



TWML – Twinity Markup Language

Release 2.0 (Draft)

January 28, 2010

CHAPTER 1: CONTENT MARK UP SPECIFICATION	4
1.1 INTRODUCTION	4
1.2 OVERVIEW	4
CHAPTER 2: DATATYPES	5
CHAPTER 3: ELEMENTS	7
3.1 HIERARCHY	7
3.2 PACKAGE	8
3.2.1 DESCRIPTION	9
3.2.2 OBJECT	10
3.2.2.1 ACTION	11
3.2.2.1.1 OBJECT_ANIMATION	12
3.2.2.1.2 AVATAR_ANIMATION	13
3.2.2.1.3 ORIGIN	15
3.2.2.2 MATERIAL_SLOT	16
3.2.2.3 FRAME	17
3.2.2.3.1 FRAME_CONTENT	18
3.2.2.4 LIGHT	19
3.2.2.5 RADIO	20
3.2.3 CLOTHES	21
3.2.3.1 CLOTHING_SLOT	22
3.2.4 ATTACHMENT	24
3.2.4.1 ATTACHMENT_SLOT	25
CHAPTER 4: EXAMPLES	27
4.1 SIMPLE STATIC MODEL	27
4.2 OBJECTS WITH ANIMATION	27
4.2.1 CONTINUOUSLY SELF ANIMATED OBJECT	27
4.2.2 SELF ANIMATED OBJECT, THAT IS TRIGGERED BY CLICK	27
4.2.3 SELF ANIMATED OBJECT, THAT IS TRIGGERED BY APPROACH	27
4.3 STATIC OBJECTS WITH SPECIAL PROPERTIES	28
4.3.1 RADIO	28
4.3.2 LIGHT	28
4.3.3 FRAME	28
4.3.4 MIRROR WITH CUSTOMIZABLE BORDER MATERIAL	28
4.3.5 MULTI FUNCTIONAL FRAME	29
4.4 SPECIAL OBJECTS	29
4.4.1 CLOTHES	29

4.4.1.1 MALE PANTS	29
4.4.1.2 UNISEX WRIST WATCH	29
4.4.2 ATTACHMENT	30
4.4.3 ANIMATIONS	30
4.4.3.1 AVATAR ANIMATION	30
4.4.3.2 AVATAR ANIMATION WITH FANCY PARAMETERS	30
4.4.4 OBJECTS THAT TRIGGER AVATAR ANIMATIONS	30
4.4.4.1 A DANCE PAD	30
4.4.4.2 A CHAIR WITH ONE SEAT POINT	31
4.4.4.3 A CHAIR WITH DIFFERENT SIT DOWN ANIMATION FOR MALE AND FEMALE SKELETON	31
4.4.4.4 A BENCH WITH TWO SEAT POINT	31
4.4.4.5 A HUG-COUCH	32

Chapter 1: Content mark up specification

1.1 Introduction

The Twinity content pipeline is build for users that want to import 3d-models and animations from other third-party tools or platforms like Google Warehouse into Twinity. Our goal is to create an open content format specification that opens up as much functionality if the Twinity game engine to content creators as possible.

We choose Collada as basic file format for 3d-models and animations, as it is supported by all major content creation tools (e.g. 3d Studio Max, Maya, Google Sketch-up, Blender, Google Warehouse). Any Collada model that fits our technical requirements can be uploaded to Twinity as static content.

Similar to HTML, which adds “markup” to static image and text information, we invented a new, open, XML-based file-format called “Twinity markup language” (TWML) to allow users to add additional markup/functionality to their static Collada 3d-models. This document contains a detailed specification of the TWML specification.

1.2 Overview

The Twinity Markup Language (TWML) is an XML-based open specification that allows adding meta-information to Collada 3d-Models. Some example meta-information:

- Name
- Description
- Type of object (e.g. Radio, TV, Light, Chair, etc.)
- Light properties
- Animation playback properties
- Chair properties (seat-point, attached animations)
- Cloth properties (clothing slot, gender, etc.)

