07-03-2012

Technical Paper

The Shadow Framework 2.0: Data

Alessandro Martinelli

Draft

Abstract

Data management is an important task in the Framework. Through an abstract data layer, any module of the framework can be stored in datafile or transfered to datastream and loaded at any time.

Basic data Concepts

The main module of the data package is **SFDataObject**. **SFDataObject** is an interface for any object which contains data:

- it can be **written** on an **SFOutputStream**;
- it can be **loaded** (read) from an **SFInputStream**;
- It is a based on the ***Composite Pattern:*** it may be a simple **dataObject**, or a composite one with more **son-Objects**. On Composite ones it is assumed that son-Objects are stored and loaded from the *writeOnStream* and *readFromStream* method of the object itself.

Of course, the other important interfaces are **SFInputStream** and **SFOutputStream**. This interfaces are somehow duals. Whatever SFInputStream read, SFOutputStream writes. Implementations must both have a method to read/write floats, ints, bytes and Strings. When an **SFInputStream** is chosen, **SFOutputStream** must be taken to support the same representation of each primitive data. *The framework will ignore the effective data format which is used to write primitive data into a file or upon a stream, and to retrieve them after*. Note that this may even be exploited to build data conversion routines.

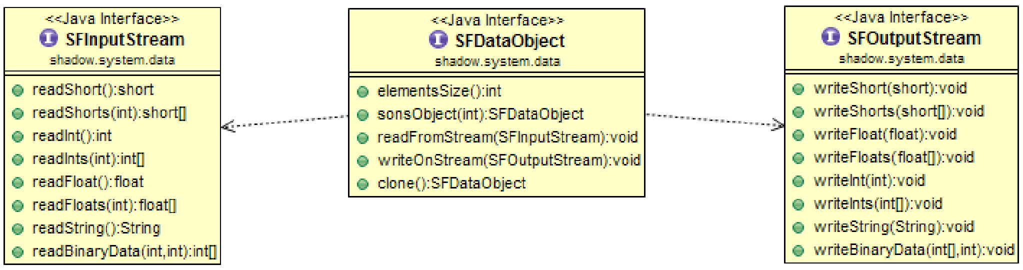

Fig. 1 The basic interfaces in shadow.system.data

Example 1. Storing some data into an outputStream (Java Code)
SFString *string*=new **SFString**("Hello");
SFInt *number*=new **SFInt**(23);
string.writeOnStream(*outputStream*);

```
number.writeOnStream(outputStream);
```

Example 2. Retrieving some data from an inputStream (Java Code)

```
SFString string=new SFString();
SFInt number=new SFInt();
string.readFromStream(inputStream);
number.readFromStream (inputStream);
```

Within The framework there are two implementation of SFInputStream and SFOutputStream: a ***Java oriented*** and a ***Web oriented***.
The Java oriented implementation can be found in ***shadow.system.data.java*** and has two important characteristics:

- its based on standard java OutputStream and InputStream; so it may be used to save/load data to/from a file as to manage data through any other stream.
- all data is stored in a compact binary format.

The other implementation, the ***Web oriented***, can be found in **shadow.system.data.js** and it's based on String representation of every primitive data.

NOTE
The main version of the ShadowFramework is developed with Java Language. For java programmers, it is a common solution to exploit *Reflection* and *Serialization* to store object's data into streams. The reason for which we avoid that here is that this mechanism is strictly java based, while the SF is intended to be cross-language.

Data Objects

Each SF component will use Data Objects to encapsulate its data, so that data can be read or written using streams. The most used Data Object to encapsulate a class data is **SFCompositeDataArray**, because its behavior higly resemble the concept of structure in C (or of attributes-set in Object Oriented Programming)

Simple data Objects

- **SFPrimitiveType** : a generic **SFDataObject** , base class for Data Objects with no subObjects.
- **SFFloat** : a SFDataObject containing a float value.
- **SFInt** : a SFDataObject containing an int value.
- **SFShort** : a SFDataObject containing a short value.
- **SFString** : a SFDataObject containing a String.
- **SFFloatArray** : a data Object containing an Array of floats. The array has a fixed number of items
- **SFIntArray**: a data Object containing an Array of ints. The array has a fixed number of items
- **SFShortArray**: a data Object containing an Array of shorts. The array has a fixed number of items
- **SFVectorData** : like SFFloatArray, but cannot be instantiated due to a protected constructor. Base class for constant size array like vertexes.

