

CyberToolbox for Java3D

Release 1.2

User's Guide

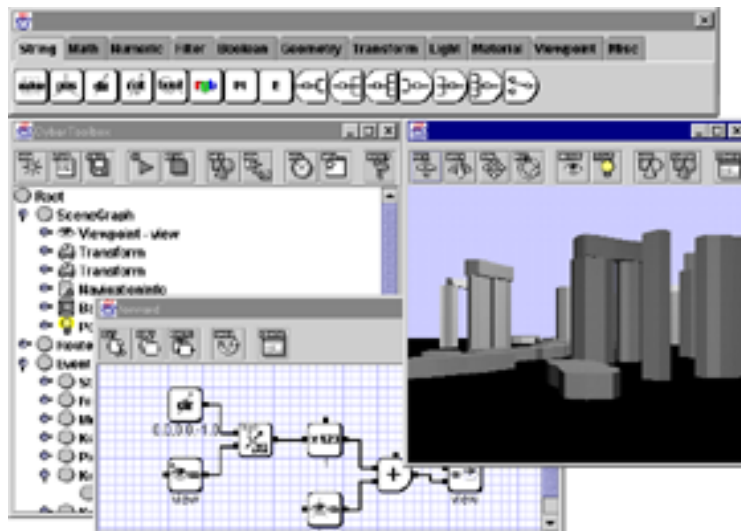
日本語版

CyberToolbox for Java3D について.....	3
インストール.....	5
チュートリアル.....	6
足し算.....	6
1. 加算動作を行うダイアグラムの定義	6
2. シミュレーションの実行	8
回転するボール.....	9
1. ボールオブジェクトの追加	9
2. オブジェクトを回転させるダイアグラムの定義	10
3. シミュレーションの実行	12
光るボール.....	13
1. ボールオブジェクトの追加	13
2. ピッキングイベントの追加	14
3. オブジェクト色を初期化するダイアグラムの定義	15
4. クリックされた時にボールが光るダイアグラムの定義	16
5. シミュレーションの実行	17
操作概要.....	18
ワールドウインドウ.....	19
シーングラフツリー	20
ルートツリー	21
イベントツリー	22
ツールバー	24
透視投影ウインドウ.....	26
ツールバー	26
ダイアグラム / モジュールウインドウ.....	30
ツールバー	33
空間動作概要.....	34
イベント.....	35
システムイベント	36
ユーザーイベント	38
ダイアグラム.....	43

モジュール	43
モジュール動作	45
<i>String</i>	45
<i>Numeric</i>	52
<i>Math</i>	59
<i>Filter</i>	69
<i>Boolean</i>	75
<i>Geometry</i>	79
<i>Transform</i>	82
<i>Material</i>	85
<i>Light</i>	90
<i>Viewpoint</i>	95
<i>Interpolator</i>	98
<i>Sensor</i>	102
<i>Misc</i>	105
オリジナルモジュールの追加	111
Module クラス	112
public void initialize()	112
public void shutdown()	112
public void processData()	112
例1 Misc::GetTime	113
例2 Math::Abs	114
対応ファイル形式	115
VRML97	115
Autodesk 3DS	115
Wavefront OBJ.....	115
LightWave3D LWS.....	115
SENSE8 NFF.....	116
CyberToolbox アプレット	117
appletviewer での設定.....	117
Microsoft / Netscape ブラウザでの設定.....	118

CyberToolbox for Java3D について

CyberToolbox for Java3D¹⁾は Java2 および Java3D プラットフォーム用のバーチャルリアリティオーサリングツールです。CyberToolbox は VRML97 のシーングラフをベースにしていますが、コンテンツがより簡単に制作できるように、VRML97 とは異なる独自のイベントモデルを採用したビジュアルプログラミング言語が搭載されています。



CyberToolbox は制作したコンテンツを Web ブラウザで見るためのアプレットパッケージと一緒に配布されています。このアプレットを利用すれば、インターネット上で多くの人が Microsoft Internet Explorer や Netscape Communicator を用いて、CyberToolbox で制作されたコンテンツを見ることができます。



¹ 「CyberToolbox for Java3D」は、完全に VRML97 に準拠している「CyberToolbox for Java」とは別の製品です。

CyberToolbox は Java 開発パッケージである CyberVRML97 と一緒に開発されています。もし、このような VRML や Java3D アプリケーション開発に興味があるのであれば、いかのサイトで詳しい情報が見れます。

<http://www.cyber.koganei.tokyo.jp>.

インストール

CyberToolbox for Java3D を利用するには、最新の Java2(JDK 1.2)および Java3D パッケージのインストールが必要です。またバーチャルリアリティ(Virtual Reality)デバイスの Sensor モジュールを利用するのであれば、最新の Java Communications API のインストールも合わせて必要です。これらのパッケージは Sun の Java サイト(<http://java.sun.com>)で入手可能です。

もし Java3D and VRML Working group の VRML-Java3D パッケージをインストールしている場合には、CyberToolbox のインストールを簡単にするために、このパッケージを取り除いて下さい。これは、以下の VRML97 仕様にあるクラスが、VRML-Java3D パッケージと CyberToolbox 付属の VRML パッケージで競合しているためです。

<http://www.vrml.org/Specifications/VRML97/part1/java.html#B.9>

CyberToolbox の登録ユーザーで、CyberToolbox にモジュールを追加したい場合には”toos.jar”パッケージを、あなたの JDK または JRE ディレクトリにコピーする必要があります。このパッケージは JDK と一緒に配布されています。

CyberToolbox は ZIP 形式ファイルとして配布されています。このパッケージを展開するには JDK 付属の Jar ユーティリティや WinZip プログラムを利用して下さい。例えば Jar ユーティリティを用いて展開するには、以下のコマンドを入力して下さい。

```
jar xvf ctbj3d110.zip
```

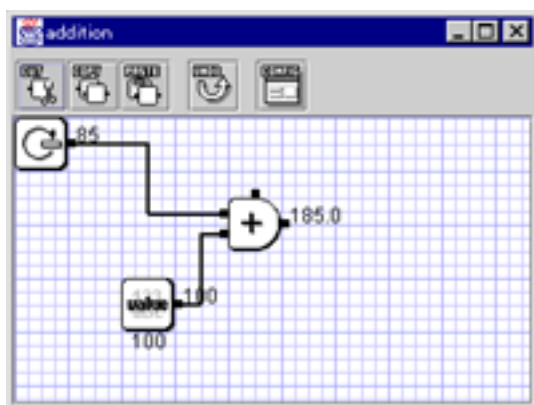
CyberToolbox を実行するには、以下のコマンドを実行して下さい。

```
cd cybertoolbox  
run (or run.bat)
```

チュートリアル

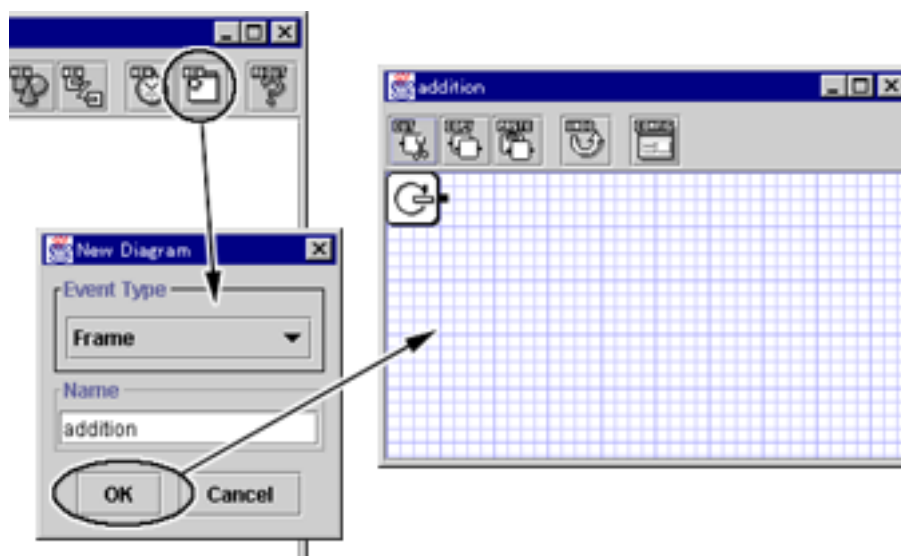
足し算

このチュートリアルでは2つの値を加算する簡単な動作の制作方法を示します。





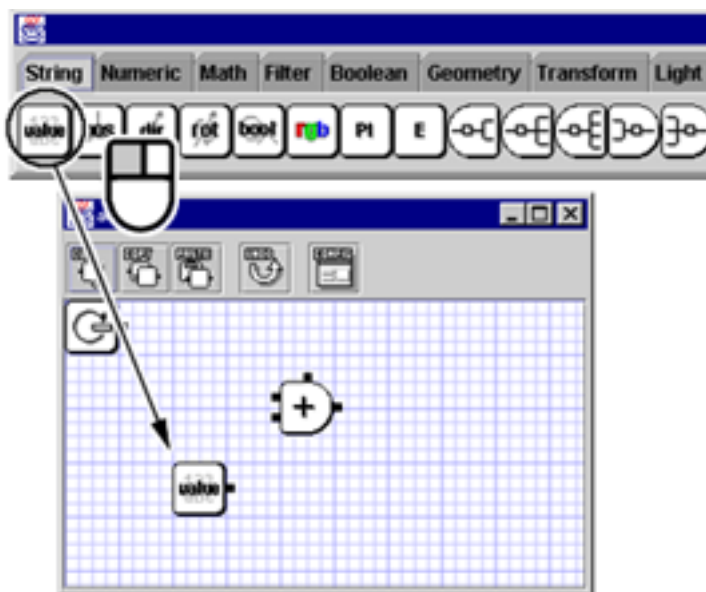
1. 加算動作を行うダイアグラムの定義

Frame イベントで実行されるダイアグラムを追加します。このイベントはシミュレーションが開始されてから、1秒間に10回発生します。このダイアグラムは現在のフレーム数を出力するシステムモジュールをもちます。ダイアグラムを追加するにはワールドウィンドウの「New Diagram」ボタンを押し、イベントタイプ(Event Type)に Frame を選択し、名前(名前)に”addition”と指定して下さい。



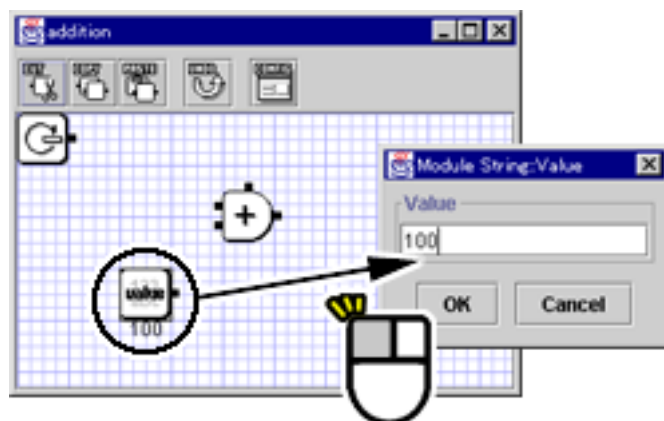
以下の2つのモジュールを、モジュールウィンドウから追加したダイアグラムに追加して下さい。

	アイコン	クラス	名前
1		String	Value
2		Numeric	Add

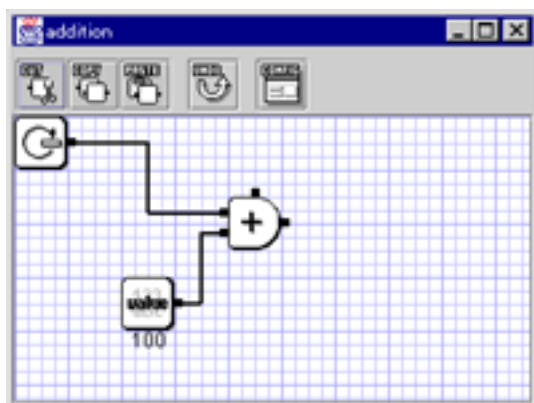


以下のモジュールをダブルクリックし、その値を 100 に設定して下さい。

	アイコン	設定値
1		100

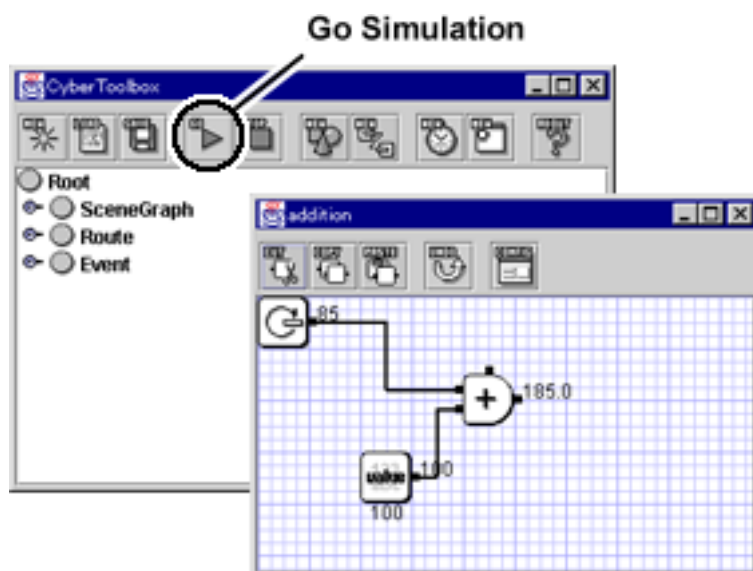


以下のようにデータフローラインを接続して下さい。



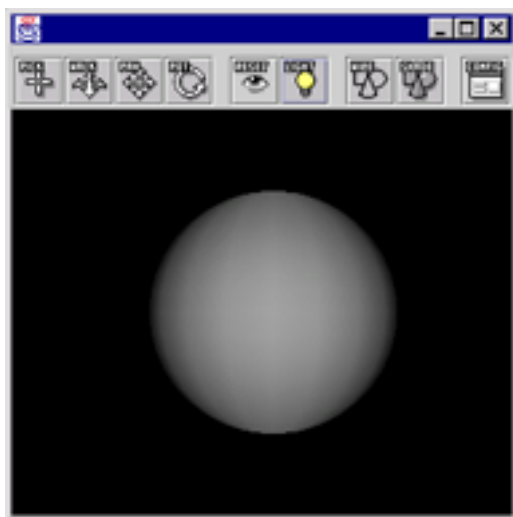
2. シミュレーションの実行

この空間動作を実行するには、ワールドウィンドウの「Go Simulation」ボタンを押して下さい。シミュレーションが開始されると、最終的に現在のフレーム数に 100 を加算した値がモジュールから出力されます。



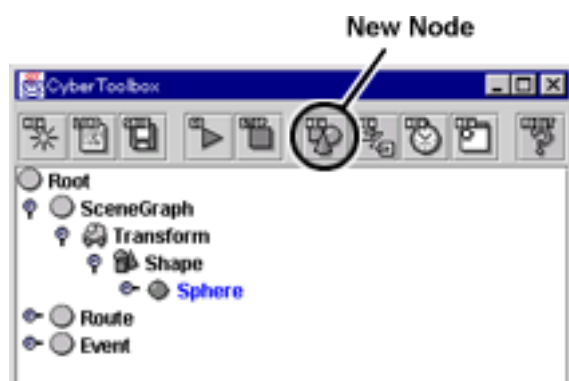
回転するボール

このチュートリアルではボールが回転する空間動作の制作方法を示します。このコンテンツを制作するには最初にボールオブジェクトを追加し、その空間動作のダイアグラムを追加する必要があります。



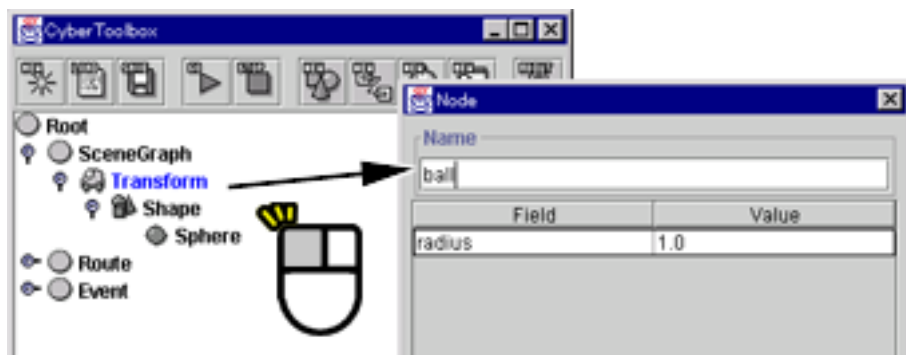
1. ボールオブジェクトの追加

ボールオブジェクトを追加するには、ワールドウインドウの「New Node」ボタンを用いて、現在のシーングラフに3つのノードを追加する必要があります。最初にシーングラフのトップノードに Transform ノードを追加し、次にその Transform ノードに Shape ノードを追加し、最後にこの Shape ノードに Sphere ノードを追加して下さい。

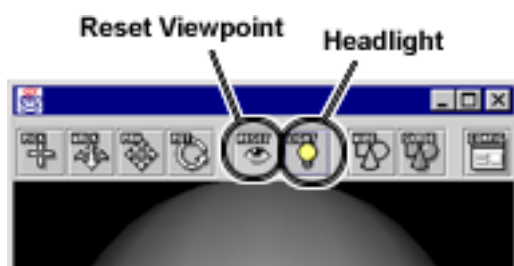


新しいノードの追加方法の詳細は「操作概要」を参考にして下さい。

ボールを回転させるには追加した Transform ノードに名前をつける必要があります。
Transform ノードをクリックし、名前を「ball」と設定して下さい。