TWML describes engine specific properties of a 3d model. It can reference external resource files like models, animations, textures, sounds, scripts etc.

Based on the examples provided in this document, content creators can create their own TWML-files with a text editor.

Twinity can both import stand-alone Collada 3d-models and TWML-files, which contain references to Collada models and animations.

TWML documents are files with “.twml” extension.

Chapter 2: Datatypes

Every attribute in a XML document (and therefore TWML document) is of some datatype. A datatype determines what data can be put in an attribute value and defines its format.

TWML uses common XML Schema datatypes for attributes or customized datatypes derived from them. You can read more about XML datatypes on web: [XML Schema Part 2: Datatypes Second Edition](#).

This chapter describes all datatypes used in TWML.

bool

Represents Boolean data – a flag that can be turned off and on. Allowed values are: “true”, “false”, “0” and “1”. This datatype is the same as [xs:boolean](#) in XML Schema.

Examples

```
<someelement visible="true" />
<animation mixed="1" />
<material_slot recursive="false" />
```

int

Represents signed integer value. This datatype is the same as [xs:int](#) in XML Schema.

Examples

```
<radio volume="50" />
<animation repeat="5" />
```

float

Represents single precision (32 bit) floating point number. This datatype is the same as [xs:float](#) in XML Schema. For details on the format and allowed values please read [xs:float](#) documentation.

Examples

```
<origin x="0" y="23.4" z="-1E4" />
<scale>12.78e-2 1.5 10.1</scale>
```

string

Represents single-line string values. This datatype is the same as [xs:string](#) in XML Schema. Please read [Character Encoding in Entities](#) chapter of XML documentation about encoding special and non-ASCII characters.

Examples:

```
<material_slot material="#ScreenPlaneMat" caption="Screen" type="frame"/>
<description>Customizable geometry - mirror/picture/screen.</description>
```

name

Represents string values that define some naming for an object. Character set of such string values is very limited and allows only English alphabet characters ('a'..'z', 'A'..'Z'), underline ('_') and minus sign ('-'). Spaces and other characters now allowed. This datatype is the same as [xs:ID](#) in XML Schema.

This datatype is used for naming various internal parts of TWML document for further referencing them from other parts of document or scripts. Names are **case sensitive**.

Examples

```
<material name="ScreenPlaneMat" />
<action name="AttachToLHand">
```

url

Represents URL link to some external (local or remote) or internal (in package) resource. This datatype is the same as [xs:anyURI](#) in XML Schema.

Depending on the context, the URL can point to resource on remote site or can be limited only to local URLs (within local file system). If the protocol part of the URL is missing, then it assumed that URL points to resource in local file system. If URL describes local path of file name, then it's relative to path of TWML document.

Examples

```
<radio url="http://radio.com.ua:877/xxx.m3u" range="30.0" volume="50" type="3D" />
<model path="model.dae" />
<model path="file:///model.dae" />
```

color

Represents color. The value format is: #RRGGBB in hex.

Examples

```
<light color="#00FF00"/>
```

vector

Represents vector of 3 [floats](#). The value format is: "float float float".

Examples

```
<position position="0 0 22.3"/>
```

Chapter 3: Elements

3.1 Hierarchy

Below you find the TWML elements hierarchy. It's **not** a working example but an outline tree of all possible elements in TWML file. For working examples see [Examples](#) chapter.

```
<?xml version="1.0" encoding="utf-8"?>
<package>
  <description />
  <object>
    <action>
      <object_animation />
      <avatar_animation />
      <origin />
    </action>
    <light />
    <radio />
    <material_slot />
    <frame>
      <frame_content>
    </frame>
  </object>
  <avatar_animation />
  <clothes>
    <clothing_slot />
  </clothes>
  <attachment>
    <slot />
  </attachment>
</package>
```

3.2 package

Introduction

Declares the root of the document that holds every other element.

Concepts

The TWML schema is XML based; therefore, it must have **exactly one** document root element or document entity to be a well-formed XML document. The `<package>` element serves that purpose.

