The Mumie
MathletFactory Tutorials

—

Creating Mathlets

Contents

1 First steps

1.1 Setting up the environment

For compiling and running java applets you need the Java Software Development Kit (JDK/SDK) with the version 1.4.2 or newer.
For viewing applets in your internet browser, you need a Java-plugin.
Both can be obtained for free from `http://java.sun.com`.
The Java APIDOC, the Java tutorial and the following extensions are also available there:

- Java3D for 3-dimensional graphics
- JavaMediaFramework (JMF) for screen capture support

You have to put the MathletFactory library (JAR-files) into the Java classpath where the compiler and the runtime will search for the classes (see your Java documentation for information about setting the classpath).

The use of a professional Integrated Development Editor (IDE) instead of a simple text editor is strongly encouraged. Here you can include the APIDOC and the sources into the editor for an easier programming.

1.2 Applet Skeleton Framework

With providing a flexible and powerful "skeleton" framework for applets, the developer has neither to "reinvent the wheel again", nor to transfer whole passages of code for new programs. Instead, basing on an applet skeleton, mathlets have a generic interface for both the code itself and the placement of code. This simplifies the understanding of mathlet code (even if it is not the "own" code) and thus reduces developing and debugging time.

1.2.1 Structure of a mathlet

The class `BaseApplet` is the base for all mathlets. It defines a common design where the components are placed into predefined panels/panes (containers able to hold GUI-components). The 3 main panels of a mathlet are (read from top to bottom):

- title pane containing a text panel with the mathlet's title
- center pane containing the canvases and/or a control-panel
- button pane containing the help, reset, screenshot and animation buttons

The center pane itself is divided into (from top to bottom)

- canvas pane containing a single canvas or an arrangement of several canvases
- controls pane pane containing a `ControlPanel`

1.2.2 The choice of a mathlet type

The mathlet-class needs to extend one of the template classes below. There are mathlet template types for the most common needs. The main distinction of theses templates is made by the number, the arrangement and the dimension of their canvases.
The following template types are available:

- **"No-Canvas"** – only 1 ControlPanel

  $$NoCanvasApplet^1$$

- **"Single"** – 1 single canvas and 1 ControlPanel below

  $$2D : SingleG2DCanvasApplet^2, 3D : SingleJ3DCanvasApplet^3$$

- **"Side-By-Side"** – 2 canvases arranged horizontally and 1 ControlPanel below

  $$2D : SideBySideG2DCanvasApplet^2, 3D : SideBySideJ3DCanvasApplet^3$$

- **"Upper-Lower"** – 2 canvases arranged vertically and 1 ControlPanel below

  $$2D : UpperLowerG2DCanvasApplet^2$$

- **"Upper-Middle-Lower"** – 3 canvases arranged vertically and 1 ControlPanel below

  $$2D : UpperMiddleLowerG2DCanvasApplet^2$$

The ControlPanel resides below the canvases, except for the `NoCanvasApplet`: there it covers the whole mathlet's space.

[1]net.mumie.mathletfactory.appletskeleton
[2]net.mumie.mathletfactory.appletskeleton.g2d
[3]net.mumie.mathletfactory.appletskeleton.j3d

1.2.3 Creating the applet class

The ”entry point” of every applet is the `init()` method which will be called by the web-browser to load and initialize the applet.
This is also true for a mathlet and its runtime: a call to the `init()` method of the super class is necessary for initializing the system.
An empty mathlet can look like this (e.g. for a ”Single-Canvas”-applet):

```
import net.mumie.mathletfactory.appletskeleton.g2d.*;

public class MyApplet extends SingleG2DCanvasApplet {
   public void init() {
      super.init() // needed to initialize the runtime
      ...  // your entry point
   }
}
```

Note that the call `super.init()` is needed to initialize the mathlet's runtime.

1.3 Basic User Interface

Every mathlet shares the same basic user interface with the following common features.

1.3.1 Title

The title is shown at top of the mathlet, i.e. above the canvases and the control panel. It can be set with

```
setTitle(String)
```

defined in the `BaseApplet` class.

