッグ可視化データ.HPB 軸表示サイズ.[I.軸指定]) 出力値=A.Bullet ヒンジ拘束 ※1 左手・行ベクトル系から右手・列ベクトル系へ変換 ※2 左手系から右手系へ変換 ※3 setBreakingImpulseThreshold ※4 setDbgDrawSize	
出力	処理結果（true： 正常終了／false： 異常終了）	Bool

機能	スライダー拘束生成	
入力	剛体 A の剛体関連データ	const RbRelation&
	剛体 B の剛体関連データ	const RbRelation&
	直動データ	const DynRot&
	演算誤差データ	const DynJointData::SpecialParams&
	デバッグ可視化データ	const DynJointData::DebugParams&
処理	〔I.剛体 A の剛体関連データを A 関連データ、I.剛体 B の剛体関連データを B 関連データと記載する〕 A.フレーム=スライダー フレーム データ::＜コンストラクタ＞(A 関連データ，B 関連データ) A.フレーム.(警告データ).＜警告確認＞() == true #.警告配列.＜追加＞(A.フレーム.(警告データ)) A.Bullet スライダー拘束= new btGeneric6DofSpring2Constraint(I.剛体 A の剛体関連データ.Bullet 剛体, I.剛体 B の剛体関連データ.Bullet 剛体, btTransform({座標系変換※1}←A.フレーム.A_Fr.sqmat)，{座標系変換※2}←A.A_Fr.off), btTransform({座標系変換※1}←A.フレーム.B_Fr.sqmat)，{座標系変換※2}←A.B_Fr.off)) A.スライダー拘束 != nullptr 〔直動/回転制限設定処理〕 ＜直動制限設定(X 軸)＞(A.Bullet スライダー拘束，I.直動データ) ＜回転固定設定(全軸)＞(A.Bullet スライダー拘束) 〔駆動設定設定処理〕 ＜直動駆動設定＞(A.Bullet スライダー拘束，I.直動データ) ＜回転駆動 OFF 設定(全軸)＞(A.Bullet スライダー拘束) 〔演算誤差設定処理〕 〔A.Bullet ヒンジ拘束を A.拘束、演算誤差データを$と記載する〕 @=ERP/STOP ERP/CFM/STOP CFM の処理 $.@有効指定 == true、＜演算誤差設定＞(A.拘束，@の識別子，$.E 軸の@) $.破断閾値有効指定 == true、A.拘束.＜破断閾値設定※3＞($.E 軸の破断閾値) 〔デバッグ設定処理〕 A.Bullet スライダー拘束.＜可視化サイズ設定※4＞(I.デバッグ可視化データ.E 軸表示サイズ) 出力値=A.Bullet スライダー拘束 ※1 左手・行ベクトル系から右手・列ベクトル系へ変換 ※2 左手系から右手系へ変換 ※3 setBreakingImpulseThreshold ※4 setDbgDrawSize	
出力	処理結果（true： 正常終了／false： 異常終了）	Bool

機能	カルダン拘束生成	
入力	剛体 A の剛体関連データ	const RbRelation&
	剛体 B の剛体関連データ	const RbRelation&
	H 軸回転データ	const DynRot&
	P 軸回転データ	const DynRot&
	演算誤差データ	const DynJointData::SpecialParams&
	デバッグ可視化データ	const DynJointData::DebugParams&
処理	〔I.剛体 A の剛体関連データを A 関連データ、I.剛体 B の剛体関連データを B 関連データと記載する〕 A.フレーム=カルダン フレーム データ::＜コンストラクタ＞(A 関連データ、B 関連データ) A.フレーム.(警告データ).＜警告確認＞() == true #.警告配列.＜追加＞(A.フレーム.(警告データ)) A.Bullet カルダン拘束= new btGeneric6DofSpring2Constraint(I.剛体 A の剛体関連データ.Bullet 剛体, I.剛体 B の剛体関連データ.Bullet 剛体, btTransform({座標系変換※1}←A.フレーム.A_Fr.sqmat), {座標系変換※2}←A.A_Fr.off), btTransform({座標系変換※1}←A.フレーム.B_Fr.sqmat), {座標系変換※2}←A.B_Fr.off)) A.Bullet カルダン拘束 != nullptr 〔直動/回転制限設定処理〕 ＜直動固定設定(全軸)＞(A.Bullet カルダン拘束) ＜回転制限設定(軸指定)＞(A.Bullet カルダン拘束，Z 軸指定値=2，I.H 軸回転データ) ＜回転固定設定(軸指定)＞(A.Bullet カルダン拘束，Y 軸指定値=1) ＜回転制限設定(軸指定)＞(A.Bullet カルダン拘束，X 軸指定値=0，I.P 軸回転データ) 〔駆動設定設定処理〕 ＜直動駆動 OFF 設定(全軸)＞(A.Bullet カルダン拘束) ＜回転駆動設定(軸指定)＞(A.Bullet カルダン拘束，Z 軸指定値=2，I.H 軸回転データ) ＜回転駆動 OFF 設定(軸指定)＞(A.Bullet カルダン拘束，Y 軸指定値=1) ＜回転駆動設定(軸指定)＞(A.Bullet カルダン拘束，X 軸指定値=0，I.P 軸回転データ) 〔演算誤差設定処理〕 〔A.Bullet カルダン拘束を A.拘束、演算誤差データを$と記載する〕 @=ERP/STOP ERP/CFM/STOP CFM の処理 $.@有効指定 == true、＜演算誤差設定＞(A.拘束，@の識別子，$.HPB 軸の@.[P 軸]) $.破断閾値有効指定 == true、A.拘束.＜破断閾値設定※3＞($.HPB 軸の破断閾値.[P 軸]) 〔デバッグ設定処理〕 A.Bullet カルダン拘束.＜可視化サイズ設定※4＞(I.デバッグ可視化データ.HPB 軸表示サイズ.[P 軸]) 出力値=A.Bullet カルダン拘束 ※1 左手・行ベクトル系から右手・列ベクトル系へ変換 ※2 左手系から右手系へ変換 ※3 setBreakingImpulseThreshold ※4 setDbgDrawSize	
出力	処理結果（true： 正常終了／false： 異常終了）	Bool

機能	3DOF 拘束生成	
入力	剛体 A の剛体関連データ	const RbRelation&
	剛体 B の剛体関連データ	const RbRelation&
	H 軸回転データ	const DynRot&
	P 軸回転データ	const DynRot&
	B 軸回転データ	const DynRot&
	演算誤差データ	const DynJointData::SpecialParams&
	デバッグ可視化データ	const DynJointData::DebugParams&
処理	〔I.剛体 A の剛体関連データを A 関連データ、I.剛体 B の剛体関連データを B 関連データと記載する〕 A.フレーム=3DOF フレーム データ::＜コンストラクタ＞(A 関連データ、B 関連データ) A.フレーム.(警告データ).＜警告確認＞() == true #.警告配列.＜追加＞(A.フレーム.(警告データ)) A.Bullet 3DOF 拘束= new btGeneric6DofSpring2Constraint(I.剛体 A の剛体関連データ.Bullet 剛体, I.剛体 B の剛体関連データ.Bullet 剛体, btTransform({座標系変換※1}←A.フレーム.A_Fr.sqmat), {座標系変換※2}←A.A_Fr.off), btTransform({座標系変換※1}←A.フレーム.B_Fr.sqmat), {座標系変換※2}←A.B_Fr.off)) A.3DOF 拘束 != nullptr 〔直動/回転制限設定処理〕 ＜直動固定設定(全軸)＞(A.Bullet 3DOF 拘束) ＜回転制限設定(軸指定)＞(A.Bullet 3DOF 拘束, Z 軸指定値=2, I.H 軸回転データ) ＜回転制限設定(軸指定)＞(A.Bullet 3DOF 拘束, Y 軸指定値=1, I.P 軸回転データ) ＜回転制限設定(軸指定)＞(A.Bullet 3DOF 拘束, X 軸指定値=0, I.B 軸回転データ) 〔駆動設定設定処理〕 ＜直動駆動 OFF 設定(全軸)＞(A.Bullet 3DOF 拘束) ＜回転駆動設定(軸指定)＞(A.Bullet 3DOF 拘束, Z 軸指定値=2, I.H 軸回転データ) ＜回転駆動設定(軸指定)＞(A.Bullet 3DOF 拘束, Y 軸指定値=1, I.P 軸回転データ) ＜回転駆動設定(軸指定)＞(A.Bullet 3DOF 拘束, X 軸指定値=0, I.B 軸回転データ) 〔演算誤差設定処理〕 〔A.Bullet3DOF 拘束を A.拘束、演算誤差データを$と記載する〕 @=ERP/STOP ERP/CFM/STOP CFM の処理 $.@有効指定 == true、＜演算誤差設定＞(A.拘束, @の識別子, $.HPB 軸の@.[B 軸]) $.破断閾値有効指定 == true、A.拘束.＜破断閾値設定※3＞($.HPB 軸の破断閾値.[B 軸]) 〔デバッグ設定処理〕 A.Bullet 3DOF 拘束.＜可視化サイズ設定※4＞(I.デバッグ可視化データ.HPB 軸表示サイズ.[B 軸]) 出力値=A.Bullet 3DOF 拘束 ※1 左手・行ベクトル系から右手・列ベクトル系へ変換 ※2 左手系から右手系へ変換 ※3 setBreakingImpulseThreshold ※4 setDbgDrawSize	
出力	処理結果（true： 正常終了／false： 異常終了）	Bool

機能	アンカーの剛体関連データ取得	
入力	アンカー モデル	BaseObject*
処理	@=#.静的剛体関連配列[*]の処理 @.モデル == I.アンカー モデル、終了(出力値=@) @=#.動的剛体関連配列[*]の処理 @.モデル == I.アンカー モデル、終了(出力値=@) 出力値=nullptr	
出力	アンカーの剛体関連データ	DynWorldBt::RbRelation*

機能	直動固定設定(全軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	A.相対位置＝I.Bullet 拘束.＜相対位置取得※1＞(@=X/Y/Z) I.Bullet 拘束.＜下限の直動制限設定※2＞(A.相対位置) I.Bullet 拘束.＜上限の直動制限設定※3＞(A.相対位置) ※1 btGeneric6DofSpring2Constraint::getRelativePivotPosition() ※2 btGeneric6DofSpring2Constraint:: setLinearUpperLimit() ※3 btGeneric6DofSpring2Constraint::setLinearLowerLimit()	

機能	直動制限設定(X 軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	直動データ	const DynLin&
処理	I.直動データ.直動タイプ別の処理	

直動タイプ	処理
固定	A.上限位置 ＝A.下限位置 ＝Bullet 拘束.＜相対位置取得※1＞(X 軸=0)
長さ制限	A.下限位置 ＝#.(単位変換データ).＜MtoU＞(I.直動データ.制限の範囲.下限) A.上限位置 ＝#.(単位変換データ).＜MtoU＞(I.直動データ.制限の範囲.上限)

A.Y 軸の位置 ＝I.Bullet 拘束.＜相対位置取得※1＞(Y 軸=1)
A.Z 軸の位置 ＝I.Bullet 拘束.＜相対位置取得※1＞(Z 軸=2)

I.Bullet 拘束.＜下限の直動制限設定※2＞(3 次元ベクター※4 (A.下限値、A.Y 軸の位置、A.Z 軸の位置))
I.Bullet 拘束.＜上限の直動制限設定※3＞(3 次元ベクター※4 (A.上限値、A.Y 軸の位置、A.Z 軸の位置))