There is can be only one collada document per `<package>`

Attributes

The `<package>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
caption	string	Visual name of package.	Required.
icon	url	Path to icon file. Icon must be in any image format supported by Twinity engine.	Optional. No default.

Related Elements

The `<package>` element relates to the following elements:

Parent elements	No parent elements
Child Elements	See the following subsection
Other	None

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><description></code>	Long multi-line textual description of package.	0 or 1
<code><object></code>	Declares object.	0 or 1
<code><avatar_animation></code>	Declares avatar animation object.	0 or 1
<code><clothes></code>	Declares clothes object.	0 or 1
<code><attachment></code>	Declares attachment object.	0 or 1

3.2.1 description

Introduction

Contains long multi-line description of package.

Concepts

Long description is shown in client when browsing objects.

Attributes

The `<description>` element has no attributes.

Related Elements

The `<description>` element relates to the following elements:

Parent elements	<code><package></code>
Child Elements	None
Other	None

Examples

See [examples](#).

3.2.2 object

Introduction

Declares an object which can be placed in the world.

Concepts

The `<object>` element corresponds always to a Collada 3d-model. Special functionality of this model is defined in the sub-nodes.

Attributes

The `<object>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for object, allowing referencing it from script.	Optional
path	url	Path link to COLLADA file with static model.	Required.
physics	name	Name of a collision mesh from COLLADA file.	Optional.

Related Elements

The `<object>` element relates to the following elements:

Parent elements	<code><package></code>
Child Elements	See the following subsection
Other	None.

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><action></code>	Declares action logic.	0 or more
<code><light></code>	Declares a light.	0 or more
<code><radio></code>	Declares radio (streamable audio) logic.	0 or 1
<code><material_slot></code>	Declares customizable material slot logic.	0 or more
<code><frame></code>	Declares customizable frame logic.	0 or more

Examples

```
<object>
  <light />
  <radio />
</object>
```

3.2.2.1 action

Introduction

Declares action-logic-trigger that can be activated by user or script and performs some animations on an object and/or player.

Concepts

This element helps to link models with animations and avatars. It declares the way how they will be executed. If `<avatar_animation>` is present - `<origin>` tag should be present to. So `<origin>` tag is needed only for actions with avatar animations.

Attributes

The `<action>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for action, allowing referencing it from script.	Optional.
caption	string	Visual name of action.	Required.
trigger	string	Event on which action is activated. Can be: <ul style="list-style-type: none">• “init” – activates automatically, on object creation/placement;• “click” – player manually activates object by clicking;• “approach” – activates when player get close to object• “idle” – activates automatically after init animation is finished	Required.
range	float	Range at which action can be triggered by player when trigger set to “approach” .	Optional. Default: 1.0

Related Elements

The `<action>` element relates to the following elements:

Parent elements	<code><object></code>
Child Elements	See the following subsection
Other	None

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><object_animation></code>	Plays animation on object	0 or more
<code><avatar_animation></code>	Plays animation on avatar	0 or more
<code><origin></code>	Sets player to this origin point at the activation.	0 or 1

Examples

See `<action>` [examples](#).

3.2.2.1.1 object_animation

Introduction

Declares an animation for an object (as opposed to an avatar animation) that can be used in action or script.

Concepts

Actions can have multiple animations each specified in its own COLLADA file.

Attributes

The `<object_animation>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
path	url	Path link to COLLADA file.	Required.
name	name	Declares name for animation, allowing referencing it from script.	Optional.
repeat	int	Repeat count of animation (0 – infinite).	Optional. Default: 0
blend_in	int	Blend-in time in milliseconds	Optional. Default: 0
blend_out	int	Blend-out time in milliseconds	Optional. Default: 0
blend_mode	string	Blending mode. Allowed values: <ul style="list-style-type: none">• “overwrite” Overwrite mode. This can be used to switch from for example walk into run;• “additive”. Additive mode. This can be used to add the given motion relatively to the current result.	Optional. Default: “overwrite”

Related Elements

The `<animation>` element relates to the following elements:

Parent elements	<code><action></code>
Child Elements	None
Other	None

Examples

See `<action>` [examples](#).