1.3.2 Reset Button

A generic reset button can be used to re-initialize a scene.
The button itself can be added by simply calling `addResetButton()` inside an extended BaseApplet-class. By pressing the button, the method `public void reset()` will be called in the mathlet. To react on this event you will have to overwrite this method in the applet-class.

1.3.3 Dynamic Reset Button

1.3.4 Screenshot Button

A generic screenshot button can be used to create screenshots and videos by capturing graphical scenes are the complete mathlet.
The button will be visible when calling `addScreenShotButton()` inside an extended BaseApplet-class.

1.3.5 Using Tabs

It is possible to open a new tab in the control-panel-, canvas- and center-pane. This is done by calling `add(String, Component)` on the following getters:

- getCanvasTabbedPane()
- getCenterTabbedPane()
- getControlTabbedPane()

These methods return a customized instance of a JTabbedPane[4] (an instance of `TabbedPanel` more precisly) which will show the real tab (with the tab's title) only when more than one tab has been added.
Note: all of the BaseApplet's tabbed panes have allready a tab (containg the canvases, etc.). But the tab's titles are not visible because they are the only tabs!

The add-method of these tabbed panes request a string for the tab's title and an instance of Component[5] which can be e.g. a new ControlPanel, Canvas or JPanel (they all extends the class `Component`).
The title of the first (default) tab can be set through the `TabbedPanel`-proper method `setTitleAt(int, String)`.
The instance of this underlying SWING-class can be accessed through `getTabbedPane()` in the class `TabbedPanel`.

1.4 Error Handling

Every action inside a mathlet can use the internal error handling system to provide a *quality feedback* of unexpected errors during the execution.
In order to *catch* an error, the following block must surround an action inside a mathlet:

```
try {
   ...
} catch(Throwable t) {
   reportError(t);
}
```

The only actions during execution are the applet instance methods and actions inside the user interface (i.e. *events*).

[4]javax.swing.JTabbedPane
[5]java.awt.Component

2 MM-Objects

MM-Objects (Multimedial Mathematical Objects) are the primary entities with which an application programmer of the MathletFactory has to deal with. The MathletFactory tutorial gives an overview of the framework that provides the simple use of MM-Objects . This can be summarized in one sentence: MM-Objects encapsulate the mathematical and interactivity state and are mapped onto graphic primitives (so called *drawables*) and panel representation (so called *MMPanels*) by the use of specialised *transformers*. The figure below shows the common role allocation in such a *Modell-View-Controller architecture*.

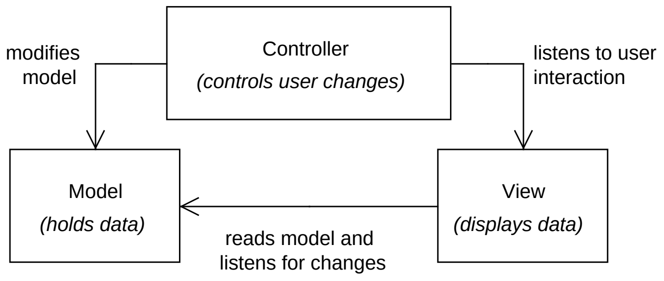
modifies model
Controller
(controls user changes)
listens to user interaction
Model
(holds data)
View
(displays data)
reads model and listens for changes

Figure 1: Common role allocation in a MVC architecture

An MM-Object is either a super class of a mathematical class (located in `net.mumie.mathletfactory.math`) implementing the `MMObjectIF` interface or an extension of the class `MMDefaultCanvasObject` or `DefaultMMObject`, implementing itself this interface and its necessary methods.
Note:

- – every MM-Object 's name starts with the two capital letters "MM"
- – every MM-Object is located in the tree branch `net.mumie.mathletfactory.mmobject`

2.1 Displaying MMObjects

A single MM-Object can be displayed in multiple instances of container- and canvas drawables. (a "Container" means a box where GUI-elements such as buttons, labels and textfields can be added and displayed)(e.g. the `ControlPanel` or a simple `JPanel`).
Thereby it is possible to display the MM-Object 's content for different transform types in separate drawables and even in multiple drawable instances for the same mathematical entity.

2.1.1 Representing content in Containers

Each time one of the methods

- getAsContainerContent()
- getAsContainerContent(int)

is called, a new container drawable instance will be created and returned. The first method returns the default container drawable, the second returns the drawable designated by the transform type. It is necessary to store it in a variable because these methods will never return this same instance again. The returned component can then by casted to the drawables runtime class to gain access to class specific functionalities.