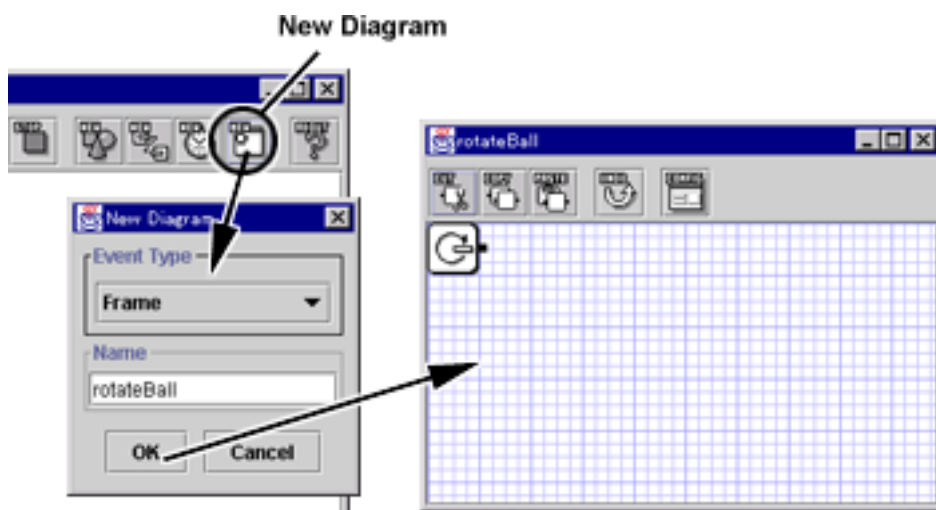


追加したボールを透視投影ウィンドウに表示させるには、「Reset Viewpoint」ボタンを押し
「XY Plane View」を選択し、「Headlight」ボタンを押してヘッドライトを ON にして下さ
い。







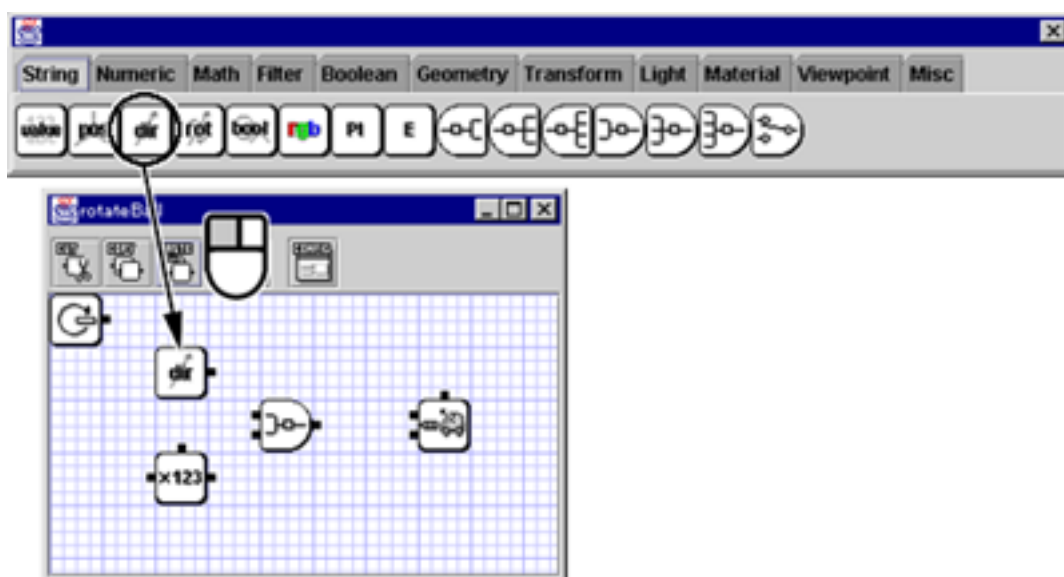
2. オブジェクトを回転させるダイアグラムの定義

オブジェクトを回転させるには、Frame イベントで実行されるダイアグラムを追加し、4
つのモジュールで動作を定義します。このダイアグラムを追加するには、「New Diagram」
ボタンを押し、イベントタイプに「Frame」、名前に「rotateBall」を指定して下さい。

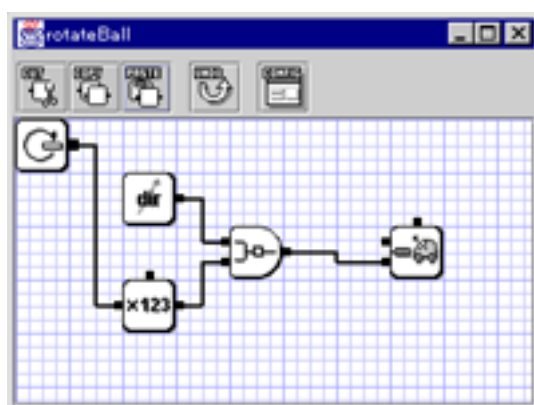


追加したダイアグラムに以下の4つのモジュールを追加します。




	アイコン	クラス	名前
1		String	Direction
2		Filter	Scale
3		String	Mearge2Values
4		Transform	SetRotation

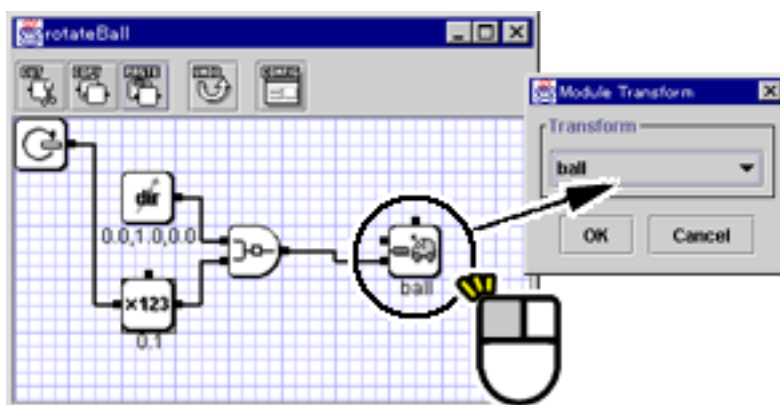


データフローラインを、以下のように接続します。



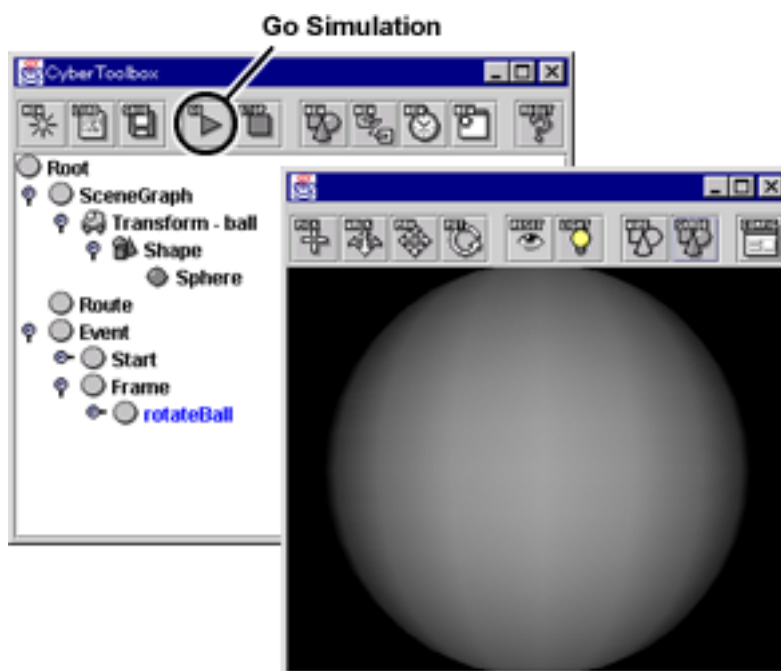
以下のモジュールをダブルクリックして値を設定します。

	アイコン	設定値
1		X=0, Y=1, Z=0 (0,1,0)
2		0.1
4		ball



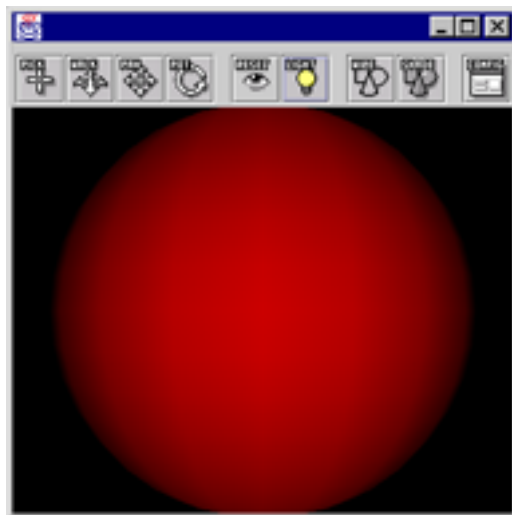
3. シミュレーションの実行

この空間動作を実行するには、ワールドウィンドウの「Go Simulation」ボタンを押して下さい。シミュレーションが開始されると、透視投影ウィンドウでボールが回転します。



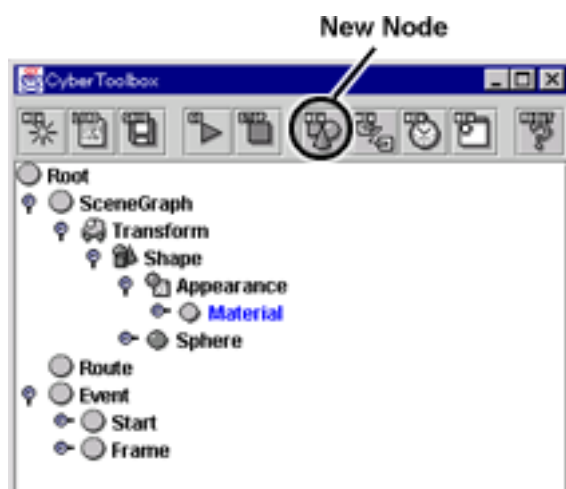
光るボール

このチュートリアルではボールがクリックされると光る空間動作の制作方法を示します。
このコンテンツを制作するには、最初にボールを追加し、次にそのピッキングイベントを追加し、ボールが光るダイアグラムを定義します。

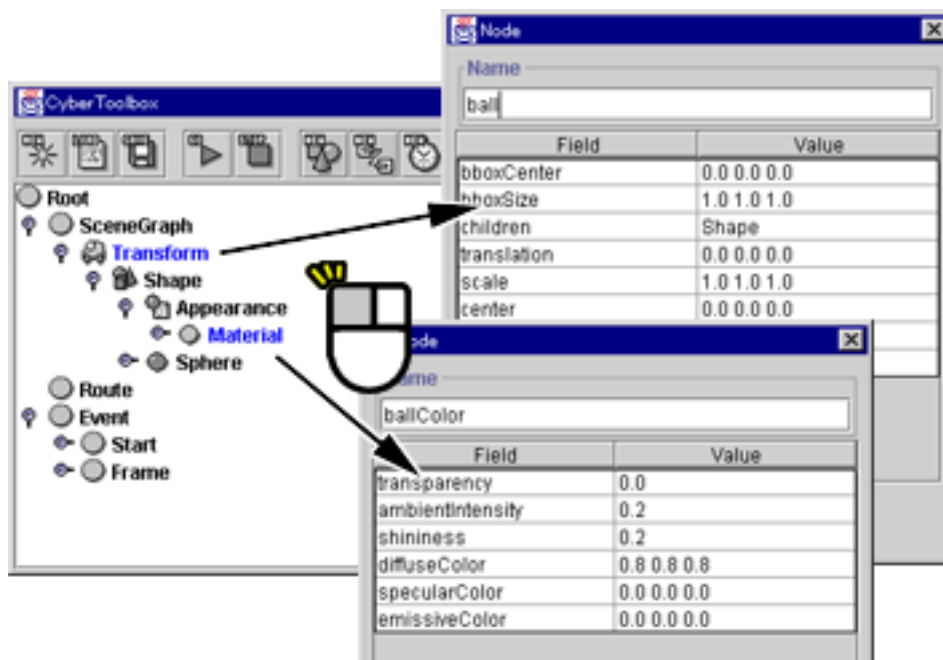


1. ボールオブジェクトの追加

ボールオブジェクトを追加するには、「New Node」ボタンで5つのノードをシーングラフに追加する必要があります。最初に Transform ノードを最上位階層に追加し、この Transform ノードに Shape ノードを追加し、この Shape ノードに Appearance ノードと Sphere ノードを追加し、この Appearance ノードに Material ノードを追加して下さい。

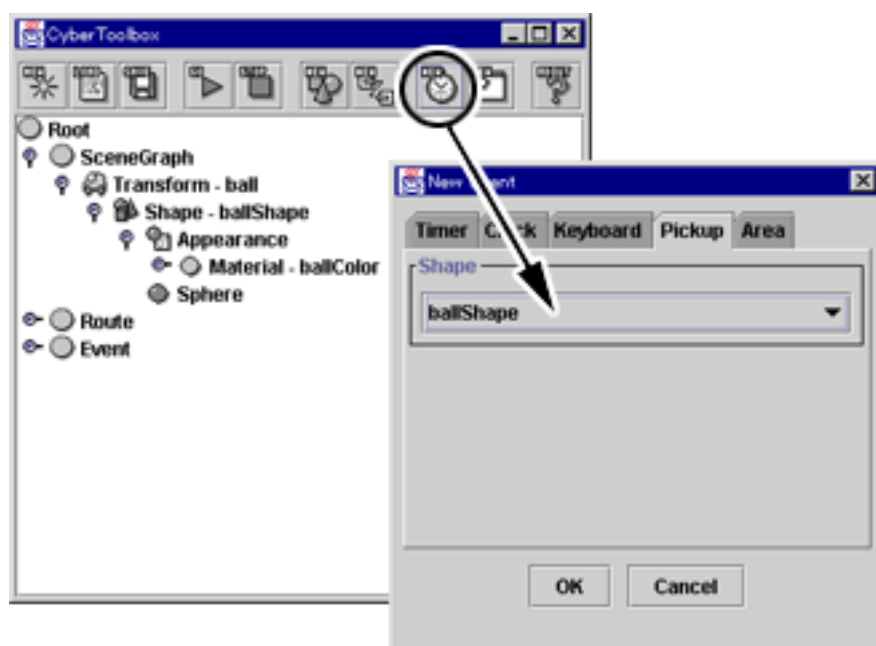


そして追加した Shape ノードは「ballShape」、Material ノードには「ballColor」と名前を設定して下さい。



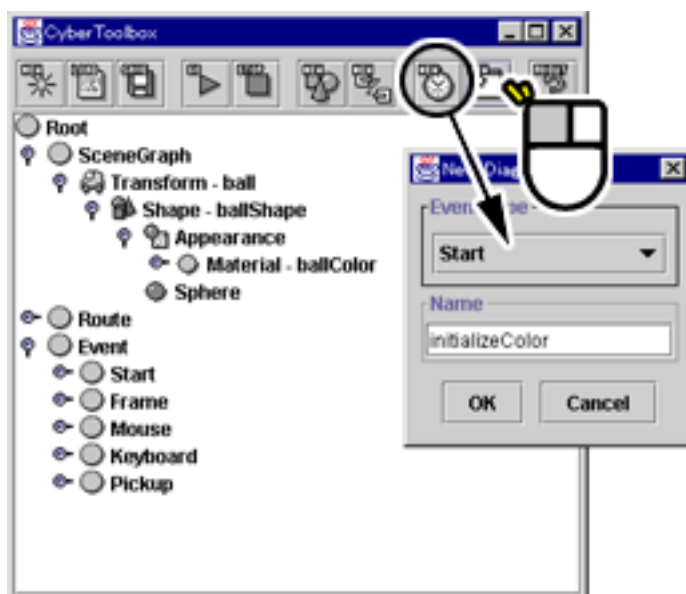
2. ピッキングイベントの追加

空間動作を定義するには、トリガとなるピッキングイベントを追加する必要があります。このイベントを追加するには、「New Event」ボタンを押し「Pickup」タブを選択し「ballShape」を選択して下さい。





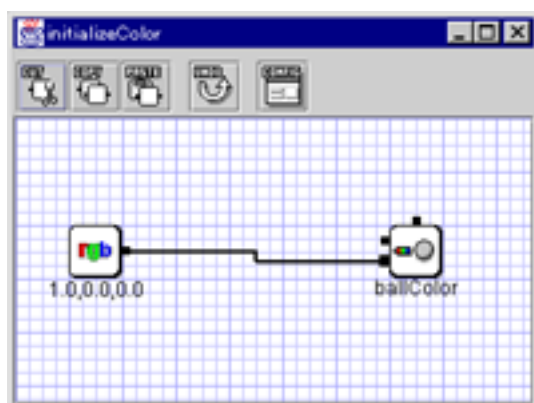
3. オブジェクト色を初期化するダイアグラムの定義

シミュレーションが開始した時にオブジェクトのマテリアル色を赤に設定するには、Start イベントで実行されるダイアログを追加する必要があります。このダイアグラムを追加するには「New Diagram」ボタンを押し、「Start」イベントを選択し、名前を「setColor」と指定して下さい。



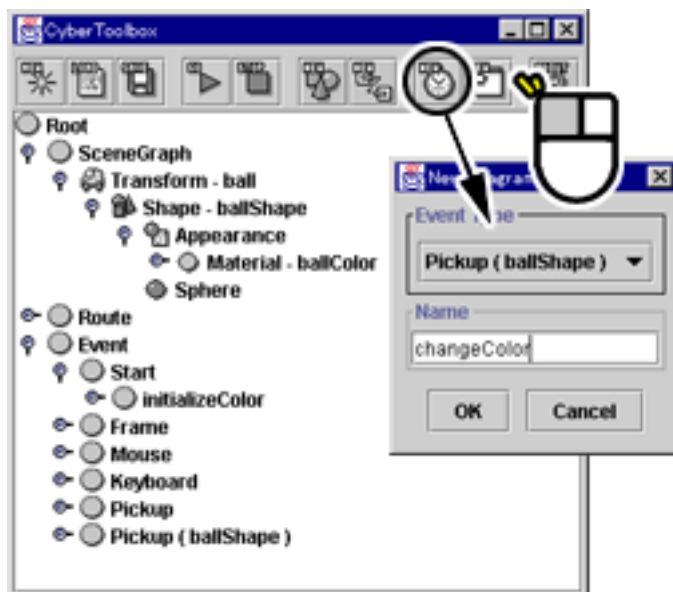
このダイアグラムに以下の2つのモジュールを追加し、値を設定し、データフローラインを接続して下さい。

