DFKI, EMBODIED AGENTS RESEARCH GROUP

EMBR Documentation

A Realtime Animation Engine For Interactive Embodied Agents

Alexis Heloir

5/5/2010

EMBR is a realtime character animation engine which offers a high degree of animation control via the EMBRScript language.

Table of Contents

Introduction 1

Who Should Read This Document 2

What you Will Learn from this Document 2

How to use This Document 2

How this Document is Organized 2

How to Contact the Author 2

Quickstart 3

Download 3

Installing 3

Installing on windows 3

Testing the sample scripts 5

The EMBRScript language 6

Syntax 6

Defining a `K_POSE_SEQUENCE` 7

Defining a `K_POSE_SEQUENCE` using a pre-recorded animation 7

Defining a `K_POSE_SEQUENCE` using a sequence of poses 7

Defining a pose 8

CONCLUSION 13

Introduction

Turning virtual humans into believable, and thus acceptable, communication partners requires highly natural verbal and nonverbal behavior. This problem can be seen from two sides: creating intelligent behavior (planning context-dependent messages) and producing corresponding surface realizations (speech, gesture, facial expression etc.). The former is usually considered an AI problem; the latter can be considered a computer graphics and animation problem (for nonverbal output). While previous embodied agents systems created their own solutions for transitioning from behavior planning to graphical realization, recent research has identified three fundamental layers of processing which facilitates the creation of generic software components: intent planner, behavior planner and surface realizer (see Fig. 1). This general architecture allows the implementation of various embodied agents realizers that can be used by the research community through a unified interface.

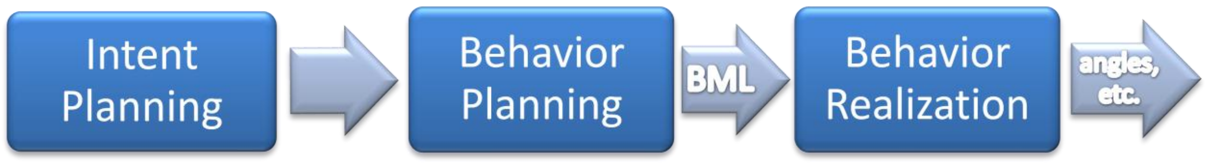


Figure 1: The SAIBA Framework defines three processing stages from intent planning to behavior realization

Although such a framework has the advantage of isolating different high level concerns apart from each other, it does not precisely explain when, where or how agents should finally be animated. The last Block "Behavior Realization" shall indeed deal which much more issues than solely animation: it should first plan the best animation corresponding to the input behavior described before figuring out "how to make the agent move". Furthermore, because no clean interface has been proposed between planning and animation, users have no possibility to fine tune the animation before it is displayed.

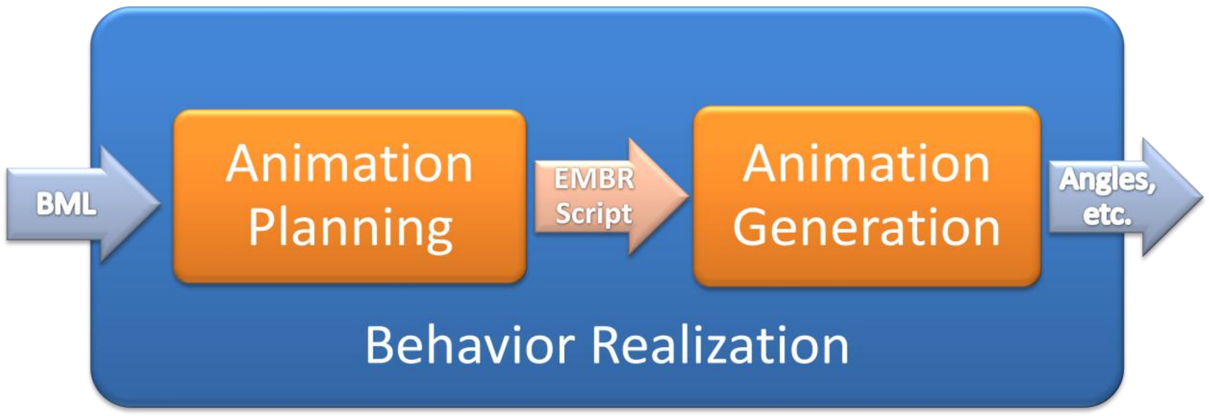


Figure 2: The behavior realization step can itself be decomposed into Animation Planning and Generation