2.1.2 Representing content in Canvases

Each time one of the methods

- getAsCanvasContent()
- getAsCanvasContent(int)

is called, a new container drawable instance will be created and returned. The first method returns the default container drawable, the second returns the drawable designated by the transform type. It is necessary to store it in a variable because these methods will never return this same instance again. The returned component can then by casted to the drawables runtime class to gain access to class-specific functionality.

2.2 Display Properties

Each MM-Object has its own display properties which define appearance related settings such as colors, fonts or transparency. They can be returned and set with

- `getDisplayProperties()`
- `setDisplayProperties(DisplayProperties)`

Both methods are defined in the interface `MMObjectIF`.
Moreover some MM-classes have extended properties where special settings are possible. By casting to the `DisplayProperties`-runtime class they offer additional methods to change other useful settings such as the line width, point radius or the basic shape of a drawable:

```
MMAffine2DPoint p1;
// ... initializing the point ...
PointDisplayProperties pdp = new PointDisplayProperties(p1.getDisplayProperties());
pdp.setPointRadius(10);
```

Note that no copying of the properties of "p1" is actually done but a referencing of the internal property maps. It is possible to call `clone()` on an instance of `DisplayProperties` to copy the settings into a new independant instance:
`DisplayProperties dp2 = (DisplayProperties) dp1.clone();`

The following table shows the available DisplayProperties in MM-Objects and Drawables:

Properties-Class	implementing MM-Class	using Drawable

`LineDisplayProperties`	`MMAffine2DLine`
	`MMAffine2DLineSegment`
	`MMAffine2DRay`
	`MMAffine3DLine`
	`MMAffine3DLineSegment`
	`MMCoordinateSystem`
	`MMDefaultRNVector`
	`MMVectorField2DOverR2-`
	`DefByComponents`
	`MMVectorField2DOverR2-`
	`DefByExpression`
		`G2DLineDrawable`
		`J3DLineSegmentDrawable`
		`J3DPolyLineDrawable`

`PointDisplayProperties`	`MMAffine2DPoint`
	`MMAffine3DPoint`
	`MMDefaultRN`
	`MMBezierPolynomialAdvanced`
		`G2DPointDrawable`
		`J3DPointDrawable`

`PolygonDisplayProperties`	`MMAffine2DPolygon`
	`MMFunctionDefByOp`
	`MMFunctionDefinedByExpression`
	`MMFunctionDefinedBySamples`
	`MMPiecewiseFunction`
	`MMStepFunction`
	`MMBezierPolynomialAdvanced`
	`MMPolynomial`
	`MMParametricFunctionInR2`
	`MMOneChainInR2`
		`G2DPolygonDrawable`

`SurfaceDisplayProperties`	`MMFunctionOverR2`
	`MMParametricFunctionInR3`
	`MMAffine3DPlane`
	`MMAffine3DSubspace`
		(no explicit drawable)

2.3 Number class

Most MM-Objects (or maybe their underlying mathematical classes) are dependant of a number type, means calculations are made through the number-class-proper arithmetic operations/methods.
All MM-classes need this number class as parameter for their constructors for initializing their internal fields with numbers of this class. A change of the class outside the constructor (i.e. after initializing) is generally not possible.
The available number classes are (located in `net.mumie.mathletfactory.math.number`): `MDouble, MComplex, MRational, MComplexRational, MBigRational, MInteger, MNatural, MRealNumber, Zmod5.`

2.4 Rendering cycle

Each rendering cycle is started either by a canvas, by the mathlet author (calling a manual repaint on the MM-Object with the `render()` method) or by a handler (calling itself the `render()` method).
The MM-Object forwards the repaint request to all of its working transformers which will set the internal data of the MM-Object 's drawables accordingly to the actual mathematical content.
Calling `render()` on a MM-Object is therefore equivalent to calling `render()` on every transformer instance hold by the MM-Object .