※1 btGeneric6DofSpring2Constraint::getRelativePivotPosition()
※2 btGeneric6DofSpring2Constraint:: setLinearUpperLimit()
※3 btGeneric6DofSpring2Constraint::setLinearLowerLimit()
※4 btVector3

機能	回転固定設定(全軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜回転固定設定(軸指定)＞(I.Bullet 拘束、Z 軸=2) ＜回転固定設定(軸指定)＞(I.Bullet 拘束、Y 軸=1) ＜回転固定設定(軸指定)＞(I.Bullet 拘束、X 軸=0)	

機能	回転固定設定(軸指定)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	軸指定 (X=0/Y=1/Z=2)	Int32
処理	A.相対角度 ＝I.Bullet 拘束.＜拘束.相対角度取得※1＞(I.軸指定) Bullet 拘束.＜制限設定※2＞(3+I.軸指定、A.相対角度、A.相対角度) ※1 btGeneric6DofSpring2Constraint::getRelativePivotPosition() ※2 btGeneric6DofSpring2Constraint::setLimit()	

機能	回転制限設定(軸指定)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	軸指定 (X=0/Y=1/Z=2)	Int32
	回転データ	const DynRot&
処理	I.回転データ.回転タイプ別の処理	

回転タイプ	処理
固定	＜回転固定設定(軸指定)＞(I.Bullet 拘束、I.軸指定)
制限なし	I.Bullet 拘束.＜制限設定※1＞(3+I.軸指定、下限値=PI×0.5、上限値=-PI×0.5)
角度制限	I.軸指定 == Z 軸 I.Bullet 拘束.＜制限設定※1＞(3+I.軸指定、I.回転データ.制限の範囲.下限値、I.回転データ.制限の範囲.上限値) I.軸指定 != Z 軸 I.Bullet 拘束.＜制限反転設定※2＞(3+I.軸指定、I.回転データ.制限の範囲.下限値、I.回転データ.制限の範囲.上限値)

※1 btGeneric6DofSpring2Constraint::setLimit()
※2 btGeneric6DofSpring2Constraint::setLimitReversed()

機能	直動駆動 OFF 設定(全軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Z 軸=2) ＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Y 軸=1) ＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、X 軸=0)	

機能	直動駆動 OFF 設定(軸指定)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	軸指定 (X=0/Y=1/Z=2)	Int32
処理	I.Bullet 拘束.＜最大フォース設定※1＞(I.軸指定、0) I.Bullet 拘束.＜目標速度設定※2＞(I.軸指定、0) I.Bullet 拘束.＜サーボ目標設定※3＞(I.軸指定、0) I.Bullet 拘束.＜サーボ設定※4＞(I.軸指定、false) I.Bullet 拘束.＜モータ設定※5＞(I.軸指定、false) ※1 btGeneric6DofSpring2Constraint::setMaxMotorForce() ※2 btGeneric6DofSpring2Constraint::setTargetVelocity() ※3 btGeneric6DofSpring2Constraint::setServoTarget() ※4 btGeneric6DofSpring2Constraint::setServo() ※5 btGeneric6DofSpring2Constraint::enableMotor()	

機能	直動駆動設定(X 軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	直動データ	const DynLin&
処理	I.直動データ.直動タイプ != 固定 I.直動データ.駆動タイプ別の処理	

駆動タイプ	処理
アイドル	＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、X 軸=0)
フォース	I.Bullet 拘束.＜最大フォース設定※1＞(X 軸=0、絶対値(直動データ.印加フォース)×C※6) I.Bullet 拘束.＜目標速度設定※2＞(X 軸=0、－直動データ.印加速度×C※6) I.Bullet 拘束.＜サーボ目標設定※3＞(X 軸=0、0) I.Bullet 拘束.＜サーボ設定※4＞(X 軸=0、false) I.Bullet 拘束.＜モータ設定※5＞(X 軸=0、true)
サーボ	I.Bullet 拘束.＜最大フォース設定※1＞(X 軸=0、絶対値(直動データ.印加フォース)×C※6) I.Bullet 拘束.＜目標速度設定※2＞(X 軸=0、絶対値(直動データ.印加速度)×C※6) I.Bullet 拘束.＜サーボ目標設定※3＞(X 軸=0、直動データ.サーボ目標×C※6) I.Bullet 拘束.＜サーボ設定※4＞(X 軸=0、true) I.Bullet 拘束.＜モータ設定※5＞(X 軸=0、true)

そうではない
＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、X 軸=0)

＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Y 軸=1)
＜直動駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Z 軸=2)

※1~5 直動駆動 OFF 設定(軸指定)の※1~5 と同じ
※6 C：#.(単位変換データ).メートル単位 → システム単位変換係数

機能	回転駆動 OFF 設定(全軸)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜回転駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Z 軸=2) ＜回転駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、Y 軸=1) ＜回転駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、X 軸=0)	

機能	回転駆動 OFF 設定(軸指定)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	軸指定 (X=0/Y=1/Z=2)	Int32
処理	A.Index = 3 + I.軸指定 I.Bullet 拘束.＜最大フォース設定※1＞(A.Index、0) I.Bullet 拘束.＜目標速度設定※2＞(A.Index、0) I.Bullet 拘束.＜サーボ目標設定※3＞(A.Index、0) I.Bullet 拘束.＜サーボ設定※4＞(A.Index、false) I.Bullet 拘束.＜モータ設定※5＞(A.Index、false) ※1~5 直動駆動 OFF 設定(軸指定)の※1~5 と同じ	

機能	回転駆動設定(軸指定)	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	軸指定 (X=0/Y=1/Z=2)	Int32
	回転データ	const DynRot&
処理	I.回転データ.回転タイプ != 固定 I.回転データ.駆動タイプ別の処理	

駆動タイプ	処理
アイドル	＜回転駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、I.軸指定)
トルク	A.Sign = I.軸指定 == Z 軸 ? 1 : -1 A.Index = 3 + I.軸指定 I.Bullet 拘束.＜最大トルク設定※1＞(I.軸指定、絶対値(回転データ.印加トルク)×C※6) I.Bullet 拘束.＜目標速度設定※2＞(I.軸指定、回転データ.印加速度×A.Sign) I.Bullet 拘束.＜サーボ目標設定※3＞(I.軸指定、0) I.Bullet 拘束.＜サーボ設定※4＞(I.軸指定、false) I.Bullet 拘束.＜モータ設定※5＞(I.軸指定、true)
サーボ	A.Sign = I.軸指定 == Z 軸 ? 1 : -1 A.Index = 3 + I.軸指定 I.Bullet 拘束.＜最大トルク設定※1＞(I.軸指定、絶対値(回転データ.印加トルク)×C※6) I.Bullet 拘束.＜目標速度設定※2＞(I.軸指定、絶対値(回転データ.印加速度)) I.Bullet 拘束.＜サーボ目標設定※3＞(I.軸指定、回転データ.サーボ目標×A.Sign) I.Bullet 拘束.＜サーボ設定※4＞(I.軸指定、true) I.Bullet 拘束.＜モータ設定※5＞(I.軸指定、true)

そうではない
＜回転駆動 OFF 設定(軸指定)＞(I.Bullet 拘束、X 軸=0)

※1~5 直動駆動 OFF 設定(軸指定)の※1~5 と同じ
※6 C : #.(単位変換データ).メートル単位 → システム単位変換係数

機能	演算誤差設定	
入力	Bullet 拘束	btGeneric6DofSpring2Constraint*
	識別子	Int32
	設定値	Float
処理	@=全直動軸と全回転軸〔0~5〕の処理 Bullet 拘束.＜パラメータ設定＞(識別子，設定値，@)	

機能	ジョイント駆動設定更新
処理	@=#.ジョイント関連配列[*]の処理 @.ジョイント タイプ別の処理

ジョイント タイプ	処理
組合せヒンジ	＜組合せヒンジ＋スライダー駆動設定更新＞(@)
カルダン＋ヒンジ	＜カルダン＋ヒンジ＋スライダー駆動設定更新＞(@)
3DOF	＜3DOF＋ヒンジ＋スライダー駆動設定更新＞(@)

機能	組合せヒンジ＋スライダー駆動設定更新	
入力	ジョイント関連データ	const JtRelation&
処理	〔ジョイント関連データを Jt 関連データと記載する〕 @=H/P/B の処理 I.Jt 関連データ.@軸拘束 != nullptr ＜ヒンジ駆動設定更新＞(@軸指定=0/1/2、I.Jt 関連データ.@軸回転データ、I.Jt 関連データ.@軸拘束) I.Jt 関連.E 軸拘束 != nullptr ＜スライダー駆動設定更新＞(I.Jt 関連.E 軸直動データ、I.Jt 関連データ.E 軸拘束)	

機能	カルダン＋スライダー駆動設定更新	
入力	ジョイント関連データ	const JtRelation&
処理	〔ジョイント関連データを Jt 関連データと記載する〕 I.Jt 関連データ.P 軸拘束 != nullptr ＜カルダン駆動設定更新＞(I.Jt 関連データ.[H,P]軸回転データ、I.Jt 関連データ.P 軸拘束) I.Jt 関連データ.B 軸拘束 != nullptr ＜ヒンジ駆動設定更新＞(B 軸指定=2、I.Jt 関連データ.B 軸回転データ、I.Jt 関連データ.B 軸拘束) I.Jt 関連データ.E 軸拘束 != nullptr ＜スライダー駆動設定更新＞(I.Jt 関連データ.E 軸直動データ、I.Jt 関連データ.E 軸拘束)	

機能	3DOF＋スライダー駆動設定更新	
入力	ジョイント関連データ	const JtRelation&
処理	〔ジョイント関連データを Jt 関連と記載する〕 I.Jt 関連.B 軸拘束 != nullptr ＜3DOF 駆動設定更新＞(I.Jt 関連.[H,P,B]軸回転データ、I.Jt 関連.B 軸拘束) I.Jt 関連.E 軸拘束 != nullptr ＜スライダー駆動設定更新＞(I.Jt 関連データ.E 軸直動データ、I.Jt 関連データ.E 軸拘束)	

機能	ヒンジ駆動設定更新	
入力	軸指定	rbxSys::Axis
	回転データ	const DynRot&
	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.回転データ)	

機能	スライダー駆動設定更新	
入力	直動データ	const DynRot&
	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜直動制限設定(X 軸)＞(I.Bullet 拘束、I.直動データ) ＜直動駆動設定(X 軸)＞(I.Bullet 拘束、I.直動データ)	

機能	カルダン駆動設定更新	
入力	H 軸回転データ	const DynRot&
	P 軸回転データ	const DynRot&
	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.H 軸回転データ) ＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=0、I.P 軸回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.H 軸回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=0、I.P 軸回転データ)	

機能	3DOF 駆動設定更新	
入力	H 軸回転データ	const DynRot&
	P 軸回転データ	const DynRot&
	B 軸回転データ	const DynRot&
	Bullet 拘束	btGeneric6DofSpring2Constraint*
処理	＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.H 軸回転データ) ＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=1、I.P 軸回転データ) ＜回転制限設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=0、I.B 軸回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=2、I.H 軸回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=1、I.P 軸回転データ) ＜回転駆動設定(軸指定)＞(I.Bullet 拘束、Z 軸指定値=0、I.B 軸回転データ)	

