How to construct a MMFG

The Multimedia Feature Graph (MMFG) is the central data structure for Smart Multimedia Information Retrieval. As it is a graph, its structure can vary according to the features it contains and the purpose of its construction.

In its simplest form, a MMFG is created with

```
MMFG mmfg = new MMFG();
```

However, when using MMFGs within the Generic Multimedia Analysis Framework, the framework will create and prepare such MMFGs for the developer.

The most important methods of a MMFG are shown here:

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	addLocation(de.swa.mmfg.Location loc)	add a location of the represented multimedia object
void	addNode(de.swa.mmfg.Node n)	adds a node
void	addTimeRange(de.swa.mmfg.Timerange ta)	returns all time ranges where the MMFG has features from
void	addToCollection(MMFG m)	adds a sub-mmfg to the mmfg
java.util.Vector<de.swa.mmfg.Node>	getAllNodes()	returns the complete lost of nodes
de.swa.mmfg.Timerange	getBegin()	returns the smallest time range of a feature
java.util.Vector<MMFG>	getCollectionElements()	returns the sub-mmfgs
de.swa.mmfg.Node	getCurrentNode()	returns the current node, which is processed
de.swa.mmfg.Timerange	getEnd()	returns the greates time range of a feature
de.swa.mmfg.GeneralMetadata	getGeneralMetadata()	returns the general metadata object
java.util.UUID	getId()	returns the UUID of the MMFG
java.util.Vector<de.swa.mmfg.Location>	getLocations()	returns the location of the represented multimedia object
java.util.Vector<de.swa.mmfg.Node>	getNodes()	returns a list of all nodes
java.util.Vector<de.swa.mmfg.Node>	getNodesByTerm(java.lang.String term)	returns all nodes with a given term
de.swa.mmfg.Timerange	getOrCreateTimerangeAt(java.util.Date d)	returns a new or existing time range object at a specific date
de.swa.mmfg.Security	getSecurity()	returns the security information object

In the simplest form, a Node (i.e. the representation of a multimedia feature) is added to the MMFG:

```
Node n = new Node();

n.setName("color");

n.setValue("red");

mmfg.addNode(n);
```

Convenience constructors are available. The same result can be produced by

```
Node n = new Node("color", "red", mmfg);
```

Nodes can contain child nodes and thus span subgraphs:

```
Node child = new Node("color-detail", "RGB(1,2,3)");

n.addChildNode(child);
```

And nodes can have relationships to various other objects. Amongst these are

- TechnicalAttributes describing the metadata of the multimedia recording
- SemanticRelationships connecting a node to semantic systems, like e.g. Wikidata
- CompositionRelationships describing spacial information like (“ABOVE”, “BEHIND”, etc.)
- Timeranges which indicate the time, where the node is relevant within the object
- AssetLinks linking to additional resources

Partucularly, the Timerange object is important for the representation of realtime or streaming multimedia objects.

```
Timerange startOfShow = new Date();

// wait until the show is ended

Timerange endOfShow = new Date();

Node speaker = new Node("TV-Host", "Mike", mmfg, new Timerange(startOfShow, endOfShow));
```

MMFGs automatically maintain features and their given time ranges for later querying.

In general it is recommended to keep MMFGs flat. This means, every scene should be represented by a Node and what is in the scene should be directly attached to the scene-node. Although deeper structures are possible, flat structures have better performance. For a video, you would have

```
MMFG
```

- Scene Node 1
  - Scene Feature Description 1.1
  - Scene Feature Description 1.2
  - ...
- Scene Node 2
  - Scene Feature Description 2.1
  - Scene Feature Description 2.2
  - ...

An example from the Shazam-Plugin, which extracts music information from a given audio track shows this in more detail:

```
Node n = new Node("Music", mmfg);

n.addChildNode(new Node("Title", title, mmfg));

n.addChildNode(new Node("Artist", artist, mmfg));

n.addChildNode(new Node("Link", link, mmfg));

n.addChildNode(new Node("Writers", writers, mmfg));

n.addChildNode(new Node("Image", image, mmfg));

n.addChildNode(new Node("Label", label, mmfg));

n.addChildNode(new Node("Released", released, mmfg));

n.addChildNode(new Node("Album", album, mmfg));

n.addChildNode(new Node("Isrc", isrc, mmfg));

n.addChildNode(new Node("Genre", genre, mmfg));

n.addChildNode(new Node("Lyrics", lyrics, mmfg));

n.addChildNode(new Node("Preview", image, mmfg));

mmfg.addNode(n);
```

If the music would be playing right now, you might want to add

```
n.setTimerange(new Timerange(new Date(), new Date());
```

The corresponding MMFG then would provide the overall time range information

```
Timerange overallStart = mmfg.getBegin();

Timerange overallEnd = mmfg.getEnd();
```

For the representation of textual information, it is recommended to at least maintain the document structure by representing the extracted features of a paragraph. However, it is also possible to represent sentences of each paragraph by a sub-structure. This depends on the selected algorithm for processing.

```
MMFG
```

- Paragraph 1
  - Terms, Sentiments, Topics of Paragraph 1
- Paragraph 2
  - Terms, Sentiments, Topics of Paragraph 2
- Paragraph 3
  - Sentence 1
    - Terms, Sentiments, Topics of Paragraph 3, Sentence 1
  - Sentence 2
    - Terms, Sentiments, Topics of Paragraph 3, Sentence 2

MMFGs can be exported and imported to numerous formats. To achieve this, a flatten / unflatten utility is provided based on two interfaces:

`public interface Flattener`

interface to represent flatteners for exporting MMFGs into various formats

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
`java.lang.String`	`endFile()`	writes some footer (if required)
`java.lang.String`	`flatten(MMFG fv)`	this method is called, when a MMFG should be exported
`java.lang.String`	`getFileExtension()`	returns the file extension for this flattening process
`java.lang.String`	`startFile()`	writes some header (if required)

`public interface Unflattener`

interface to be used for unflattening (i.e. importing) MMFGs based on a String

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
`MMFG`	`unflatten(java.lang.String s)`	returns the MMFG contained in the string

For example, to export a MMFG as XML, you can simply use the XMLEncodeDecode class, which implements both interfaces:

```
XMLEncodeDecode xml = new XMLEncodeDecode();

String xmlString = xml.flatten(mmfg);

System.out.println(xmlString);

MMFG otherMmfg = xml.decode(xmlString);
```

Further exports are

- Json (de.swa.mmfg.builder.JsonFlattener)
- Mpeg7 (de.swa.mmfg.builder.Mpeg7IO)
- GraphML (de.swa.mmfg.builder.GraphMLFlattener)
- HTML (de.swa.mmfg.builder.HTMLFlattener)
- Neo4J (de.swa.mmfg.builder.Neo4JFlattener)
- RDF (de.swa.mmfg.builder.RDFFlattener)

Further imports are

- Mpeg7 (de.swa.mmfg.builder.Mpeg7IO)