3.2.2.1.2 avatar_animation

Introduction

Declares a single avatar animation that can be used in action or script.

Concepts

Actions can have multiple avatar animations each specified in its own COLLADA file.

Attributes

The `<avatar_animation>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
path	url	Path link to COLLADA file.	Required.
name	name	Declares name for animation, allowing referencing it from script.	Optional.
trigger	string	Event on which animation is started. Can be: <ul style="list-style-type: none">• “begin” – player manually activates animation by clicking;• “idle” – activates automatically after “begin” animation is finished. If it is not present - “sync” animation should start playing immediately;• “sync” – activates automatically after “idle” animation synchronously with other “sync” animations on the same object;• “end” – player manually activates animation by moving (stand up);	Optional. Default: idle
repeat	int	Repeat count of animation (0 – infinite).	Optional. Default: 0
blend_in	int	Blend-in time in milliseconds	Optional. Default: 0
blend_out	int	Blend-out time in milliseconds	Optional. Default: 0
blend_mode	string	Blending mode. Allowed values: <ul style="list-style-type: none">• “overwrite” - Overwrite mode. This can be used to switch from for example walk into run;• “additive” - Additive mode. This can be used to add the given motion relatively to the current result.	Optional. Default: “overwrite”
skeleton	string	Defines what avatars this animation for. Can be: <ul style="list-style-type: none">• “male” – allows male avatars;• “female” – allows female avatars;	Optional. Default: all

		<ul style="list-style-type: none"> • “all” – allows any avatar. 	
--	--	---	--

Related Elements

The `<avatar_animation>` element relates to the following elements:

Parent elements	<code><action></code>
Child Elements	None
Other	None

Examples

See `<action>` [examples](#).

3.2.2.1.3 origin

Introduction

Teleports avatar to some point relative to object's origin before starting an animation

Concepts

This element teleports the avatar to some point with coordinates relative to object's origin, and give it new facing angle (yaw), before the animation is started. This can be needed, for example, for seats actions that positions player at some point and then plays the sit down animation on avatar.

Attributes

The `<origin>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
position	vector	x, y, z coordinates.	Required.
yaw	float	yaw angle (facing direction).	Required.

Related Elements

The `<origin>` element relates to the following elements:

Parent elements	<code><action></code>
Child Elements	None
Other	None

Examples

See `<action>` [examples](#).

3.2.2.2 material_slot

Introduction

Declare customizable material slot logic. Is used to specify materials that can be customized by users.

Concepts

In Twinity, users can upload images from their hard-drive to customize objects. The uploaded images just replace the pre-modeled textures on the object. If multiple faces in the object use the same material, all faces will be changed to the new material.

Attributes

The `<material_slot>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for material slot, allowing referencing it from script.	Optional.
caption	string	Visual name of material slot.	Optional. Material-name will be used if not specified.
material	name	Name of material in the COLLADA file.	Required.

Related Elements

The `<material_slot>` element relates to the following elements:

Parent elements	<code><object></code>
Child Elements	None
Other	None

Examples

See `<material_slot>` [examples](#).

3.2.2.3 frame

Introduction

Adds customizable frame logic to an object. Frames in Twinity are 2d-surfaces that can be used as Picture, Web-Browser, Mirror, and Video-Player.

Concepts

In Twinity, frames are a quite popular and simple way to customize apartments. Frames are just 3d-models with a rectangular surface that has a special function in the game engine. The function is defined by the sub-elements. If multiple-functions are defined the user can choose one of them from a context menu.

Attributes

The `<frame>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for frame, allowing referencing it from script.	Optional.
material	material	ID of material in the COLLADA file.	Required.

Related Elements

The `<frame>` element relates to the following elements:

Parent elements	<code><object></code>
Child Elements	See the following subsection
Other	None

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><frame_content></code>	Defines content sub logic.	1 or more

Examples

See `<frame>` [examples](#).

