

Simulator Bob

3D Simulation Environment for Mobile Robots

Version: 1.4.1

USER & DEVELOPER GUIDE

by

Dipl.-Ing. Patrik Stellmann
stellmann@users.sourceforge.net

Last modified February 6, 2006

Copyright ©2005 Patrik Stellmann.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

1	Introduction	1
I	User Guide	3
2	The Idea of Components	4
2.1	Overview of ComponentBaseTypes	4
3	The Simulators GUI	6
3.1	3D-View and Camera Control	6
3.2	Toolbars	6
3.2.1	Standard	6
3.2.2	Mode*	6
3.2.3	Simulation	7
3.2.4	Record*	7
3.2.5	Log Window	7
3.2.6	Status Bar	7
3.3	Settings	7
3.3.1	Global Settings	7
3.3.2	Mouse Settings	8
3.3.3	View Quality	8

3.3.4	Scene Settings	9
3.3.5	Camera Settings	10
3.4	SimMouseHandlers	10
3.4.1	SelectionMouseHandler*	10
3.4.2	ObjectMouseHandler*	10
3.4.3	PullMouseHandler*	10
3.4.4	CameraMouseHandler	11
3.4.5	ContextMenuMouseHandler*	11
3.5	CameraComponents	12
3.5.1	StaticCamera	12
3.5.2	MountedCamera*	12
3.5.3	SpotlightCamera*	12
3.5.4	TrackingCamera*	13
4	Serialization	14
4.1	The INI File Format	14
4.2	The XML File Format	16
4.3	String Conversion of Basic Datatypes	17
5	Simulation File	20
5.1	Camera List	20
5.2	Scene Settings	20
5.3	Component List	21
6	Component Types	22
6.1	BodyComponents	22
6.1.1	RCStaticRigidBody	22
6.1.2	RCMobileRigidBody	23

6.1.3	RCRodBody*	23
6.1.4	RCConstruction	24
6.1.5	RCImportBody	24
6.1.6	RCBodyClone*	25
6.2	GlobalSimComponents	25
6.2.1	RCGround	25
6.2.2	RCSky	26
6.2.3	RCSandBox	27
6.2.4	RCRobot	27
6.2.5	RCImportRefList	28
6.3	ModelComponents	28
6.3.1	RCCuboidModel	28
6.3.2	RCPlaneModel	29
6.3.3	RCEllipsoidModel	30
6.3.4	RCCylinderModel	30
6.3.5	RCConeModel	31
6.3.6	RCEmptyModel	32
6.4	SkinComponent	32
6.4.1	RCColoredSkin	32
6.4.2	RCTexturedSkin	33
6.4.3	RCColTexSkin	33
6.5	GeomComponent	33
6.5.1	RCTransformGeom	33
6.5.2	RCCuboidGeom	34
6.5.3	RCPlaneGeom	34
6.5.4	RCSphereGeom	34
6.5.5	RCCylinderGeom	34

6.5.6	RCCappedCylinderGeom	34
6.5.7	RCModelMatchGeom*	35
6.5.8	RCModelWrapGeom*	35
6.6	JointComponent	35
6.6.1	RCBallSocketJoint	36
6.6.2	RCHingeJoint	36
6.6.3	RCSliderJoint	36
6.6.4	RCFixJoint	36
6.7	MaterialComponents	37
6.7.1	RCGenericMaterial	37
6.8	PoseComponents	37
6.8.1	RCGlobalCoordinates	37
6.8.2	RCLocalCoordinates	37
6.9	MotionComponents	38
6.9.1	RCGlobalMotion	38
6.10	InertiaComponents	38
6.10.1	RCGenericInertia	38
6.10.2	RCInertiaByGeom*	38
6.11	PeripheryComponents	39
6.11.1	RCGenericDrive	39
6.11.2	RCGenericServo	40
6.11.3	RCGenericVelControlledDrive	40
6.11.4	RCProximitySensor	41
6.11.5	RCImportJoint	41
6.11.6	RCLocalSocketCommunicator	42
6.12	ControllerComponents	42
6.12.1	RCDriveTest	42
6.12.2	RCServoTest	43
6.12.3	RCTwoWheelMotionControl	43

7 Simulation Engine	44
7.1 Physics	44
7.2 Synchronization	44
II Developer Guide	46
8 Basic Design Concepts	47
8.1 Module Hierarchy	47
8.2 Directory Structure	47
8.3 Parameter Handling	47
8.4 Extension Handling	47
9 Development Tutorials	48
9.1 Using the Class Creator	48
9.2 Implementing a new Extension DLL	49
9.3 Implementing a new ComponentType	49
9.3.1 BodyComponent	49
9.3.2 GlobalSimComponent	49
9.3.3 ModelComponent	49
9.3.4 SkinComponent	50
9.3.5 GeomComponent	50
9.3.6 JointComponent	50
9.3.7 MaterialComponent	50
9.3.8 PoseComponent	50
9.3.9 MotionComponent	50
9.3.10 InertiaComponent	50
9.3.11 ControllerComponent	50
9.3.12 PeripheryComponent	51

<i>CONTENTS</i>	vii
III Appendix	52
10 Coding Conventions	53
11 GNU Free Documentation License	54
12 Document History	64

Chapter 1

Introduction

Simulator Bob¹

This project is an open-source 3D simulation environment designed for - but not limited to - the simulation of mobile robots. It uses ODE² to simulate the rigid body dynamics and OpenGL³ for the 3D visualization. The GUI is realized by using MFC⁴ and, thus, the software is limited to Win32 platforms.

The software can simulate mechatronical systems that can be built up by composing a number of different components of which the most important are:

- **mechanics** – rigid bodies connected by joints,
- **actuators** – applying forces and torques to bodies,
- **sensors** – calculating "measurements" based on the simulation state and
- **controllers** – software module accessing with sensors, actuators and other controllers.

The application was originally written in the scope of my final thesis but now I'm using – and extending – it for my research in the field of cooperation in a system of mobile

¹homepage: <http://simbob.sourceforge.net>

²OPEN DYNAMICS ENGINE®: <http://ode.org/>

³OPENGL®: <http://www.opengl.org/>

⁴Microsoft Foundation Classes – a C++ library wrapping the MS-Windows API

autonomous robots as well as for educational projects. Between version 1.3.1 and 1.4.0a I dramatically restrucured the software resulting in several features beeing not yet available again and the old simulations are no more readably by the newer versions.

This Document

This document should provide the necessary information to decide if Simulator Bob is the right software for your requirements and show you how to use and extend it. Additionally to this document there is html documentation of the API accessable through the project homepage. I will try to update both, this document and the API documentation with every new release of the application.

There are some elements that are not available yet but will be implemented soon. I still mentioned them here to give the reader a better idea where the development will go to and marked them with a "*".

Please help me to improve this document as well as the application itself by dropping me an e-mail with your comments and not yet sufficiently answered question. I will also setup a discussion forum and/or mailing list once there is reasonable activity to be expected.