To increase animation control while keeping high level behavior description simple, we specified and implemented an intermediate layer: the animation layer. This animation layer, thanks to the EMBRScript language, gives access to animation parameters that are close to common motion generation mechanisms like pre-recorded animation playback and key frame animation based on spatio-temporal constraints. Although EMBR has been introduced in the context of embodied conversational agents, we believe it would be very well suited for a broader range of application like video games, virtual puppetry and theater or even motion picture industry. This document describes how to set up the EMBR Character Animation Engine and ho to use its associated script language EMBRScript.

Who Should Read This Document

This document is intended for people who want/need to create quality character animation using a fine grained description language. Such people probably fall into four basic groups:

- people involved in the Research and development of Embodied Conversational Agents,
- students interested in computer animation,
- game developers,
- Animators looking for high level procedural motion generation.

What you Will Learn from this Document

This document is concerned with how to get/install and run the EMBR animation engine and how to control the EMBR animation engine using the EMBRScript language. It also explains how to send bug reports to authors because EMBR is still in heavy development phase.

How to use This Document

For the most part, you can work with this documentation in whatever order you choose. Each section addresses different areas of EMBR's functionality and so can be read independently from the rest of the document. In some case, reference is made to material covered previously, but this is in general not crucial to understanding the subject at hand.

How this Document is Organized

Have a look at the table of content.

How to Contact the Author

If you run into trouble at any point in reading this book, or if you have any insights or tips you would like to share, the first place we recommend to turn for quick responses and knowledgeable feedback is writing an Email to the main author: alexis.heloir@dfki.de.

Quickstart

Download

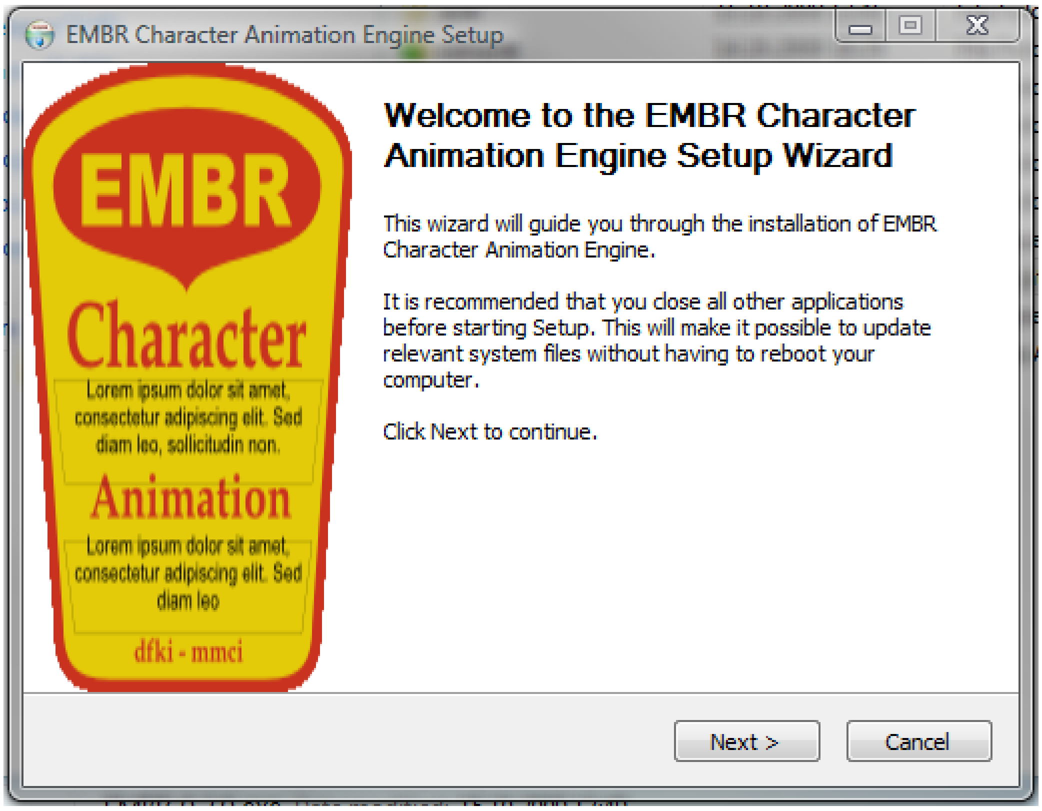
Windows users should download the windows installer from the EMBR website. Mac OSX and Linux users should wait a little bit.