COLLADA note

To change pictures in frame, material in COLLADA should have texture assigned to it.

3.2.2.3.1 frame_content

Introduction

Declare customizable frame sub logic type.

Attributes

The `<frame_content>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
type	string	Specifies sub logic type. Supported values: <ul style="list-style-type: none">• "picture" – custom picture• "mirror" – mirror sub logic• "browser" – web browser sub logic• "flash_video" – flash video sub logic (f.e. from youtube.com)• "quicktime_video" – streaming quicktime video	Required.
preview	bool	Enable preview for this object	Optional. Default: "true" .

Related Elements

The `<frame_content>` element relates to the following elements:

Parent elements	<code><frame></code>
Child Elements	None
Other	None

Examples

See `<frame_content>` [examples](#).

3.2.2.4 light

Introduction

Adds customizable light logic to an object.

Attributes

The `<light>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for light logic, allowing referencing it from script.	Optional.
type	string	Specifies light type for custom lights. Can be: <ul style="list-style-type: none">• "spot" – Spot light, has position, direction, cone angle etc;• "point" – Point light, has position, shines equally in all direction.	Required
color	color	Defines light color.	Optional. Default: "#FFFFFF".
falloff_angle	float	Defines falloff angle of spot light. Range: "0.0 - 90.0"	Optional. Required for light if type="spot".
radius	float	Defines light radius. Range: "0.0 - 10.0"	Required.
position	vector	Defines the light position relative to object's origin.	Optional. Default: "0.0 0.0 0.0".
direction	vector	Second point which defines the light direction relative to object's origin.	Optional. Required for light if type="spot".

Related Elements

The `<light>` element relates to the following elements:

Parent elements	<code><object></code>
Child Elements	None
Other	None

Examples

See `<light>` [examples](#).

3.2.2.5 radio

Introduction

Adds radio (streamable audio) logic to an object.

Concepts

Radio logic allows users to playback internet radio streams at the object's origin.

Attributes

The `<radio>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for radio logic, allowing referencing it from script.	Optional.
url	url	Default URL for audio stream.	Optional. No default.
volume	int	Default volume in percents	Optional. Default: 100
type	string	Sound emitter type. Can be: <ul style="list-style-type: none">• "2D" – 2D-audio without positioning in world;• "3D" – 3D-audio.	Optional. Default: 3D

Related Elements

The `<radio>` element relates to the following elements:

Parent elements	<code><object></code>
Child Elements	None
Other	None

Examples

See `<radio>` [examples](#).

3.2.3 clothes

Introduction

Declares a clothes object.

Concepts

Clothes is an wearable object that replaces a certain part of the avatar.

Attributes

The `<clothes>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for clothes, allowing referencing it from script.	Optional
path	url	Path link to COLLADA file with cloth model.	Required.
skeleton	string	Defines what avatars this clothes for. Can be: <ul style="list-style-type: none">• “male” – allows male avatars;• “female” – allows female avatars;• “all” – allows any avatar.	Optional. Default: all

Related Elements

The `<clothes>` element relates to the following elements:

Parent elements	<code><package></code>
Child Elements	See the following subsection
Other	None

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><clothing_slot></code>	Defines to what slots that clothes are applied.	1 or more

Examples

See `<clothes>` [examples](#).

3.2.3.1 clothing_slot

Introduction

Defines to what slots a clothes object will be applied.

Concepts

Clothes are attached to some logical slots. When new clothes are attached to avatar it replaced all clothes that were in its slots before.

Attributes

The `<clothing_slot>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
type	string	Specifies clothes slot type. Supported values: <ul style="list-style-type: none">• "headwear"• "hair"• "left_brow"• "right_brow"• "glasses"• "earswear"• "earrings"• "nosewear"• "mouthwear"• "beard"• "neckwear"• "top"• "hands"• "left_wrist"• "left_hand"• "l_little_finger"• "l_third_finger"• "l_middle_finger"• "l_index_finger"• "l_thumb_finger"• "right_wrist"• "right_hand"• "r_little_finger"• "r_third_finger"• "r_middle_finger"• "r_index_finger"• "r_thumb_finger"• "bag"• "belt"• "bottom"• "feet"• "right_ankle"	Required.