Part I

User Guide

Chapter 2

The Idea of Components

In short words: A **ComponentBaseType** is an interface class (e.g. `CameraComponent`), a **ComponentType** is a class implementing – and thus derived from – a `ComponentBaseType` (e.g. `StaticCamera`) and a **Component** is an instance of a `ComponentBaseType`. So far this is a general OOP concept. Specific for this component-framework is, that:

- Components can dynamically created at runtime (there exists a factory for each `ComponentBaseType`).
- They can have parameters (optionally also inherited by their `ComponentBaseType`) being able to be set through a serializer or through an automatically generated dialog.
- New `ComponentTypes` can be added through (extension-)DLLs.

2.1 Overview of ComponentBaseTypes

SimMouseHandler – Handles mouse actions in a 3D view.

CameraComponent – Provides parameters (esp. the cameras pose) used by a 3D view to display a simulation.

BodyComponent – A rigid body usually consisting of a visible hull (`ModelComponent`) and a physical shape (`GeomComponent`) together with other parameters.

GlobalSimComponent – Can be anything - from a simple rigid body to a file loader that registers components loaded from another file (e.g. commonly used materials and skins).

ModelComponent – Describes the visual hull of a body. Has only influence on the Simulation view – not on the physics engine!

SkinComponent – Describes the appearance of a models surface.

GeomComponent – Describes the physical shape of a body used to calculate contacts between bodies.

JointComponent – Connects two BodyComponents applying constraints to their relative motion.

MaterialComponent – Describes the physical properties of a body required by the physics engine.

PoseComponent – Provides global or local (relative to a parent object) pose information (position and orientation).

MotionComponent – Provides global or local (relative to a parent object) motion information (linear and angular velocity).

InertiaComponent – Provides information of the physical properties of a mobile rigid body (mass and inertia tensor).

PeripheryComponent – Simulates a sensor or actuator (or a combination of both).

ControllerComponent – Represents a controlling software optionally running into one or several own threads accessing Peripheries and/or other controllers. Controllers are not bound to the simulation environment but could be used on a real robot without modifications (provided that the real robot has the same framework as the simulator).

SimControllerComponent – Has the same purpose as a ControllerComponent but can access the simulation data. Thus, it is not portable to a real robot but can simplify the development of a controllers that take long to implement based on sensor data (e.g. a localizer). Later it could be easily replaced with a real ControllerComponent.

Chapter 3

The Simulators GUI

3.1 3D-View and Camera Control

You can change the cameras position and orientation by clicking with the mouse in the 3D view and move it. The effect of the mouse buttons and keys can be customized in the menu item Settings->Mouse. With the default settings you have to click with the middle mouse button and can change the cameras X- and Y- coordinate. To change the Z-coordinate hold the shift-key. By holding the ctrl-key while moving the mouse you can adapt the orientation.

3.2 Toolbars

3.2.1 Standard

Standard buttons for loading and saving a simulation. The Reload function is usefull when creating new simulation to rapidly display the current stage but comes in handy also to reset the simulation since there is no undo-function.

3.2.2 Mode*

This toolbar holds buttons to switch between different modes. In the simulation mode the physics engine is active to apply the constraints resulting from joints and collisions. In the editor mode the physics engienis turned off and you can freely move all objects.

3.2.3 Simulation

Allows control of the simulation:

- **Start** Starts the simulation.
- **Pause** Pauses the simulation.
- **Stop** Same as Pause but additionally the velocities of all mobile bodies are set to zero.

3.2.4 Record*

This toolbar holds the buttons to control the recording of a simulation to an AVI file with the camera used by the active view.

3.2.5 Log Window

All log messages generated by the application are displayed in the log window. If warning or error message is received while the log window is hidden it is shown automatically.

3.2.6 Status Bar

The status bar currently only displays a descriptive text for the selected menu entry or toolbar button.

3.3 Settings

Description of the settings dialogs available through the Settings Menu item - partially only available when a simulation is opened.

3.3.1 Global Settings

- **DocumentIni**
If this option is turned when loading a simulation it is checked for an ini file with the same name and the camera settings (later possibly the view settings as well) are loaded overwriting the information in the simulation file. When the simulation is closed the corresponding ini file is updated.

- **LogFile**

All log messages are stored in a log file as well so you have the chance to check for possible reasons on a crash. Since there are already log messages generated before the global settings are loaded and these messages are be important in case of a crash during initialization the log file is created first in the applications root directory and – after the global settings were loaded – moved to the location specified here.

- **DocumentFolder** Here you can specify the initial folder of a file dialog to open or close a simulation.
- **ExtensionFolder** Specifies the folder with the extension DLLs (plug-ins).
- **TextureFolder** Specifies the folder for image files of textures.

3.3.2 Mouse Settings

Currently there exists only a single mouse handler: `RCCameraMouseHandler`. See section 3.4.4 for the meaning of its parameters.

3.3.3 View Quality

These settings always refer to the camera being used by the active view.

- **Level**

- **Frame**

- A rectangular wire frame is drawn around the dimensions of each model.

- **Wire**

- The models are drawn as wire frame.

- **Plane**

- The models are filled in a single color depending on their skin.

- **Flat** The models are shaded but each segment is filled with a single color.

- **Smooth** The segments are smoothly shaded through interpolation.

- **Textures**

- Projection of textures on the surfaces.

- **Shadows**

- Projection of a shadow on specific planes depending on the light position.

- **Transparency**
Transparent drawing of Models with a corresponding skin. Due to a primitive drawing algorithms it can come to errors when looking through more than one transparent surface.
- **Fog** Fog blends a fog color with each rasterized pixel fragment's posttexturing color using a blending factor depth depending f . The fog parameters are adjustable in the menu item Settings-Scene.

3.3.4 Scene Settings

- **LightPos**
Position of the light used for shading the models.
- **LightColor**
color of the light used for shading the models.
- **ClearColor**
Color with which the background is filled.
- **ShadowIntensity**
Intensity of the shadow projected on shadow planes.
- **FogSettings-Mode**
The mode selects the formula used to calculate the blending factor f depending of the depths z .
 - **Linear** – $f = \frac{end-z}{end-start}$
 - **Exponential** – $f = e^{(-density \cdot z)}$
 - **Exponential2** – $f = e^{(-density \cdot z)^2}$
- **FogSettings-Density**
Value of *density* in the formula.
- **FogSettings-Start**
Value of *start* in the formula.
- **FogSettings-End**
Value of *end* in the formula.
- **FogSettings-Color**
color of the fog.

3.3.5 Camera Settings

Currently there is only one `CameraType` implemented: `RCStaticCamera`. See the meaning of its parameters below. Later you will be able to have several cameras of different types and be able to select one for each view individually.

3.4 SimMouseHandlers

All mouse handlers are stored in a list. When a mouse action was noticed the first handler in the list is notified and it can decide weather further handlers should be notified as well and weather a mouse tracking should start now. A mouse tracking is always completed (keeping the results) by releasing the mouse button. It is aborted (canceling the changes made through the tracking) by pressing another mouse button or the ESC key.