	アイコン	クラス	名前	Value
1		String	Color	Red (255, 0, 0) = (1.0, 0.0, 0.0)
2		Material	SetDiffuseColor	ballColor



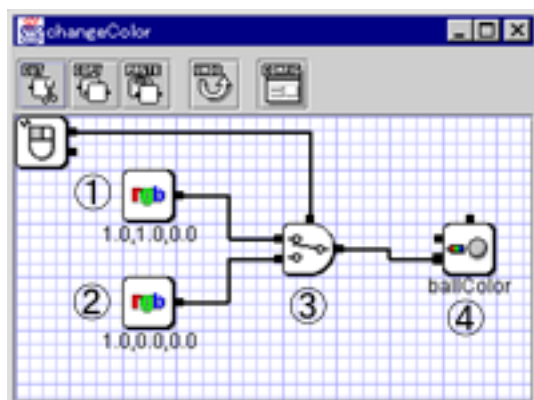
4. クリックされた時にボールが光るダイアグラムの定義

ボールがクリックされた時に黄色に、リリースされた時に赤色にマテリアルを変更するダイアグラムを定義するには、ピックアップイベントで実行されるダイアグラムを追加します。このダイアグラムを追加するには「New Diagram」ボタンを押し、「Pickup (ballShape)」イベントを選択し、名前を「changeColor」と指定して下さい。



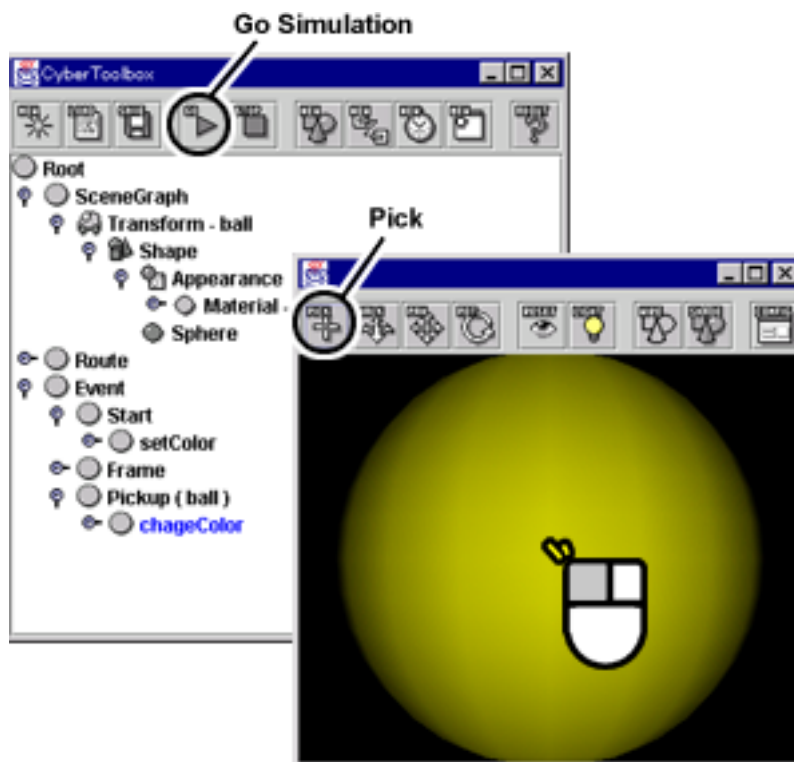
このダイアグラムに以下の4つのモジュールを追加し、値を設定し、データフローラインを接続して下さい。

	アイコン	クラス	名前	Value
1		String	Color	Yellow (255, 255, 0) = (1.0, 1.0, 0.0)
2		String	Color	Red (255, 0, 0) = (1.0, 0.0, 0.0)
3		String	Selector	
4		Material	SetDiffuseColor	ballColor



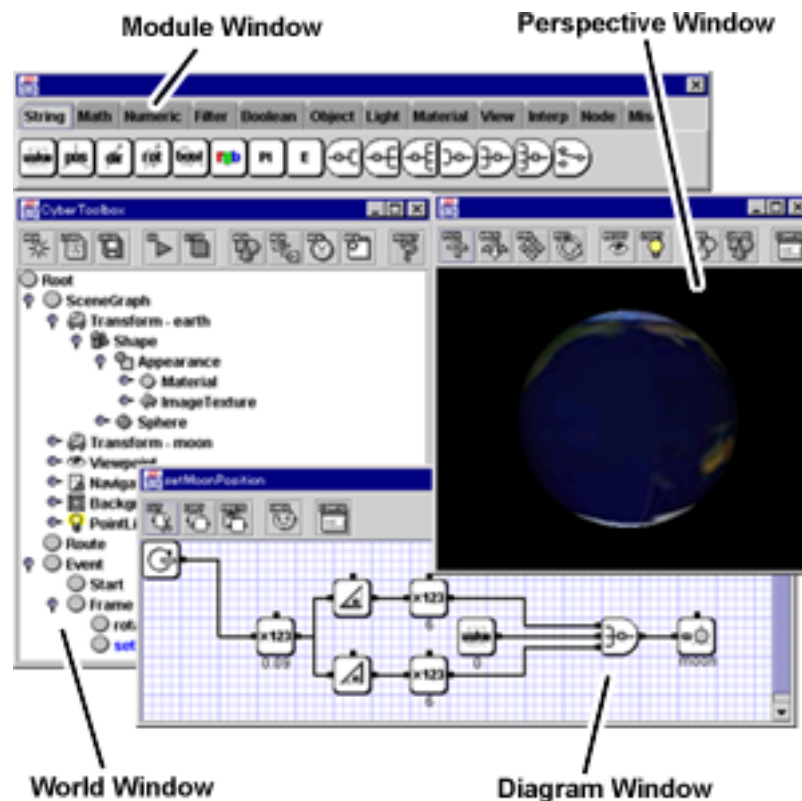
5. シミュレーションの実行

この空間動作を確認するには、ワールドウインドウの「Go Simulation」ボタンを押し、透視投影ウインドウの「Pick」ボタンを押して下さい。透視投影ウインドウでボールをクリックすると、その色が黄色に変更します。



操作概要

CyberToolbox は4つのメインウインドウ、ワールドウインドウ (World Window)、透視投影ウインドウ (Perspective window)、ダイアグラムウインドウ (Diagram window)、モジュールウインドウ (Module window)をもちます。



ワールドウインドウには、VRML ノード、ROUTE、イベント、ダイアグラム情報などの、すべての空間情報が表示されています。このウインドウで、すべての VRML や空間動作情報の編集ができます。

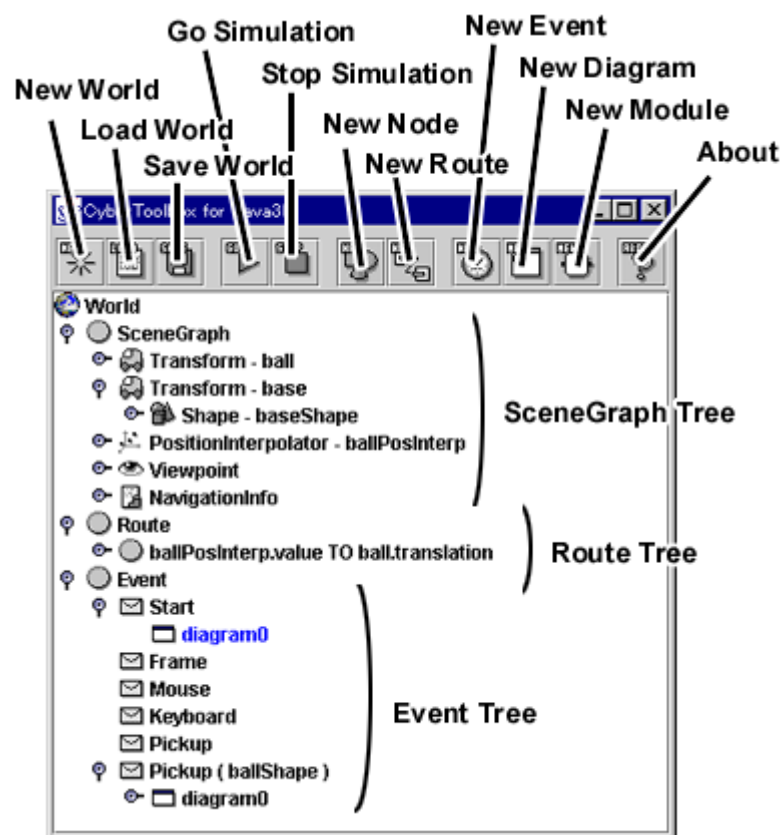
透視投影ウインドウは現在の空間のグラフィカルオブジェクトを表示します。シミュレーションが開始されていれば空間動作を含めた確認ができ、マウスで空間を移動したり、オブジェクトをピックアップしたりできます。

ダイアグラムウインドウは、モジュールウインドウのモジュールを用いて空間動作を制作する編集領域です。マウス操作で簡単に空間動作が制作できます。

ワールドウインドウ

ワールドウインドウは、現在のシーングラフおよび空間動作情報を表示し、ジオメトリファイルをロードし、そのシーングラフ情報を現在のシーングラフに追加、現在の空間情報を VRML97 ファイルに保存、新しいノード / イベント / ダイアグラムの追加、シミュレーションの開始 / 停止などができます。

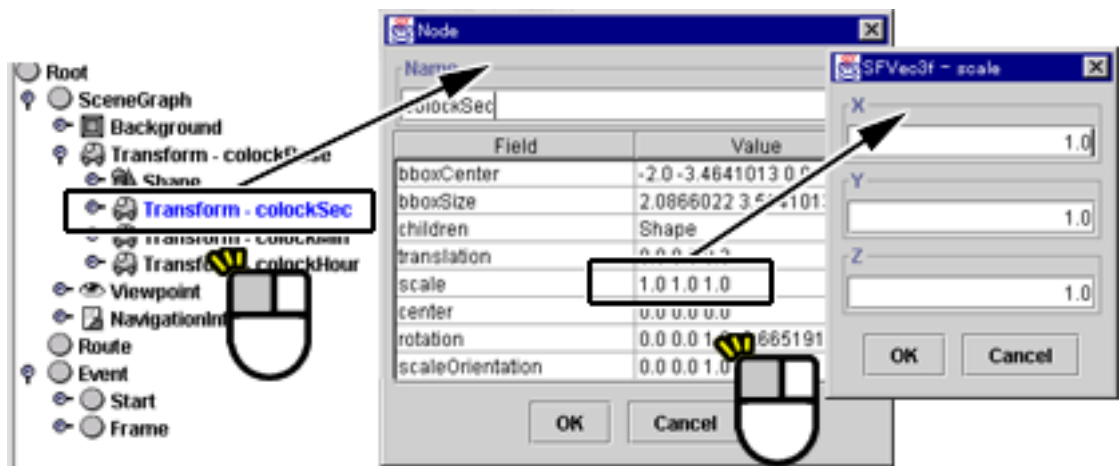
このウインドウはシーングラフ (SceneGrpah)、ルート (Route)、イベント (Event)の 3 つのツリーをもちます。



シーングラフツリーは現在の空間にある全ての VRML ノード情報、ルートツリーは Route 情報、イベントツリーはイベントとダイアグラム情報を表示します。

シーングラフツリー

シーングラフツリーは現在の空間にある VRML ノード情報を表示します。ノード情報を確認するには、そのノードアイテムをダブルクリックして設定ダイアログを開いて下さい。フィールド情報を変更するには、そのフィールドアイテムをダブルクリックして下さい。



ノードを他の親ノードの階層下に移動するには、そのノードアイテムをドラッグし新しい親ノードの上でドロップして下さい。ノードをシーングラフの最上位階層に移動したい場合には、ツリーアイテムの「SceneGraph」の上にドロップして下さい。



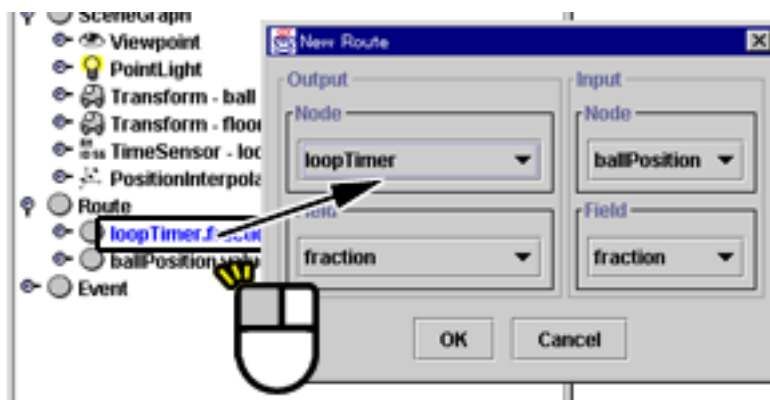
ノードを削除する場合には、そのノードアイテムをクリックして選択し、削除キーを押して下さい。表示される確認ダイアログで OK を押すと削除されます。



ツールバーの「New Node」ボタンは、新しい VRML ノードを空間に追加する場合に利用して下さい。詳細は「ツールバー」の項目を参照して下さい。

ルートツリー

ルートツリーには現在の空間にある VRML ルート情報が表示されます。ルート情報を編集するには、そのルートアイテムをダブルクリックして設定ダイアログを開いて下さい。



ルートを削除する場合には、そのルートアイテムをクリックして選択し、削除キーを押して下さい。表示される確認ダイアログで OK を押すと削除されます。



ツールバーの「New Route」ボタンは、新しいルートを空間に追加する場合に利用して下さい。詳細は「ツールバー」の項目を参照して下さい。

イベントツリー

イベントツリーには、現在の空間にあるイベントおよびダイアグラム情報が表示されます。イベントは空間動作を実行する最初のトリガーで、イベントは関連するダイアグラムを実行します。詳細は「空間動作概要」を参照して下さい。

ユーザーにより追加されたユーザーイベントのオプションを変更するには、そのイベントアイテムをダブルクリックして設定ダイアログを開いて下さい。CyberToolbox により最初から追加されているシステムイベントは編集できません。

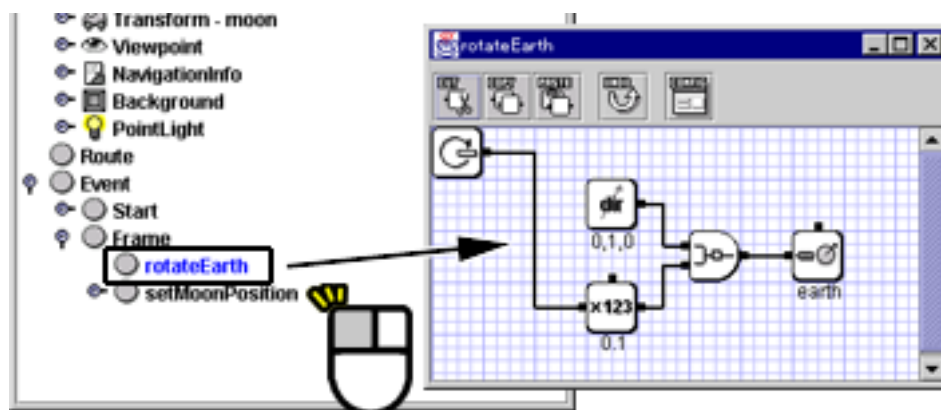


ユーザーイベントを削除するには、そのイベントアイテムをクリックして選択し、削除キーを押して下さい。表示される確認ダイアログで OK を押すと削除されます。ユーザーイベントが削除されると、その関連したダイアグラムも一緒に削除されます。



ツールバーの「New Event」ボタンは、新しいユーザーイベントを空間に追加する場合に利用して下さい。詳細は「ツールバー」の項目を参照して下さい。

ダイアグラムの編集ウィンドウを開くには、そのダイアログアイテムをダブルクリックして下さい。このウィンドウで、空間動作を編集できます。空間動作の詳細は「空間動作概要」を参照して下さい。



ダイアグラムを削除する場合には、そのダイアグラムアイテムをクリックして選択し、削除キーを押して下さい。表示される確認ダイアログで OK を押すと削除されます。.