		• "left_ankle".	
--	--	-----------------	--

Related Elements

The `<clothing_slot>` element relates to the following elements:

Parent elements	<code><clothes></code>
Child Elements	None
Other	None

Examples

See `<clothes>` [examples](#)

3.2.4 attachment

Introduction

Declares an object that can be attached to the avatar.

Concepts

Attachment is an object that contains some mesh that can be attached to attachment points of the avatar. Potentially it can replace another attachment

Attributes

The `<attachment>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
name	name	Declares name for attachment, allowing referencing it from script.	Optional
path	url	Path link to COLLADA file with attachment model.	Required.
skeleton	string	Defines what avatars this attachment for. Can be: <ul style="list-style-type: none">• “male” – allows male avatars;• “female” – allows female avatars;• “all” – allows any avatar.	Optional. Default: all

Related Elements

The `< attachment>` element relates to the following elements:

Parent elements	<code><package></code>
Child Elements	See the following subsection
Other	None

Child Elements

List of allowed child elements:

Element	Description	Occurrences
<code><attachment_slot></code>	Defines to what slots that attachment are applied, it's relative position and orientation.	1 or more

Examples

See `<attachment>` [examples](#)

3.2.4.1 attachment_slot

Introduction

Defines to what slots an attachment object will be applied.

Concepts

Attachments are attached to some logical slots. When new attachment are attached to avatar it replaced all attachments that were in its slots before.

Attributes

The `<attachment_slot>` element has the following attributes:

Attribute	Type	Description	Requirement/Default
type	string	Specifies clothes slot type. Supported values: <ul style="list-style-type: none">• "headwear"• "hair"• "left_brow"• "right_brow"• "glasses"• "earswear"• "earrings"• "nosewear"• "mouthwear"• "beard"• "neckwear"• "top"• "hands"• "left_wrist"• "left_hand"• "l_little_finger"• "l_third_finger"• "l_middle_finger"• "l_index_finger"• "l_thumb_finger"• "right_wrist"• "right_hand"• "r_little_finger"• "r_third_finger"• "r_middle_finger"• "r_index_finger"• "r_thumb_finger"• "bag"• "belt"• "bottom"• "feet"• "right_ankle"	Required.

		• “left_ankle” .	
position	vector	Point which defines the attached object position relative to object’s origin.	Required.
direction	vector	Second point which defines the attached object direction relative to object’s origin.	Required.
yaw	float	yaw angle (facing direction).	Required.

Related Elements

The `<attachment_slot>` element relates to the following elements:

Parent elements	<code><attachment></code>
Child Elements	None
Other	None

Examples

See `<attachment>` [examples](#)

Chapter 4: Examples

4.1 Simple static model

```
<package caption="My Table" icon="table.jpg">
  <description>A black wooden Table</description>
  <object path="table.dae"/>
</package>
```

4.2 Objects with animation

4.2.1 Continuously self animated object

```
<package caption="My Fan" icon="fan.jpg">
  <description>A rotating fan</description>
  <object path="fan-with-rotation.dae">
    <action caption="On-load action" trigger="init">
      <object_animation path="fan-with-rotation.dae" repeat="0"/>
    </action>
  </object>
</package>
```

4.2.2 Self animated object, that is triggered by click

```
<package caption="My Talking Fish" icon="fish.jpg">
  <description>This fish moves when you click it</description>
  <object path="fish-with-animation.dae">
    <action caption="Click action" trigger="click" range="2" caption="Play">
      <object_animation path="fish-with-animation.dae" repeat="2"/>
    </action>
  </object>
</package>
```