Since currently there exists only a single mouse handler it is also not possible to edit the list, yet. Later there will also be the ability for moduls to add their own mousehandler for their related document – e.g. to select a position in the simulation where a robot should go to – which will be notified before the global handler list.

3.4.1 SelectionMouseHandler*

It selects an object by clicking on it. When using the same mouse button for selecting as for moving an object you could specify in this handler weather the `ObjectMouseHandler` should only by notified when you click again on an already selected object avoiding an unintended moving while selection.

3.4.2 ObjectMouseHandler*

It changes the position and orientation of the selected object in editor mode ignoring any joints and collisions.

3.4.3 PullMouseHandler*

It applies a force to the selected object in simulation mode to allow – same as the `ObjectMouseHandler` – changing the position and orientation of an object but with consideration of constraints like joints and collisions.

3.4.4 CameraMouseHandler

It changes the position and orientation of the active camera. Note that the angle respectively the horizontal axis at right angle to the view axis is limited to ± 90 deg and changing the angle respectively the view axis can not be changed at all.

The parameter you can specify for this mouse handlers and their meaning are:

- **Button**
Mouse button that needs to be pressed for moving the camera.
- **MoveKey**
Status key that needs to be pressed additionally to change the X- and Y-coordinate respectively the orientation of the camera by moving the mouse.
- **MoveZKey**
Status key that needs to be pressed to change the Z- instead of the Y-coordinate by moving the mouse.
- **RotateKey**
Status key that needs to be pressed to change the orientation rather than the position of the camera.
- **MovementStep**
Factor to calculate the camera shift from the number of pixels the mouse has been moved (in meter per pixel).
- **RotationStep**
Factor to calculate the camera rotation from the number of pixels the mouse has been moved (in rad per pixel).

During rotation the center of the rotation is the origin of the global coordinate system. Later it might become the center of the selected object.

3.4.5 ContextMenuMouseHandler*

It opens a context menu for the selected object to make common function easily available.

3.5 CameraComponents

CameraComponent provide the visual engine with the pose of a virtual camera from which view the simulation is to be rendered. Additionally they also provide following information that – depending on the CameraType – can be constant and adjustable through a dialog or being adapted dynamically:

- **ViewAngle**
Horizontal view angle of the camera. The vertical angle results from the width-height-ratio of the window.
- **NearClip**
Distance from the camera of the near clipping plane. Everything the is infront of this plane will not be rendered.
- **FarClip**
Distance from the camera of the far clipping plane. Everything the is behind this plane will not be rendered.

Additional parameters to allow an orthogonal (in contrast to a perspicitive) view might be added once the visual engine can handle them.

3.5.1 StaticCamera

This camera holds its pose as global coordinates and, thus, is independent from any other objects. You can adapt the position and orientation freely through the CameraMouse-Handler.

3.5.2 MountedCamera*

This camera holds its pose as local coordinates relative to an BodyComponent. So when the Body moves the camera moves as well as if it was mounted on that object.

3.5.3 SpotlightCamera*

This camera holds always on a specifeable BodyComponent while keeping it's position. Optionally the view angle can be adapted automatically to display the object always in roughly the same size.

3.5.4 TrackingCamera*

Similar like the SpotlightCamera this one needs to be linked to a BodyComponent and continuously adapts its orientation to center that object as well. But furthermore it keeps a constant distant from to the object as if it was connected to it with a rod.

Chapter 4

Serialization

The underlying framework provides the ability to serialize a wide range of datatypes while basic types are handles explicitly and complex types by breaking their down to basis types. Currently the INI and the XML format is supported. Basically you could serialize any information to any format but the INI format is more suitable for more simple data like settings and the XML format is more comfortable for complex data like a simulation description.

4.1 The INI File Format

An INI file consists of a list of assignments with the format

$$\textit{variable} = \textit{value}.$$

variable represents here the key (case-insensitive) of an entry and *value* the string representation of the value. In general the string needs not to be put in quotation marks unless at least one of the following the conditions is true for the string to be assigned:

- It is empty.
- It starts and/or ends with a quotation mark.
- It stat and/or ends with a space or tabulator.

The entries in such a file are not necessarily sorted. Usually every new entry that has not been assigned before is added at the end.

To avoid conflicts with same variable keys for different variables and also to structurize the file it is possible to define sections:

[*section*].

<section> is the key (case-insensitive) for the section and all following assignments up to the next section marker are declared to be in this section. Assignment made before any section marker are part of the global section.

To avoid too complex strings, which would make the modification of an ini file very uncomfortable, more complex data types are serialized with several assignments where the key of the complex data type is set in front of the member key separated by a “.”. For instance a variable with the key `TestStruct` and the members `NumMember` and `StringMember` could be serialized to:

```
TestStruct.NumMember = 1
TestStruct.StringMember = whatever
```

To be able to handle lists with a variable length the actual number of elements needs to be stored explicitly. For instance a list with the key `StructList` consisting of the datatype from the previous example could be serialized to:

```
StructCount = 2
Struct0.NumMember = 1
Struct0.StringMember = whatever
Struct1.NumMember = 2
Struct1.StringMember = anything
```

For components additionally to their specific parameters always their type and key needs to be specified. This is done like the assignment of any other parameter. For instance for a `ModelComponent`:

```
Model.Type = RCCuboidModel
Model.Key = BoxModel
[...]
```

4.2 The XML File Format

To generate and edit the XML files used by this application you don't need to have a great understanding of this format. Thus, I will only give a short description of what you need to know about it for this purpose.

An XML file describes a tree while each element starts with a tag of the format `<key>` and ends with a tag – having the same key – `</key>`. Between these two tags its child elements are placed. If an element has no child elements the end tag can be dropped and the start tag looks like this: `<key/>`.

Simple types are serialized from/to this format as a single element with the string representation as attribute of this element:

```
<key Value="value"/>
```

where `textttValue` is case sensitive. Complex types are serialized as an element with its members as child elements. Applying this to the example from the INI file format result in this:

```
<TestStruct>
  <NumMember Value="1"/>
  <StringMember Value="whatever"/>
</TestStruct>
```

Since in this format the end the description for an entry is clearly defined, lists don't need to specify the number of entries explicitly. Instead you can just write the elements as child of the list:

```
<StructList>
  <Struct>
    <NumMember Value="1"/>
    <StringMember Value="whatever"/>
  </Struct>
  <Struct>
    <NumMember Value="2"/>
    <StringMember Value="anything"/>
  </Struct>
</StructList>
```

For components an extra element with the actual component type as key and the components key as attribute is added:

```

<Model>
  <RCCuboidModel Key="BoxModel">
    [...]
  </RCCuboidModel>
</Model>

```

To refer to an already defined component you can specify the fully qualified key (meaning the full path to the component while the component keys are separated by a “.”) of the component you want to refer to:

```

<Model Ref="Pendulum1.Model"/>

```

Furthermore for some component types there are macros defined. Using them will create a new component but you don’t have to specify the parameters:

```

<Geom Macro="NoGeom"/>

```

4.3 String Conversion of Basic Datatypes

- **Boolean**

Converting a string to a boolean following values are interpreted as true: "True", "1", "On" and "Enable". These values are interpreted as false: "False", "0", "Off" and "Disable". The conversion is not case-sensitive. When converting a boolean to a string the respectively first string from the list is used.