Example: 2D point
The `MMAffine2DTransformer` gets the coordinates of its "master" (here: `Affine2DPoint`) and passes them to its drawable (here: `G2DPointDrawable`) which will draw a point at the new coordinates in the canvas.
When dragging the point with the mouse, an instance of the class `Affine2DMouseTranslateHandler` will set the new coordinates in the master (a `MMAffine2DPoint`) and the rendering cycle will continue as described above.

3 Graphical Scenes with `Canvas`

A canvas is a paint board where MM-OBJECTS , images and text can be displayed. It can hold either 2- or 3-dimensional objects, their class is respectively `MMG2DCanvas` or `MMJ3DCanvas`, subclasses of `MM2DCanvas` and `MM3DCanvas`.[6] The common class of these two different canvas types is `MMCanvas`, where the biggest part of functionality is implemented.

3.1 Common Functionalities

Objects can be added with
`addObject(MMCanvasObjectIF`[7]`)`
defined in `MMCanvas`[8].

3.2 2D Canvas

3.3 3D Canvas

[6]The distinction of two 2D and 3D canvas classes was made to allow in future releases the usage of other graphics implementations than the Java-proper ones (*Graphics2D* and *Java3D*).

[7]net.mumie.mathletfactory.mmobject

[8]net.mumie.mathletfactory.display

4 User Interfaces with `ControlPanel`

The `ControlPanel`[9] represents a container for GUI(Graphical User Interface)-components, providing a flexible text editor-like layout manager. It allows the programmer to add components and to lay out them in the mathlet as easy as writing a text: the ControlPanel's height will be divided upon all lines where each of them can have a different horizontal alignment (left, right and center). Their height will be determined by the preferred sizes of their child components.

The `ControlPanel` is a base functionality of the BaseApplet and all of its extending classes. In most cases it is positioned under the canvases (except for the NoCanvasApplet: there it fills the whole vertical space). It can be accessed through `BaseApplet.getControlPanel()` or used indirectly by one of the delegate methods defined in the BaseApplet. These delegaters are named after their implementations in the ControlPanel, except the `add()` method: its corresponding delegate method is `addControl()`.[10]
It is possible to create a new ControlPanel-instance (e.g. in a standalone applet or in a new tab) but it is not necessary to do this in the predefined mathlet classes.

4.1 Layout Mechanism

At the begin (means no object has been added yet) a "line pointer" starts at the first line and adds all following components to it. A line will be terminated by a line break. The pointer will then jump to the second line.
Note that line alignments can only be done for the current row.
It is possible to use tabstops and other space holders for a more exact positioning.

4.2 Method Summary

- `add(JComponent)` – adds a component to the actual line's end
- `addText(String)` – adds a `TextPanel` with the given text
- `addText(String, Color)` – adds a `TextPanel` with the given text and color
- `addImage(String)` – adds an image with the given file location
- `insertLineBreak()` – jumps to the next line
- `insertLineBreak(int)` – jumps `int` lines downwards

[9]net.mumie.mathletfactory.appletskeleton.util
[10]This distinction was made to show the difference between the add-method defined in the Applet-class and this one.

- `insertTab()` – inserts a single tabstop
- `insertTab(int)` – inserts `int` tabstops
- `insertHSpace(int)` – inserts a place holder with `int` pixels width
- `insertVSpace(int)` – inserts a place holder with `int` pixels height
- `setLeftAlignment(), setCenterAlignment(), setRightAlignment()` – sets left/center/right alignment of the current and all following lines
- `setScrollable(boolean)` – sets whether the `ControlPanel` should be scrollable
- `getCurrentLine()` – returns the container for the current line
- `getLine(int n)` – returns the container for the n-th line

5 Styled Texts

Styled texts can contain both descriptions, style and layout information. They can be used in almost all applet technologies, except ”raw” SWING components (e.g. `JLabel`).

5.1 Capable Components

5.1.1 `TextPanel`

Package: `net.mumie.mathletfactory.display.util`
This is the base component for all styled texts (i.e. TeX and HTML). It is widely used in applets for e.g. titles and texts in `ControlPanels`. It can be used as an replacement for `JLabel`, which only supports HTML.