7.6.2.1 剛体関連データ

Bullet 物理ワールド管理コア クラス::剛体関連データ(DynWorldBt::RbRelation)の構成を示す。

(1) 管理情報

表 7-48 剛体関連データ型

番号	項目	初期値	型
1	ネイティブ剛体の情報	－	－
1.1	モデル	nullptr	BaseObject*
1.2	Ml 有効・無効（true： 有効 / false： 無効）	false	Bool
1.3	慣性主軸 Ml（L 座標系→慣性主軸座標系）	＜－＞	Matrix
1.4	L 原点 Ml（慣性主軸座標系→L 座標系）	＜－＞	Matrix
1.5	回転逆 Mg（G 座標系の角度→L 座標系の角度）	＜－＞	SqMat3<Vector>
1.6	モニター データ	nullptr	const DynBodyMonData*
2	Bullet 剛体の情報	－	－
2.1	剛体	nullptr	btRigidBody*

(2) 外部関数

機能	デフォルト コンストラクタ
処理	表 7-48 の初期値から剛体管理データを生成する。

機能	コンストラクタ	
入力	剛体モデル	BaseObject*
	モニター データ	const DynBodyMonData*
	Bullet 剛体	btRigidBody*
処理	#.@ = I.*〔入力で指定されない項目は初期値を設定〕 #.ローカル マトリクス有効・無効=false ＜モニター データ初期化＞()	

機能	コンストラクタ	
入力	剛体モデル	BaseObject*
	慣性主軸 Ml	const Matrix&
	L 原点 Ml	const Matrix&
	モニター データ	const DynBodyMonData*
	Bullet 剛体	btRigidBody*
処理	#.@ = I.* #.ローカル マトリクス有効・無効=true ＜モニター データ初期化＞()	

機能	モニター データ初期化
処理	#.モニター データ != nullptr 　#.モニター データ.＜書き込み可能確認＞() == true 　　#.モニター データ.* = 0

機能	ネイティブ モデルの位置と角度更新©	
入力	単位変換データ	const DynUnitCnv&
処理	〔I.単位変換データ.システム単位 → メートル単位変換係数を UtoM と記載する〕 A.剛体の Mg={右手・列ベクトル系→左手・行ベクトル系変換}←#.Bullet 剛体の Mg #.ローカル マトリクス有効・無効 == true 　A.剛体の Mg = A.剛体の Mg×#.L 原点 Ml #.ネイティブ モデルの Mg = A.剛体の Mg #.回転逆 Mg ={転置}←A.剛体の Mg.sqmat #.モニター データ != nullptr 　#.モニター データ.＜書き込み可能確認＞() == true 　　#.モニター データ.線形速度 = ＜直動速度取得＞()×UtoM 　　#.モニター データ.回転速度 = ＜HPB 回転速度取得＞()	

機能	直動速度取得©	
処理	出力値={右手系→左手系変換}←#.Bullet 剛体.直動速度	
出力	直動速度（単位：U/S）	Vector

機能	XYZ 回転速度取得©	
処理	出力値={右手系→左手系変換}←#.Bullet 剛体.回転速度	
出力	XYZ 回転速度（単位：Rad/S）	Vector

機能	HPB 回転速度取得©	
処理	出力値={XYZ→HPB 変換}←{右手系→左手系変換}←#.Bullet 剛体.回転速度	
出力	HPB 回転速度（単位：Rad/S）	Vector

7.6.2.2 ジョイント関連データ

Bullet 物理ワールド管理コア クラス:: ジョイント関連データ(DynWorldBt::JtRelation)の構成を示す。

(1) 管理情報

表 7-49 ジョイント関連データ型

番号	項目	初期値	型
1	ネイティブジョイントの情報	－	－
1.1	ジョイント タイプ	-1	DynJointType
1.2	ルート指定	false	Bool
1.3	分離リグ指定	false	Bool
1.4	ジョイント モデル	nullptr	BaseObject*
1.5	位置参照モデル（分離リグ指定が true の場合のみ有効）	nullptr	BaseObject*
1.6	回転/直動モニター更新指定データ	nullptr	DynRotLinMonUpdate*
1.7	H 軸回転データ※1	nullptr	const DynRot*
1.8	P 軸回転データ※1	nullptr	
1.9	B 軸回転データ※1	nullptr	
1.10	E 軸直動データ※2	nullptr	const DynLin*
2	Bullet のジョイントの情報	－	－
2.1	H 軸拘束	nullptr	btGeneric6DofSpring2 Constraint*
2.2	P 軸拘束	nullptr	
2.3	B 軸拘束	nullptr	
2.4	E 軸拘束	nullptr	
2.5	H 節剛体関連データ	＜－＞	RbRelation
2.6	P 節剛体関連データ	＜－＞	RbRelation
2.7	B 節剛体関連データ	＜－＞	RbRelation
2.8	E 節剛体関連データ	＜－＞	RbRelation
2.9	前節剛体関連データ（前節剛体が静的剛体の場合は nullptr）	nullptr	RbRelation*

※1 ショート ブレーキ指定、抵抗トルク、最大トルクと最高角速度は参照しない。
※2 ショート ブレーキ指定、抵抗フォース、最大フォースと最高速度は参照しない。

(2) 外部関数

機能	デフォルト コンストラクタ
処理	表 7-49 の初期値からジョイント関連データを生成する。 ＜モニター データ初期化＞()

機能	モニター データ初期化
処理	@=H/P/B の処理 #.@軸回転データ != nullptr #.@軸回転データ.＜カレント データ領域書き込み可能確認＞() == true #.@軸回転データ.＜カレント データ クリアー＞() #.E 軸直動データ != nullptr #.E 軸直動データ.＜カレント データ領域書き込み可能確認＞() == true #.@軸直動データ.＜カレント データ クリアー＞()

機能	末尾の節の剛体関連データ取得©	
処理	@=E/B/P/H の処理 #.@節剛体関連データ != nullptr の場合、終了(出力値=#.@節剛体関連データ) 出力値=nullptr	
出力	末尾の節の剛体関連データ	const RbRelation*

<table>
<tr><td>機能</td><td colspan="2">ネイティブ モデルの位置と角度更新©</td></tr>
<tr><td>入力</td><td>単位変換データ</td><td>const DynUnitCnv&</td></tr>
<tr><td>処理</td><td colspan="2">@=H/P/B の処理
#.@節剛体関連データ.＜ネイティブ モデルの位置と角度更新＞(I.単位変換データ)

#.分離リグ指定 == true
#.分離リグ ジョイントの位置参照モデル != nullptr
#.ジョイント モデル.Mg.off = #.分離リグ ジョイントの位置参照モデル.Mg.off
ジョイント タイプ別の処理
<table><tr><th>ジョイント タイプ</th><th>処理</th></tr><tr><td>組合せヒンジ</td><td>＜組み合わせヒンジの角度取得＞(A.相対角度)
＜組み合わせヒンジ＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr><tr><td>カルダン＋ヒンジ</td><td>＜カルダン＋ヒンジの角度取得＞(A.相対角度)
＜カルダン＋ヒンジ＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr><tr><td>3DOF</td><td>＜3DOF の角度取得＞(A.相対角度)
＜3DOF＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr></table>#.ジョイント モデル.相対角度=A.相対角度
そうではない
ジョイント タイプ別の処理
<table><tr><th>ジョイント タイプ</th><th>処理</th></tr><tr><td>組合せヒンジ</td><td>＜組み合わせヒンジ＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr><tr><td>カルダン＋ヒンジ</td><td>＜カルダン＋ヒンジ＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr><tr><td>3DOF</td><td>＜3DOF＋スライダーのモニター データ更新＞(I.単位変換データ)</td></tr></table></td></tr>
</table>

機能	組み合わせヒンジの角度取得©	
入力	相対角度	Vector&
処理	I.相対角度=#.ジョイント モデル.相対角度 #.H 軸拘束 != nullptr I.相対角度.x = #.H 軸拘束.Z 軸の角度 #.P 軸拘束 != nullptr I.相対角度.y = #.P 軸拘束.Z 軸の角度 #.B 軸拘束 != nullptr I.相対角度.z = #.B 軸拘束.Z 軸の角度	

機能	カルダン＋ヒンジの角度取得©	
入力	相対角度	Vector&
処理	I.相対角度=#.ジョイント モデル.相対角度 #.P 軸拘束 != nullptr I.相対角度.x = #.P 軸拘束.Z 軸の角度 I.相対角度.y = #.P 軸拘束.X 軸の角度×-1 #.B 軸拘束 != nullptr I.相対角度.z = #.B 軸拘束.Z 軸の角度	

機能	3DOF の角度取得©	
入力	相対角度	Vector&
処理	#.B 軸拘束 != nullptr I.相対角度.x = #.P 軸拘束.Z 軸の角度 I.相対角度.y = #.P 軸拘束.Y 軸の角度×-1 I.相対角度.z = #.P 軸拘束.X 軸の角度×-1 #.B 軸拘束 == nullptr I.相対角度=#.ジョイント モデル.相対角度	

機能	組み合わせヒンジ＋スライダーのモニター データ更新©	
入力	単位変換データ	const DynUnitCnv&
処理	@=H/P/B、@xyz=Y/X/Z={XYZ に変換}←@の処理〔@の前の節を@と記載する〕 #.@節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.@ == true A.@節の回転速度=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※2).@xyz A.@節の回転速度=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※3).@xyz A.回転速度= A.@節の回転速度－A.@節の回転速度 A.トルク= {#.@軸回転データ.印加トルクと印加角速度からの線形変換}←A.回転速度 #.@軸回転データ.＜カレント データ設定＞(A.回転速度と A.トルク) @=E の処理〔@の前の節を@と記載する〕 #.@節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.@ == true A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※4).z A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※5).z A.直動速度= A.@節の直動速度－A.@節の直動速度 A.フォース= {#.@軸直動データ.印加フォースと印加速度からの線形変換}←A.直動速度 #.@軸直動データ.＜カレント データ設定＞(A.直動速度と A.フォース) ※1 #.@節剛体関連データ.回転逆 Mg×G 座標系の回転速度ベクトル ※2 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※3 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※4 #.@節剛体関連データ.＜直動速度取得＞() ※5 #.@節剛体関連データ.＜直動速度取得＞()	

機能	カルダン＋ヒンジ＋スライダーのモニター データ更新©	
入力	単位変換データ	const DynUnitCnv&
処理	@=HP、@xy=YX={XYZ に変換}←@の処理〔@の前の節を@と記載する〕 #.P 節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.[H,P] == true A.@節の回転速度.HP=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※2).@xy A.@節の回転速度.HP=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※3).@xy A.回転速度.HP= A.@節の回転速度.HP－A.@節の回転速度.HP A.トルク.HP= {#.@軸回転データ.印加トルクと印加角速度からの線形変換}←A.回転速度.HP #.回転/直動モニター更新指定.H == true #.H 軸回転データ.＜カレント データ設定＞(A.回転速度.H と A.トルク.H) #.回転/直動モニター更新指定.P == true #.P 軸回転データ.＜カレント データ設定＞(A.回転速度.P と A.トルク.P) @=B の処理〔@の前の節を@と記載する〕 #.B 節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.B == true A.@節の回転速度=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※2).z A.@節の回転速度=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※3).z A.回転速度= A.@節の回転速度－A.@節の回転速度 A.トルク= {#.@軸回転データ.印加トルクと印加角速度からの線形変換}←A.回転速度 #.B 軸回転データ.＜カレント データ設定＞(A.回転速度と A.トルク) @=E の処理〔@の前の節を@と記載する〕 #.@節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.@ == true A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※4).z A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※5).z A.直動速度= A.@節の直動速度－A.@節の直動速度 A.フォース= {#.@軸直動データ.印加フォースと印加速度からの線形変換}←A.直動速度 #.@軸直動データ.＜カレント データ設定＞(A.直動速度と A.フォース) ※1 #.@節剛体関連データ.回転逆 Mg×G 座標系の回転速度ベクトル ※2 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※3 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※4 #.@節剛体関連データ.＜直動速度取得＞() ※5 #.@節剛体関連データ.＜直動速度取得＞()	