- **Numerical Value**

All numerical datatypes use the same conversion since when reading a string it is first converted into double and then converted to the specific datatype. The first character needs to be a plus or minus sign, a digit or a “.”. You can add an order of magnitude by adding an “e” followed by the power to the value.

- **String**

The following replacements are done when converting an internal string to a string for serialization: "\t" ⇒ tabulator, "\n" ⇒ newline. Any other character behind a backslash is taken over as it is, thus, "\\\" will convert into a single backslash.

- **numerical value list**

This conversion doesn’t refer to a specific type but is used by several types. When reading a string into a value list there are very few requirements to the format. In

front of, between and after the values any number of the following characters can be written: “ ”, “, ”, “; ”, “\ ”, “(”, “) ”. The elements are converted the same way as a single numerical value.

- **Triple**

A triple holds three values. If all values are zero it converts into a single ”0”. Otherwise the values are converted into a list of, seperated by a comma and surrounded by braces. If the last value is zero it is not written. When converting a string into a triple the values are read as a numerical value list.

- **Vector**

A vactor holds 4 elements while the 4th element is just a scaling factor for the others. When writing a vector it is always normalized (settings the 4th element to 1) before – unless the 4th element is zero indicating that the vector defines a direction and, thus, can’t be normalized. A vector is read as a numerical value list where the 4th element can be any value.

- **Quaternion**

A quaternion is the extension of a complex number with two additional dimensions which makes it very suitable to represent an orientation ain the threedimensional space. It is written and read as a numerical lit with exactly four elements.

- **Pose**

A pose describes a position and orientation a threedimensional space. Internally it holds the position as a vector and the orientation as quaternion. It converts to and from a string always as a list of six elements: the three coordinates of the position and the three Euler angles of the orientation in degrees.

- **Matrix**

The matrix has a dimension of 4×4 . To simplify the representation as a string there are four possible formats all as nnumerical value lists but differing in the number of elements. When the matrix is converted into a string the shortest possible is used:

$$\text{– 3 elements: diagonal matrix } \mathbf{M} = \begin{pmatrix} x_0 & 0 & 0 & 0 \\ 0 & x_1 & 0 & 0 \\ 0 & 0 & x_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{– 9 elements: } 3 \times 3 \text{ matrix } \mathbf{M} = \begin{pmatrix} x_0 & x_1 & x_2 & 0 \\ x_3 & x_4 & x_5 & 0 \\ x_6 & x_7 & x_8 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned}
 - 12 \text{ elements: } 3 \times 4 \text{ matrix } \mathbf{M} &= \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
 - 3 \text{ elements: } 4 \times 4 \text{ matrix } \mathbf{M} &= \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}
 \end{aligned}$$

- **Cuboid**

A Cuboid hold the position of two opposite corners. When converting a string into a cuboid a numerical value list with six elements is expected where the first three elements specify one corner and the next three element the other corner.

- **Color**

A color is defined by a vector of four elements each in the interval $[0, 1]$. The first three elements specify the intensity of the colored red, green and blue and the 4th value (calls “alpha”) refers to the transparency. An alpha of zero results in a totally transparent (invisible) color and value of 1 results in a solid skin. In some cases it also makes sense not to specify a color – e.g. to use a default value. The string “invalid” converts into such an undefined color.

- **Enum**

Enums are often used in the application to select from a number of possibilities. Internally it is stored as a number but such a number is usually not very meaningful. For this reason there are keys assigned to each possible entry that depend on the enum type. The conversion from a string to an enum is in general case-insensitive.

- **Bitmask**

Bitmasks are stored as numbers that have the same disadvantage as for enums: the meaning is hard to guess from it. Thus, the same idea is used for enums with the difference that one bitmask is converted into a list of keys, each representing a subset of the bits set in the mask. They are separated by a “|”. Generating a string from the bitmask it is always tried to create the minimal combination of enums that express the value. If there exists no representation the missing bits are added as hexadecimal value with an additional “0x” in front of it.

Chapter 5

Simulation File

The simulationfile should be created using the XML format since it is much more comfortable to use for this purpose. You can use any text editor but an XML editor will probably make it more comfortable. I can recommend the opensource application Notepad++ together with the plugins "Insertion" and "XML Check". Visual Studio does a good job as well.

5.1 Camera List

The first Entry is CameraList containing a list of CameraComponents to be used by the views for this simulation. If the list is empty or missing a default camera is created automatically.

NOTE: Currently it is not possible to select or create a camera from the GUI. Thus, the first camera in this list will always be used for all views of this simulation.

5.2 Scene Settings

The next entry is SceneSettings containing the parameters that can be edited in the GUI from the menu item Settings->Scene. When this element is missing reasonable default values are used so don't worry about this part when creating your own simulation.

5.3 Component List

The Entry ComponentList is the actual description of the simulation itself. It can contain any number of components of the types BodyComponent and GlobalSimComponent. Later it will support SkinComponent, MaterialComponent and ModelComponent as well to be able to refer to. For information regarding the description of the individual component types refer to section 6.

Chapter 6

Component Types

This section contains list of all components of the base application that can be used in a simulation. To be easily able to keep this list up-to-date including extensions the software might be added by a tool to extract the information from the application generating the corresponding latex- (or optionally html-) code automatically. Of course, this would require to include additional explanation for the components and their parameters in the source code.

6.1 BodyComponents

6.1.1 RCStaticRigidBody

An immovable rigid body with a visible model and a physical geometry:

- **Pose**
A PoseComponent specifying the pose of the body in global coordinates.
- **Geom**
A GeomComponent specifying the geometry of the body used by the physics engine for collision detection.
- **Material**
A MaterialComponent specifying the physical properties of the body.
- **Model**
A ModelComponent specifying the appearance of the body in a 3D view.

6.1.2 RCMobileRigidBody

A rigid body similar to the RCStaticRigidBody but this one can move and, thus, requires additional parameters to calculate the motion.

- **Pose**
A PoseComponent specifying the initial pose of the body in global coordinates.
- **Motion**
A MotionComponent specifying the initial motion of the body in global coordinates.
- **Geom**
A GeomComponent specifying the geometry of the body used by the physics engine for collision detection.
- **Inertia**
A InertiaComponent specifying the mass and inertia tensor of the body.
- **Material**
A MaterialComponent specifying the physical properties of the body.
- **Model**
A ModelComponent specifying the appearance of the body in a 3D view.

6.1.3 RCRodBody*

This body is a mobile rigid body as well and adds no new functionality at all but makes the construction easier in the case you need a cylindric body connecting two points, since with this component you just need to specify the coordinates of these points and a few additional parameters rather than calculating the pose of the resulting body yourself.