ツールバーの「New Diagram」ボタンは、新しいダイアグラムを空間に追加する場合に利用して下さい。詳細は「ツールバー」の項目を参照して下さい。

ツールバー



New World

空間を初期化します。空間が初期化されると、すべてのノード、ルート、イベント、ダイアグラムが削除され、空間が空の状態になります。



Load World

ジオメトリファイルをロードし、現在の空間に追加します。ジオメトリファイルの形式が VRML97 で、空間動作情報をもっている場合には、その空間動作ファイルも読み込まれます。サポートされているファイル形式についての詳細は、対応ファイル形式を参照して下さい。



Save World

現在の空間情報をファイルに保存します。そのシーングラフ情報は VRML97 ファイル（拡張子：wrl）、空間動作情報は独自の空間動作ファイル（拡張子：cbf）に保存されます。



Go Simulation

シミュレーションを開始し空間動作を実行します。シミュレーション実行中は、新しいイベントやダイアグラムを追加したり、ダイアグラムを編集することはできません。これらの操作を行いたい場合には、シミュレーションを停止させて下さい。



Stop Simulation

シミュレーションを停止させます。



New Node

現在選択されているノードの子ノードとして、新しいノードを追加します。ダイアログには、選択されているノードの子ノードとして追加できるノード種類のみ表示されます。最上位の階層にノードを追加するには、シーングラフツリーの「SceneGraph」アイテムを選択して下さい。



New Route

新しいルートを追加します。既に同じルートが追加されている場合には無視されます。



New Event

新しいユーザーイベントを追加します。ユーザーイベントを追加するには、追加したいイベントタブを選択し、オプション値を設定または選択して下さい。既に同じイベントが追加されている場合には無視されます。ユーザーイベントについての詳細は「空間動作概要」を参照して下さい。



New Diagram

新しいダイアグラムを追加します。ダイアグラムを追加するには、このダイアグラムを実行するトリガーとなるイベントを選択し、名前を入力して下さい。既に同じダイアグラムが追加されている場合には無視されます。ダイアグラムが追加されると、編集ウインドウが開きます。



New Module

新しいモジュールを CyberToolbox に追加します。このユーザーモジュールについては「オリジナルモジュールの追加」を参照して下さい。

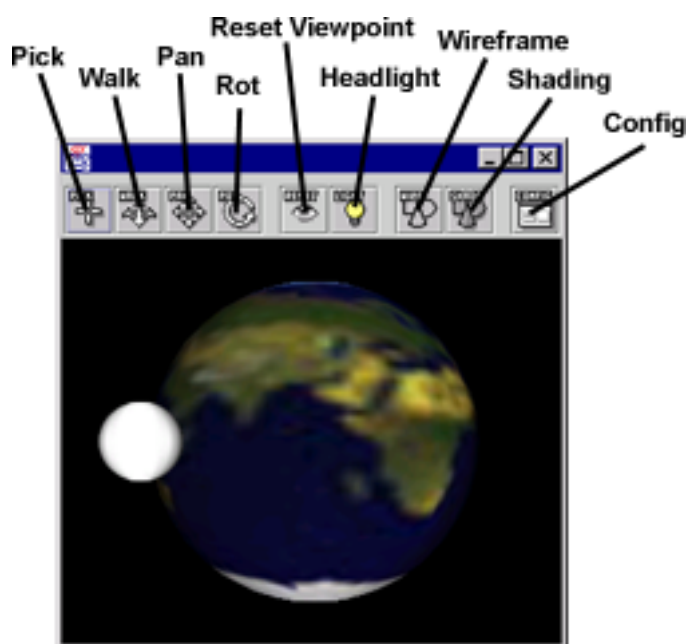


About

CyberToolbox のリリース情報を表示します。

透視投影ウィンドウ

透視投影ウィンドウは、現在の空間を Java3D を利用して表示します。シミュレーションが実行中であれば、このウィンドウは空間動作により更新されます。またマウス操作により、空間を移動したりオブジェクトをピックアップできます。



ツールバー



Pick

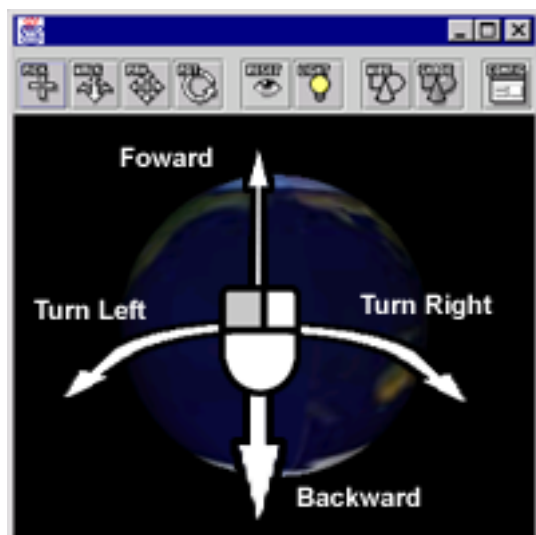
選択モードに変更します。選択モードでは、マウスでウィンドウのオブジェクトをピックアップできます。シミュレーションが実行されておりオブジェクトがピックアップされた場合にはイベントが発生します。詳細は「空間動作概要」を参照して下さい。.





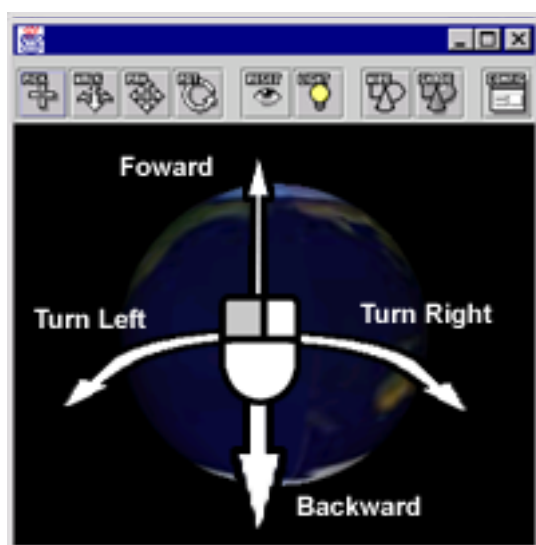
Walk

歩行モードに変更します。歩行モードでは、マウスで空間を移動できます。マウスの左ボタンを押しながら、前進したい場合にはカーソルをウインドウ上部、更新したい場合にはウインドウ下部、左に回転したい場合にはウインドウ左部、右に回転したい場合にはウインドウ右部に移動して下さい。



Pan

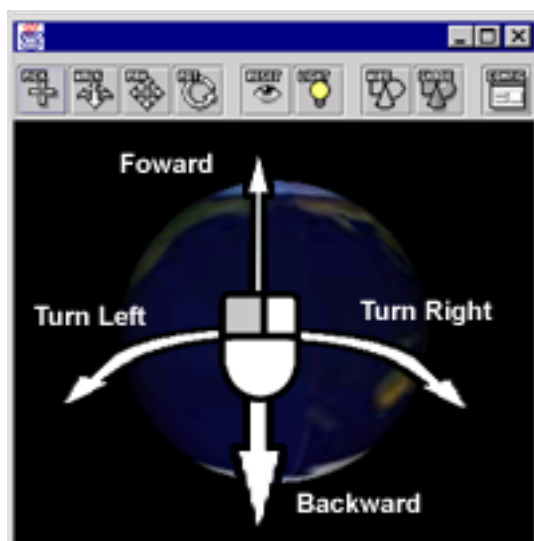
平行移動モードに変更します。平行移動モードでは、マウスで空間を上下左右に平行に移動できます。マウスの左ボタンを押しながら、上昇したい場合にはカーソルをウインドウ上部、下降したい場合にはウインドウ下部、左にスライドしたい場合にはウインドウ左部、右にスライドしたい場合にはウインドウ右部に移動して下さい。





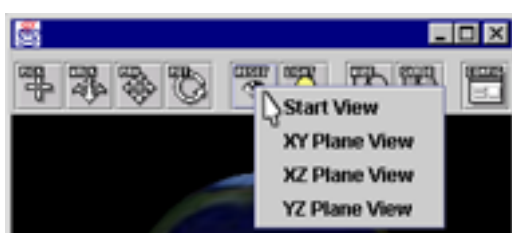
Rot

回転モードに変更します。回転モードでは、マウスで視点を前後左右に回転できます。マウスの左ボタンを押しながら、上を向きたい場合にはカーソルをウインドウ上部、下を向きたい場合にはウインドウ下部、左に傾けたい場合にはウインドウ左部、右に傾けたい場合にはウインドウ右部に移動して下さい。



Reset Viewpoint

現在の視点を、初期の座標または空間のオブジェクトがすべて視界にはいる位置に移動させます。



Headlight

ヘッドライトを ON/OFF します。空間に光源がない場合などに利用して下さい。



Wire-frame

レンダリングスタイルをワイヤーフレームにします。レンダリングスタイルがワイヤーフレームの場合には、すべてのオブジェクトは線で描画されます。



Shading

レンダリングスタイルを通常モードにします。レンダリングスタイルが通常モードの場合には、すべてのオブジェクトは色やテクスチャを利用して描画されます。



Config

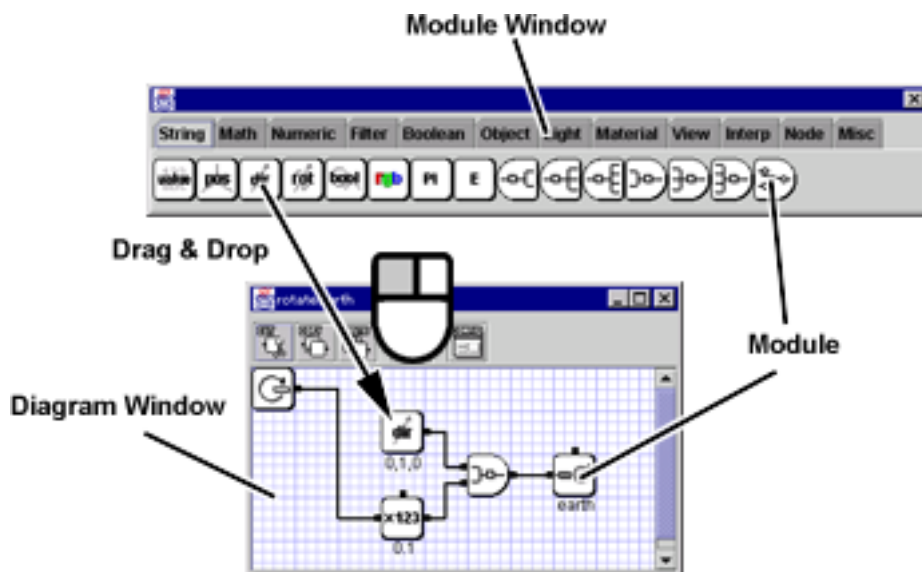
レンダリングスタイル、移動スピード、ヘッドライトのオプションを設定します。移動スピードは、マウスでの空間移動速度に影響します。



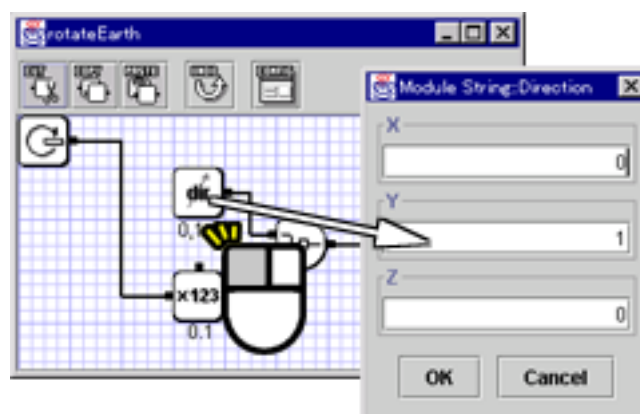
ダイアグラム / モジュールウィンドウ

ダイアグラムウィンドウは空間動作を制作する編集領域で、モジュールウィンドウからドロップしたモジュール同士を接続することにより空間動作を定義していきます。

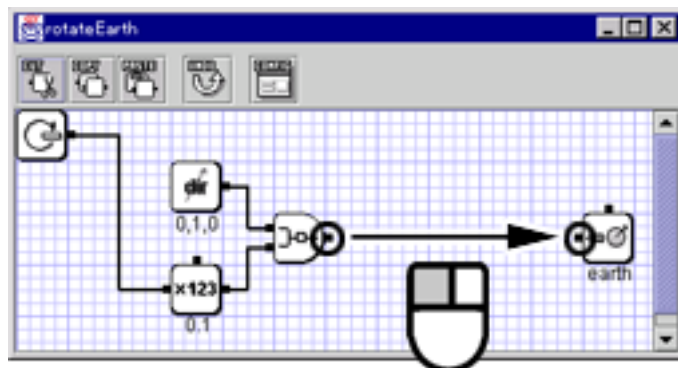
モジュールをダイアグラムに追加するには、モジュールウィンドウの追加したいモジュールをドラッグし、ダイアグラムウィンドウの上でドロップして下さい。



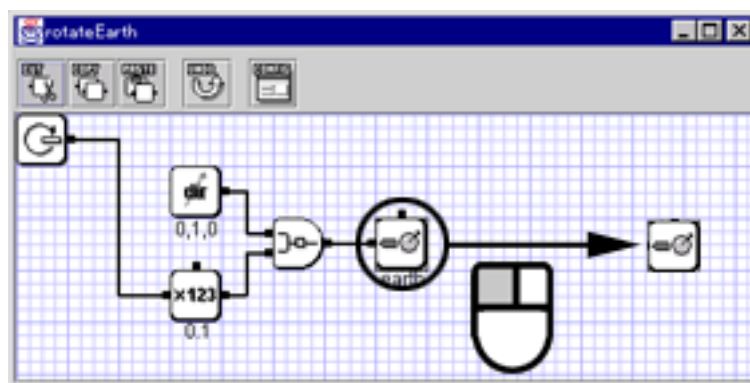
モジュールは、対象ノードや内部データを設定するための設定ダイアログをもつ場合があります。モジュールをダブルクリックすることにより、そのダイアログを開くことができます。



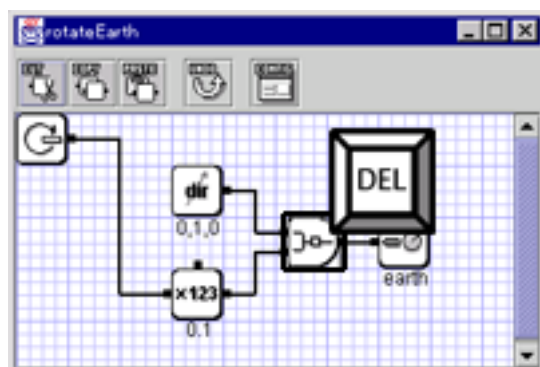
モジュール間のデータフローラインを接続するには、一方のノードを選択しドラッグしながら、もう一方のノードの上でドロップして下さい。



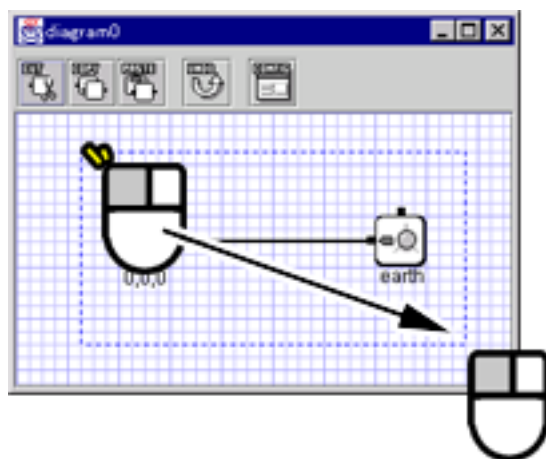
モジュール位置を移動するには、モジュールをドラッグして下さい。 .



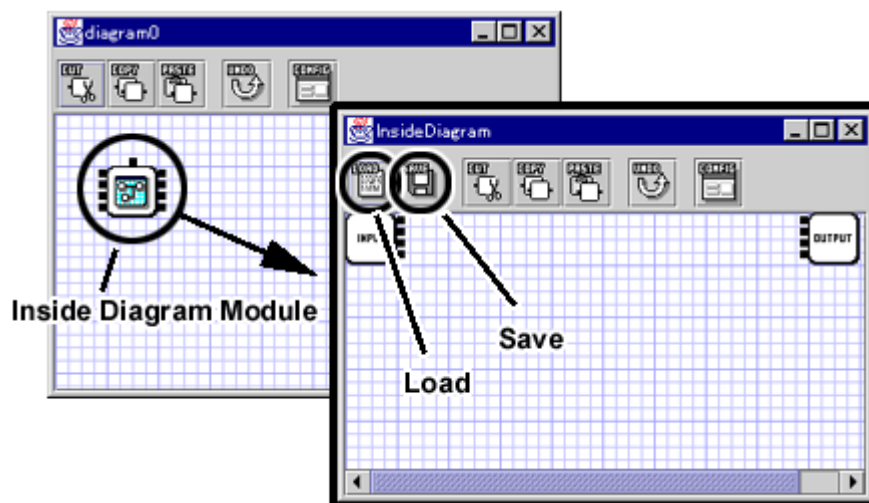
モジュールまたはデータフローラインを削除するには、そのモジュールまたは一方のノードをクリックし、削除キーを押して下さい。



選択範囲の中にあるモジュールおよびデータフローラインを、クリップボードにカットまたはコピーすることができます。範囲を選択するには、開始点をクリックし、終点までドラッグして下さい。



モジュールには内部にダイアログをもつことができるものがあります。この内部ダイアグラムを編集するには、モジュールをダブルクリックして編集ウインドウを開いて下さい。この編集ウインドウはダイアログ定義を入出力のために特別に”Load”と”Save”ボタンをもちます。”Load”ボタンはダイアグラム定義をファイルからロードするために、”Save”ボタンは現在のダイアグラム定義をファイルに保存するために用いて下さい。この標準ファイル拡張子は”*.dgm”です。



