M3X
Implementation notes

Jyrki Saarinen
jyrki.saarinen@ardites.com

November 13, 2008

1 Introduction

This document is meant mainly for the M3X project developers. Currently (November 13, 2008) it is briefly discussed how things have been implemented, and what is missing.

2 The m3x.m3g package

This package follows the class hierarchy specified in the M3G 1.0 specification.

`M3GObject`, `Section` classes and the abstract `Object3D` class are the most important pieces. All concrete M3G classes are derived from the `Object3D`, as it is modeled in the specification.

Between the `Section` and `Object3D` classes there is an `ObjectChunk` class 'layer'. The class serves as a data container only. The rationale between this design decision was that having the responsibilites of `ObjectChunk` in `Object3D` would have meant major amount of bookkeeping in `Object3D` class. Now the bookkeeping is done by the `java.io` streams instead of manual bookkeeping. Also this decision follows the M3G specification nicely.

All M3G classes implement the interface `M3GSerializable`. This interface specifies serialization and deserialization to and from streams. Concrete classes implement `M3GTypedObject` which is derived from `M3GSerializable`.

2.1 What is implemented

All M3G 1.0 classes are implemented following the specification. There are also JUnit test cases for every M3G class, and an end-to-end test (from M3G object to serialized form and back).

2.2 What is to be done

Everything should be done now November 13, 2008.

3 The m3x.translation package

3.1 What is implemented

Translation from M3X to M3G is implemented for all classes.

3.2 What is to be done

All M3X to M3G translation classes do not have a JUnit test cases. In the test classes test only the fields defined in the subclass. Testing inherited fields does not make sense in all translation test classes.

All M3G to M3X translation methods are missing from the translator classes (returning `null`).