- **Pos1**
A PoseComponent specifying the initial position of one end of the body in global coordinates. The specified orientation has no influence.
- **Pos2**
A PoseComponent specifying the initial position of the other end of the body in global coordinates. The specified orientation has no influence.
- **Radius**
The radius of the generated cylinder used for both, the model and the collision geometry.

- **Capped**
Boolean that determines whether a flat ended cylinder or one with spherical caps should be created.
- **HullPositions**
Boolean that determines whether the specified positions should be used as the most external points of the body or the body should be extended by the radius on both sides.
- **Motion**
A MotionComponent specifying the initial motion of the body in global coordinates.
- **Mass**
The mass of the Body in kg. The inertia tensor is calculated automatically according to the geometry.
- **Material**
A MaterialComponent specifying the physical properties of the body.
- **Skin**
A SkinComponent specifying the appearance of the model's surface.

6.1.4 RCConstruction

This BodyComponent gives the ability to create systems of bodies connected by joints and attached peripheries.

- **Body**
A BodyComponent representing the root of this construction. Although it is possible to use here a body of type RCConstruction I doubt that it makes any sense.
- **ChildList**
A list of JointComponents and PeripheryComponents linked to this body.

6.1.5 RCImportBody

This body enables you to reuse the definition of a body for several simulations or create multiple instances within the same simulation. You can specify a file that holds the body definition in xml-format (root element is "Body"). The pose and motion specified in that file will be overwritten by the parameters made for this component.

- **Pose**
A PoseComponent specifying the initial pose of the body in global coordinates.
- **Motion**
A MotionComponent specifying the initial motion of the body in global coordinates.
If the imported body is static
- **File**
The file name that holds the definition of the body to be imported. If the path specified is relative it is interpreted relative to the importing file's directory.

6.1.6 RigidBodyClone*

In more complex simulations it will probably often happen to have several bodies with identical properties. To avoid the necessity of copy/paste in the simulation file, the component RigidBodyClone allows you to create a copy of an already defined body and specify a new key and pose for it.

- **Pose**
A PoseComponent specifying the initial pose of the body in global coordinates.
- **Original**
Fully qualified key of the body that should be cloned.

6.2 GlobalSimComponents

6.2.1 RCGround

The ground is a static rigid body consisting of a horizontal plane through the worlds origin.

- **Material**
A MaterialComponent specifying the physical properties of the ground body.
- **Skin**
A SkinComponent specifying the appearance of the ground's surface.
- **Extension**
The visible extension of the plane. The physical extension is always infinite.

- **TexScale**
scaling factor for the texture. In case of a negative value the reference value is the extension of the model ($-0.5 \Rightarrow$ the texture image is mapped two times in both dimensions), otherwise the reference value is one meter ($0.5 \Rightarrow$ the dimension of one mapped texture image is 0.5m).
- **MaxCellSize**
Value specifying the maximal dimension of a cell the plane is constructed of. A negative value results in a plane consisting only of a single cell. This parameter is important for large planes when the camera gets close to it: with a too large cell size it can come to a visual tremble of the ground due to rounding errors.

6.2.2 RCSky

The sky is a static rigid body consisting of a horizontal plane at a specifiable height visible from the downside and physically cutting off the upper part.

- **Material**
A MaterialComponent specifying the physical properties of the ground body.
- **Skin**
A SkinComponent specifying the appearance of the sky surface.
- **Extension**
The visible extension of the plane. The physical extension is always infinite.
- **Height**
Height (i.e. the y coordinate) of the plane.
- **TexScale**
Scaling factor for the texture mapping with the same meaning as for RCGround described in section 6.2.1.
- **MaxCellSize**
Value specifying the maximal dimension of a cell the plane is constructed of with the same meaning as for RCGround described in section 6.2.1.
- **MotionX**
Motion of texture (no physical effect) in x direction.
- **MotionY**
Motion of texture (no physical effect) in y direction.

6.2.3 RCSandBox

For some simulations it might be useful to have a the area limited. To simplyfy this, the SandBox will create 4 vertical physical planes and visible walls for this. This component adds no ground or sky itself - use the corresponding components if you needed.

- **Width**
Width of the sandbox (extension in x direction).
- **Depth**
Depth of the sandbox (extension in z direction).
- **Thickness**
Thickness of the visible walls - has no effect on the physics, the walls are treated as having no thickness.
- **Height**
Height of the visible walls - has no effect on the physics, the walls are treates as infitie.
- **Wall**
A SkinComponent specifying the appearence of the walls' surface.
- **TexScale**
Scaling factor for the texture of the cuboids modelling the walls.
- **Material**
A MaterialComponent specifying the physical properties of the walls.

6.2.4 RCRobot

A simulated robot in the sense that a robot is the combination of hardware and software controlling its hardware:

- **Construction**
A BodyComponent – usually of type RCConstruction – representing the robots mechanics and peripheries.
- **ControllerList**
A list of ControllerComponents and SimControllerComponents representing the robots software.

6.2.5 RCImportRefList

Several components – especially materials and skins, sometimes models as well – will be used in more than one simulation. To avoid the necessity of copying there description from one file to another this component gives you the ability to load a file containing a list similar to the ComponentList of the simulation file in xml-format (root element is “RefList”) with the component types MaterialComponent, SkinComponent, ModelCponent.

- **File**

Name of the file holding the list of components to be imported. If the path specified is relative it is interpreted relative to the importing file’s directory.

6.3 ModelComponents

Models can currently only created by composing basic geometrical forms. Later the ability to load mesh models created with any model editor might be added.

Common for all model components are the following parameters:

- **Pose**

A PoseComponent specifying the pose of the model in the local coordinates of the object the model is drawn for. Since one model can be used for several objects there is no reasonable convention for the meaning of global coordinates and, thus, specifying global coordinates for a root model result in an error. For child models (added as childs to another model) ”global coordinates” refer to the coordinate system used by the root model – that is, relative to the body linked to this model.

- **Skin**

A SkinComponent specifying the appearance of the models surface.

- **ChildList**

A list of additional ModelComponents beeing childs of this model.

6.3.1 RCCuboidModel

Model with the shape of a cuboid:

- **Cuboid**

A Cuboid specifying the dimension of the model.

- **TexScale**
Scaling factor for the texture mapping with the same meaning as for RCGround described in section 6.2.1.
- **SideMask**
bitmask specifying which sides should be visible. A combination of the following values is valid:
 - **Left** / **MinX** – left side
 - **Right** / **MaxX** – right side
 - **Bottom** / **MinY** – top side
 - **Top** / **MaxY** – bottom side
 - **Back** / **MinZ** – back side
 - **Front** / **MaxZ** – front side
 - **X** – left and right side
 - **Y** – top and bottom side
 - **Z** – back and front side
 - **All** – all six sides
- **Wire**
Boolean determining whether the model should be always drawn as a wire mesh.

6.3.2 RCPlaneModel

A plane model with limited extension perpendicular to the Z axis.

