M3X
Technical Design Document

Jacques Gasselin de Richebourg
jacques.gasselin@ironmonkeystudios.com

June 23, 2008

Abstract

The M3X project aims to provide tools for asset manipulation for the Mobile 3D Graphics API (M3G). M3X is being developed as a complement to Java Specification Request (JSR) #184 and #297.

JSR184 (M3G 1.0) and JSR297 (M3G 2.0) provide a solid interface for presenting 3D assets on Java capable devices supporting Connected Limited Device Configuration (CLDC) 1.1 .

A binary format for importing assets is clearly defined in the specification of both APIs. The proper ordering of references is defined as are the required binary data structures. A well defined approach to creating the binary asset is intentionally left out of the specification.

The M3X project supplies tools and a specification for creating binary assets that comply with M3G 1.0 and M3G 2.0. The intention is to remove a mechanical step from the asset pipeline; using an XML schema that presents a human readable interface to the M3G binary format.

XML files are easily edited by hand or using transformation scripts. Text format files also behave better than binary format files in most Version Control Systems. This is an important aspect for software development.

Version based change logs can be tracked. XML files can be easily validated to conform to a schema. Using the M3X tools; XML to binary conversion can be integrated into build pipelines.

Chapter 1

Application Programming Interface

The m3x API provides access to data conversion facilities through three packages. These three packages are each responsible for one form of input and output of data.

- The `m3x.xml` package uses the m3x XML Schema.
- The `m3x.m3g` package uses the M3G 1.0 and 2.0 binary format.
- The `m3x.translation` package provides application programming access.

The inclusion of each package is dictated by what each tool needs to do. Tools that want to convert between XML and binary formats need to use all the packages to achieve the conversion. Conversion from external data to XML or binary may only need two of the packages.

1.1 m3x.xml

The `m3x.xml` package use the Java API for XML Data Bindings (JAXB). The bindings are automatically created from the m3x XML Schema. Therefore an XML file that validates to the m3x Schema is valid input to this package.

The `ANT` build script uses the `xjc` JAXB implementation compiler to generate a package of classes that map the XML to runtime structures. JAXB handles serialization which removes an error prone part of the XML handling.

The element nodes in the `m3x` Schema share the same name as the corresponding class in the `M3G` API. `Group` maps to `<Group>` and so on. Because an XML Schema only allows inheritance for types and not elements; the naming convention of the classes in the `m3x.xml` package have a 'Type' suffix for most classes. `Objec3D` maps to `Object3DType` and so on.

In the `m3x.xml` package all `M3G` constants are handled by their string name, not their value as in the `M3G` API. Conversion between the two is the responsibility of the `m3x.translation` package.

Deserialisation

To deserialise an XML document one must create a JAXB `Unmarshaller` object. The following steps will obtain one using the m3x Schema:

1. Create a JAXB context.
   `JAXBContext context = JAXBContext.newInstance("m3x.xml");`
2. Create an Unmarshaller.
   `Unmarshaller unmarshaller = context.createUnmarshaller();`
3. Deserialise an XML document from an input stream, `i`.
   `m3x.xml.M3G root = (m3x.xml.M3G)unmarshaller.unmarshal(i);`

Summary

An XML data file conforming to the m3x XML Schema can be manipulated using the `m3x.xml` package.

1.2 m3x.m3g

Summary

A binary data file conforming either M3G 1.0 or M3G 2.0 can be manipulated using the `m3x.m3g` package.

1.3 m3x.translation

The `m3x.translation` package uses `m3x.xml` and `m3x.m3g` to translate between formats. The package can be used by a third-party application to create `m3x` or `m3g` content.

Basics

All translation classes implement the `Translator` interface. The interface defines the methods to create a translation object from an XML or binary class. It also defines the methods to create an XML or a binary class from a translation object.

`interface Translator`

- `void set(m3x.m3g.Object3D)`
  Sets the values from an M3G object.
- `void set(m3x.xml.Object3DType)`
  Sets the values from an XML object.
- `m3x.m3g.Object3D toM3G()`
  Gets a converted M3G object.
- `m3x.xml.Object3DType toXML()`
  Gets a converted XML object.

Each class in the `m3x.translation` package must able to be instantiated by using `Class.newInstance()`. That means that the objects created with the default constructor must be able to execute `toM3G()` and `toXML()`. It must be a complete object able to be converted without throwing any exceptions related to the state of the object. It may still be an object that is not valid in the target XML or M3G format.

Summary

External data can be converted to M3G or XML data using the translation layer in the `m3x.translation` package.