This document corresponds to the version 0.5.2 of EMBR.

Installing

Installing on windows

After having downloaded the EMBR installer, double click on its icon, an installer window similar to the one depicted in Figure should appear:


EMBR Character Animation Engine Setup

Welcome to the EMBR Character Animation Engine Setup Wizard

This wizard will guide you through the installation of EMBR Character Animation Engine.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

EMBR
Character
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed diam leo, sollicitudin non.
Animation
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed diam leo
dfki - mmci
Next >
Cancel

Figure 3: EMR installation wizard

Read the software license agreement (LGPL) and, if you agree, click *"I agree"* choose the installation path (this could be for instance `C:\` of `C:\Program Files`). Click install, the installation process starts.

Once installed, EMBR installation wizard asks you if you want to run the EMBR engine. The first time you run EMBR, it may crash. Run it again. If it still crashes, one reason is that the current OpenGL context initialization chooses a pixel format that is not affordable on your machine. As a workaround, you can edit the Config.prc file in the `/etc` directory of your EMBR installation directory (`{$EMBR_DIR}/etc/Config.prc`) and add the following line at the beginning of the file:

```
gl-force-pixfmt 5
```

Another workaround is to ask for/buy a better graphic card/laptop!

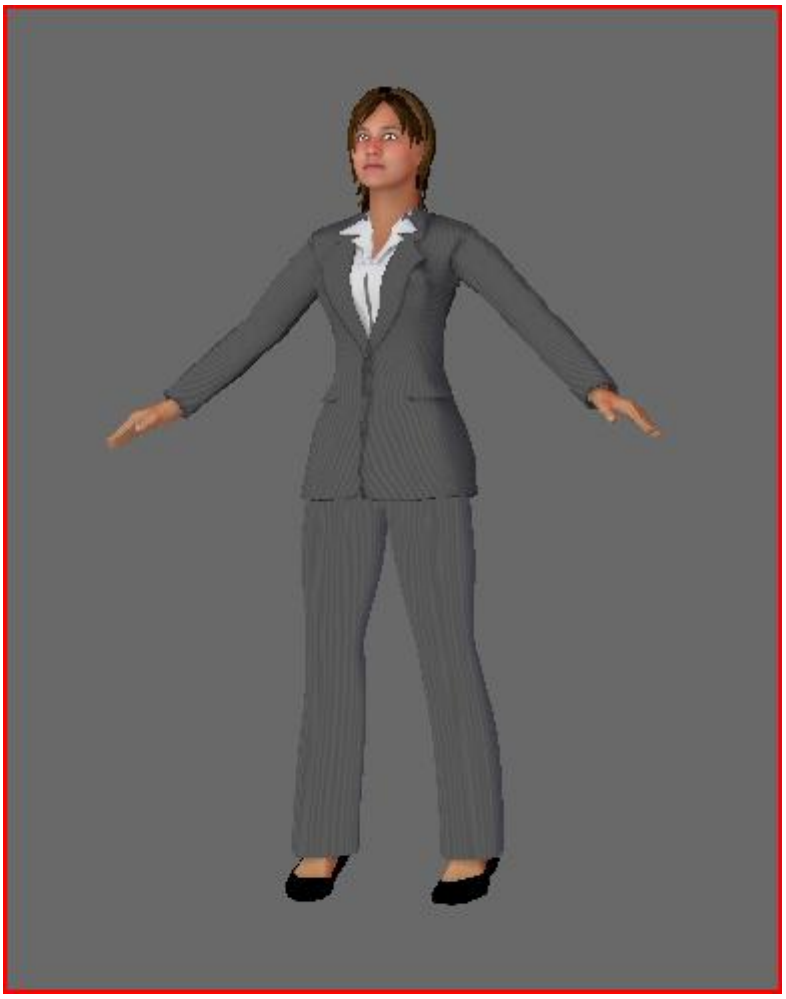


Figure 4: AMBER, the Default EMBR agent

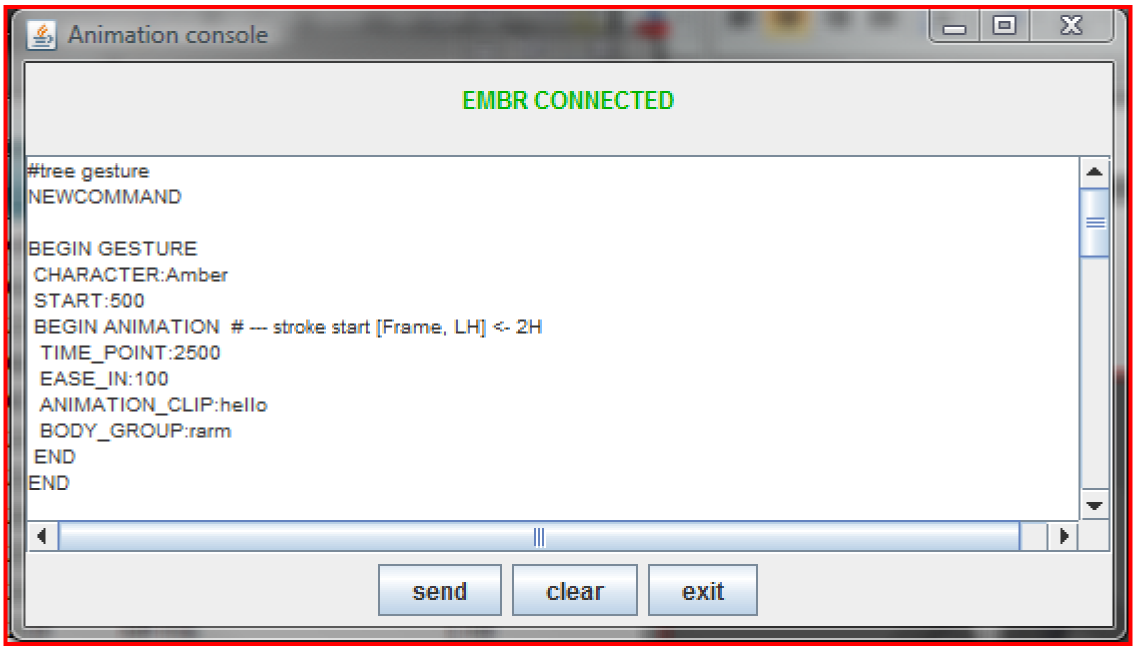
At this point, you should see a running EMBR environment featuring our current agent: Amber. Amber should be breathing at regular interval and convey some kind of ambient movement.

You should be able to move the camera around the scene using the mouse:

- by pressing mouse right button, you control zoom in/out,
- by pressing mouse middle button, you control the camera rotation around the scene
- by pressing mouse left button, you control the camera position

While loading the visualization window, EMBR starts a server listening on the tcp port number 5555. By sending EMBRScript commands to this port, you should be able to control the Agent AMBER.