- **SFVertex2fData** : a vector with 2 components, data for a SFVertex2f
- **SFVertex3fData** : a vector with 3 components, data for a SFVertex3f
- **SFVertex4fData** : a vector with 4 components, data for a SFVertex4f
- **SFVoidData**: a Data Object containing no data.

Composite Data Objects

- **SFCompositeDataArray** : a Data Object containing a list of DataObjects; data Object beholding to the list are defined during construction and cannot be changed at a second time.
- **SFDataList** : a Data Object which is a List of Data Object. The method `public ArrayList<T> getDataObject()` allows to extract the list of data objects, which can be inserted or removed at any time.
- **SFBinaryDataList** : a list of binary values of the same size; a binary value is a positive integer value in the range **$[0,2^{size}-1]$** where size is an assigned bitsize. Since all binary values within the same list have the same size, size should be consider a list property rather than a value property.

SFDataset

A **Dataset** is an object which keeps the knowledge of a **DataObject** and of its own **type**. **Type** is represented with a **String**. Type should always be a class property, that is a String which identify the class itself. Most of **Datasets** will use their class name as type.
A Collection of **Datasets** can be stored and retrieved from a file. The basic procedure for data storing will be something like this:

```
outputStream.writeInt(datasetsCollection.getSize());
for each ( dataset  in datasetsCollection )
        outputStream.writeString(dataset.getType());
        dataset.getSFDataObject().writeOnStream(outputStream);
```

The basic procedure to restore a collection of **Datasets** from an inputStream will require an addictional module to generate proper instances for each type, that is some kind of Dataset Factory. Such procedure will work like this:

```
int n=inputStream.readInt();
for(i in [0,N-1])
        String type = inputStream.readString();
        SFDataset dataset = datasetFactory.createDataset(type);
        dataset.getSfDataObject().readFromStream(inputStream);
        datasetCollection.add(dataset);
```

For this reason the from define an abstraction called SFAbstractDatasetFactory with the simple method:

```
public SFDataset createDataset(String typeName);
```

which should be used as already shown.

Dataset can be encapsulated into DataObject using the **SFDatasetObject**.

SFDataCenter

DataCenter is the central node for data management. It is a singleton, so each instance of the SF running on any device will have only 1 **DataCenter**. SF content viewers will always access its unique **DataCenter** to retrieve **Datasets**. **DataCenter** provides an important abstraction on how data are effectively retrieved; in order to make a DataCenter work, it must be provide an implementation for:

- A **SFAbstractDatasetFactory**.
- A **SFIDataCenter**.

The Interface Data Center provide an abstraction on a ***Map of Datasets*** which are identified through their name. With IdataCenter we can ask for a Dataset to be retrieved, if we need it, or to be released, if we don't need anymore. **DataCenter** will not immediately return a required dataset, but will send them to any implementation of **SFDataCenterListener** when ready.

SFObjectsLibrary

SFDataCenterListener will mostly be like a component using **ObjectsLibraries**. **SFObjectsLibrary** is used to collect a set of **Dataset** where each **Dataset** is assigned a name. **ObjectsLibraries** are **Datasets** themselves, so they can be nested into other **ObjectsLibraries**. In order to insert an element into the library you simply need to put it into the library:

```
public void put(String name,SFDataset dataset)
```

SFLibraryReference

Another important element which is part of the **DataCenter** Architecture is the **LibraryReference**. A **LibraryReference** is a **DataObject**, which can be used by components which have a reference to other **Datasets** stored into a library. The **LibraryReference** simply stores the **Dataset** name in the library, but it can evenctually act as a **SFDataCenterListener** to the static **DataCenter** in order to retrieve (`public SFDataset retrieveDataset()`) the dataset element.