機能	3DOF＋スライダーのモニター データ更新©	
入力	単位変換データ	const DynUnitCnv&
処理	@=HPB、@xyz=YXZ={XYZ に変換}←@の処理〔@の前の節を@と記載する〕 #.B 節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.(HPB の何れか) == true A.@節の回転速度.HPB=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※2).@xyz A.@節の回転速度.HPB=({@節の L 座標系へ変換※1}←@節の G 座標系の回転速度※3).@xyz A.回転速度.HPB= A.@節の回転速度.HPB－A.@節の回転速度.HPB A.トルク.HPB= {#.@軸回転データ.印加トルクと印加角速度からの線形変換}←A.回転速度.HPB #.回転/直動モニター更新指定.H == true #.H 軸回転データ.＜カレント データ設定＞(A.回転速度.H と A.トルク.H) #.回転/直動モニター更新指定.P == true #.P 軸回転データ.＜カレント データ設定＞(A.回転速度.P と A.トルク.P) #.回転/直動モニター更新指定.B == true #.B 軸回転データ.＜カレント データ設定＞(A.回転速度.B と A.トルク.B) @=E の処理〔@の前の節を@と記載する〕 #.@節剛体関連データ.モデル != nullptr #.回転/直動モニター更新指定.@ == true A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※4).z A.@節の直動速度=({@節の L 座標系へ変換※1}←@節の G 座標系の直動速度※5).z A.直動速度= A.@節の直動速度－A.@節の直動速度 A.フォース= {#.@軸直動データ.印加フォースと印加速度からの線形変換}←A.直動速度 #.@軸直動データ.＜カレント データ設定＞(A.直動速度と A.フォース) ※1 #.@節剛体関連データ.回転逆 Mg×G 座標系の回転速度ベクトル ※2 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※3 #.@節剛体関連データ.＜XYZ 回転速度取得＞() ※4 #.@節剛体関連データ.＜直動速度取得＞() ※5 #.@節剛体関連データ.＜直動速度取得＞()	

7.6.2.3 ヒンジ フレーム データ

Bullet 物理ワールド管理コア クラス::ヒンジ フレーム データ(DynWorldBt::HingeFrame：DynWorld::WarningInfo)の構成を示す。

(1) 管理情報

表 7-50 ヒンジ フレーム データの管理情報

番号	項目	初期値	型
1	物理ワールド管理コア クラスの警告データの管理データ	＜－＞	DynWorld::WarningInfo
2	剛体 A のフレーム マトリクス（A_Fr）	＜－＞	Matrix
3	剛体 B のフレーム マトリクス（B_Fr）	＜－＞	Matrix

(2) 外部関数

<table>
<tr><td>機能</td><td colspan="2">コンストラクタ</td></tr>
<tr><td rowspan="3">入力</td><td>剛体 A の剛体関連データ（関連 A）</td><td>const RbRelation&</td></tr>
<tr><td>剛体 B の剛体関連データ（関連 B）</td><td>const RbRelation&</td></tr>
<tr><td>軸指定</td><td>rbxSys::Axis</td></tr>
<tr><td>処理</td><td colspan="2">〔I.関連 A.モデルを剛体 A、I.関連 B.モデルを剛体 B と記載する〕
A.剛体 B の基準 Mg〔H/P/B=0 度の Mg〕＝rbxSys::getFrozenRotMg(剛体 B、A.剛体 B の Mg)
A.FrA〔フレーム A の Ml〕=Inv(剛体 A の Mg×関連 A.慣性主軸 Ml)×A.剛体 B の基準 Mg
A.FrB〔フレーム B の Ml〕=Inv(剛体 B の Mg×関連 B.慣性主軸 Ml)×A.剛体 B の Mg
軸指定別の処理
<table>
<tr><th>軸指定</th><th>処理</th></tr>
<tr><td>H</td><td>〔軸の向きを変更する(Y→Z, X→Y, Z→X)〕
#.A_Fr=Mat3<Vector>(A.FrA.off, A.FrA.sqmat.v3, A.FrA.sqmat.v1, A.FrA.sqmat.v2)
#.B_Fr=Mat3<Vector>(A.FrB.off, A.FrB.sqmat.v3, A.FrB.sqmat.v1, A.FrB.sqmat.v2)
剛体 B.相対角度.(y 又は z) != 0
＜警告設定＞(剛体 B.名称、座標変換の固定異常)</td></tr>
<tr><td>P</td><td>〔軸の向きを変更する(X→Z, -Y→Y, Z→X)〕
#.A_Fr=Mat3<Vector>(A.FrA.off, A.FrA.sqmat.v3, -A.FrA.sqmat.v2, A.FrA.sqmat.v1)
#.B_Fr=Mat3<Vector>(A.FrB.off, A.FrB.sqmat.v3, -A.FrB.sqmat.v2, A.FrB.sqmat.v1)
剛体 B.相対角度.(x 又は z) != 0
＜警告設定＞(剛体 B.名称、座標変換の固定異常)</td></tr>
<tr><td>B</td><td>#.剛体 A_Fr=Mat3<Vector>(#.FrA.off, #.FrA.sqmat)
#.剛体 B_Fr=Mat3<Vector>(#.FrB.off, #.FrB.sqmat)
剛体 B.相対角度.(x 又は y) != 0
＜警告設定＞(剛体 B.名称、座標変換の固定異常)</td></tr>
</table></td></tr>
</table>

7.6.2.4 スライダー フレーム データ

Bullet 物理ワールド管理コア クラス::スライダー フレーム データ(DynWorldBt::SliderFrame：DynWorld::WarningInfo)の構成を示す。

(1) 管理情報

表 7-51 スライダー フレーム データの管理情報

番号	項目	初期値	型
1	物理ワールド管理コア クラスの警告データの管理データ	＜－＞	DynWorld::WarningInfo
2	剛体 A のフレーム マトリクス（A_Fr）	＜－＞	Matrix
3	剛体 B のフレーム マトリクス（B_Fr）	＜－＞	Matrix

(2) 外部関数

機能	コンストラクタ	
入力	剛体 A の剛体関連データ（関連 A）	const RbRelation&
	剛体 B の剛体関連データ（関連 B）	const RbRelation&
処理	〔I.関連 A.モデルを剛体 A、I.関連 B.モデルを剛体 B と記載する〕 A.剛体 B の基準 Mg＝rbxSys::getFrozenRotMg(剛体 B、A.剛体 B の Mg) A.FrA〔フレーム A の Ml〕=Inv(剛体 A の Mg×関連 A.慣性主軸 Ml)×A.剛体 B の基準 Mg A.FrB〔フレーム B の Ml〕=Inv(剛体 B の Mg×関連 B.慣性主軸 Ml)×A.剛体 B の Mg 〔軸の向きを変更する(-X→Z, Y→Y, Z→X)〕 #.A_Fr=Mat3<Vector>(A.FrA.off, A.FrA.sqmat.v3, A.FrA.sqmat.v2, -A.FrA.sqmat.v1) #.B_Fr=Mat3<Vector>(A.FrB.off, A.FrB.sqmat.v3, A.FrB.sqmat.v2, -A.FrB.sqmat.v1) 剛体 B.相対角度.(x,y,z の何れか) != 0 ＜警告設定＞(剛体 B.名称、座標変換の固定異常)	

7.6.2.5 カルダン フレーム データ

Bullet 物理ワールド管理コア クラス::カルダン フレーム データ(DynWorldBt::CardanFrame：DynWorld::WarningInfo)の構成を示す。

(1) 管理情報

表 7-52 カルダン フレーム データの管理情報

番号	項目	初期値	型
1	物理ワールド管理コア クラスの警告データの管理データ	＜－＞	DynWorld::WarningInfo
2	剛体 A のフレーム マトリクス（A_Fr）	＜－＞	Matrix
3	剛体 B のフレーム マトリクス（B_Fr）	＜－＞	Matrix

(2) 外部関数

機能	コンストラクタ	
入力	剛体 A の剛体関連データ（関連 A）	const RbRelation&
	剛体 B の剛体関連データ（関連 B）	const RbRelation&
処理	〔I.関連 A.モデルを剛体 A、I.関連 B.モデルを剛体 B と記載する〕 A.剛体 B の基準 Mg〔H/P/B=0 度の Mg〕＝rbxSys::getFrozenRotMg(剛体 B、A.剛体 B の Mg) A.FrA〔フレーム A の Ml〕=Inv(剛体 A の Mg×関連 A.慣性主軸 Ml)×A.剛体 B の基準 Mg A.FrB〔フレーム B の Ml〕=Inv(剛体 B の Mg×関連 B.慣性主軸 Ml)×A.剛体 B の Mg 〔軸の向きを変更する(Y→Z, X→X, -Z→Y)〕 #.A_Fr=Mat3<Vector>(A.FrA.off, A.FrA.sqmat.v1, -A.FrA.sqmat.v3, A.FrA.sqmat.v2) #.B_Fr=Mat3<Vector>(A.FrB.off, A.FrB.sqmat.v1, -A.FrB.sqmat.v3, A.FrB.sqmat.v2) 剛体 B.相対角度.z != 0 ＜警告設定＞(剛体 B.名称、座標変換の固定異常)	

7.6.2.6 3DOF フレーム データ

Bullet 物理ワールド管理コア クラス::3DOF フレーム データ(DynWorldBt::Ge3DofFrame：DynWorld::WarningInfo)の構成を示す。

(1) 管理情報

表 7-53 3DOF フレーム データの管理情報

番号	項目	初期値	型
1	物理ワールド管理コア クラスの警告データの管理データ	＜－＞	DynWorld::WarningInfo
2	剛体 A のフレーム マトリクス（A_Fr）	＜－＞	Matrix
3	剛体 B のフレーム マトリクス（B_Fr）	＜－＞	Matrix

(2) 外部関数

機能	コンストラクタ	
入力	剛体 A の剛体関連データ（関連 A）	const RbRelation&
	剛体 B の剛体関連データ（関連 B）	const RbRelation&
処理	〔I.関連 A.モデルを剛体 A、I.関連 B.モデルを剛体 B と記載する〕 A.剛体 B の基準 Mg〔H/P/B=0 度の Mg〕＝rbxSys::getFrozenRotMg(剛体 B、A.剛体 B の Mg) A.FrA〔フレーム A の Ml〕=Inv(剛体 A の Mg×関連 A.慣性主軸 Ml)×A.剛体 B の基準 Mg A.FrB〔フレーム B の Ml〕=Inv(剛体 B の Mg×関連 B.慣性主軸 Ml)×A.剛体 B の Mg 〔軸の向きを変更する(Y→Z, X→Y, Z→X)〕 #.A_Fr=Mat3<Vector>(A.FrA.off, A.FrA.sqmat.v3, A.FrA.sqmat.v1, A.FrA.sqmat.v2) #.B_Fr=Mat3<Vector>(A.FrB.off, A.FrB.sqmat.v3, A.FrB.sqmat.v1, A.FrB.sqmat.v2)	

7.6.2.7 Bullet デバッグ描画クラス

Bullet デバッグ描画クラスの(dynBulletDebugDraw：btIDebugDraw)の構成を示す。

(1) 管理情報

表 7-54 Bullet デバッグ描画クラスの管理情報

番号	項目	初期値	型
1	Bullet 物理ワールド	＜・＞	btDiscreteDynamicsWorld*
2	デバッグ モード	＜・＞	int
3	描画コンテキスト	＜・＞	BaseDraw*