Although we encourage you to write your own EMBR client, we still provide two minimal clients which let you send EMBR commands to a running EMBR server instance on the same computer. The first client, client.exe is a minimal console application which takes an EMBRScript file as argument. The second client (Figure 5) is a java application which lets you edit the EMBRScript sequence in an interactive frame before sending it.



```
#tree gesture
NEWCOMMAND

BEGIN GESTURE
 CHARACTER:Amber
 START:500
 BEGIN ANIMATION  # --- stroke start [Frame, LH] <- 2H
  TIME_POINT:2500
  EASE_IN:100
  ANIMATION_CLIP:hello
  BODY_GROUP:rarm
 END
END
```

Figure 5: the EMBR animation console GUI

Testing the sample scripts

At this point, you should be able to run the different sample scripts we included in the current distribution. Find them in the `gestures` folder in the `console` folder of the download package. Before running a script, make sure that you have a running EMBR server, and then run the Animation Console java application and copy-paste the content of the desired script into the application's GUI. You should see the agent performing the gestures described in the EMBRScript chunk you just sent. Following is a brief description of the scripts example script we provide:

- **tripleClap.embr:** a simple clapping gesture introducing the use of interruptive events using the "asap" timestamp,
- **testInverseKinematics.embr:** this script shows you how to define K-poses using kinematic constraints. It introduces position, orientation and swivel constraints, it shows also how to use the TIME_WARP element.
- **testMorphTargets.embr**: this script shows you how to define a K-pose using morph targets. You can use morph targets for facial expressions and lip-sync,
- **testAutonomousBehavior.embr**: this script shows you how to change the parameters exposed by an agent's autonomous behaviors. In this case you can vary breathing amplitude and breathing frequency.
- **testShader.embr**: this script shows you how to change the parameters exposed by an agent's programmable shaders. In this case, you can vary the amount of blushing of the agent.
- **tree.embr:** this script shows you how to specify complex gestures using multiple geometric constraints at the same time. The sign depicted in this script correspond to the standard sign meaning TREE in French Sign Language
- **wipe.embr:** simple wipe gesture defined using spatial constraints
- **index.embr:** "index raised up" gesture using a pre-stored hand configuration and spatial constraints
- **testRecordedPose.embr:** a succession of pre-recorded hand configurations, using recorded poses

- **circle.embr:** a circular gesture using spatial constraints
- **calm.embr:** a “calming down” emblematic gesture, using spatial constraints

The EMBRScript language

EMBRScript can be regarded as a thin wrapper around the animation engine with the following most important characteristics:

- Animations are specified using key-poses
- Key-poses are specified in absolute time
- Spatial constraints are specified in agent’s local space

EMBRScript's main principle is that every animation is described as a succession of key poses. A key pose describes the state of the character at a specific point in time (**`TIME_POINT`**), which can be held still for a period of time (**`HOLD`**). For animation, EMBR performs interpolation between neighboring poses. The user can select interpolation method and apply temporal modifiers. A pose can be specified using one of four principal methods: skeleton configuration (e.g. reaching for a point in space, bending forward), morph targets (e.g. smiling and blinking with one eye), shaders (e.g. blushing or paling) or autonomous behaviors (e.g. breathing).

Syntax

Here follows the simplest script you can send to EMBR, it actually does nothing:

```
TIME_RESET
BEGIN K_POSE_SEQUENCE
  TIME_WARP:EXP;1.0
  CHARACTER:Amber
  START:1000
  FADE_IN:2000
  FADE_OUT:2000
END K_POSE_SEQUENCE
```

`TIME_RESET`: reset EMBR’s internal timer: all subsequent time stamped commands sent to EMBR are specified according to the time of the last call to **`TIME_RESET`**.

`K_POSE_SEQUENCE`: represents an element describing an animation. Animation may be described using a sequence of poses or a pre-recorded animation (using a motion capture file for example) and are described in the following subsection.

`TIME_WARP`: EMBR supports time warp profiles that can be applied on any animation element and correspond to the curves depicted in Figure 2. Time warp profiles conveying ease in, ease out and ease it and ease out can be specified in the EMBR language with a combination of two parameters: function family and slope steepness.