- **ExtensionX**
Extension in local X direction.
- **ExtensionY**
Extension in local Y direction.
- **IsShadowPlane**
Boolean determining whether a shadow of all other objects should be projected on this plane. (Usually applies only to the ground plane since it is computationally very expensive.)
- **TexScale**
Scaling factor for the texture mapping with the same meaning as for RCGround described in section 6.2.1.

- **MaxCellSize**

Value specifying the maximal dimension of a cell the plane is constructed of with the same meaning as for RCGround described in section 6.2.1.

6.3.3 RCEllipsoidModel

- **Triangles**

Boolean specifying whether the model is to be constructed of triangles or quadrangles. The triangles are created by deviding the segments of a duodecaeder resulting in an even accuracy over the surface. The disadvantage is, that so far no skins with textures can be applied to such a model.

- **Accuracy**

Value specifying the accuracy with which the mathematical ellipsoid should be approximated. The greater the value the more segements are generated. In case of triangles the value specifies the recursion depths of splitting the segements with a maximum of 4. Thus, the number of segments n beeing generated is $n = 12 \cdot 4^{Accuracy}$. When generating quadrangles a step size for the spherical coordinates of $\frac{\pi}{8 \cdot Accuracy}$ ist used resulting in a number of segments of $n = 32 \cdot Accuracy^2$.

- **RadiusX**

Radius in local X direction.

- **RadiusY**

Radius in local Y direction.

- **RadiusZ**

Radius in local Z direction.

6.3.4 RCCylinderModel

A cylindric model with its rotation axis beeing aligned with the local Z axis.

- **Radius**

Radius of the cylinder.

- **Length**

Length of the cylinder.

- **AngTexScale**

Scaling factor for the texture along the perimeter.

- **LinTexScale**
Scaling factor for the texture along the rotation axis.
- **ComponentMask**
Bitmask specifying which components of the cylinder should be visible. A combination of the following values is valid:
 - **None** – no components are visible
 - **Jacket** – the jacket of the cylinder
 - **FlatFront** – a flat front side
 - **FlatBack** – a flat back side
 - **CappedFront** – a spherical cap at the front side
 - **CappedBack** – a spherical cap at the back side
 - **FlatCylinder** – a cylinder with flat front and back side
 - **CappedCylinder** – a cylinder with spherical caps at the front and back side
- **Accuracy**
Value specifying the accuracy the same way as for the RCEllipsoidModel described in section 6.3.3.

6.3.5 RCConeModel

A cone model with its rotation axis being aligned with the local Z axis.

- **Radius**
Radius of the cone.
- **Length**
Length of the cone.
- **LinTexScale**
Scaling factor for texture mapping.
- **DrawBase**
Boolean determining whether the base of the cone should be visible or not.
- **Accuracy**
Value specifying the accuracy the same way as for the RCEllipsoidModel described in section 6.3.3.

6.3.6 RCEmptyModel

When an object requires a model but should not be visible you can use an RCEmptyModel for it.

6.4 SkinComponent

6.4.1 RCColoredSkin

- **Color**
Base color of this skin. It is used when shading is disabled due to the view quality level and also as a reference color to calculate the others when they are not specified.
- **Ambient**
Color used for ambiente reflection. If not specified it is set to $\frac{1}{2} \cdot color$.
- **Diffuse**
Color used for diffuse reflection. If not specified it is set to $\frac{1}{2} \cdot color$.
- **Specular**
Color used for specular reflection. If not specified it is set to $2 \cdot color$.
- **Emission**
Color of emitted light. If not specified it is set to black.
- **ColorMode**
Specifies the color mode of the skin. Can be one of the following:
 - **Color** – only the base color is used
 - **Shading** – shading is turned on
 - **TransparentColor** – the skin is transparent according to its alpha vale but shading is turned off.
 - **TransparentLighing** – shading is turned on and the alpha value determines the transparency.
- **Shininess**
Value determining the shininess of the skin.

6.4.2 RCTexturedSkin

- **Texture**
Name of the texture mapped on the surface. The visual engine will look for an image file (currently only bitmaps are supported) in the texture folder (set in the global settings, see section 3.3.1) and load it if such a file exists and it was not previously loaded.
- **ColorMode**
Same as for RCColoredTexture described in section 6.4.1
- **Shininess**

6.4.3 RCColTexSkin

A combination of RCColoredSkin (described in section 6.4.1) and RCTextureSkin (described in section 6.4.2) blending the base color and texture. It has all the parameters of both the components.

6.5 GeomComponent

6.5.1 RCTransformGeom

Usually the GeomComponents have their origin in their center to make it easy to model objects with an even density. In situation where this is not the case you can use the RCTransformGeom to allow an offset in the position and orientation relative to the body linked to it.

- **Pose**
Pose of the child geom in the coordinate system of the body linked to it.
- **Sub Geom**
A GeomComponent specifying the geometry of the body used by the physics engine for collision detection.

6.5.2 RCCuboidGeom

Cuibold geometry with the center of mass in its origin.

- **Width**
Extension of the cuboid along the X axis.
- **Height**
Extension of the cuboid along the Y axis.
- **Depth**
Extension of the cuboid along the Z axis.

6.5.3 RCPlaneGeom

- **Pose**

6.5.4 RCSphereGeom

- **Radius**

6.5.5 RCCylinderGeom

- **Radius**
- **Length**

6.5.6 RCCappedCylinderGeom

- **Radius**
- **Length**

6.5.7 RCMoelMatchGeom*

To simplify and speed up the description of a body this geom tries to get all required information from the model of the body and creates a one of the other geoms that exactly matches with it – which is, of course, not possible for all models.

- **IgnoreChilds**
Boolean determining if the childs of the models should be ignored.

6.5.8 RCMoelWrapGeom*

Creates a geometry with the extensions chosen that the model is completely wrapped by the geometry.

- **GeomType**
Specifies with which kind a geom the model should be wrapped. Valid values are:
 - **Cuboid**
 - **Sphere**
 - **Cylinder**
 - **CappedCylinder**
- **Alignment**
In case of a cylindric geometry beeing created you need to specify the orientation of that geom. Valid values are:
 - **XAligned** – the X axis is the rotation axis
 - **YAligned** – the Y axis is the rotation axis
 - **ZAligned** – the Z axis is the rotation axis
- **IgnoreChilds**
Boolean determining if the childs of the models should be ignored.

6.6 JointComponent

Joints always connect two rigid bodies. They have their own coordinate system to allow the specification of coordinates for an anchor or the second body they connect in this coordinate system. The common parameters are:

- **Pose**
A PoseComponent specifying the reference position of the joint in global coordinates.
- **Body**
A BodyComponent the joints parent body is connected to.

6.6.1 RCBallSocketJoint

The ball-socket-joint allows rotation respectively the common anchor point.

- **Anchor**
A PoseComponent specifying the anchor point at which the two bodies are connected in global coordinates – usually this is the origin of the joint. The specified orientation has no influence.

6.6.2 RCHingeJoint

The hinge joint allows a rotation respectively a single common axis.