ツールバー



Load

ダイアログ定義をファイルから読み込む時にクリックして下さい。インサイドダイアグラムだけが、このボタンをもちます。



Save

現在のダイアログ定義をファイルに保存する時にクリックして下さい。インサイドダイアグラムだけが、このボタンをもちます。



Cut

選択されている範囲のモジュールおよびデータフローラインをシステムクリップボードにカットします。



Copy

選択されている範囲のモジュールおよびデータフローラインをシステムクリップボードにコピーします。



Paste

システムクリップボードにあるモジュールおよびデータフローラインを、ダイアグラムにペーストします。



Undo

最後の操作を取り消します。



Config

ダイアログ名やデータフローラインの描画方法などのダイアグラム属性を設定します。

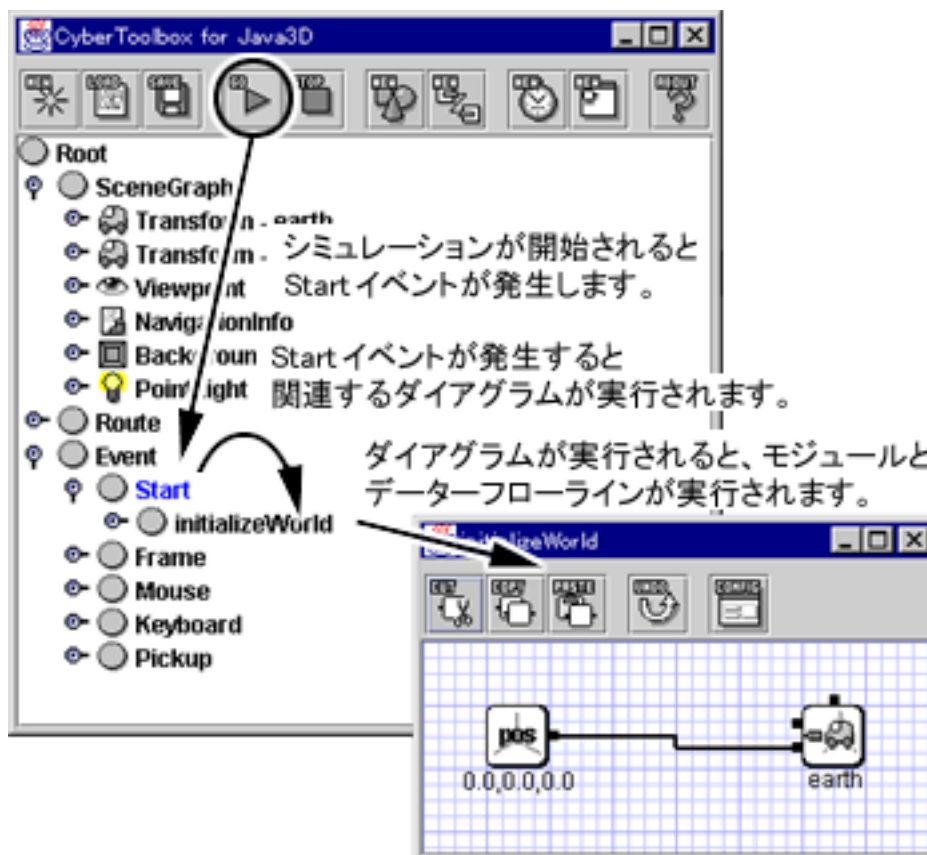
空間動作概要

CyberToolbox は空間動作を定義するためのビジュアルプログラミング言語を搭載しています。この特別な言語を利用して、面白い空間動作を簡単に定義することができます。

この空間動作は空間で発生するトリガーによって実行されます。このトリガーを CyberToolbox ではイベントと呼びます。例えば Start イベントはシミュレーションが開始された時に発生し、ピックアップイベントはオブジェクトがマウスでクリックされた時に発生します。

イベントが発生すると、関連するダイアグラムが実行されます。このダイアグラムは空間動作を定義しているモジュールとデータフローラインから構成され、このダイアグラムが実行される時に、これらのモジュールとデータフローラインも実行されます。

例えば、以下の空間で Start イベントが発生すると、「initializeWorld」ダイアグラムが実行されます。

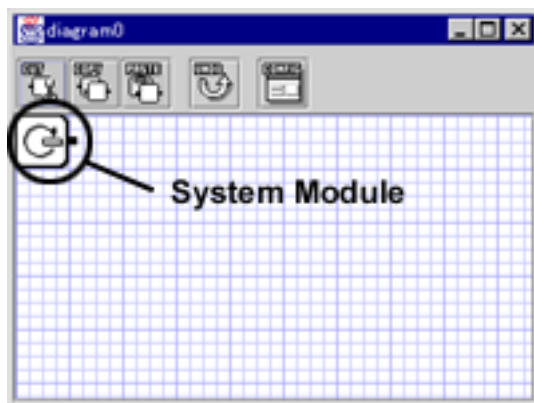


イベント

CyberToolbox には、現在 5 種類のシステムイベントと 7 種類のユーザーイベントが用意されています。システムイベントは CyberToolbox により最初から追加されていて、ユーザーイベントを利用して、ユーザーが独自のイベントを定義できます。

	名前	発生条件
システム	Start	シミュレーション開始時
	Frame	フレーム毎
	Mouse	マウス情報が変化した時
	Keyboard	キーが押された/離された時
	Pickup	Shape オブジェクトがクリックされた時
ユーザー	Timer	指定した時間が経過した時
	Clock	指定時間毎
	Keyboard	指定したキーが押された時
	Pickup	指定した Shape オブジェクトがクリックされた時
	Collision	指定した 2 つのオブジェクトが接触した時
	Area	指定したエリアに視点が入った/出た時

ダイアグラムを追加した場合に、そのダイアグラムがシステムモジュールをもつ場合があります。このモジュールは、そのダイアグラムに関連するイベント情報を出力します。このモジュールはダイアグラムから削除することはできません。.



システムイベント

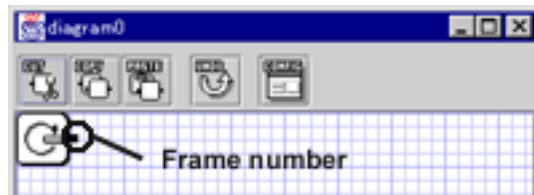
Start

このイベントは、ワールドウィンドウの「Go Simulation」ボタンを押した時に一回だけ発生します。このイベントは、何かしらの初期化目的の空間動作を定義する場合に利用して下さい。このイベントに関連するダイアグラムはシステムモジュールをもちません。 .



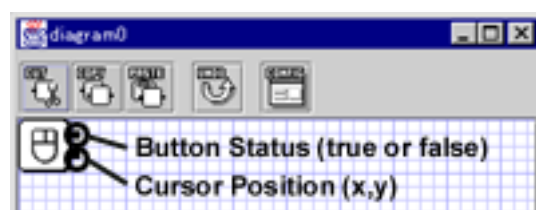
Frame

このイベントは、シミュレーションが開始してから 1 秒間に 10 回発生します。このイベントに関連するダイアグラムは、現在のフレーム数を出力するシステムモジュールをもちます。



Mouse

このイベントは、透視投影ウィンドウでマウスがクリックされた時に発生します。このイベントに関連するダイアグラムは、ボタン状態とカーソル位置を出力するシステムモジュールをもちます。マウスがドラッグされている期間は、カーソル位置情報が更新されます。 .

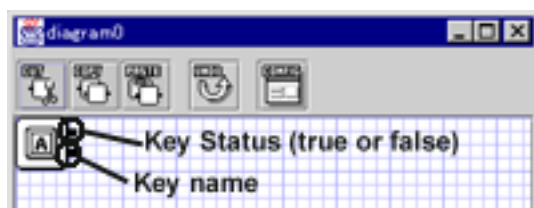


システムモジュールをクリックして、カーソル位置の出力座標フォーマットを選択できます。正規化された位置が欲しい場合には、"Normalized"を選択して下さい

い。標準の出力形式はフレーム座標系です。

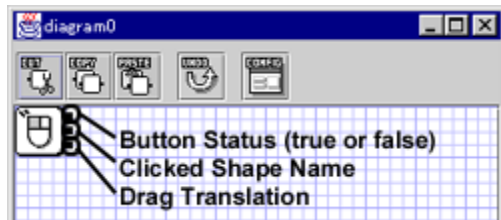
Keyboard

このイベントは、透視投影ウインドウでキーが押された、あるいは離された時に発生します。このイベントに関連するダイアグラムは、押されたキーとキー状態を出力するシステムモジュールをもちます。



Pickup

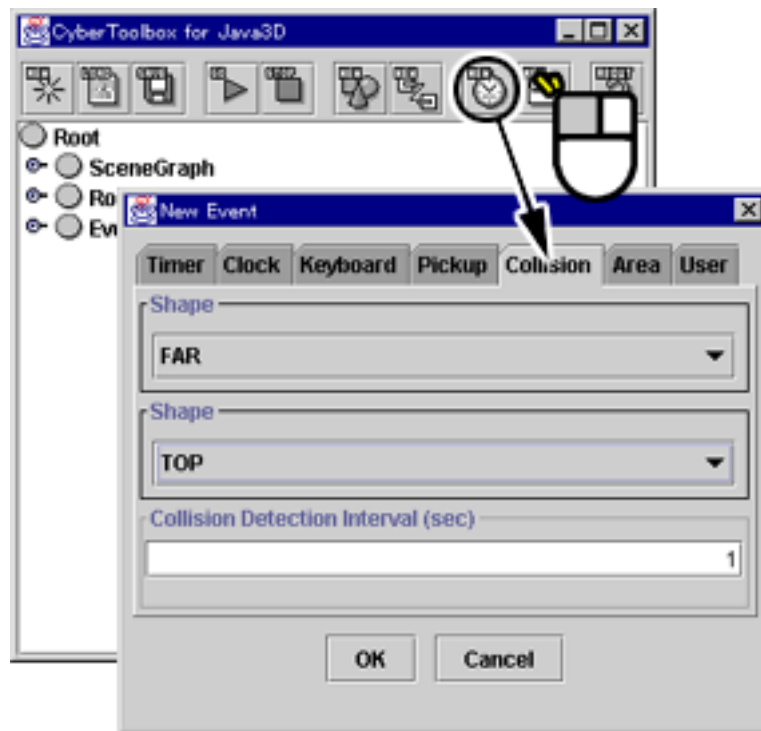
このイベントは、透視投影ウインドウでオブジェクトがマウスでクリックされた、あるいは離された時に発生します。このイベントに関連するダイアグラムは、マウスのボタン状態とクリックされたオブジェクトの Shape ノード名を出力するシステムモジュールをもちます。 .



システムモジュールをクリックして、ドラッグ変位の出力座標フォーマットを選択できます。正規化された変位が欲しい場合には、”Normalized”を選択して下さい。標準の出力形式はフレーム座標系です。

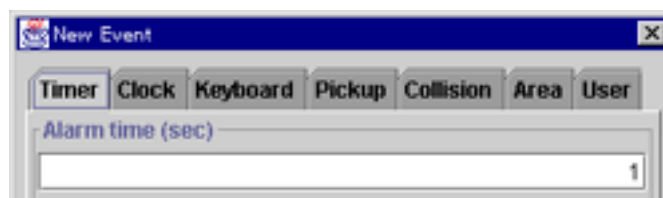
ユーザーイベント

CyberToolbox はユーザが独自のイベントを追加するための7種類のユーザーイベントが用意されています。新しいユーザーイベントを追加するには、ワールドウィンドウの「New Event」ボタンを押し、そのイベントタブを選択しオプションを入力して下さい。



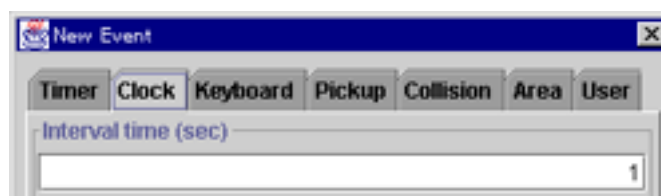
Timer

このイベントは、指定された時間が経過した時に発生します。このイベントに関連するダイアグラムにシステムモジュールはありません。

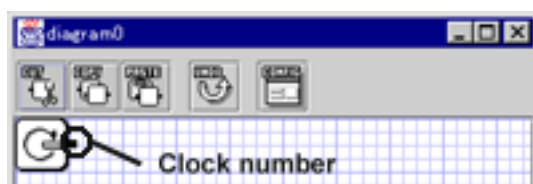


Clock

このイベントは Frame イベントのように、指定されたインターバル時間毎に発生します。このイベントを生成するには、このインターバル時間を設定して下さい。

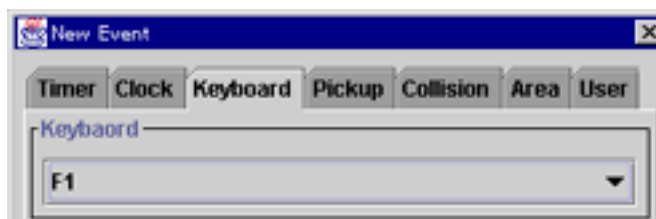


このイベントに関連するダイアグラムは、現在のフレーム数を出力するシステムモジュールをもちます。

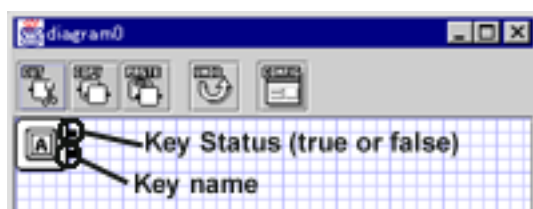


Keyboard

このイベントは、透視投影ウィンドウで指定されたキーが押された、あるいは離された場合に発生します。

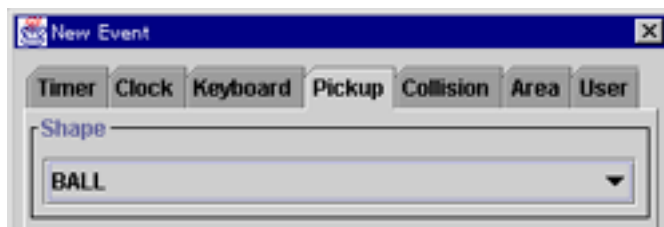


このイベントに関連したダイアグラムは、押されたキーとキー状態を出力するシステムモジュールをもちます。

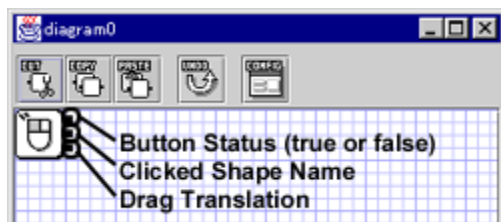


Pickup

このイベントは、透視投影ウインドウでオブジェクトがマウスでクリックされた、あるいは離された時に発生します。.



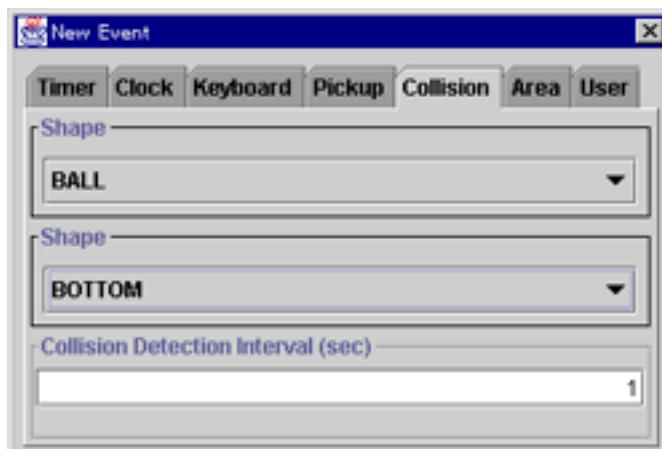
このイベントに関連するダイアグラムは、マウスのボタン状態と、クリックされたオブジェクトの Shape ノード名、ドラッグ変位を出力するシステムモジュールをもちます。.



システムモジュールをクリックして、ドラッグ変位の出力座標フォーマットを選択できます。正規化された変位が欲しい場合には、"Normalized"を選択して下さい。標準の出力形式はフレーム座標系です。

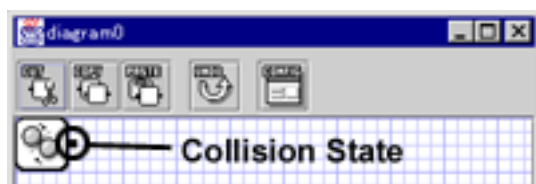
Collision

このイベントは指定された2つの Shape ノードが接触した場合に発生します。



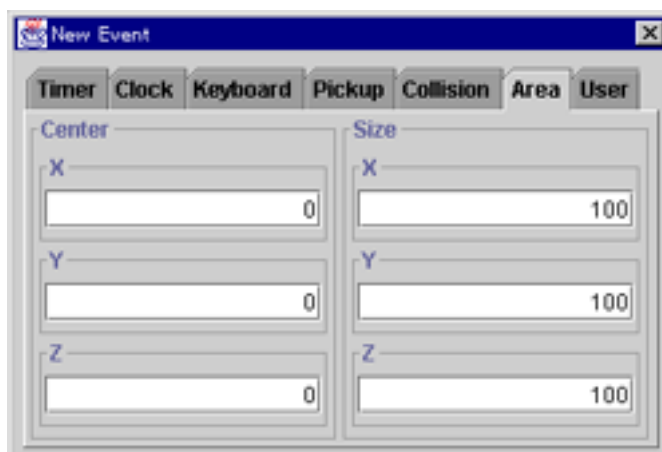
このイベントに関連するダイアグラムは、接触状態を出力するシステムモジュール

ルをもちます。このモジュールは接触した場合に”true”、離れた場合に”false”を出力します。.

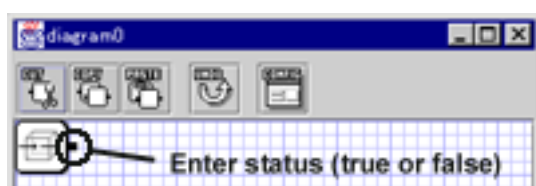


Area

このイベントは空間の指定した範囲にユーザーが入ったり出た場合に発生します。このイベントを追加するには、その中心位置とサイズを入力して下さい。

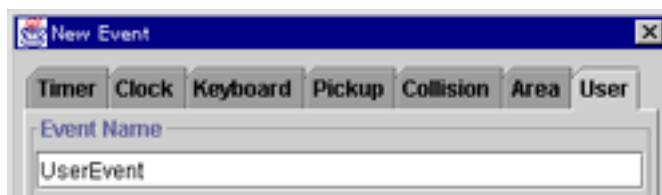


このイベントに関連したダイアグラムは、範囲に入った時に”true”、範囲を出た時に”false”を出力するシステムモジュールをもちます。

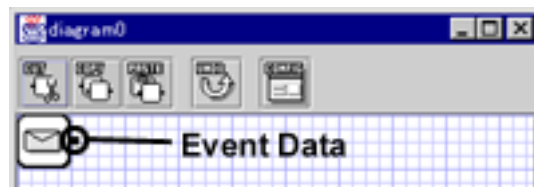


User

このイベントは、「Misc::SendMessage」モジュールから、このイベントにメッセージが送られてきた時に発生します。.