(2) 外部関数

機能	コンストラクタ（構築完了時にデバッグ描画を行う）	
入力	Bullet 物理ワールド	btDiscreteDynamicsWorld*
	デバッグ モード	int
	描画コンテキスト	BaseDraw*
処理	#.* = I.* #.描画コンテキスト.＜座標変換行列設定＞(nullptr, Matrix()) #.Bullet 物理ワールド.＜デバッグ描画オブジェクト設定※1＞(#) #.Bullet 物理ワールド.＜デバッグ描画※2＞() ※1 btDiscreteDynamicsWorld::setDebugDrawer() ※2 btDiscreteDynamicsWorld::debugDrawWorld()	

機能	デストラクタ	
処理	#.描画コンテキスト= nullptr #.描画コンテキスト != nullptr #.Bullet 物理ワールド.＜デバッグ描画オブジェクト設定※1＞(nullptr) #.Bullet 物理ワールド= nullptr ※1 btDiscreteDynamicsWorld::setDebugDrawer()	

(3) 内部関数

機能	ⓥ直線描画	
入力	開始位置	const btVector3&
	終了位置	const btVector3&
	描画色	const btVector3&
処理	#.描画コンテキスト != nullptr #.描画コンテキスト.＜線色設定＞(I.描画色) #.描画コンテキスト.＜直線描画＞(I.開始位置※1、I.終了位置※1、Z クリップ) ※1 右手系から左手系への変換を実施する	

機能	ⓥ接点描画	
入力	位置	const btVector3&
	法線	const btVector3&
	距離（未使用）	btScalar
	ライフタイム（未使用）	int
	描画色	const btVector3&
処理	#.描画コンテキスト != nullptr #.描画コンテキスト.＜線色設定＞(I.描画色) A.位置={座標変換※1}←I.位置 A.球サイズ=(1,1,1) #.描画コンテキスト.＜球描画＞(A.位置、A.球サイズ、I.描画色、Z クリップ) #.デバッグ モード.DBG_DrawNormals == 1 A.法線={座標変換※1}←I.法線 A.終了位置= A.位置＋A.法線×A.球サイズ×5 #.描画コンテキスト.＜直線描画＞(A.位置、A.終了位置、Z クリップ) ※1 右手系から左手系へ変換	

機能	ⓥエラーと警告の報告	
入力	エラーと警告を表す文字列	const char*
処理	無処理	

機能	ⓥテキスト描画	
入力	位置	const btVector3&
	テキスト	const char*
処理	無処理	

機能	ⓥデバッグ モード設定	
入力	デバッグ モード	int
処理	#.デバッグ モード=I.デバッグ モード	

機能	ⓥデバッグ モード取得©	
処理	出力値=#.デバッグ モード	
出力	デバッグ モード	int

7.6.3 PhysX 物理ワールド管理コア クラス

PhysX 物理演算エンジン用の物理ワールド管理を実現する PhysX 物理ワールド管理コア クラス(DynWorldPx：public DynWorld)の構成を示す。

(1) 管理情報

表 7-55 PhysX 物理ワールド管理コア クラスの管理情報

番号	項目	初期値	型
1	剛体関連データ(型)※1	－	RbRelation
2	動的剛体関連配列	＜－＞	BaseArray<RbRelation>
3	静的剛体関連配列	＜－＞	BaseArray<RbRelation>
4	動作可能状態	false	Bool
5	PhysX 基幹オブジェクト	－	－
5.1	メモリ アロケーター（16 バイト境界アライン）	＜－＞	PxDefaultAllocator
5.2	エラー コールバック	＜－＞	PxDefaultErrorCallback
5.3	物理基礎	nullptr	PxFoundation*
5.4	物理要素	nullptr	PxPhysics*
5.5	CPU ディスパッチャー	nullptr	PxDefaultCpuDispatcher*
5.6	物理シーン	nullptr	PxScene*
5.7	ポリゴン メッシュ操作	nullptr	PxCooking*

(2) 外部関数

機能	Ⓢ物理ワールド管理コア生成	
処理	新規メモリ領域に物理ワールド管理コアを生成する。	
出力	物理ワールド管理コア	DynWorldPx*

機能	Ⓥ物理エンジン種別取得Ⓒ	
処理	出力値＝PE::PHYSX	
出力	物理エンジン種別	PE

機能	Ⓥ物理ワールド初期化	
入力	物理ワールド モデル	BaseObject*
	単位変換データ	const DynUnitCnv&
	物理ワールド要素候補データ	const Candidates&
処理	#.警告配列.サイズ=0 #.動作可能状態 != true の場合、終了(出力値=false) #.(単位変換データ).＜単位変換データ設定＞(I.単位変換データ) ＜クリーン アップ＞() ＜動的剛体追加＞(物理ワールド要素候補データ.動的剛体候補配列)※1 ＜静的剛体追加＞(物理ワールド要素候補データ.静的剛体候補配列) ※1 出力値=true ※1 処理で異常が発生した場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

機能	ⓥ動的剛体削除	
入力	削除対象剛体配列（配列要素のインデックス番号は昇順）	const BaseArray<IndexedModel>&
処理	@=削除対象剛体配列[*]の処理〔降順に処理〕 A.PhysX 動的剛体=#.動的剛体関連配列[@.インデックス番号].＜PhysX 動的剛体取得＞() #.物理シーン.＜剛体除去＞(A.PhysX 動的剛体) A.PhysX 動的剛体を削除 #.動的剛体関連配列.＜除去＞(@.インデックス番号)	

機能	ⓥ静的剛体削除	
入力	削除対象剛体配列（配列要素のインデックス番号は昇順）	const BaseArray<IndexedModel>&
処理	@=削除対象剛体配列[*]の処理〔降順に処理〕 A.PhysX 静的剛体=#.静的剛体関連配列[@.インデックス番号].＜PhysX 静的剛体取得＞() #.物理シーン.＜剛体除去＞(A.PhysX 静的剛体) A.PhysX 静的剛体を削除 #.静的剛体関連配列.＜除去＞(@.インデックス番号)	

機能	ⓥモニター データ無効化	
入力	モニター データ	const DynBodyMonData*
処理	@=動的剛体関連配列[*]と静的剛体関連配列[*]の処理 @.モニター データ == I.モニター データの場合、@.モニター データ=nullptr	

機能	ⓥシミュレーション進行	
入力	物理シミュレーション パラメータ	const DynSimParams&
処理	〔物理シミュレーション パラメータを I と記載する〕 #.動作可能状態 != true、終了 #.物理シーン.＜重力設定＞(btVector3(0, #.MtoU(-I. 重力加速度), 0)) I.時間フレーム当たりのステップ数回の繰り返し処理 #.物理シーン.＜シミュレーション進行＞(I.ステップ時間) #.物理シーン.＜結果読み取り＞()	

機能	ⓥシミュレーション結果更新
処理	#.動作可能状態 != true、終了 A.単位変換データ=#.(単位変換データ).＜読み出し専用単位変換データ取得＞() @=#.動的剛体関連配列[*]の処理 @.＜ネイティブ モデルの位置と角度更新＞(A.単位変換データ)

機能	ⓥクリーン アップ
処理	#.動作可能状態 != true、終了 #.動的剛体関連配列.サイズ=0 #.静的剛体関連配列.サイズ=0 @=#.物理シーン.剛体[*]の処理※1 #.物理シーン.＜剛体除去＞(@) @を削除 ※1 物理シーン内の剛体の取得は PxScene::getNbActors()、PxScene::getActors()を使用する

(3) 内部関数

機能	コンストラクタ
処理	PhysX 基幹オブジェクトを生成 #.動作可能状態 = PhysX 基幹オブジェクト生成完了 ? true : false

機能	デストラクタ
処理	<クリーン アップ>() PhysX 基幹オブジェクトを削除

処理	動的剛体追加	
入力	動的剛体候補配列	const BaseArray<DynBodyData>&
処理	#.動的剛体関連配列.サイズ=I.動的剛体候補配列.サイズ※1 @=0 から動的剛体候補配列数-1 の処理 A.剛体データ=動的剛体候補配列[@] A.PhysX 剛体=<剛体生成>(剛体データ)※2 #.物理シーン.<剛体追加>(A.PhysX 剛体) A.剛体データ.Ml 有効・無効 == false 動的剛体関連配列[@].<コンストラクタ>(A.剛体データ.実形状のＳＣモデルD.モデル、 A.剛体データ.剛体モニター データ、A.PhysX 剛体) そうではない 動的剛体関連配列[@].<コンストラクタ>(A.剛体データ.実形状のＳＣモデルD.モデル、 A.剛体データ.慣性主軸 Ml、A.剛体データ.L 原点 Ml、 A.剛体データ.剛体モニター データ、A.PhysX 剛体) ※1 配列のサイズ変更できない場合は終了(出力値=false) ※2 剛体を生成できない場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

処理	静的剛体追加	
入力	静的剛体候補配列	const BaseArray<DynBodyData>&
処理	#.静的剛体関連配列.サイズ=I.静的剛体候補配列.サイズ※1 @=0 から静的剛体候補配列数-1 の処理 A.剛体データ=静的剛体候補配列[@] A.PhysX 剛体=<剛体生成>(剛体データ)※2 #.物理シーン.<剛体追加>(A.PhysX 剛体) 静的剛体関連配列[@].<コンストラクタ>(A.剛体データ.実形状のＳＣモデルD.モデル、nullptr、A.PhysX 剛体) ※1 配列のサイズ変更できない場合は終了(出力値=false) ※2 剛体を生成できない場合は終了(出力値=false)	
出力	処理結果（true： 正常終了 / false： 異常終了）	Bool

機能	剛体生成	
入力	剛体データ	const DynBodyData&
処理	〔I.剛体データ.実形状のＳＣモデルDを実ＳＣモデルD、 I.剛体データ.衝突判定形状のＳＣモデルDを偽ＳＣモデルD、 I.剛体データ.動力学データを動力学データと記載する〕 A.PhysX ジオメトリ=<ジオメトリ生成>(I.偽ＳＣモデルD、I.動力学データ.L 原点 Ml)※1 A.剛体の慣性主軸 Mg=I.実ＳＣモデルD.モデル.Mg I.動力学データ.Ml 有効・無効 == true A.剛体の慣性主軸 Mg＝A.剛体の慣性主軸 Mg×I.動力学データ.慣性主軸 Ml 出力値=<剛体生成>(A.剛体の慣性主軸 Mg、A.PhysX ジオメトリ、I.動力学データ)※2 A.PhysX ジオメトリを削除 ※1 処理異常の場合は終了(出力値=nullptr)	
出力	PhysX 剛体データ（処理異常の場合は nullptr）	PxRigidActor*