- **Hinge**
A PoseComponent specifying the anchor point at which the two bodies are connected in global coordinates. The local Z axis is the common rotationaxis.

6.6.3 RCSliderJoint

The slider joints allows a relative shift of the two body relative to each other.

- **Axis**
The local Z axis specifies the axis along which a shift is allowed. The specified position ignored.

6.6.4 RCFixJoint

The fixed joint prevents any relative motion of the connected bodies and requires no parameters. Currently it can handle only a different position of the connected bodies but no difference in the orientation.

6.7 MaterialComponents

6.7.1 RCGenericMaterial

Allows the specification of all parameters explicitly:

- **Friction**
Friction coefficient of the material.
- **Bounceness**
Value representing the elasticity of the material.
- **MinBounceVel**
Minimal velocity that is necessary that a body of this material bounces of from another.

See section 7.1 for information on how the physics engine uses these parameters

6.8 PoseComponents

A PoseComponent can return a pose in local and global coordinates. Usually it is linked to an object with an own pose that defines the local coordinate system. If such an object is missing a conversion between local and global coordinates is impossible and results in an error if tried.

6.8.1 RCGlobalCoordinates

Allows you to specify coordinates in the global coordinate system.

- **Pose**
Pose in global coordinates.

6.8.2 RCLocalCoordinates

Allows you to specify coordinates in the local coordinate system.

- **Pose**
Pose in local coordinates.

6.9 MotionComponents

6.9.1 RCGlobalMotion

Allows you to specify the motion of a body relative to the global coordinate system

- **LinVelocity**
A triple specifying the linear velocity along the three axis in $\frac{m}{s}$.
- **AngVelocity**
A triple specifying the angular velocity $\vec{\omega}$ while the norm of $\vec{\omega}$ specifies the rotation speed and the direction specifies the rotation axis.

6.10 InertiaComponents

6.10.1 RCGenericInertia

Allows the specification of all parameters explicitly:

- **Mass**
Mass of the body in kg.
- **Inertia**
A 3×3 matrix specifying the inertia tensor of the body.

6.10.2 RCInertiaByGeom*

Requires you just to specify either the mass or the density of the body. The inertia tensor – and if the density is specified the mass as well – is calculated automatically according to the geometry of the body.

- **Mass**
Mass of the body in kg. If the value is negative the Mass is calculated from the density according to the geometry of the body.
- **Density**
Average density of the body. If a value ≥ 0 for Mass is specified this value is ignored.

6.11 PeripheryComponents

To model hardware other than mechanics (especially sensors and actuators) you can use Peripheries. They can be attached to constructions and usually provide an interface that allows controllers (or also other peripheries) to access them – be it for controlling or reading sensor data. All parameters are type specific.

6.11.1 RCGenericDrive

A simulated drive that allows the linked controller to specify a voltage (through the interface RCDriver) and applies the corresponding power (force or torque, depending on the joint type) on the bodies connected by the joint the drive is linked to. The power applied by the drive is proportional to the voltage. This linear characteristic is multiplied by the stall-factor that is 1 for $-v_{stall} < v < v_{stall}$ with $v_{stall} = stall \cdot v_{max}$ and 0 for $v \leq -v_{max}$ and $v \geq v_{max}$ being interpolated linearly in between. This results in the drive not overcoming the specified maximal velocity by its own power.

- **Joint**
The JointComponent the drive should be linked to. This must be either a hinge joint (to create an angular drive) or a slider joint (to create a linear drive).
- **MaxPower**
Maximal force (in N) or torque (in Nm) that can be applied by the drive.
- **MaxVelocity**
Maximal velocity (in m/s or rad/s) that can be reached by the drive.
- **MinPos**
Minimal position of the drive (in m or rad) – set it to “-Infinite” to disable it.
- **MaxPos**
Maximal position of the drive (in m or rad) – set it to “Infinite” to disable it.
- **Stall**
Factor to calculate v_{stall} – the velocity at which the power starts to be scaled down.

6.11.2 RCGenericServo

A simulated drive that allows the linked controller to specify a destination position (through the interface RCPosControl) and applies the corresponding power (force or torque, depending on the joint type) on the bodies connected by the joint the drive is linked to, so that the position is reached as fast as possible.

- **Joint**
The JointComponent the servo should be linked to. This must be either a hinge joint (to create an angular servo) or a slider joint (to create a linear servo).
- **MaxPower**
Maximal force (in N) or torque (in Nm) that can be applied by the servo.
- **MaxVelocity**
Maximal velocity (in m/s or rad/s) that can be reached by the servo.
- **MinPos**
Minimal position of the drive (in m or rad) – set it to “-Infinite” to disable it.
- **MaxPos**
Maximal position of the drive (in m or rad) – set it to “Infinite” to disable it.
- **Smooth**
This number (needs to be > 0) specifies the velocity reduction when getting closer to the destination position.

6.11.3 RCGenericVelControlledDrive

A simulated drive that allows the linked controller to specify a destination velocity (through the interface RCVelControl) and applies the corresponding power (force or torque, depending on the joint type) on the bodies connected by the joint the drive is linked to, so that the velocity is reached as fast as possible.

- **Joint**
The JointComponent the servo should be linked to. This must be either a hinge joint (to create an angular servo) or a slider joint (to create a linear servo).
- **MaxPower**
Maximal force (in N) or torque (in Nm) that can be applied by the servo.

- **MaxVelocity**
Maximal velocity (in m/s or rad/s) that can be reached by the servo.
- **MinPos**
Minimal position of the drive (in m or rad) – set it to “-Infinite” to disable it.
- **MaxPos**
Maximal position of the drive (in m or rad) – set it to “Infinite” to disable it.

6.11.4 RCProximitySensor

A simulated proximity sensor that allows the linked controller to check if an object was detected (through the interface RCContactSensor). It uses internally the collision detection of the physics engine and simply checks if the specified `GeomComponent` collides with any other body of the simulation. Note that `GeomComponent` is virtual, meaning that it will not bounce off from objects but simple detect that collision.

- **Pose**
A `PoseComponent` specifying the reference position of the joint in global coordinates.
- **Geom**
A `GeomComponent` specifying the geometry used by the physics engine for collision detection.

6.11.5 RCImportJoint

This joint enables you (analog to `RCImportBody`) to reuse the definition of a joint for several simulations or create multiple instances within the same simulation. You can specify a file that holds the body definition in xml-format (root element is “Joint”). The pose specified in that file will be overwritten by the parameter made for this component.

- **Pose**
A `BodyComponent` representing the root of this construction. Although it is possible to use here a body of type `RCConstruction` I doubt that it makes any sense.
- **File**
The file name that holds the definition of the joint to be imported. If the path specified is relative it is interpreted relative to the importing file’s directory.

6.11.6 RCLocalSocketCommunicator

This periphery allows communication with other applications running on the same computer via sockets. If the component is running as server it will wait for a client trying to contact on the specified port. When the connection is terminated it will start waiting for a new one. Running in client mode it tries to contact a waiting server on that port. If there is no server running it will give an error message and not retry. Thus, you better make sure the server is always started first!