4.2.3 Self animated object, that is triggered by approach

```
<package caption="My Alarm" icon="alarm.jpg">
  <description>If you get close to this object the alarm animation will be
  triggered</description>
  <object path="gate-with-animation.dae">
    <action caption="Approach action" trigger="approach" range="1.5">
      <object_animation path="gate-with-animation.dae" repeat="1"/>
    </action>
  </object>
```

```
</package>
```

4.3 Static objects with special properties

4.3.1 Radio

```
<package caption="My Radio" icon="radio.jpg">  
  <description>A virtual radio</description>  
  <object path="radio.dae">  
    <radio url="http://radio.com.ua:877/xxx.m3u" volume="50" type="2d"/>  
  </object>  
</package>
```

4.3.2 Light

```
<package caption="My Light" icon="light.jpg">  
  <description>A virtual Light</description>  
  <object path="light.dae">  
    <light type="spot" color="#00FF00" falloff_angle="180.0" radius="10.0"  
position="10.0 15.0 100" direction="1 0.5 0"/>  
  </object>  
</package>
```

4.3.3 Frame

```
<package caption="My Frame" icon="frame.jpg">  
  <description>A picture frame</description>  
  <object path="frame.dae">  
    <frame material="surface_material">  
      <frame_content type="picture"/>  
    </frame>  
  </object>  
</package>
```

4.3.4 Mirror with customizable border material

```
<package caption="My Mirror" icon="mirror.jpg">  
  <description>A Mirror with customizable border material</description>  
  <object path="frame.dae">  
    <frame material="surface_material">
```

```
        <frame_content type="mirror"/>
    </frame>
    <material_slot material="border_material" caption="Border Material"/>
</object>
</package>
```

4.3.5 Multi functional frame

```
<package caption="My Super Frame" icon="frame.jpg">
  <description>A frame with lots of possibilities</description>
  <object path="frame.dae">
    <frame material="surface_material">
      <frame_content type="picture"/>
      <frame_content type="mirror"/>
      <frame_content type="browser"/>
      <frame_content type="flash_video"/>
      <frame_content type="quicktime_video"/>
    </frame>
  </object>
</package>
```

4.4 Special objects

4.4.1 Clothes

4.4.1.1 Male Pants

```
<package caption="Pants" icon="pants.jpg">
  <description>My male Pants</description>
  <clothes path="pants.dae" skeleton="male">
    <clothing_slot type="bottom"/>
  </clothes>
</package>
```

4.4.1.2 Unisex wrist watch

```
<package caption="Watch" icon="watch.jpg">
  <description>My Watch</description>
  <clothes path="watch.dae">
    <clothing_slot type="right_wrist"/>
    <clothing_slot type="left_wrist"/>
  </clothes>
```

```
</package>
```

4.4.2 Attachment

```
<package caption="Gun" icon="gun.jpg">  
  <description>The AK-47 gun</description>  
  <attachment path="ak47.dae">  
    <attachment_slot type="right_hand" position="1 1 0" direction="1 0 0"  
yaw="30" />  
    <attachment_slot type="left_hand" position="1 1 0" direction="1 0 0"  
yaw="30" />  
  </attachment>  
</package>
```

4.4.3 Animations

4.4.3.1 Avatar animation

```
<package caption="Breakdance" icon="breakdance.jpg">  
  <description>Let's do the headspin</description>  
  <avatar_animation path="male_breakdance.dae" repeat="0" skeleton="male"/>  
  <avatar_animation path="female_breakdance.dae" repeat="0" skeleton="female"/>  
</package>
```

4.4.3.2 Avatar Animation with fancy parameters

```
<package caption="Cool walk" icon="walk.jpg">  
  <description>Cool walking animation</description>  
  <avatar_animation path="walk.dae" repeat="0" mixed="true" blend_in="200"  
blend_out="200" blend_mode="additive"/>  
</package>
```