機能	剛体生成	
入力	慣性主軸 Mg	const Matrix&
	PhysX ジオメトリ	PxGeometry*
	動力学データ	const DynData&
処理	A.PhysX マテリアル=#.物理要素.＜マテリアル生成＞(　I.動力学データ.摩擦、I.動力学データ.摩擦、I.動力学データ.反発)※1 A.PhysX 形状=#.物理要素.＜形状生成＞(I.PhysX ジオメトリ、A.PhysX マテリアル)※1 A.PhysX 座標変換={左手系から右手系変換}←I.慣性主軸 Mg I.動力学データ.＜静的性質確認＞() == true〔静的剛体の場合〕 　A.PhysX 静的剛体 ＝#.物理要素.＜静的剛体生成＞(A.PhysX 座標変換)※1 　A.PhysX 静的剛体.＜形状アタッチ＞(A.PhysX 形状)※1 　A.PhysX マテリアルと A.PhysX 形状を削除 　終了(出力値=A.PhysX 静的剛体) A.PhysX 動的剛体 ＝#.物理要素.＜動的剛体生成＞(A.PhysX 座標変換)※1 A.PhysX 動的剛体.＜形状アタッチ＞(A.PhysX 形状)※1 A.PhysX マテリアルと A.PhysX 形状を削除 A.PhysX 動的剛体.＜質量設定＞(I.動力学データ.質量) A.主慣性モーメント= 　I.動力学データ.主慣性モーメント× 　#.(単位変換データ).メートル単位の 2 乗 → システム単位 2 乗変換係数× 　I.動力学データ.(並進･回転性質データ).慣性モーメント L スケール ² A.PhysX 動的剛体.＜主慣性モーメント設定＞(A.主慣性モーメント) A.PhysX 動的剛体.＜線形ダンピング設定＞(I.動力学データ.線形ダンピング) A.PhysX 動的剛体.＜回転ダンピング設定＞(I.動力学データ.回転ダンピング) A.線形速度={左手系から右手系変換}← 　(I.動力学データ.初期線形速度×#.(単位変換データ).メートル単位 → システム単位変換係数) A.PhysX 動的剛体.＜初期線形速度設定＞(A.線形速度) A.PhysX 動的剛体.＜初期回転速度設定＞(I.動力学データ.初期回転速度) A.PhysX マテリアルと A.PhysX 形状を削除 終了(出力値=A.PhysX 動的剛体) ※1 処理異常の場合は、A.PhysX マテリアルと A.PhysX 形状を削除後に終了(出力値=nullptr)	
出力	PhysX 剛体データ （処理異常の場合は nullptr）	PxRigidActor*

機能	ジオメトリ生成	
入力	ＳＣモデルD	const ScaledObject&
	L 原点 Ml（慣性主軸座標系→ローカル座標系）	const Matrix&
処理	出力値=表 6-16 の内容に従って PhysX ジオメトリを生成〔I.ＳＣモデルD.モデルがプリミティブでない場合は、頂点座標を慣性主軸座標系（右手系）に変換する。〕	
出力	PhysX ジオメトリ （処理異常の場合は nullptr）	PxGeometry*

7.6.3.1 剛体関連データ

PhysX 物理ワールド管理コア クラス::剛体関連データ(DynWorldPx::RbRelation)の構成を示す。

(1) 管理情報

表 7-56 剛体関連データ型

番号	項目	初期値	型
1	ネイティブ剛体の情報	－	－
1.1	モデル	nullptr	BaseObject*
1.2	Ml 有効・無効（true： 有効 / false： 無効）	false	Bool
1.3	慣性主軸 Ml（L 座標系→慣性主軸座標系）	＜－＞	Matrix
1.4	L 原点 Ml（慣性主軸座標系→L 座標系）	＜－＞	Matrix
1.5	回転逆 Mg（G 座標系の角度→L 座標系の角度）	＜－＞	SqMat3<Vector>
1.6	モニター データ	nullptr	const DynBodyMonData*
2	PhysX 剛体の情報	－	－
2.1	剛体	nullptr	PxRigidActor*

(2) 外部関数

機能	デフォルト コンストラクタ
処理	表 7-56 の初期値から剛体関連データを生成する。

機能	コンストラクタ	
入力	剛体モデル	BaseObject*
	モニター データ	const DynBodyMonData*
	PhysX 剛体	PxRigidActor*
処理	#.@ = I.*〔入力で指定されない項目は初期値を設定〕 #.ローカル マトリクス有効・無効=false ＜モニター データ初期化＞()	

機能	コンストラクタ	
入力	剛体モデル	BaseObject*
	慣性主軸 Ml	const Matrix&
	L 原点 Ml	const Matrix&
	モニター データ	const DynBodyMonData*
	PhysX 剛体	PxRigidActor*
処理	#.@ = I.* #.ローカル マトリクス有効・無効=true ＜モニター データ初期化＞()	

機能	PhysX 動的剛体取得（キャスト）	
処理	#.PhysX 剛体.＜タイプ取得＞() != 動的剛体、アサート 出力値=PhysX 剛体	
出力	PhysX 動的剛体	PxRigidDynamic*

機能	PhysX 静的剛体取得（キャスト）	
処理	#.PhysX 剛体.＜タイプ取得＞() != 静的剛体、アサート 出力値=PhysX 剛体	
出力	PhysX 動的剛体	PxRigidStatic*

機能	モニター データ初期化
処理	#.モニター データ != nullptr 　#.モニター データ.＜書き込み可能確認＞() == true 　　#.モニター データ.* = 0

機能	ネイティブ モデルの位置と角度更新©	
入力	単位変換データ	const DynUnitCnv&
処理	〔I.単位変換データ.システム単位 → メートル単位変換係数を UtoM と記載する〕 A.剛体の Mg={右手系→左手系変換}←#.PhysX 剛体の Mg #.ローカル マトリクス有効・無効 == true A.剛体の Mg = A.剛体の Mg×#.L 原点 Ml #.ネイティブ モデルの Mg = A.剛体の Mg #.回転逆 Mg ={転置}←A.剛体の Mg.sqmat #.モニター データ != nullptr #.モニター データ.＜書き込み可能確認＞() == true #.モニター データ.線形速度 = ＜直動速度取得＞()×UtoM #.モニター データ.回転速度 = ＜HPB 回転速度取得＞()	

機能	直動速度取得©	
処理	出力値={右手系→左手系変換}←#.PhysX 剛体.直動速度	
出力	直動速度（単位：U/S）	Vector

機能	XYZ 回転速度取得©	
処理	出力値={右手系→左手系変換}←#.PhysX 剛体.回転速度	
出力	XYZ 回転速度（単位：Rad/S）	Vector

機能	HPB 回転速度取得©	
処理	出力値={XYZ→HPB 変換}←{右手系→左手系変換}←#.PhysX 剛体.回転速度	
出力	HPB 回転速度（単位：Rad/S）	Vector

7.6.4 RoboBio-X 物理ワールド管理コア クラス

RoboBio-X 物理演算エンジン用の物理ワールド管理を実現する RoboBio-X 物理ワールド管理コア クラス(DynWorldRbx：public DynWorld)の構成は以下の点を除き Bullet 物理ワールド管理コア クラスと同等とする。

(1) 物理演算エンジンのオブジェクト型の差異

表 7-57 Bullet と RoboBio-X 物理演算エンジンのオブジェクト型の差異

番号	オブジェクト	Bullet	RoboBio-X
1	基幹オブジェクト	－	－
1.1	衝突検出(広域)	btDbvtBroadphase	rbxDbvtBroadphase
1.2	衝突検出の構成	btDefaultCollisionConfiguration	rbxCollisionConfiguration
1.3	衝突検出(狭域)ﾃﾞｨｽﾊﾟｯﾁｬｰ	btCollisionDispatcher	rbxCollisionDispatcher
1.4	ソルバー	btSequentialImpulseConstraintSolver	rbxConstraintSolver
1.5	物理ワールド	btDiscreteDynamicsWorld	rbxDynamicsWorld
2	剛体	btRigidBody	rbxRigidBody
3	形状	表 6-12 と表 6-13 を参照のこと。	表 6-18 と表 6-19 を参照のこと。
4	拘束	表 6-14 を参照のこと。	表 6-20 を参照のこと。

(2) 衝突判定の対象となる形状の差異

表 6-12 から表 6-13 と、表 6-18 から表 6-19 を参照のこと。

(3) 物理演算対象のジョイント タイプの差異

表 6-14 と表 6-20 を参照のこと。

(4) ジョイント関連データ中の回転データと直動データの非参照項目の差異

RoboBio-X 物理ワールド管理コアと RoboBio-X 物理演算エンジンは表 7-49 の※1,2 で指定されている項目を物理演算の項として参照する。

7.6.5 物理ワールド管理クラス

動作環境に応じた物理ワールド管理を実現する物理ワールド管理クラス(DynWorldNode：public ObjectData, protected DynUnitCnv)の構成を示す。

(1) 管理情報

表 7-58 物理ワールド管理クラスの管理情報

番号	項目	初期値	型
1	状態（0：ロード/1：コピー/3：有効/4：ジョイント構成異常）	3	enum class State
2	物理演算エンジン種別	1(Bullet)	PE
3	時間スケール	1	Float
4	重力加速度(m/s²)	9.80665	Float
5	1 フレームあたりのステップ数	5	Int32
6	1 フレームあたりの最大演算回数	10	Int32
7	前回処理のフレーム位置	-1	Int32
8	物理ワールド コア	nullptr	DynWorld*
9	物理ワールド要素リスト	－	－
9.1	動的剛体リスト	＜－＞	BaseArray<BaseObject*>
9.2	静的剛体リスト	＜－＞	BaseArray<BaseObject*>
9.3	ジョイント リスト	＜－＞	BaseArray<BaseObject*>

(2) UI コンテナ（シミュレーション環境のみ）

図 7-24 にジョイント管理クラスの UI コンテナを示す。（青枠で囲んだ項目は入力変更不可。）

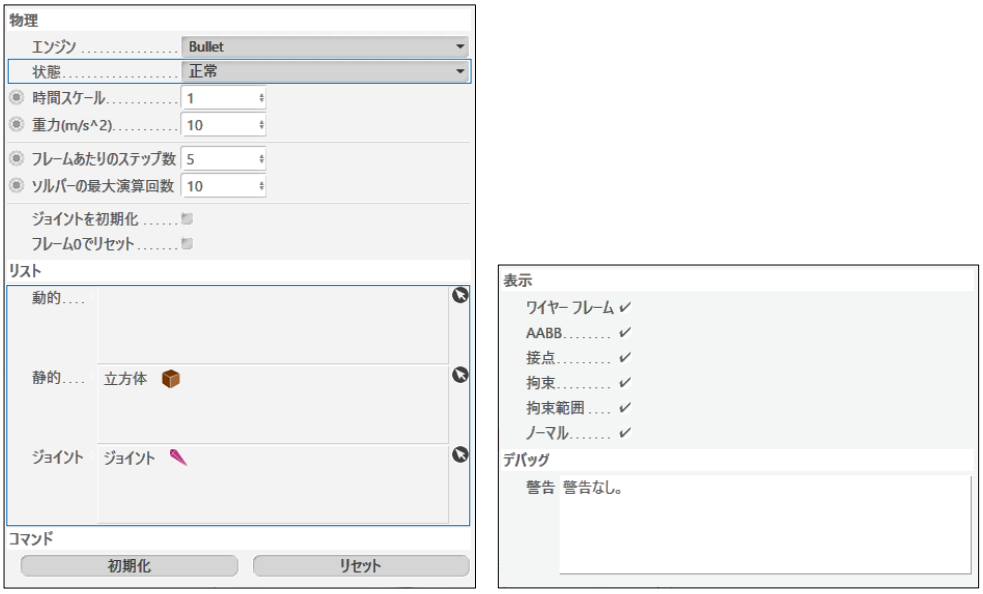

図 7-24 物理ワールド管理の UI コンテナ

(3) 外部関数

機能	Ⓢ物理ワールド管理生成	
処理	新規メモリ領域に物理ワールド管理を生成する。	
出力	物理ワールド管理	NodeData*

機能	ⓥUI コンテナの初期化（シミュレーション環境のみ）	
入力	物理ワールド モデル	GeListNode*
処理	#.(単位変換データ).＜初期化＞() I.物理ワールド モデル.UI コンテナ.* = 初期値	
出力	処理結果（常に true）	Bool