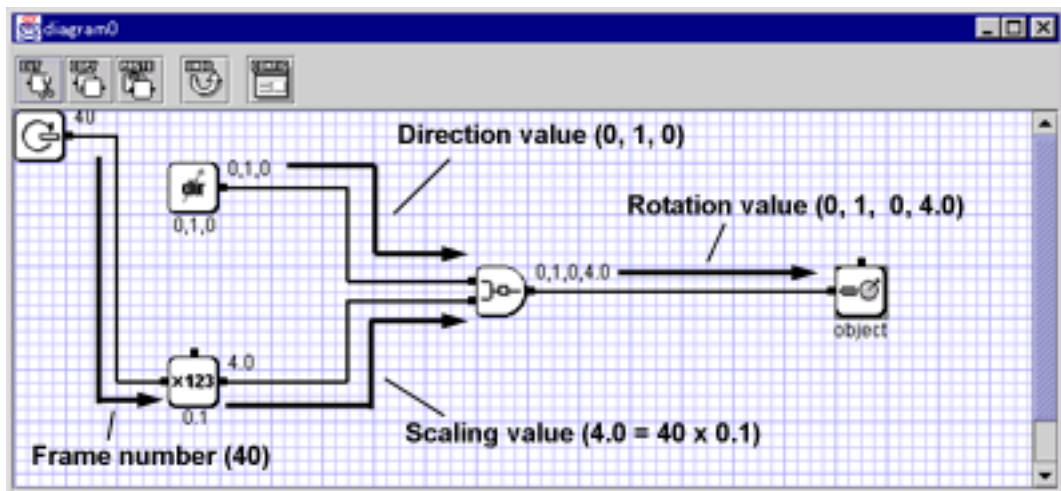


このイベントに関連したダイアグラムは、モジュールから送られてきたメッセージを出力するシステムモジュールをもちます。e.



ダイアグラム

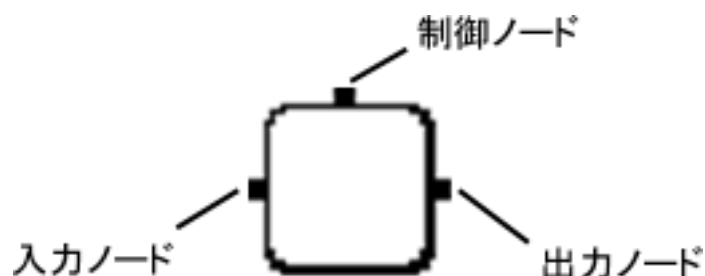
ダイアグラムウィンドウは空間動作を制作する編集領域で、モジュールウィンドウからドロップしたモジュール同士を接続することにより空間動作を定義していきます。モジュール間を接続している線はデータフローラインで、モジュールはこのデータフローラインに沿ってデータを出力ノードから他のモジュールの入力ノードに送ります。



データフローの最上位のモジュールが最初の実行され、このモジュールは出力ノードから、この出力ノードに接続されているモジュールにデータを送ります。同様に最上位モジュールからデータを受け取ったモジュールは、データフローラインに沿って他のモジュールにデータを送ります。

モジュール

モジュールは空間動作を定義する最小単位で、モジュールは3種類のノードタイプ、入力ノード、出力ノード、制御ノードをもちます。



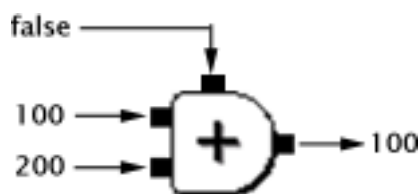
入力ノードは他のモジュールの出力ノードからデータを受け取り、出力ノードはその受け取ったデータや内部データから生成されたデータを出力します。

例えば、以下のモジュールは2つの入力ノードと1つの出力ノードをもち、その出力結果は、その2つの入力データを加算した結果です。



制御ノードを利用して、モジュールの実行を制御できます。制御ノードに他のモジュールからのデータフローラインが接続されていない場合には、このモジュールは実行されます。制御ノードが接続されていて、その入力データが”true”である場合にはモジュールが実行され、そうでなければ実行されません。

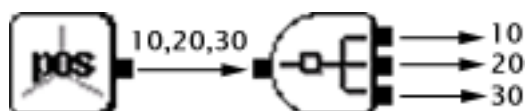
例えば、以下のモジュールは、2つの入力ノード、1つの出力ノード、1つの制御ノードをもち、その制御ノードは”false”を受け取ります。その出力ノードは、制御ノードによりモジュールが実行されないために、最初の入力ノードと同じ値を出力します。



全てのノードのデータ形式は文字列です。モジュールがノードの文字列データを数値として計算する必要がある場合には、そのモジュールは文字列データを数値に変換しその計算を開始します。

カンマ文字で複数の数字を連結することにより、1つの文字データに複数のデータを含むことができます。String クラスのモジュールを利用して、複数のデータを1つの文字列に連結したり、1つの文字列から複数のデータへ分割することができます。

例えば、以下の左側のモジュールは3つの数値を含む位置文字列を出力していて、右側のモジュールはこの文字列を3つの数字文字列に分割します。



モジュール動作

モジュールは 13 種類のクラス、String, Numeric, Math, Filter, Boolean, Geometry, Transform, Material, Light, Viewpoint, Interpolator, Sensor, Misc に分類されます。

クラス 名前	Function
String	指定された値を出力。入力文字列の連結 / 分割など.
Numeric	入力値の加算、減算など ...
Math	入力値の増減、絶対値への変換など.
Filter	入力値のスケーリング、切り捨てなど
Boolean	入力値の比較など
Geometry	入力ベクトルの正規化、回転など
Transform	Transform ノードのフィールド値の設定 / 取得
Material	Material ノードのフィールド値の設定 / 取得
Light	光源関連ノードのフィールド値の設定 / 取得
Viewpoint	Viewpoint ノードのフィールド値の設定 / 取得
Interpolator	Interpolator 関連ノードの再生など
Sensor	VR 関連デバイス情報の取得
Misc	グローバルデータの設定 / 取得など

String



Value

指定された文字列を出力します。.

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値



Position

指定された位置情報を出力します。.

入力ノード名	-
--------	---

出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値 (x, y, z)



Direction

指定された方向情報を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値 (x, y, z)



Rotation

指定された回転情報を出力します。.

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値 (x, y, z, angle)



Bool

指定された真偽情報を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値 (true or false)



Color

指定された色情報を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	OutValue = ユーザー設定値 (r, g, b)



PI

円周率を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = PI



E

自然定数を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = E



Gravity

重力定数を出力します。

入力ノード名	-
出力ノード名	OutValue
制御ノード	-

設定ダイアログ	-
対象ノード	-
結果	OutValue = 9.8



Divide2Values

入力文字列を 2 つに分割し出力します。

入力ノード名	InValue (value1,value2)
出力ノード名	OutValue1 OutValue2
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue1 = value1 OutValue2 = value2
例	InValue = 100,200 OutValue1 = 100 OutValue2 = 200



Divide3Values

入力文字列を 3 つに分割し出力します。 .

入力ノード名	InValue (value1,value2, value3)
出力ノード名	OutValue1 OutValue2 OutValue3
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue1 = value1 OutValue2 = value2 OutValue3 = value3
例	InValue = 100,200,300 OutValue1 = 100 OutValue2 = 200 OutValue3 = 300



Divide4Values

入力文字列を 4 つに分割し出力します。 .

入力ノード名	InValue (value1,value2, value3,value4)
出力ノード名	OutValue1 OutValue2 OutValue3 OutValue4
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue1 = value1 OutValue2 = value2 OutValue3 = value3 OutValue4 = value4
例	InValue = 100,200,300,400 OutValue1 = 100 OutValue2 = 200 OutValue3 = 300 OutValue4 = 400



Merge2Values

2 つの入力文字列を連結し出力します。 .

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = InValue1,InValue2
例	InValue1 = 100 InValue2 = 200 OutValue1 = 100,200



Merge3Values

3つの入力文字列を連結し出力します。.

入力ノード名	InValue1 InValue2 InValue3
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = InValue1,InValue2,InValue3
例	InValue1 = 100 InValue2 = 200 InValue3 = 300 OutValue1 = 100,200,300



Merge4Values

4つの入力文字列を連結し出力します。

入力ノード名	InValue1 InValue2 InValue3 InValue4
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = InValue1,InValue2,InValue3,InValue4
例	InValue1 = 100 InValue2 = 200 InValue3 = 300 InValue4 = 400 OutValue1 = 100,200,300,400



Selector

2つの入力文字列の一方を出力します。制御ノードにデータフローラインが接続されていない場合や、制御ノードが“true”を受け取る場合には1番目のノードの文字列を出力し、そうでなければ2番目のノードの文字列を出力します。.

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = InValue1 else { if (制御ノードデータ is “true”) OutValue = InValue1 else OutValue = InValue2 }</pre>
例	<pre>InValue1 = 100 InValue2 = 200 制御ノード = “false” OutValue = 200</pre>

Numeric

Numeric クラスのすべてのモジュールは制御ノードをもちます。制御ノードにデータフローラインが接続されていない場合や、制御ノードが”true”文字列を受け取る場合には、モジュールは計算結果を出力ノードに出力します。そうでない場合には入力ノードの値をそのまま出力します。.



2つの入力ノードの加算結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	if (制御ノード is not connected) OutValue = InValue1 + InValue2 else { if (制御ノードデータ is “true”) OutValue = InValue1 + InValue2 Else OutValue = InValue1 }
例	例 1: InValue1 = 100 InValue2 = 200 OutValue = 300 例 2: InValue1 = 100,200,300 InValue2 = 400, 500,600 OutValue = 500,700,900



Sub

入力ノード 1 から入力ノード 2 の値を引いた結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	if (制御ノード is not connected) OutValue = InValue1 - InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 - InValue2 Else OutValue = InValue1 }
例	例 1: InValue1 = 200 InValue2 = 100 OutValue = 100 例 2: InValue1 = 600,700,800 InValue2 = 400, 500,600 OutValue = 200,200,200



Mul

入力ノード 1 と入力ノード 2 の値を掛け合せた結果を出力します。 .

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) OutValue = InValue1 x InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 x InValue2 Else OutValue = InValue1 } </pre>
例	<p>例 1:</p> <p>InValue1 = 20 InValue2 = 30 OutValue = 600</p> <p>例 2:</p> <p>InValue1 = 100,200,300 (pos or vector) InValue2 = 2 OutValue = 200,400,600</p> <p>例 3:</p> <p>InValue1 = 0,0,1 (vector) InValue2 = 0,1,0,1.57 (rotation) OutValue = 1,0,0</p>



Divide

入力ノード 1 を入力ノード 2 の値で割った結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	○
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) OutValue = InValue1 / InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 / InValue2 else OutValue = InValue1 } </pre>
例	<p>例 1:</p> <p>InValue1 = 600 InValue2 = 30 OutValue = 20</p> <p>例 2:</p> <p>InValue1 = 200,400,600 (pos or vector) InValue2 = 2 OutValue = 100,200,300</p>



Mod

入力ノード 1 を入力ノード 2 の値で割った結果の、商を出力ノード 1、あまりを出力ノード 2 に出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue1 OutValue2
制御ノード	○
設定ダイアログ	-
対象ノード	-

結果セ	<pre> if (制御ノード is not connected) { OutValue1 = (InValue1 - (InValue1 % InValue2)) / InValue2 OutValue2 = InValue1 % InValue2 } else { if (制御ノードデータ is "true") { OutValue1 = (InValue1 - (InValue1 % InValue2)) / InValue2 OutValue2 = InValue1 % InValue2 } else { OutValue1 = InValue1 OutValue2 = InValue2 } } </pre>
例	InValue1 = 10 InValue2 = 3 OutValue = 1



And

2つの入力ノードの AND 演算結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = InValue1 & InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 & InValue2 else OutValue = InValue1 } </pre>

例	InValue1 = 1 InValue2 = 2 OutValue = 0
---	--



Or

2つの入力ノードの OR 演算結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = InValue1 InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 InValue2 else OutValue = InValue1 } </pre>
例	InValue1 = 1 InValue2 = 2 OutValue = 3



Xor

2つの入力ノードの XOR 演算結果を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) OutValue = InValue1 ^ InValue2 else { if (制御ノードデータ is "true") OutValue = InValue1 ^ InValue2 else OutValue = InValue1 } </pre>
例	<pre> InValue1 = 1 InValue2 = 2 OutValue = 3 </pre>

Math

Math クラスのすべてのモジュールは制御ノードをもちます。制御ノードにデータフローラインが接続されていない場合や、制御ノードが”true”文字列を受け取る場合には、モジュールは計算結果を出力ノードに出力します。そうでない場合には入力ノードの値をそのまま出力します。.



Increment

入力ノードの値に 1 を加算した結果を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = InValue + 1 else { if (制御ノードデータ is "true") OutValue = InValue + 1 else OutValue = InValue }</pre>
例	<pre>InValue = 1.1 OutValue = 2.1</pre>



Decrement

入力ノードの値に 1 を減算した結果を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) OutValue = InValue - 1 else { if (制御ノードデータ is "true") OutValue = InValue - 1 else OutValue = InValue } </pre>
例	InValue = 1 OutValue = 0



Abs

入力ノードの値の絶対値を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	○
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = InValue else { if (制御ノードデータ is "true") OutValue = InValue else OutValue = InValue } </pre>
例	InValue = -1 OutValue = 1



Negative

入力ノードの値の符号を反転した結果を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = - InValue else { if (制御ノードデータ is "true") OutValue = - InValue else OutValue = InValue }</pre>
例	<pre>InValue = 1 OutValue = -1</pre>



Pow

入力ノード1を入力ノード2の値で累乗した結果を出力します。

入力ノード名	<pre>InValue1 InValue2</pre>
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = pow(InValue1, InValue2) else { if (制御ノードデータ is "true") OutValue = pow(InValue1, InValue2) else OutValue = InValue }</pre>

例	InValue1 = 2 InValue2 = 3 OutValue = 8
---	--



Sqrt

入力ノードの値の平方を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = sqrt(InValue) else { if (制御ノードデータ is "true") OutValue = sqrt(InValue) else OutValue = InValue } </pre>
例	InValue = 9 OutValue = 3



Min

入力ノードの小さいほうの値を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) { if (InValue1 < InValue2) OutValue = InValue1 else OutValue = InValue2 OutValue = sqrt(InValue) } else { if (制御ノードデータ is "true") { if (InValue1 < InValue2) OutValue = InValue1 else OutValue = InValue2 } else OutValue = InValue } </pre>
例	InValue1 = 100 InValue2 = 200 OutValue = 100



Max

入力ノードの大きいほうの値を出力します。.

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-

結果	<pre> if (制御ノード is not connected) { if (InValue1 > InValue2) OutValue = InValue1 else OutValue = InValue2 OutValue = sqrt(InValue) } else { if (制御ノードデータ is "true") { if (InValue1 > InValue2) OutValue = InValue1 else OutValue = InValue2 } else OutValue = InValue } </pre>
例	<pre> InValue1 = 100 InValue2 = 200 OutValue = 100 </pre>



Log

入力ノードの値の対数を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = log(InValue) else { if (制御ノードデータ is "true") OutValue = log(InValue) else OutValue = InValue } </pre>



Exp

入力ノードの値の指数を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = exp(InValue) else { if (制御ノードデータ is "true") OutValue = exp(InValue) else OutValue = InValue }</pre>



Sin

入力ノードの値の正弦を出力します。

入力ノード名	RadianAngle
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = sin(RadianAngle) else { if (制御ノードデータ is "true") OutValue = sin(RadianAngle) else OutValue = RadianAngle }</pre>



Cos

入力ノードの値の余弦を出力します。

入力ノード名	RadianAngle
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = cos(RadianAngle) else { if (制御ノードデータ is "true") OutValue = cos(RadianAngle) else OutValue = RadianAngle }</pre>



Tan

入力ノードの値の正接を出力します。

入力ノード名	Radian
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = tan(RadianAngle) else { if (制御ノードデータ is "true") OutValue = tan(RadianAngle) else OutValue = RadianAngle }</pre>



ASin

入力ノードの値の逆正弦を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = asin(InValue) else { if (制御ノードデータ is "true") OutValue = asin(InValue) else OutValue = InValue }</pre>



ACos

入力ノードの値の逆余弦を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = acos(InValue) else { if (制御ノードデータ is "true") OutValue = acos(InValue) else OutValue = InValue }</pre>



ATan

入力ノードの値の逆正接を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = atan(InValue) else { if (制御ノードデータ is "true") OutValue = atan(InValue) else OutValue = InValue } </pre>



Degree2Radiun

入力値を弧度角からラジアン角に変換し出力します。

入力ノード名	DegreeAngle
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	$OutValue = DegreeAngle / 180 \times \pi$



Radiun2Degree

入力値をラジアン角から弧度角に変換し出力します。

入力ノード名	DegreeAngle
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	$OutValue = DegreeAngle / 180 \times \pi$

Filter

Filter クラスのすべてのモジュールは制御ノードをもちます。制御ノードにデータフローラインが接続されていない場合や、制御ノードが”true”文字列を受け取る場合には、モジュールは計算結果を出力ノードに出力します。そうでない場合には入力ノードの値をそのまま出力します。.