Special methods:

- `setText(String)` sets the text to be displayed
- `setWidth(int)` sets the *preferred width* of this text component; setting this property will enable *word wrapping*; `-1` will restore the default size
- `setHeight(int)` sets the *preferred height* of this text component; `-1` will restore the default size
- `setHorizontalTextAlignment(int)` sets the alignment along the x axis by using one of the `SwingConstants`

5.1.2 `StyledTextButton`

Package: `net.mumie.mathletfactory.display.util`
This component combines `JButtons` along with the capability to display styled texts.

5.2 Styled Texts Using TeX

Descriptive texts with font styles and formulas can be easily created using a *TeX dialect*. It orientates on the common TeX commands and syntax but uses some simplifications and harmonisations. It is designed to easily add new commands if they are needed. By now only the most used commands are implemented.

5.2.1 Special Considerations

- *Environments* must be defined in the following form:
  `\command_name{argument_1}{argument_2}{...}`

- *Packages* must not be declared
- *Math Environment* (i.e. ”$”) is not used
- *Blanks and Line Breaks* are ignored
- *Constants* are always expressed in uppercase letters
  (e.g. color constant ”RED”).

5.2.2 Using Special Characters

- **Java** - the backslash character (”\”) must be double quoted, i.e. ”\\”
- **MM-TeX** - the backslash character (”\”) must be expressed with ”\backslash{}”
- **Plain Text** (e.g. *message files*, see ...) - no restrictions

5.2.3 Adding Space

Line Breaks

- \\ - adds a line break

Blanks

- \`quad` - adds a non-breaking space
- \`qquad` - adds a double non-breaking space

5.2.4 Font Formattings

Font Size

- `tiny`, `small`, `normalsize`, `large`, `Large`, `LARGE`, `huge`
  Example: "`\large{Text}`" ⇒ Text

Font Style

- `\textbf{...}` - bold font
  Example: "`\textbf{Text}`" ⇒ **Text**

- `\textit{...}` - italic font
  Example: "`\textit{Text}`" ⇒ *Text*

Superscripts

- `^{...}` - superscript
  Example: "`a^2`" ⇒ a^2

- `_{...}` - subscript
  Example: "`a_3`" ⇒ a_3

- `_{...}^{...}` - both super- and subscript
  Example: "`a^2_3`" ⇒ a^2_3

Note: The paranthesises can be omitted if only a single character should be scripted.

Font Color

- `\textcolor{`*color*`}{...}`
  with *color* as a color constant (see below).
  Example: "`\textcolor{BLUE}{Text}`" ⇒ Text

The available colors constants are:

```
WHITE, BLACK
GRAY, LIGHT_GRAY, DARK_GRAY
RED, LIGHT_RED, DARK_RED
PINK, LIGHT_PINK, DARK_PINK
ORANGE, LIGHT_ORANGE, DARK_ORANGE
YELLOW, LIGHT_YELLOW, DARK_YELLOW
GREEN, LIGHT_GREEN, DARK_GREEN
MAGENTA, LIGHT_MAGENTA, DARK_MAGENTA
CYAN, LIGHT_CYAN, DARK_CYAN
BLUE, LIGHT_BLUE, DARK_BLUE
```

5.2.5 Mathematical Environments

- `\frac`{*nominator*}{*denominator*} - fractions
  Example: "`\frac{a}{b}`" ⇒ $\frac{a}{b}$
- `\vec`{...} - vector arrows
  Example: "`\vec{v}`" ⇒ $\vec{v}$
- `\norm`{...} - vector norm
  Example: "`\norm{a}`" ⇒ $||a||$
- `\det`{...} - determinant border (matrices)
  Example: "`\det{a}`" ⇒ $|a|$
- `\abs`{...} - absolute value
  Example: "`\abs{a}`" ⇒ $|a|$
- `\dot`{...} - single dotted characters
  Example: "`\dot{a}`" ⇒ $\dot{a}$
- `\ddot`{...} - double dotted characters
  Example: "`\ddot{a}`" ⇒ $\ddot{a}$
- `\bar`{...} - horizontal top bar
  Example: "`\bar{a}`" ⇒ $\bar{a}$
- `\tilde`{...} - tilde top bar
  Example: "`\tilde{a}`" ⇒ $\tilde{a}$