機能	ⓥメッセージ ハンドラ	
入力	物理ワールド モデル	GeListNode*
	タイプ	Int32
	データ	void*
処理	入力値から以下のトリガーを検出 トリガー別の処理	

トリガー	処理
ロード	＜ロード＞(I.タグ，I.データ.空間管理)
UI 項目値変更	＜UI 項目値変更＞(I.データ.UI 項目 ID，I.物理ワールド モデル，I.データ.空間管理)
コマンド発行	＜コマンド実行＞(I.物理ワールド モデル，I.データ.コマンド ID)
モニター データ無効化	＜モニター データ無効化＞(I.データ)

出力	処理結果（常に true）	Bool

機能	ⓥUI 項目の有効性取得（シミュレーション環境のみ）	
入力	物理ワールド モデル	GeListNode*
	UI 項目 ID	const DescID&
	パラメータ データ（未使用）	const GeData&
	フラグ（未使用）	DESCFLAGS_ENABLE
	ディスクリプション コンテナ（未使用）	const BaseContainer*
処理	出力値=true	
出力	編集可否設定値（true：編集可能／false：編集不可）	Bool

機能	ⓥ実行タイミング追加（シミュレーション環境のみ）	
入力	物理ワールド モデル	BaseObject*
	優先度リスト	PriorityList*
処理	I.優先度リスト.＜追加＞(I.物理ワールド モデル，EXECUTIONPRIORITY_GENERATOR + 999，起動フラグ=0)	
出力	処理結果（常に true）	Bool

機能	ⓥ実行ハンドラ	
入力	物理ワールド モデル	BaseObject*
	空間管理	BaseDocument*
	スレッド（未使用）	BaseThread*
	優先度（未使用）	Int32
	起動フラグ（未使用）	EXECUTIONFLAGS
処理	A.UI コンテナ=I.物理ワールド モデル.UI コンテナ ＜状態遷移＞(I.物理ワールド モデル，I.空間管理，A.UI コンテナ) #.状態 == 通常 ＜通常状態の実行＞(I.物理ワールド モデル，I.空間管理，A.UI コンテナ) A.UI コンテナ.状態=#.状態 終了(EXECUTIONRESULT::OK)	
出力	実行結果(EXECUTIONRESULT::OK)	EXECUTIONRESULT

機能	ⓥ複製	
入力	複製先物理ワールド管理	NodeData*
	複製元物理ワールド モデル（本関数では未使用）	GeListNode*
	複製先物理ワールド モデル（本関数では未使用）	GeListNode*
	複製フラグ（本関数では未使用）	COPYFLAGS
	エイリアス トランスレータ（本関数では未使用）	AliasTrans*
処理	出力値＝基本クラス定義の＜複製＞(I.*) 出力値 == true I.複製先物理ワールド管理.状態=コピー	
出力	処理結果（true：正常終了/false：異常終了）	Bool

機能	ⓥ可視エレメント描画（シミュレーション環境のみ）	
入力	物理ワールド モデル	BaseObject*
	描画パス	DRAWPASS
	描画コンテキスト	BaseDraw*
	描画ヘルパー	BaseDrawHelp*
処理	I.描画パス != オブジェクト、終了(出力値=DRAWRESULT::OK) A.UI コンテナ=物理ワールド モデル.UI コンテナ A.デバッグ描画指定データ=DebugDraw.＜コンストラクタ＞(A.UI コンテナ.表示/ワイヤー フレーム, A.UI コンテナ.表示/AABB, A.UI コンテナ.表示/接点, A.UI コンテナ.表示/拘束, A.UI コンテナ.表示/拘束範囲, A.UI コンテナ.表示/ノーマル) A.デバッグ描画指定データ.＜デバッグ描画確認＞() == true ＜物理ワールド管理コア選択＞(I.物理ワールド モデル，A.UI コンテナ) #.物理ワールド コア.＜デバッグ描画＞(I.描画コンテキスト，A.デバッグ描画指定データ) 出力値= DRAWRESULT::OK	
出力	処理結果（常に DRAWRESULT::OK）	DRAWRESULT

機能	物理演算エンジン種別取得	
処理	出力値=#.物理演算エンジン種別	
出力	物理演算エンジン種別	PE

(4) 内部関数

機能	コンストラクタ
処理	表 7-58 の初期値から物理ワールド管理を生成する。

機能	デストラクタ
処理	#.物理ワールド コア != nullptr (rbxSys::RefSingleThrMng::)＜開放＞(#.物理ワールド コア)

機能	キャッシュ データ更新	
入力	UI コンテナ	BaseContainer*
処理	#.[2~6] = I.UI コンテナ.@	

機能	警告更新	
入力	UI コンテナ	BaseContainer*
処理	#.物理ワールド コア != nullptr I.UI コンテナ.デバッグ/警告 ={文字列化}←#.物理ワールド コア.警告配列[*]	

機能	ロード	
入力	物理ワールド モデル	BaseObject*
	空間管理	BaseDocument*
処理	#.状態=ロード	

機能	UI 項目値変更	
入力	UI 項目 ID	Int32
	物理ワールド モデル	GeListNode*
	空間管理：未使用	BaseDocument*
処理	@=[図 7-24 中の入力変更不可の項目]の処理 I.UI 項目 ID == @.ID I.物理ワールド モデル.UI コンテナ.[I.UI 項目 ID の項目]=#.$〔入力変更を取り消す〕	

機能	コマンド実行	
入力	物理ワールド モデル	GeListNode*
	コマンド ID	Int32
処理	コマンド ID 別の処理	

コマンド ID	処理
初期化	I.物理ワールド モデル.UI コンテナ.ジョイントを初期化 == true 物理ワールド モデル.空間管理.＜マルチ メッセージ送信＞(ブロード キャスト，ジョイント初期化指示，I.物理ワールド モデル) ＜物理ワールド初期化＞(I.物理ワールド モデル，I.物理ワールド モデル.空間管理)※1
リセット	＜物理ワールド リセット＞(I.物理ワールド モデル)※1

※1 処理異常、又は警告ありの場合、その旨を通知する。

機能	物理ワールド初期化	
入力	物理ワールド モデル	GeListNode*
	空間管理	BaseDocument*
	初期の位置と角度保存指定（true：保存する／false：保存しない／省略時：true）	Bool
処理	(単位変換データ.)＜初期化＞() A.UI コンテナ=I.物理ワールド モデル.UI コンテナ ＜物理ワールド管理コア選択＞(I.物理ワールド モデル，A.UI コンテナ) A.物理ワールド要素候補データ=空リスト〔物理ワールド要素候補データを候補データと記載する〕 A.先頭モデル=物理ワールド モデル.最初の子 A.先頭モデル != nullptr ＜物理ワールド要素の候補リスト生成＞(A.先頭モデル，A.候補データ，I. 空間管理)※1 A.単位変換データ=(単位変換データ).＜読み出し専用単位変換データ取得＞() #.物理ワールド コア.＜物理ワールド初期化＞(物理ワールド モデル，A.単位変換データ，A.候補データ)※2 物理ワールド初期化 == 正常完了 ＜物理ワールド要素リスト生成＞(A.候補データ，A.UI コンテナ)※3 物理ワールド要素リスト生成 == 正常完了、且つ I.初期の位置と角度保存指定 == true ＜初期の位置と角度を保存＞() ＜警告更新＞(A.UI コンテナ) 出力値=※2 と※3 が正常完了 ? true : false ※1 処理異常の場合、終了(出力値=false)	
出力	処理結果	Bool

機能	物理ワールド リセット	
入力	物理ワールド モデル	GeListNode*
処理	A.UI コンテナ=I.物理ワールド モデル.UI コンテナ A.空間管理=I.物理ワールド モデル.空間管理 ＜物理ワールド再構成＞(A.UI コンテナ，A.空間管理) 物理ワールド再構成 != 正常完了、終了(出力値=false) ＜位置と角度をリセット＞() ＜物理ワールド初期化＞(I.物理ワールド モデル，A.空間管理、false) 出力値=true	
出力	処理結果	Bool

<table>
<tr><td>機能</td><td colspan="2">物理ワールド要素候補リスト生成</td></tr>
<tr><td rowspan="4">入力</td><td>モデル</td><td>BaseObject*</td></tr>
<tr><td>物理ワールド要素候補データ</td><td>DynWorld::Candidates&</td></tr>
<tr><td>空間管理</td><td>BaseDocument*</td></tr>
<tr><td>親ジョイントの要素番号（無指定：-1）</td><td>Int32</td></tr>
<tr><td>処理</td><td colspan="2">〔I.物理ワールド要素候補データを候補データと記載する〕

@=I.モデルと同階層の全てのモデル〔順序は昇順〕の処理
A.親ジョイントの要素番号=I.親ジョイントの要素番号
@にアタッチされたタグ別の処理
<table><tr><th>タグ</th><th>処理</th></tr><tr><td>剛体
管理</td><td>タグが有効
A.剛体管理=@.タグ.剛体管理
A.剛体管理.＜所属確認＞() == false〔節に属していない場合〕
A.剛体管理.＜データ キャシュ更新＞(空間管理)
A.剛体管理.＜動的剛体確認＞() == true
＜剛体候補追加＞(@、A.剛体管理、I.候補データ.動的剛体候補配列)※1
そうでない場合
＜剛体候補追加＞(@、A.剛体管理、I.候補データ.静的剛体候補配列)※1</td></tr><tr><td>ｼﾞｮｲﾝﾄ
管理</td><td>タグが有効
A.ジョイント管理=@.タグ.ジョイント管理
A.ジョイント管理.＜ジョイント設定の有効確認＞() == true
A.ジョイント管理.＜データ キャシュ更新＞(空間管理)
A.要素番号=I.候補データ.ジョイント候補配列.要素数
I.候補データ.ジョイント候補配列を 1 要素分拡張※1
A.Jt 候補データ=I.候補データ.ジョイント候補配列の末尾の参照
A.Jt 候補データ.ジョイント データ=A.ジョイント管理.＜ジョイント データ取得＞()
A.Jt 候補データ.ジョイント データ.ルート指定 == false
A.Jt 候補データ.親ジョイント候補データの番号=A.親ジョイントの要素番号
A.親ジョイントの要素番号=A.要素番号</td></tr></table>
@の子モデルが存在する場合
＜物理ワールド要素候補リスト生成＞(
@の最初の子モデル、I.候補データ、I.空間管理、A.親ジョイントの要素番号)※1

出力値=true

※1 処理異常の場合、終了(出力値=false)</td></tr>
<tr><td>出力</td><td>処理結果（true：正常終了／false：異常終了）</td><td>Bool</td></tr>
</table>

<table>
<tr><td>機能</td><td colspan="2">Ⓢ剛体要素候補追加</td></tr>
<tr><td rowspan="3">入力</td><td>モデル</td><td>BaseObject*</td></tr>
<tr><td>剛体管理</td><td>class TagDynBody*</td></tr>
<tr><td>剛体候補配列</td><td>BaseArray<DynBodyData>&</td></tr>
<tr><td>処理</td><td colspan="2">A.動力学データ=I.剛体管理.＜動力学データ取得＞(A.衝突判定形状モデル、A.剛体モニター データ)
A.衝突判定形状モデル == nullptr
A.剛体データ=剛体データ::コンストラクタ(
SMDF::＜実形状ＳＣモデルＤ生成＞(I.モデル、I.剛体管理)、
A.動力学データ、A.剛体モニター データ)
I.剛体候補配列.＜末尾に追加＞(A.剛体データ)※1
そうでない場合
A.剛体データ=剛体データ::コンストラクタ(
SMDF::＜実形状ＳＣモデルＤ生成＞(I.モデル、I.剛体管理)、
SMDF::＜衝突判定形状ＳＣモデルＤ生成＞(A.衝突判定形状モデル)、
A.動力学データ、A.剛体モニター データ)
I.剛体候補配列.＜末尾に追加＞(A.剛体データ)※1