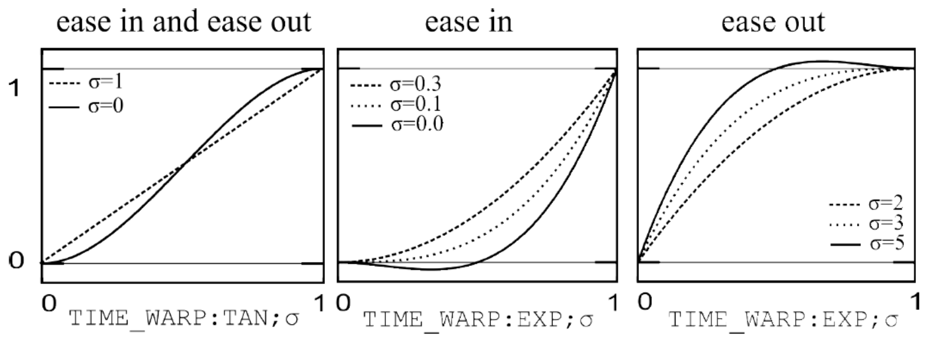


Figure 6: Time warp profiles can be used to model ease in, ease out and ease in and ease out.

CHARACTER: a **K_POSE_SEQUENCE** is defined character-wise, which means that several characters may be animated inside a single the EMBR instance.

START: The desired start time for the animation depicted in the **K-POSE_SEQUENCE**: Relative to the time of the last call to TIME**_RESET**, specified in milliseconds.

FADE_IN: linear ramp defining a progressive appearance of the animation described in the **K_POSE_SEQUENCE**. **FADE_IN** duration is specified in milliseconds.

FADE_OUT: linear ramp defining a progressive disappearance of the animation described in the **K_POSE_SEQUENCE**. **FADE_OUT** duration is specified in milliseconds.

Defining a K_POSE_SEQUENCE

There is two ways do define a **K_POSE_SEQUENCE**: either using a pre-recorded animation, either by defining a sequence of **K_POSE**'s which will be interpolated.

Defining a K_POSE_SEQUENCE using a pre-recorded animation

A **K_POSE_SEQUENCE** may contain a pre-recorded clip that is to be played on a part of the animation agent defined by **BODY_GROUP**:

```
BEGIN K_POSE_SEQUENCE
 CHARACTER:Amber
 START:1000
 FADE_IN:2000
 FADE_OUT:2000
 BEGIN ANIMATION  # --- stroke start [Frame, LH] <- 2H
  TIME_POINT:10000
  ANIMATION_CLIP:idle0
  BODY_GROUP:all
 END
END
```

TIME_POINT: desired duration for the recorded animation in milliseconds. The pre-recorded animation will be shrunk or extended to fit the **TIME_POINT** requirement.

ANIMATION_CLIP: unique identifier for the pre-recorded animation.

BODY_GROUP: although animations may be defined over the whole character, they may be played back only on subparts of the character. For instance, a motion captured walking motion capture over the whole body may only be applied on the lower part of the body. Amber can currently accept the following body groups:

- **head**
- **headNeck**
- **headThorax**
- **headAbdomen**
- **upperBodyNoArms**
- **lhand**
- **rhand**
- **larm**
- **rarm**

Defining a K_POSE_SEQUENCE using a sequence of poses

A **K_POSE SEQUENCE** may be described using a sequence of **K_POSE**'s. The resulting Animation will be obtained by interpolating between the specified **K-POSE**'s; a **K_POSE** describes the state of a **BODY_GROUP** at a certain point **TIME_POINT**.

TODO: give the possibility to change the interpolation method

```
TIME_RESET
BEGIN K_POSE_SEQUENCE
 BEGIN K-POSE
 TIME_POINT:1000
 END K-POSE
 BEGIN K-POSE
 TIME_POINT:2000
 END K-POSE
END
```

Defining a pose

There are two ways to define a pose: by recalling a pre-recorded pose, of by specifying it using a set of constraints.

Defining a pose by recalling a pre-recorded pose:

```
BEGIN K-POSE
  TIME_POINT:2000
  BEGIN POSE_TARGET
    BODY_GROUP:rhand
    POSE_KEY:hands_claw
  END
END
```

`POSE_TARGET:` unique identifier for the pre-recorded pose.