4.4.4 Objects that trigger avatar animations

4.4.4.1 A dance pad

```
<package caption="My Dance Pad" icon="dancepad.jpg">  
  <description>Step on me to start dancing</description>  
  <object path="dancepad.dae">
```

```

        <action trigger="approach" range="3">
            <avatar_animation trigger="idle" path="male_dance.dae" repeat="0"
skeleton="male"/>
            <avatar_animation trigger="idle" path="female_dance.dae" repeat="0"
skeleton="female"/>
        </action>
    </object>
</package>

```

4.4.4.2 A chair with one seat point

```

<package caption="My Chair" icon="chair.jpg">
    <description>Click on this chair to sit down</description>
    <object path="chair.dae">
        <action trigger="click" range="3" caption="Sit down">
            <origin position="1 2 3" yaw="90"/>
            <avatar_animation trigger="begin" path="chair_sitdown.dae" repeat="1"/>
            <avatar_animation trigger="idle" path="chair_idle.dae" repeat="0"/>
            <avatar_animation trigger="end" path="chair_standup.dae" repeat="1"/>
        </action>
    </object>
</package>

```

4.4.4.3 A chair with different sit down animation for male and female skeleton

```

<package caption="My Chair" icon="chair.jpg">
    <description>Click on this chair to sit down</description>
    <object path="chair.dae">
        <action trigger="click" range="3" caption="Sit down">
            <origin position="1 2 3" yaw="90"/>
            <avatar_animation trigger="begin" path="male_chair_sitdown.dae"
repeat="1" skeleton="male"/>
            <avatar_animation trigger="begin" path="female_chair_sitdown.dae"
repeat="1" skeleton="female"/>
            <avatar_animation trigger="idle" path="chair_idle.dae" repeat="0"/>
            <avatar_animation trigger="end" path="chair_standup.dae" repeat="1"/>
        </action>
    </object>
</package>

```

4.4.4.4 A bench with two seat point

```

<package caption="My Bench" icon="bench.jpg">
    <description>Click on one of the seatpoints to sit on the bench</description>
    <object path="bench.dae">
        <action trigger="click" range="3" caption="Sit down">
            <origin position="1 2 3" yaw="90"/>

```

```

    <avatar_animation trigger="begin" path="chair_sitdown.dae" repeat="1"/>
    <avatar_animation trigger="idle" path="chair_idle.dae" repeat="0"/>
    <avatar_animation trigger="end" path="chair_standup.dae" repeat="1"/>
  </action>
  <action trigger="click" range="3" caption="Sit down">
    <origin position="1 3 4.5" yaw="90"/>
    <avatar_animation trigger="begin" path="chair_sitdown.dae" repeat="1"/>
    <avatar_animation trigger="idle" path="chair_idle.dae" repeat="0"/>
    <avatar_animation trigger="end" path="chair_standup.dae" repeat="1"/>
  </action>
</object>
</package>

```

4.4.4.5 A hug-couch

```

<package caption="My Hugcouch" icon="hugcouch.jpg">
  <description>Kiss your girlfriend on this couch</description>
  <object path="couch.dae">
    <action trigger="click" range="3" caption="Make out">
      <origin position="1 2 0" yaw="90"/>
      <avatar_animation trigger="begin" path="right_hugcouch_sitdown.dae"
repeat="1"/>
      <avatar_animation trigger="idle" path="right_hugcouch_wait.dae"
repeat="0"/>
      <avatar_animation trigger="sync" path="right_hugcouch_kiss.dae"
repeat="0"/>
      <avatar_animation trigger="end" path="right_hugcouch_standup.dae"
repeat="1"/>
    </action>
    <action trigger="click" range="3" caption="Make out">
      <origin position="1 3 0" yaw="270"/>
      <avatar_animation trigger="begin" path="left_hugcouch_sitdown.dae"
repeat="1"/>
      <avatar_animation trigger="idle" path="left_hugcouch_wait.dae"
repeat="0"/>
      <avatar_animation trigger="sync" path="left_hugcouch_kiss.dae"
repeat="0"/>
      <avatar_animation trigger="end" path="left_hugcouch_standup.dae"
repeat="1"/>
    </action>
  </object>
</package>

```