出力値=true

※1 処理異常の場合、終了(出力値=false)</td></tr>
<tr><td>出力</td><td>処理結果（true：正常終了／false：異常終了）</td><td>Bool</td></tr>
</table>

機能	物理ワールド要素リスト生成	
入力	物理ワールド要素候補データ	DynWorld::Candidates&
	UI コンテナ	BaseContainer*
処理	〔I.物理ワールド要素候補データ.動的剛体候補配列を動的候補配列、 I.物理ワールド要素候補データ.静的剛体候補配列を静的候補配列、 I.物理ワールド要素候補データ.ジョイント候補配列をジョイント候補配列と記載する〕 #.動的剛体リスト[*]=I.動的候補配列[*].実形状のＳＣモデルD.モデル※1 I.UI コンテナ.動的剛体=#.動的剛体リスト[*]※1 #.静的剛体リスト[*]=I.候補データ.静的剛体候補配列[*].実形状のＳＣモデルD.モデル※1 I.UI コンテナ.静的剛体=#.静的剛体リスト[*]※1 #.ジョイント リスト[*]=I.ジョイント候補配列[*].ジョイント データ.ジョイント モデル※1 I.UI コンテナ.ジョイント=#.ジョイント リスト[*]※1 出力値=true ※1 リソース不足等の処理異常の場合は以下の処理を実行する I.UI コンテナ.[動的剛体、静的剛体、ジョイント]=空リスト #.[動的剛体リスト、静的剛体リスト、ジョイント リスト]=空リスト 終了(出力値=false)	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	初期の位置と角度を保存©
処理	@=#.動的剛体リスト[*]の処理 A.剛体管理=剛体管理::＜剛体管理取得＞(@) A.剛体管理 != nullptr の場合、A.剛体管理.＜初期の位置と角度を保存＞() @=#.ジョイント リスト[*]の処理 A.ジョイント管理=ジョイント管理::＜ジョイント管理取得＞(@) A.ジョイント管理 != nullptr の場合、A.ジョイント管理.＜初期の位置と角度を保存＞()

機能	位置と角度をリセット©
処理	@=#.動的剛体リスト[*]の処理 A.剛体管理=剛体管理::＜剛体管理取得＞(@) A.剛体管理 != nullptr の場合、A.剛体管理.＜位置と角度をリセット＞() @=#.ジョイント リスト[*]の処理 A.ジョイント管理=ジョイント管理::＜ジョイント管理取得＞(@) A.ジョイント管理 != nullptr の場合、A.ジョイント管理.＜位置と角度をリセット＞()

機能	物理ワールド再構成	
入力	UI コンテナ	BaseContainer*
	空間管理	BaseDocument*
処理	#.動的剛体リスト.要素数 ＞0 　＜削除された物理演算要素を検知＞(　　I.空間管理、I.UI コンテナ.動的剛体リスト、#.動的剛体リスト、A.削除要素リスト) 　削除要素検知.削除数 ＞0 　　#.物理ワールド コア.＜動的剛体削除＞(A.削除要素リスト) 　　I.UI コンテナ.動的剛体リスト中の nullptr の空要素を削除 #.静的剛体リスト.要素数 ＞0 　＜削除された物理演算要素を検知＞(　　I.空間管理、I.UI コンテナ.静的剛体リスト、#.静的剛体リスト、A.削除要素リスト) 　削除要素検知.削除数 ＞0 　　#.物理ワールド コア.＜静的剛体削除＞(A.削除要素リスト) 　　I.UI コンテナ.静的剛体リスト中の nullptr の空要素を削除 〔ジョイント リストを Jt リストと記載する〕 #.Jt リスト.要素数 ＞0 　＜削除された物理演算要素を検知＞(　　I.空間管理、I.UI コンテナ.Jt リスト、#.Jt リスト、A.削除要素リスト) 　削除要素検知.削除数 ＞0 の場合、終了(出力値=false) 　@=#.Jt リスト[*]の処理 　　A.ジョイント管理=剛体管理::＜ジョイント管理取得＞(@) 　　A.ジョイント管理 == nullptr〔ジョイント管理タグが削除された場合〕、終了(出力値=false) 　　A.軸節構成確認結果=A.ジョイント管理.＜データ キャッシュ チェック＞(@, I.空間管理) 　　A.軸節構成確認結果 == false の場合、終了(出力値=false) そうでない場合 　I.UI コンテナ.Jt リスト.要素数 ＞0 の場合、終了(出力値=false) 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	削除された物理演算要素を検知	
入力	空間管理	BaseDocument*
	物理演算要素リスト 1（UI コンテナのリスト）	const InExcludeData*
	物理演算要素リスト 2	BaseArray<BaseObject*>&
	削除要素リスト	BaseArray<IndexedModel>&
処理	削除要素リスト.要素数 ＞0、終了(出力値=-1) @=I.物理演算要素リスト 2.[物理演算要素リスト 1 に存在しないモデル]の処理 　削除要素リスト.＜追加＞(@.モデル，@の位置〔インデックス番号〕) 　物理演算要素リスト 2.＜除外＞(@) 出力値=削除要素リスト.要素数	
出力	処理結果（true：正常終了／false：異常終了）	Int32

機能	状態遷移	
入力	物理ワールド モデル	BaseObject*
	空間管理	BaseDocument*
	UI コンテナ	BaseContainer*
処理	状態別の処理	

状態	処理
ロード	#.前回処理のフレーム位置=現在のフレーム位置 コピーの処理へ
コピー	＜物理ワールド初期化＞(I.物理ワールド モデル、I.空間管理、false) 物理ワールド初期化 == 正常完了 　#.状態=通常 そうではない 　アサート
通常とｼﾞｮｲﾝﾄ構成異常	＜物理ワールド再構成＞(I.UI コンテナ、I.空間管理) #.状態 = 物理ワールド再構成 == 正常完了 ? 通常 : ジョイント構成異常

機能	通常状態の実行	
入力	物理ワールド モデル	BaseObject*
	空間管理	BaseDocument*
	UI コンテナ	BaseContainer*
処理	＜データ キャッシュ更新＞(I.UI コンテナ) 現在のフレーム位置 != #.前回処理のフレーム位置 現在のフレーム位置 == 0 I.UI コンテナ.フレーム 0 でリセット == true ＜物理ワールド リセット＞(I.物理ワールド モデル) そうではない、現在のフレーム位置 > #.前回処理のフレーム位置 ＜物理ワールド管理コア選択＞(I.物理ワールド モデル、I.UI コンテナ) 〔A.物理シミュレーション パラメータを@と記載する〕 @.ステップ時間=1 フレームの時間×#.時間スケール×Inv(#.1 フレームあたりのステップ数) @.重力加速度=#.重力加速度 @.最大演算回数=#.1 フレームあたりの最大演算回数 @.時間フレーム当たりのステップ数=#.1 フレームあたりの最大演算回数 #.前回処理のフレームから現在のフレームまでの処理 #.物理ワールド コア.＜シミュレーション進行＞(@) #.物理ワールド コア.＜シミュレーション結果更新＞() #.前回処理のフレーム位置=現在のフレーム位置	

機能	物理ワールド管理コア選択	
入力	物理ワールド モデル	BaseObject*
	UI コンテナ	BaseContainer*
処理	A.物理演算エンジンの選択=I.UI コンテナ.エンジン #.物理ワールド管理コア != nullptr A.物理演算エンジンの選択 == #.物理ワールド管理コア.＜物理エンジン種別取得＞()の場合、終了 #.(物理ワールド管理コア).＜物理ワールド管理コア削除＞(#.物理ワールド管理コア) 選択されている物理演算エンジン別の処理	

物理演算エンジン	処理
Bullet	Bullet 物理ワールド管理コア::＜物理ワールド管理コア生成＞()
PhysX	PhysX 物理ワールド管理コア::＜物理ワールド管理コア生成＞()
RoboBio-X	RoboBio-X 物理ワールド管理コア::＜物理ワールド管理コア生成＞()

機能	モニタ データ無効化	
入力	モニタ データ	const DynBodyMonData*
処理	#.物理ワールド コア != nullptr #.物理ワールド コア.＜モニタ データ無効化＞(I.モニタ データ)	

8. 体制

運動制御原始モジュール 順動力学演算ブロックの開発は運動制御系担当者（一人）が行う。

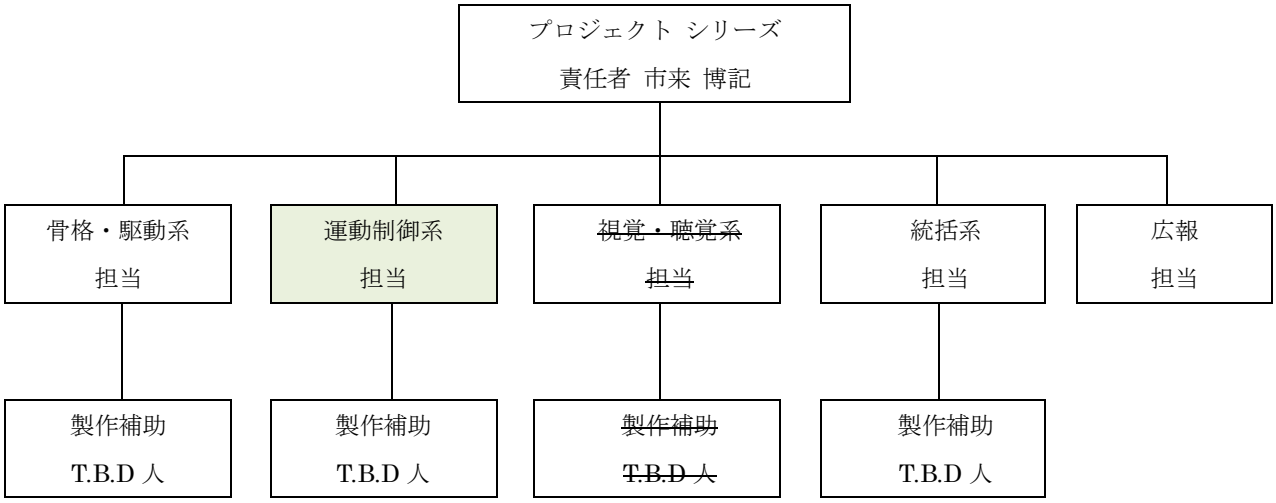

図 8-1 開発人員体制図

9. スケジュール

ロボバイオ リアルノイド オリジン - 運動制御原始モジュール開発計画書の 9 章を参照のこと。

10. 予算

開発作業は全て運動制御系担当者（一人）が行うものとし、人件費以外の費用は発生しない。

11. 問題点

現時点（2020 年 11 月）で承知している問題点と対策を表 11-1 に示します。

表 11-1 運動制御原始モジュール 順動力学演算ブロック開発における問題点

区分	内容	対策
非公開		

12. おわりに

本書 7 章に記載していないクラスの内部関数の構成を決定する際は、適切に機能分割してメンテナンス性の高いものとすること。

ニューラルソフト有限会社

改定履歴	改　　定　　内　　容	検認	照査	作成
初期作成 20/11/31		―	―	市来 博記
B 21/3/31	公開部と非公開部を分離した。	―	―	市来 博記
C 21/12/25	文書ファイルのプロパティを設定した。	―	―	市来 博記
D 22/4/30	表紙のフォーマットを変更した。 管理番号を採番した。	―	―	市来 博記