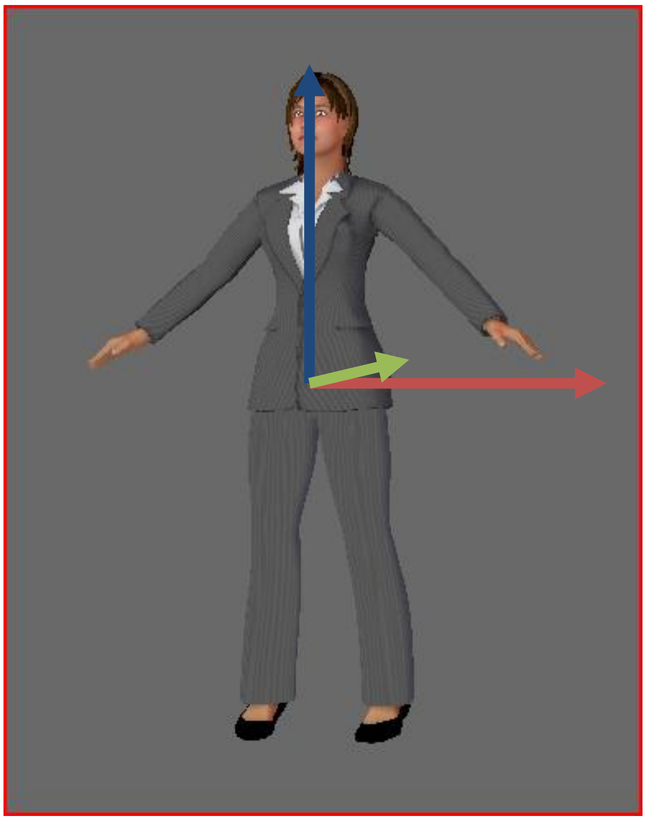
Amber can currently display the following poses:

- `hands_claw`
- `hands_fist`
- `hands_index`
- `hands_open-relaxed`
- `hands_open-spread`
- `hands_open-straight`
- `hands_purse`
- `hands_ring`
- `hands_thumbUp`

Defining a pose using kinematic constraints:

EMBR's space: units and agent coordinates:

Spatial landmarks in EMBR are defined according to the agent's local frame. The frame is Z-up, right-handed, Units are in meter and frame origin corresponds to agent's pelvis.



Position constraint:

```
BEGIN K_POSE   # +++ REST POSE +++
  TIME_POINT:1500
  HOLD:0
  BEGIN POSITION_CONSTRAINT
    BODY_GROUP:larm
    TARGET:0.5;0.0;0.4
    JOINT:lhand
    OFFSET:0.;0.0;0.
  END
```

`TARGET`: position of the kinematic target in agent's local space (meters)

`JOINT`: joint where the constraint should apply (joint should belong to BODY_GROUP)

`OFFSET`: desired offset (if the taking into account of the skin is needed for instance)

Orientation constraint:

```
BEGIN K_POSE  # position and orientation constraint
  TIME_POINT:7000
  HOLD:500
  BEGIN ORIENTATION_CONSTRAINT
    BODY_GROUP:larm
    NORMAL:Zaxis
    DIRECTION:1.0;0.5;0.0
    JOINT:lhand
  END
END
```

`DIRECTION`: direction of the orientation constraint

`JOINT`: joint where the constraint should apply (joint should belong to `BODY_GROUP`)

`NORMAL`: which local axis of the joint should be aligned with `DIRECTION`.

Swivel constraint (currently, only for right and left arm):

```
  BEGIN SWIVEL_CONSTRAINT
    BODY_GROUP:rarm
    SWIVEL_ANGLE:0
  END
```

`SWIVEL_ANGLE`: desired angle value for swivel

Look at constraint:

A `LOOK_AT_CONSTRAINT` only applies to the following groups:

- `head`
- `headNeck`
- `headThorax`
- `headAbdomen`

```
  BEGIN LOOK_AT_CONSTRAINT
    BODY_GROUP:headNeck
    TARGET:0.0;-0.5;0.8
  END
```

`TARGET`: where to look, position as a vector in 3D space

Defining a pose using other attributes:

In EMBR a pose can not only be described using skeletal element, but also using shader keys and morph target keys.