Offset

入力ノードの値に指定されたオフセット値を加算して出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = InValue + ユーザー設定値 else { if (制御ノードデータ is "true") OutValue = InValue + ユーザー設定値 Else OutValue = InValue }</pre>
例	<pre>InValue = 10 ユーザー設定値 = 1000 OutValue = 1010</pre>



Mul

入力ノードの値に指定された値を掛け合せて出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-

結果	<pre> if (制御ノード is not connected) OutValue = InValue * ユーザー設定値 else { if (制御ノードデータ is "true") OutValue = InValue * ユーザー設定値 Else OutValue = InValue } </pre>
例	<pre> InValue = 10 ユーザー設定値 = 20 OutValue = 200 </pre>



入力ノードの値に指定された値で割って出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	<pre> if (制御ノード is not connected) OutValue = InValue * ユーザー設定値 else { if (制御ノードデータ is "true") OutValue = InValue * ユーザー設定値 Else OutValue = InValue } </pre>
例	<pre> InValue = 10 ユーザー設定値 = 20 OutValue = 200 </pre>



Ceil

入力ノードの値を超える最小の整数を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = ceil(InValue) else { if (制御ノードデータ is "true") OutValue = ceil(InValue) Else OutValue = InValue }</pre>
例	<p>InValue = 12.3 OutValue = 13</p>



Floor

入力ノードの値を超えない最大の整数を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = floor(InValue) else { if (制御ノードデータ is "true") OutValue = floor(InValue) else OutValue = InValue }</pre>
例	<p>InValue = 12.3 OutValue = 12</p>



High

入力ノードの値を指定された最大値を超えない範囲で出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	<pre>if (制御ノード is not connected) { if (User specified high value < InValue) OutValue = User specified high value Else OutValue = InValue } else { if (制御ノードデータ is "true") { if (User specified hi value < InValue) OutValue = User specified high value Else OutValue = InValue } else OutValue = InValue }</pre>
例	<p>InValue = 120 User specified high value = 100 OutValue = 100</p>



Low

入力ノードの値を指定された最小値を下回らない範囲で出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	<pre> if (制御ノード is not connected) { if (InValue < User specified low value) OutValue = User specified low value else OutValue = InValue } else { if (制御ノードデータ is "true") { if (InValue M ¥¥< User specified low data) OutValue = User specified log value else OutValue = InValue } else OutValue = InValue } </pre>
例	InValue = 12.3 OutValue = 12



Ragne

入力ノードの値を指定された範囲内で出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	O
対象ノード	-

結果	<pre> if (制御ノード is not connected) { if (InValue < User specified low value) OutValue = User specified low value else OutValue = InValue if (User specified high value < InValue) OutValue = User specified high value else OutValue = InValue } else { if (制御ノードデータ is "true") { if (InValue M ¥¥< User specified low data) OutValue = User specified log value else OutValue = InValue if (User specified high value < InValue) OutValue = User specified high value else OutValue = InValue } else OutValue = InValue } </pre>
例	<pre> InValue = 12.3 OutValue = 12 </pre>

Boolean



Equal

入力ノード 1 と入力ノード 2 の値が等しい場合には”true”、そうでない場合には”false”を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 == InValue2) OutValue = “true” else OutValue = “false”



NotEqual

入力ノード 1 と入力ノード 2 の値が等しくない場合には”true”、そうでない場合には”false”を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 != InValue2) OutValue = “true” else OutValue = “false”



Greater

入力ノード 1 が入力ノード 2 の値より大きい場合には”true”、そうでない場合には”false”を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 > InValue2) OutValue = “true” else OutValue = “false”



Less

入力ノード 1 が入力ノード 2 の値より小さい場合には”true”、そうでない場合には”false”を出力します。

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 < InValue2) OutValue = “true” else OutValue = “false”



Equal Greater

入力ノード 1 が入力ノード 2 の値より大きい場合、または等しい場合には”true”、そうでない場合には”false”を出力します。 .

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 >= InValue2) OutValue = “true” else OutValue = “false”



Equal Less

入力ノード 1 が入力ノード 2 の値より小さい場合、または等しい場合には”true”、そうでない場合には”false”を出力します。 .

入力ノード名	InValue1 InValue2
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	if (InValue1 <= InValue2) OutValue = “true” else OutValue = “false”



Not

入力ノードの真偽値を反転して出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	O
設定ダイアログ	-
対象ノード	-
結果	<pre>if (制御ノード is not connected) OutValue = ! InValue else { if (制御ノードデータ is "true") { OutValue = ! InValue } else OutValue = InValue }</pre>



And

2つの入力真偽値の論理積を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$OutValue = InValue$



Or

2つの入力真偽値の論理和を出力します。

入力ノード名	InValue
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$OutValue = InValue$

Geometry



Normalize

入力されたベクトルを正規化し出力します。

入力ノード名	InValue (x, y, z)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = normalize(InValue)



Inverse

入力されたベクトルを反転し出力します。

入力ノード名	InValue (x, y, z)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = -x, -y, -z



GetLength

入力されたベクトルの長さを出力します。

入力ノード名	InValue (x, y, z)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	OutValue = sqrt(x*x + y*y + z*z)



GetDot

2つの入力ノードベクトルの内積を出力します。

入力ノード名	InValue (x1, y1, z1) InValue (x2, y2, z2)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$\text{OutValue} = (x1 \cdot x2) + (y1 \cdot y2) + (z1 \cdot z2)$



GetAngle

2つの入力ノードベクトルのなす角を出力します。

入力ノード名	InValue (x1, y1, z1) InValue (x2, y2, z2)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$\text{OutValue} = \arccos(((x1 \cdot x2) + (y1 \cdot y2) + (z1 \cdot z2)) / (\sqrt{x1^2 + y1^2 + z1^2} + \sqrt{x2^2 + y2^2 + z2^2}))$



GetVector

2つの入力ノード座標からなるベクトルを出力します。

入力ノード名	InValue (x1, y1, z1) InValue (x2, y2, z2)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$\text{OutValue} = (x2 - x1), (y2 - y1), (z2 - z1)$



GetDistance

2つの入力ノード座標の距離を出力します。

入力ノード名	InValue (x1, y1, z1) InValue (x2, y2, z2)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$OutValue = \sqrt{(x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2}$



GetCross

2つの入力ノードベクトル平面の法線を出力します。

入力ノード名	InValue (x1, y1, z1) InValue (x2, y2, z2)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$OutValue = \text{cross vector}$



Rotate

入力ノード1のベクトルを、入力ノード2の値で回転し出力します。

入力ノード名	InValue1 (x, y, z) InValue2 (x, y, z, angle)
出力ノード名	OutValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	$OutValue = \text{rotated vector}$

Transform

このクラスのモジュールは、指定された Transform ノードのフィールド値を設定または出力します。その Transform ノードを指定するために、モジュールをダブルクリックするか入力ノード 1 にノード名の文字列を入力して下さい。指定されたノード名は、モジュール下に描画されます。

モジュールに制御ノードがあり、その制御ノードに”false”が入力されている場合には、モジュールはその操作を実行しません。



GetName

指定された Transform ノード名を出力します。

入力ノード名	
出力ノード名	Node name
制御ノード	-
設定ダイアログ	O
対象ノード	Transform



SetTranslation

指定された Transform ノードの translation フィールド値を設定します。

入力ノード名	Node name Translation (x, y, z)
出力ノード名	-,
制御ノード	O
設定ダイアログ	O
対象ノード	Transform
結果	Transform :: translation = translation



SetRotation

指定された Transform ノードの rotation フィールド値を設定します。

入力ノード名	Node name Rotation (x, y, z, angle)
出力ノード名	-

制御ノード	O
設定ダイアログ	O
対象ノード	Transform
結果	Transform : : rotation = rotation



SetScale

指定された Transform ノードの scale フィールド値を設定します。

入力ノード名	Node name Scale (x, y, z)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Transform
結果	Transform : : scale= Scale



SetCenter

指定された Transform ノードの center フィールド値を設定します。

入力ノード名	Node name Center (x, y, z)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Transform
結果	Transform : : center = center



GetTranslation

指定された Transform ノードの translation フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Translation (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	Transform
結果	Translation = Transform : : translation



GetRotation

指定された Transform ノードの rotation フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Rotation (x, y, z, angle)
制御ノード	-
設定ダイアログ	O
対象ノード	Transform
結果	Rotation = Transform :: rotation



GetScale

指定された Transform ノードの scale フィールド値を出力します。 .

入力ノード名	Node name
出力ノード名	Scale (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	Transform
結果	Scale = Transform :: scale



GetCenter

指定された Transform ノードの center フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Center (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	Transform
結果	Center= Transform :: center

Material

このクラスのモジュールは、指定された Material ノードのフィールド値を設定または出力します。その Material ノードを指定するために、モジュールをダブルクリックするか入力ノード 1 にノード名の文字列を入力して下さい。指定されたノード名は、モジュール下に描画されます。

モジュールに制御ノードがあり、その制御ノードに”false”が入力されている場合には、モジュールはその操作を実行しません。



GetName

指定された Material ノード名を出力します。

入力ノード名	
出力ノード名	Node name
制御ノード	-
設定ダイアログ	O
対象ノード	Material



SetAmbientIntensity

指定された Material ノードの ambientIntensity フィールド値を設定します。

入力ノード名	Node name AmbientIntensity
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Material
結果	Material : : ambientIntensity = AmbientIntensity



SetDiffuseColor

指定された Material ノードの diffuseColor フィールド値を設定します。

入力ノード名	Node name DiffuseColor (r, g, b)
出力ノード名	-

制御ノード	○
設定ダイアログ	○
対象ノード	Material
結果	Material : : diffuseColor = DiffuseColor



SetEmissiveColor

指定された Material ノードの emissiveColor フィールド値を設定します。

入力ノード名	Node name EmissiveColor (r, g, b)
出力ノード名	-
制御ノード	○
設定ダイアログ	○
対象ノード	Material
結果	Material : : emissiveColor = EmissiveColor



SetSpecularColor

指定された Material ノードの specularColor フィールド値を設定します。

入力ノード名	Node name SpecularColor (r, g, b)
出力ノード名	-
制御ノード	○
設定ダイアログ	○
対象ノード	Material
結果	Material : : specularColor = SpecularColor



SetShininess

指定された Material ノードの shininess フィールド値を設定します。

入力ノード名	Node name Shininess
出力ノード名	-
制御ノード	○
設定ダイアログ	○
対象ノード	Material
結果	Material : : shininess = Shininess



SetTransparency

指定された Material ノードの transparency フィールド値を設定します。

入力ノード名	Node name Transparency
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Material
結果	Material :: transparency = Transparency



GetAmbientIntensity

指定された Material ノードの ambientIntensity フィールド値を出力します。

入力ノード名	Node name
出力ノード名	AmbientIntensity
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	AmbientIntensity = Material :: ambientIntensity



GetDiffuseColor

指定された Material ノードの diffuseColor フィールド値を出力します。

入力ノード名	Node name
出力ノード名	DiffuseColor (r, g, b)
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	DiffuseColor = Material :: diffuseColor



GetEmissiveColor

指定された Material ノードの emissiveColor フィールド値を出力します。

入力ノード名	Node name
出力ノード名	EmissiveColor (r, g, b)
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	EmissiveColor = Material : : emissiveColor



GetSpecularColor

指定された Material ノードの specularColor フィールド値を出力します。

入力ノード名	Node name
出力ノード名	SpecularColor (r, g, b)
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	SpecularColor = Material : : specularColor



GetShininess

指定された Material ノードの shininess フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Shininess
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	Shininess = Material : : shininess



GetTransparency

指定された Material ノードの transparency フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Transparency
制御ノード	-
設定ダイアログ	O
対象ノード	Material
結果	Transparency = Material :: transparency

Light

このクラスのモジュールは、指定された光源ノード(DirectionalLight / PointLight / SpotLight)ノードのフィールド値を設定または出力します。その光源ノードを指定するために、モジュールをダブルクリックするか入力ノード1にノード名の文字列を入力して下さい。指定されたノード名は、モジュール下に描画されます。

モジュールに制御ノードがあり、その制御ノードに”false”が入力されている場合には、モジュールはその操作を実行しません。



GetName

指定された光源ノード(DirectionalLight / PointLight / SpotLight)名を出力します。

入力ノード名	
出力ノード名	Node name
制御ノード	-
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight



SetOn

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の on フィールド値を設定します。

入力ノード名	Node name On (“true” or “false”)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	Light :: on = On



SetColor

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の color フィールド値を設定します。

入力ノード名	Node name Color (r, g, b)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	Light : : color = Color



SetIntensity

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の intensity フィールド値を設定します。

入力ノード名	Node name Intensity
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	Light : : intensity = Intensity



SetLocation

指定された光源ノード(PointLight / SpotLight)の location フィールド値を設定します。

入力ノード名	Node name Location (x, y, z)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	PointLight / SpotLight
結果	Light : : location= Location



SetDirection

指定された光源ノード(DirectionalLight / SpotLight)の direction フィールド値を設定します。

入力ノード名	Node name Node name Direction (x, y, z)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	DirectionalLight / SpotLight
結果	Light :: direction = Direction



SetRadius

指定された光源ノード(PointLight / SpotLight)の radius フィールド値を設定します。

入力ノード名	Node name Radius
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	PointLight / SpotLight
結果	Light :: radius = Radius



GetOn

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の on フィールド値を出力します。

入力ノード名	Node name
出力ノード名	On ("true" or "false")
制御ノード	-
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	On = Light :: on



GetColor

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の color フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Color (r, g, b)
制御ノード	-
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	Color = Light :: color



GetIntensity

指定された光源ノード(DirectionalLight / PointLight / SpotLight)の intensity フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Intensity
制御ノード	-
設定ダイアログ	O
対象ノード	DirectionalLight / PointLight / SpotLight
結果	Intensity = Light :: intensity



GetLocation

指定された光源ノード(PointLight / SpotLight)の location フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Location (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	PointLight / SpotLight
結果	Location = Light :: location



GetDirection

指定された光源ノード(DirectionalLight / SpotLight)の direction フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Direction (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	DirectionalLight / SpotLight
結果	Intensity = Light : : intensity



GetRadius

指定された光源ノード(PointLight / SpotLight)の radius フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Radius
制御ノード	-
設定ダイアログ	O
対象ノード	PointLight / SpotLight
結果	Radius = Light : : radius

Viewpoint

このクラスのモジュールは、指定された Material ノードのフィールド値を設定または出力します。その Material ノードを指定するために、モジュールをダブルクリックするか入力ノード 1 にノード名の文字列を入力して下さい。指定されたノード名は、モジュール下に描画されます。

モジュールに制御ノードがあり、その制御ノードに”false”が入力されている場合には、モジュールはその操作を実行しません。



GetName

指定された Viewpoint ノード名を出力します。

入力ノード名	
出力ノード名	Node name
制御ノード	-
設定ダイアログ	O
対象ノード	Viewpoint



SetPosition

指定された Viewpoint ノードの position フィールド値を設定します。

入力ノード名	Node name Position (x, y, z)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Viewpoint
結果	Viewpoint : : position = Position



SetOrientation

指定された Viewpoint ノードの orientation フィールド値を設定します。

入力ノード名	Node name Orientation (x, y, z, angle)
出力ノード名	-

制御ノード	O
設定ダイアログ	O
対象ノード	Viewpoint
結果	Viewpoint : : orientaton = Orientation



SetFOV

指定された Viewpoint ノードの fieldOfView フィールド値を設定します。

入力ノード名	Node name fov
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Viewpoint
結果	Viewpoint : : fieldOfView = fov



GetPosition

指定された Viewpoint ノードの position フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Position (x, y, z)
制御ノード	-
設定ダイアログ	O
対象ノード	Viewpoint
結果	Position = Viewpoint : : position



GetOrientation

指定された Viewpoint ノードの orientation フィールド値を出力します。

入力ノード名	Node name
出力ノード名	Orientation (x, y, z, angle)
制御ノード	-
設定ダイアログ	O
対象ノード	Viewpoint
結果	Orientation = Viewpoint : : orientaton



GetFOV

指定された Viewpoint ノードの fieldOfView フィールド値を出力します。

入力ノード名	Node name
出力ノード名	fov
制御ノード	-
設定ダイアログ	O
対象ノード	Viewpoint
結果	fov = Viewpoint : : fieldOfView

—

Interpolator

このクラスのモジュールは、Interpolator ノードをビデオデッキのように簡単に再生や停止させることができます。その Interpolator ノードを指定するために、モジュールをダブルクリックするか入力ノード 1 にノード名の文字列を入力して下さい。指定されたノード名は、モジュール下に描画されます。

Interpolator ノードの set_fraction フィールドが、このクラスのモジュールにより変更された場合には、Interpolator ノードは新たな値を value_changed フィールドに設定し、この value_changed フィールドに接続されている ROUTE を自動的に更新します。



GetName

指定された Interpolator ノード名を出力します。

入力ノード名	
出力ノード名	Node name
制御ノード	-
設定ダイアログ	<input type="radio"/>
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



Play

制御ノードが”true”を受け取った場合に、指定された Interpolator ノードの現在の fraction 位置からの再生を開始します。再生モードは設定ダイアログで指定できます。

入力ノード名	Node name
出力ノード名	-
制御ノード	<input type="radio"/>
設定ダイアログ	<input type="radio"/>
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



Stop

制御ノードが”true”を受け取った場合に、指定された Interpolator ノードが再生中であれば停止させ、fraction 位置を 0 に設定します。

入力ノード名	Node name
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



Pose

制御ノードが”true”を受け取った場合に、指定された Interpolator ノードが再生中であれば停止させます。

入力ノード名	Node name
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



IsPlaying

指定された Interpolator ノードが再生中であれば”true”、そうでなければ”false”を出力します。

入力ノード名	Node name
出力ノード名	IsPlaying
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



Rewind

制御ノードが”true”を受け取った場合に、fraction 位置を 0 に設定します。

入力ノード名	Node name
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator



Next

指定された Interpolator ノードの fraction 位置に入力ノード 2 の値を加算します。.