- **IsServer**
Boolean specifying if the component should run as server or client.
- **Port**
Port-Number for the communication.

6.12 ControllerComponents

A controller represents the controlling software of a robot. You can either write a single robot that links to all the peripherals of the construction or write a controller for every sub-task and link them through interfaces. The last style allows you to reuse the controllers for other simulations and also to easily exchange controllers e.g. to test different algorithms on the same task. All parameters are type specific.

6.12.1 RCDriveTest

A test controller linking to a drive and applying a voltage that changes sinusoidally over time.

- **Drive**
A Reference to the component that should be controlled. It needs to implement the interface RCDrive.
- **Period**
Loop period of the controller's thread.
- **Variation**
Number of sinus cycles per minute.

6.12.2 RCServoTest

A test controller linking to a position control oscillating between the minimal and maximal position with constant velocity.

- **PosControl**
A Reference to the component that should be controlled. It needs to implement the interface RCPosControl.
- **Period**
Loop period of the controller's thread.
- **Variation**
Number of oscillations per minute.

6.12.3 RCTwoWheelMotionControl

Controller that translate the specified angular and linear velocity (through the interface RC2DMotionControl) into destination velocities for the drives of two parallel wheels.

- **ControlLeft**
A Reference to the component that receives the destination velocity of the left wheel. It needs to implement the interface RCVelControl.
- **ControlRight**
A Reference to the component that receives the destination velocity of the right wheel. It needs to implement the interface RCVelControl.
- **WheelDistance**
Distance between the two wheels.
- **Variation**
Radius of the wheels.

Chapter 7

Simulation Engine

This chapter includes some information regarding the simulation engine that I consider useful to understand the meaning of some parameters and how the simulation works.

7.1 Physics

The physics engine requires for each contact detected to specify a value for the friction coefficient, the elasticity (bounceness) and the minimal velocity required so that one body actually bounces off the other. These parameters would actually need to be specified for each possible combination of colliding materials requiring a possibly quite large matrix. To overcome this need you have to specify these values only once for each material instead. When determining the parameters for a pair of colliding materials the maximum of the values for friction and bounceness and the minimum of the minimal bounce velocity is used. There is no physical principle underlying this method but it is simple to realize and proved to be sufficient so far.

7.2 Synchronization

The synchronization between the simulation and any component accessing the simulation is done on a request-grant basis. This means, that a module that needs to access the simulation data (be it a sensor or a 3D view or anything else) sends a request to the simulation specifying the simulation-time at which it wants the access. The requests are stored in a list sorted by the requested simulation-time.

If the earliest requested time is close enough to the current time (i.e. it maximal differs by `MinTimeStep`) the access is granted to the accessor and it is ensured that this is the only module accessing the simulation until it frees the access. To keep the simulation from freezing due to a not freed access the simulation will continue after a specific time (`MaxAccessTime`) has elapsed. In this case an error message is logged.

If the earliest request is too far away it performs a time step calculating all collisions and integrating the differential equations of motion up to the next requested time (considering the value of `MaxTimeStep`).

The synchronization between the real time and the simulation time is only done through the 3D views that will sleep for some time when the simulation is too fast or reduce their frame rate when it takes too slow.

Part II

Developer Guide

Chapter 8

Basic Design Concepts

TODO...

8.1 Module Hierarchy

TODO...

8.2 Directory Structure

TODO...

8.3 Parameter Handling

TODO...

8.4 Extension Handling

TODO...

Chapter 9

Development Tutorials

This chapters should give a rough idea how to write new source using the simulation environment. If you want to implement new components – and most practical simulations will require new ones – you should first create a new extension dll in which you place those classes. Of course you are allowed to modify the existing modules as well but this would make it more difficult to merge your sources with a new release.

9.1 Using the Class Creator

In all tutorials the application ClassCreatorBob will be used. It's a small tool I wrote to save me lot of time when creating new classes. You can download it from the project page of SimulatotBOB or get it from CVS. But you should NOT put the sources in the same folder as SimulatorBob since it used different configurations for the base modules!

The tool works by taking draft files, replacing key-words in it and save them as new component-specific ones. Feel free to adapt the draft files to your needs – especially the copyright note will not fit for your work, I guess. It's also possible to add new component types without modifying the source code by adapting the ini-file and add the required draft files.

9.2 Implementing a new Extension DLL

To create a new extension dll click on the “New” Button next to the Module ComboBox and type in the name of the new module. It needs to be a valid identifier but this will not be checked - otherwise the resulting project will not work, though. A new project will be created with the specified name in lower case. The namespace in it also has the specified name, but upper cased.

To create the project file a new GUID is required which will be generated by the tool `uuidgen.exe` from visual studio. The file is located in the folder “Program Files/Microsoft Visual Studio .NET 2003/Common7/Tools/Bin”. The path variable needs to be set accordingly to make it work.

When the creation was successful a new project and solution file was generated. The solution also contains all the other modules from SimulatorBob this one depends on so you should immediately be able to compile it – Although it doesn’t contain any components yet...

9.3 Implementing a new ComponentType

If there exists already a template in ClassCreatorBob for the kind of component you want to create it’s recommended to create the files with this tool to make things easier. In case you just want to play around mark the check box “No Modifying”. If it is checked the new files will not be added to the project- and header files and you can’t do any harm. Existing files won’t be overwritten! Of course you will have to adapt all the created files to do what you actually want it to do.

9.3.1 BodyComponent

TODO...

9.3.2 GlobalSimComponent

TODO...

9.3.3 ModelComponent

TODO...

9.3.4 SkinComponent

TODO...

9.3.5 GeomComponent

TODO...

9.3.6 JointComponent

TODO...

9.3.7 MaterialComponent

TODO...

9.3.8 PoseComponent

TODO...

9.3.9 MotionComponent

TODO...

9.3.10 InertiaComponent

TODO...

9.3.11 ControllerComponent

For controller components there are two templates in the class creator: active and passive controller. They are both derived from the same base class and the only difference is, that the active controller already contains the code for running an own thread while the passive one is expected to be accessed only through its interface and performs no actions on its own. Of course, a controller could also have more than one thread.

For a better idea take a look at the existing controllers - so far they are all pretty simple.

9.3.12 PeripheryComponent

TODO...

Part III

Appendix

Chapter 10

Coding Conventions

TODO...

Chapter 11

GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals;

it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps,

when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one

stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to

copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c) YEAR YOUR NAME.  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the GNU Free Documentation License, Version 1.2  
or any later version published by the Free Software Foundation;  
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.  
A copy of the license is included in the section entitled "GNU  
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the  
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Chapter 12

Document History

Date	Author	Modification
2005-11-25	Patrik Stellmann	first version of this document (incomplete)
2005-12-23	Patrik Stellmann	added description of new components of v1.4.0 (still incomplete)
2006-02-06	Patrik Stellmann	added description of new components of v1.4.1 (still incomplete) started with developer guide (extension dlls and controllers)