Shader parameters:

Modern 3d hardware gives users the possibility to modify their default rendering pipeline through the use of a shading language. Using shading language, a broad range of interesting effects can be achieved: A dedicated shader is for instance in charge of making Amber Blush. EMBR offers the possibility to modify some shader parameters that have been exposed through EMBRScript.

```
BEGIN K_POSE
  TIME_POINT:1000
  HOLD:0
  BEGIN SHADER
   SHADER_KEY:blushing
   SHADER_VALUE:1.0
  END
 END
```

SHADER_KEY: unique identifyier for the shader; Currently, Amber provides the following shader keys:

```
• blushing
```

SHADER_VALUE: decimal value between 0.0 and 1.0

MorphTarget parameters

The face is a highly important communication channel for embodied agents: Emotions can be displayed through frowning, smiling and other facial expressions, and changes in skin tone (blushing, paling) can indicate nervousness, excitement or fear. In EMBR, facial expressions are realized through morph target animation, blushing and paling are achieved through fragment-shader based animation. In EMBRScript, the MORPH TARGET label can be used to define a morph target pose and multiple morph targets can be combined using weights.

```
BEGIN K_POSE
  TIME_POINT:1000
  HOLD:0
  BEGIN MORPH_TARGET
   MORPH_KEY:ModBlinkRight
   MORPH_VALUE:1.0
  END
 END
```

MORPH_KEY: unique morph key identifier; currently, Amber accepts the following morph keys:

```
• ExpSmileClosed
• ExpAnger
• ExpDisgust
• ExpFear
• ExpSad
• ExpSurprise
• ExpSmileOpen
• ModBlinkLeft
• ModBlinkRight
• ModBrowDownLeft
• ModBrowDownRight
• ModBlinkRight
• ModBrowInRight
• ModBrowInLeft
• ModBrowUpLeft
• ModBrowUpRight
• ModLookDown
• ModLookLeft
• ModLookRight
• ModLookUp
• ModBlinkLeft
•   Phonaah
•   PhonB,M,P
•   Phonbigaah
•   Phonch,J,sh
•   PhonD,S,T
•   Phonee
•   Phoneh
•   PhonF,V
•   Phoni
•   PhonK
•   PhonN
•   Phonoh
•   Phonooh,Q
•   PhonR
•   Phonth
•   PhonW
```

`MORPH_VALUE:` decimal value between 0.0 and 1.0

Autonomous parameters:

Autonomous behaviors are very basic human behaviors that are beyond conscious control. Such autonomous behavior include breathing, eye blinking, the vestibulo-ocular reflex, eye saccades and smooth pursuit, balance control and weight shifting. Although we don't want to completely describe such behavior in EMBRScript, we still want to specify relevant parameters like breathing frequency or blinking probability.

```
BEGIN K_POSE
  TIME_POINT:6000
  HOLD:2000
  BEGIN AUTONOMOUS_BEHAVIOR
   BEHAVIOR_KEY:breathingFrequency
   BEHAVIOR_VALUE:1.2
  END
  BEGIN AUTONOMOUS_BEHAVIOR
   BEHAVIOR_KEY:breathingAmplitude
   BEHAVIOR_VALUE:1.0
  END
 END
```

`BEHAVIOR_KEY`: unique behavior key identifier; currently, Amber accepts the following henaviors:

`BEHAVIOR_VALUE`: decimal value between 0.0 and 1.0

```
•   breathing_frequency
•   breathing amplitude
```

`BEHAVIOR_VALUE:`

Currently, Amber provides the following behavior keys:

```
•   breathingFrequency
•   breathingAmplitude
```

CONCLUSION

At this point, you should know all you need to get started with the EMBR Real Time Character Animation Engine and its associated language EMBRScript. Because EMBR is still under heavy development, we would be very thankful to get feedback from your side. If you have any comment, question or found any bug, please consult EMBR's website: http://embots.dfki.de/EMBR.

Happy scripting!