入力ノード名	Node name Step
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator
結果	Interpolator :: fraction += Step



Prev

指定された Interpolator ノードの fraction 位置に入力ノード 2 の値を減算します。

入力ノード名	Node name Step
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator

結果	Interpolator :: fraction -= Step
----	----------------------------------



SetFraction

入力ノード 2 の値を、指定された Interpolator ノードの fraction 位置に設定します。

入力ノード名	Node name Fraction
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator
結果	Interpolator :: fraction = Fraction



GetFraction

指定された Interpolator ノードの fraction 位置を出力します。.

入力ノード名	Node name
出力ノード名	Fraction
制御ノード	O
設定ダイアログ	O
対象ノード	CoordinateInterpolator / NormalInterpolator / OrientationInterpolator / PositionInterpolator / ScalarInterpolator
結果	Fraction = Interpolator :: fraction

Sensor

このクラスのもジュールはバーチャルリアリティ (Virtual Reality) デバイスの情報を出力します。CyberToolbox では現在以下のデバイスをサポートしています。

Logitech Magellan (Spacecontrol Mouse)
Polhemus Fastrak / Isotrak2
Intersense IS300
BG Systems BeeBox
ジョイスティック (WIN32 プラットフォームのみ)
マウス

RS-232C ポートに接続するデバイスを利用する場合には、Java Communications API のインストールが必要になります。.



Mouse

指定されたマウスの現在の情報を出力します。

入力ノード名	
出力ノード名	Button Position (x, y)
制御ノード	-
設定ダイアログ	-



Magellan

指定された Magellan の現在の情報を出力します。

入力ノード名	
出力ノード名	Button Translation (x, y, z) Radian Angle (x, y, z)
制御ノード	-
設定ダイアログ	O



Fastrak

指定された Fastrak の現在の情報を出力します。

入力ノード名	
出力ノード名	Position (x, y, z) Radian Angle (x, y, z)
制御ノード	-
設定ダイアログ	○



Isotrak2

指定された Isotrak2 の現在の情報を出力します。

入力ノード名	
出力ノード名	Position (x, y, z) Radian Angle (x, y, z)
制御ノード	-
設定ダイアログ	○



IS300

指定された IS300 の現在の情報を出力します。

入力ノード名	
出力ノード名	Position (x, y, z) Radian Angle (x, y, z)
制御ノード	-
設定ダイアログ	○



Joystick

指定されたジョイスティックの現在の情報を出力します。

入力ノード名	
出力ノード名	Button Translation (x, y)
制御ノード	-
設定ダイアログ	○



BeeBox

指定された BeeBox の現在の情報を出力します。

入力ノード名	
出力ノード名	Button Stick (x, y) Lever
制御ノード	-
設定ダイアログ	O

Misc



SetMessage

指定されたイベントに、入力ノード2のメッセージを送信します。このモジュールで、ユーザ独自のイベントにメッセージを送信できます。

入力ノード名	Event name Message
出力ノード名	-
制御ノード	O
設定ダイアログ	O



insideDiagram

このモジュールはモジュール内にダイアグラムをもつことができます。この内部ダイアグラムを編集するには、モジュールをダブルクリックして下さい。

入力ノード名	
出力ノード名	value-
制御ノード	O
設定ダイアログ	-



SetData

指定されたデータ名に、入力ノード2の値を設定します。

入力ノード名	Data name value
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	Data 名前 :: value = value



GetData

指定されたデータ名の値を出力します。

入力ノード名	Data 名前
出力ノード名	value-
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	value = Data 名前 :: value



SetArrayData

指定されたグループ名とデータ名に、入力ノード 2 の値を設定します。

入力ノード名	Group name Data name value
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	(Group 名前, Data 名前) :: value = value



GetArrayData

指定されたグループ名とデータ名の値を出力します。

入力ノード名	Group name Data name
出力ノード名	value-
制御ノード	O
設定ダイアログ	O
対象ノード	-
結果	value = (Group 名前, Data 名前) :: value



SetSwitch

指定された Switch ノードの whichChoice フィールドを設定します。.

入力ノード名	InValue (0.0 – 1.0)
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	SwitchNode
結果	SwitchNode : : whichChoice = InValue



SetSkyColor

指定された Background ノードの skyColor フィールドを設定します。.

入力ノード名	color
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Background
結果	Background : : skyColor = color



SetText

指定された Text ノードの string フィールドを設定します。

入力ノード名	text
出力ノード名	-
制御ノード	O
設定ダイアログ	O
対象ノード	Text
結果	Text : : string[0] = text



GetTime

現在の時刻を出力します。

入力ノード名	-
出力ノード名	Hour Minute Second
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	Hour = current system hour Minute = current system minute Second = current system second



Random

0.0 から 1.0 の範囲でランダムな値を出力します。

入力ノード名	-
出力ノード名	RandomValue
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	RandomValue = 0.0 – 1.0



PlaySound

制御ノードが”true”を受け取った場合に、指定された音源ファイルを再生します。

入力ノード名	-
出力ノード名	-
制御ノード	-
設定ダイアログ	O
対象ノード	-
結果	Play a specified sound



Beep

制御ノードが”true”を受け取った場合に、ビープ音を発生させます。

入力ノード名	-
出力ノード名	-
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	Play a beep sound



ShowDocument

ブラウザやアプレットに指定する Web ページを表示させます。

入力ノード名	UR Target
出力ノード名	-
制御ノード	-
設定ダイアログ	O
対象ノード	



ShowStatus

指定された文字列をステータスウインドウに表示します。

入力ノード名	String
出力ノード名	-
制御ノード	-
設定ダイアログ	-
対象ノード	-



JavaConsole

Java コンソールに文字列を出力します。

入力ノード名	String
出力ノード名	-
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	



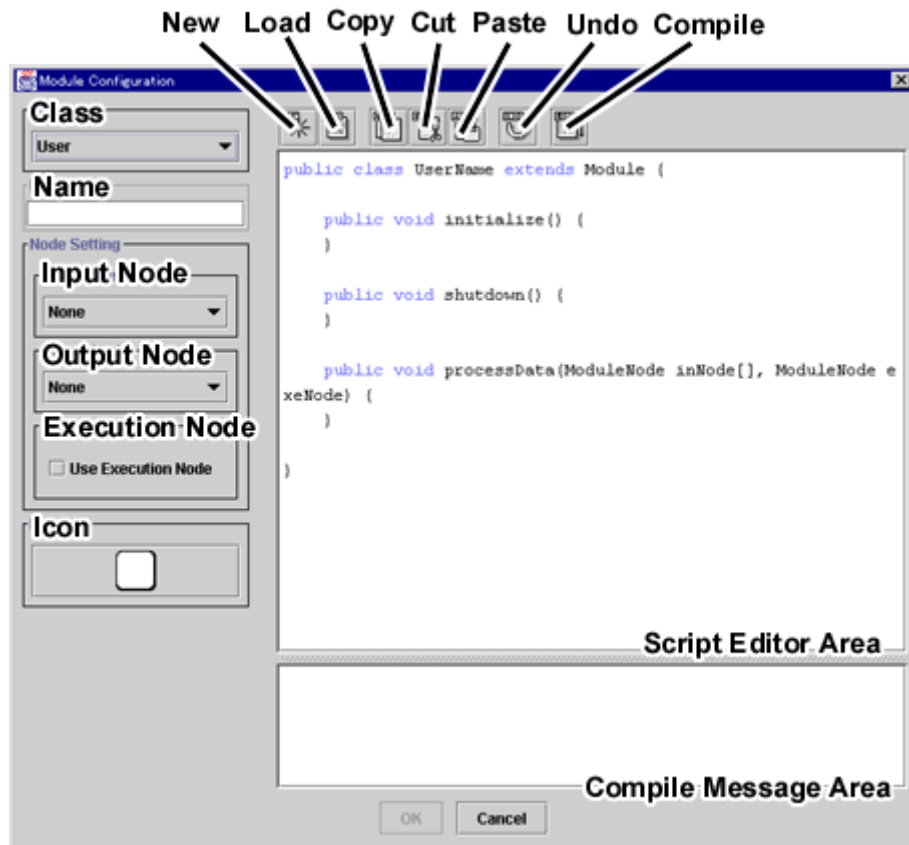
GetScreenSize

透視投影画面のサイズを出力します。

入力ノード名	-
出力ノード名	ScreenSize
制御ノード	-
設定ダイアログ	-
対象ノード	-
結果	ScreenSize = width,height

オリジナルモジュールの追加

ワールドウインドウの「New Module」ボタンで、CyberToolbox にオリジナルモジュールを追加できます。このボタンを押すと、以下のダイアログが表示されます。



新しいモジュールを追加するには、以下のダイアログフィールドを設定し、モジュールの動作コードを作成し、そのコードを正常にコンパイルする必要があります。

ダイアログフィールド	詳細
Class	モジュールクラスを選択します。
Name	モジュール名を設定します。
Input Node	入力ノード数を選択します。
Output Node	出力ノード数を選択します。
Execution Node	制御ノードをもつ場合に、チェックボックスを ON にします。
Icon	モジュールのアイコン画像を選択します。画像サイズは 32x32、画像形式は GIF である必要があります。

Module クラス

CyberToolbox の全てのモジュールは Module クラスのサブクラスです。そのため追加するモジュールのコードも、Module クラスのサブクラスである必要があります。Module クラスは抽象クラスなので、以下の3つのメソッドを定義する必要があります。

public void initialize()

このメソッドはシミュレーションが開始される前に実行されます。このメソッドには初期化コードを記述して下さい。

public void shutdown()

このメソッドはシミュレーションが開始された後に実行されます。このメソッドには終了コードを記述して下さい。

public void processData(ModuleNode inNode[], ModuleNode exeNode)

このメソッドは、このモジュールが他のモジュールからデータを受け取ったり、アクティブになった場合に実行されます。

引数 inNode は入力ノードのデータを持ち、その配列の長さは、モジュールの入力ノード数と同じです。モジュールがインプットノードをもたない場合には、この引数には null が渡されます。

引数 exeNode は制御ノードのデータを持ちます。モジュールが制御ノードをもたない場合には、この引数には null が渡されます。

ModuleNode クラスの以下のメソッドを用いて、ノード情報を取得することができます。

```
public class ModuleNode {  
    public boolean isConnected()  
    public String getStringValue()  
    public int getIntegerValue() throws NumberFormatException  
    public float getFloatValue() throws NumberFormatException  
    public double getDoubleValue() throws NumberFormatException  
    public boolean getBooleanValue()  
    public String []getStringValues()  
    public double []getDoubleValues()  
}
```

```

        public float []getFloatValues()
        public int []getIntegerValues()
    }

```

isConnected メソッドは、このノードが他のモジュールの出力ノードにデータフローラインで接続されているかを確認します。もし接続されていない場合には、このノードのデータは無視すべきです。

これらの入力ノードや制御ノードのデータから、他のモジュールへ新しいデータを出力ノード経由で送信する必要があります。このデータを出力ノードに送るには、Module クラスの以下のメソッドを利用して下さい。

```

public void sendOutNodeValue(int n, String value)
public void sendOutNodeValue(int n, int value)
public void sendOutNodeValue(int n, float value)
public void sendOutNodeValue(int n, double value)
public void sendOutNodeValue(int n, boolean value)
public void sendOutNodeValue(int n, String value[])
public void sendOutNodeValue(int n, int value[])
public void sendOutNodeValue(int n, float value[])
public void sendOutNodeValue(int n, double value[])

```

例 1 Misc::GetTime



このモジュールには入力ノードおよび制御ノードはなく、現在時間を出力する3つの出力ノードをもちます。

```

public class MiscGetTime extends Module {
    Calendar calendar = new GregorianCalendar();
    public void initialize() {
    }
    public void shutdown() {
    }
    public void processData(ModuleNode inNode[], ModuleNode exeNode){
        calendar.setTime(new Date());
        sendOutNodeValue(0, calendar.get(Calendar.HOUR_OF_DAY));
        sendOutNodeValue(1, calendar.get(Calendar.MINUTE));
    }
}

```

```

        sendOutNodeValue(2, calendar.get(Calendar.SECOND));
    }
}

```

例2 Math::Abs



このモジュールは、入力ノード、出力ノード、制御ノードを一つずつもちます。制御ノードが接続されていないか、true データを受け取っている場合には、出力ノードに入力ノードの絶対値を出力します。そうでない場合には、入力ノードのデータをそのまま出力します。

```

public class MathAbs extends Module {
    public void initialize() {
    }
    public void shutdown() {
    }
    public void processData(ModuleNode inNode[], ModuleNode exeNode){
        if (exeNode.isConnected() == true) {
            if (exeNode.getBooleanValue() == false) {
                sendOutNodeValue(0,
                    inNode[0].getStringValue());
                return;
            }
        }
        try {
            double value = inNode[0].getDoubleValue();
            sendOutNodeValue(0, Math.abs(value));
        }
        catch (NumberFormatException nfe) {
            sendOutNodeValue(0, Double.toString(Double.NaN));
        }
    }
}

```

対応ファイル形式

CyberToolbox は以下のジオメトリファイル形式の入力に対応しています。.

VRML97

CyberToolbox は指定された VRML97 ファイルのすべての情報を取得でき、そのノードをシーングラフに追加します。

Autodesk 3DS

CyberToolbox は指定された 3DS ファイルの以下の情報だけを取得し、これらの情報を VRML97 ノードに変換し、シーングラフに追加します。

Chunk ID	Description
0xA010	Material Ambient Color
0xA020	Material Diffuse Color
0xA030	Material Specular Color
0xA040	Material Shininess
0x4100	Triangle Set
0x4110	Triangle Point Set
0x4120	Triangle Face Set

Wavefront OBJ

CyberToolbox は指定された OBJ ファイルの以下の情報だけを取得し、これらの情報を VRML97 ノードに変換し、シーングラフに追加します。CyberToolbox では、マップファイルやマテリアルファイルを読み込みません。

ID	Description
v	Vertex Position
vn	Vertex Normal
f	Face Index

LightWave3D LWS

CyberToolbox は Java3D パッケージのローダークラスを用いて、LWS ファイルを読み込みます。このローダーの詳細は、Java3D のパッケージを参照して下さい。CyberToolbox は、このローダーにより読み込まれた情報を VRML97 ノードに変換し、シーングラフに追加します。

SENSE8 NFF

CyberToolbox は指定された NFF ファイルの頂点とポリゴン情報だけを取得し、これらの情報を VRML97 ノードに変換し、シーングラフに追加します。

CyberToolbox アプレット

CyberToolbox ユーザーは、CyberToolbox で制作したコンテンツをインターネットで配布できます。この目的のために CyberToolbox のアプレットパッケージが無料で配布されています。このパッケージファイル名は”ctbj3d.jar”で、CyberToolbox と同じサイトから最新のパッケージがダウンロードできます。

appletviewer での設定

JDK1.2 に付属している appletviewer でコンテンツを見る場合には、APPLET タグを利用して、CODE パラメータに “WorldApplet.class”、ARCHIVE パラメータに”ctbj3d.jar”を指定し、以下のように PARAM タグを利用してコンテンツのファイル名または URL 名を指定して下さい。

```
<HTML>
<BODY>
<CENTER>
<H1>CyberTool box Rel ease 3.0</H1>
<APPLET CODE="WorldApplet.class" ARCHIVE="ctbj3d.jar"
width=250 height=250>
  <PARAM NAME=src VALUE="world/sthenge/sthenge.wrl">
</APPLET>
</CENTER>
</BODY>
</HTML>
```

PARAM タグについては、コンテンツの VRML97 ファイルのファイル名または URL 名のみを指定して下さい。この VRML ファイルには動作定義ファイルの情報が WorldInfo ノードを利用して含まれているため、コンテンツの動作定義ファイルまで指定する必要はありません。その例を以下に示します。

```
DEF CTB_BEHAVIOR_INFO WorldInfo {
  title "CyberTool box Behavior Format V1.0"
  info [
    "sthenge.cbf"
  ]
}
```

Microsoft / Netscape ブラウザでの設定

Microsoft Internet Explore や Netscape Communicator でコンテンツを見る場合には、Java PlugIn のインストールが必要です。 .

それから、Java Plugin が「JDK1.2」または「JRE1.2」を利用するように設定し、以下のクラスファイルを CLASSPATH に追加して下さい。

j3dcore.jar, j3dutils.jar, vecmath.jar, j3daudio.jar

これらのブラウザでコンテンツ見る場合には、APPLET タグの代わりに JavaPlugin のために OBJECT タグを利用する必要があります。以下にその例を示します。

```
<HTML>
<BODY>
<CENTER>
<H1>CyberTool box for Java3D</H1>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
  WIDTH=250 HEIGHT=250
  CODEBASE="http://java.sun.com/products/plugin/1.2/
             jinstall-12-win32.cab#Version=1,2,0,0">
  <PARAM NAME=CODE VALUE="WorldApplet.class" >
  <PARAM NAME=ARCHIVE VALUE="ctbj3d.jar" >
  <PARAM NAME="type"
    VALUE="application/x-java-applet;version=1.2">
  <PARAM NAME=src VALUE="world/sthenge/sthenge.wrl">
</OBJECT>
</CENTER>
</BODY>
</HTML>
```

OBJECT タグを簡単に設定したい場合には appletviewer 用の HTML ファイルを JavaPlugin 用の HTML コンバータで変換することをお勧めします。