Author: J.F. Benckhuijsen (jfbenck@users.sourceforge.net)
Version: 1.0

Kiss System File IO

1. Introduction

The most important part of the whole MDK Application is the Kiss System, the lowlevel library to store the semantics of a system. Off course to allow a user to restore a project he's been working on, we'll have to save the Kiss System. As can be read in the File IO Design, XML has been chosen as the file format. This document describes the design of the loading and saving of these files and the format of the saved data

2. File Format

Describing the whole format for each of these classes is hardly interesting I presume. Load and save handling is mostly an internal affair of each classes, and each of these classes use the same pattern of implementation:

- Each direct property like strings, integers or enumerations are stored as properties of the main node;
- References to one of the five main classes (KissObject, KissSubject, KissAction, KissProcess or KissType) is stored as a string representing the name of these classes;
- Arrays are stored as multiple subnodes within the main node;
- Kiss classes stored in other classes (like associations in actions) are each stored as a subnode
- Inherited classes save the information of their base class in a subnode called <Base>

In practice, a saved KissAction will look like this:

```
<KissAction Comment=”.....” Type=”1”>
          <Base>
                    <Base Name=”DoSomething”>
                    </Base>
                    <Attribute ...>
                              // Attribute data goes here
                    </Attribute>
                    <Attribute ...>
                              // Attribute data goes here
                    </Attribute>
          </Base>
          <Association ...>
                    // Association data goes here
          </Association>
          <Association ...>
                    // Association data goes here
          </Association>
</KissAction>
```

3. Implementation

In general, subclasses of each class in loaded in sequential order. However, in case of the shared classes: KissAction, KissObject, KissSubject, KissProcess and KissTypes (and internally Association), this approach can fail.

References to shared objects are found using the TKissXMLNode class, which can be asked for one of these shared objects (internally it just holds a pointer to the KissSystem, which it queries for the appropiate object by name). It may happen a shared object is queried, which isn't loaded yet.

To solve this problem, an attempt is made to load a class. Upon succes, the next class in the list is simply loaded and the loaded class is removed from the list. Upon failure, the class is skipped and we move on to the next class. At the end of the list we check wheter the list is empty. If not we 'll traverse the list again, until the list is empty. Because many types of objects are shared, we traverse each of these lists once, before we try to traverse them all from the beginning.

To prevent a lot of faillure, we try to load the most often shared class (KissObject) first and the least shared class (KissSubject) last.

Besides the ability to query for references to shared objects, TKissXMLNode also offers some methods to store all kinds of datatypes, so the load and save methods don't have to bother for conversion.