5.2.6 Special Characters and Umlaute

Mathematical Operators

Glyph	Command	Glyph	Command
$\sim$	\sim	$\forall$	\forall
$\simeq$	\simeq	$\exists$	\exists
$\equiv$	\equiv	$\in$	\in
$\approx$	\approx		\empty
$\ne$	\ne	∞	\infty
$\cong$	\cong	$\perp$	\perp
$\le$	\le	$\rightarrow$	\rightarrow
$\ge$	\ge	$\Rightarrow$	\Rightarrow
$\pm$	\pm	$\leftarrow$	\leftarrow
$\mp$	\mp	$\Leftarrow$	\Leftarrow
$\cdot$	\cdot	$\leftrightarrow$	\leftrightarrow
∇	\nabla	$\Leftrightarrow$	\Leftrightarrow
∂	\partial		

Special Characters and Umlaute

Glyph	Command
á	\’a
â	\ˆa
à	\\`a
ã	\˜a
ä	\¨a
ă	\ua
æ	\ae
ç	\cc
é	\’e
ê	\ˆe
è	\\`e
ë	\¨e
í	\’i
î	\ˆi
ì	\\`i
ï	\¨i
ł	\l
ñ	\˜n
ó	\’o
ô	\ˆo
ò	\\`o
õ	\˜o
ö	\¨o
ø	\o
œ	\oe
ß	\¨s, \ss
ú	\’u
û	\ˆu
ù	\\`u
ü	\¨u
ý	\’y
ÿ	\¨y
¡	!’

Glyph	Command
Á	\’A
Â	\ˆA
À	\\`A
Ã	\˜A
Ä	\¨A
Ă	\uA
Æ	\AE
Ç	\cC
É	\’E
Ê	\ˆE
È	\\`E
Ë	\¨E
Í	\’I
Î	\ˆI
Ì	\\`I
Ï	\¨I
Ł	\L
Ñ	\˜N
Ó	\’O
Ô	\ˆO
Ò	\\`O
Õ	\˜O
Ö	\¨O
Ø	\O
Œ	\OE
Ú	\’U
Û	\ˆU
Ù	\\`U
Ü	\¨U
Ý	\’Y
Ÿ	\¨Y
¿	?’

Greek Characters

Glyph	Command	Glyph	Command
α	\alpha	A	\Alpha
β	\beta	B	\Beta
γ	\gamma	Γ	\Gamma
δ	\delta	Δ	\Delta
ϵ	\epsilon	E	\Epsilon
ζ	\zeta	Z	\Zeta
η	\eta	H	\Eta
θ	\theta	Θ	\Theta
ι	\iota	I	\Iota
κ	\kappa	K	\Kappa
λ	\lambda	Λ	\Lambda
μ	\mu	M	\Mu
ν	\nu	N	\Nu
ξ	\xi	Ξ	\Xi
o	\omicron	O	\Omicron
π	\pi	Π	\Pi
ρ	\rho	R	\Rho
σ	\sigma	Σ	\Sigma
τ	\tau	T	\tau
υ	\upsilon	Y	\Upsilon
ϕ	\phi	Φ	\Phi
χ	\chi	X	\Chi
ψ	\psi	Ψ	\Psi
ω	\omega	Ω	\Omega

5.3 Styled Texts Using HTML

6 Internationalization

Internationalization of mathlets can be done by using localizable messages (identified by keys) for e.g. the title or descriptions inside the mathlet. The mathlet developer is encouraged to use localizable messages instead of static strings in order to free the code from language dependant expressions and therefore to provide a more flexible language integration.

At start-up, the mathlet tries to load the correct message set for the executing system, else it tries to load it in the default language (english).

6.1 About Locales

In order to load the correct message set, the *locale* of the executing system must correspond to that of the message set. Each language has its own locale, e.g. "en" for english, "de" for german or "fr" for french. This language prefix identifies a set of messages/texts.
The locale of the mathlet runtime can be set via properties/parameters, otherwise the system locale will be used.

6.2 Storing messages

There are two possibilities to store messages outside a mathlet, whereas it is possible to combine them.

6.2.1 Messages Files

Message files confirm to the Java Property Format and store the messages of a single set (i.e. for a single language) in an own file. The syntax is `key = message`. The filename of a message-file is determined by the prefix "`Messages_"` followed by the locale and the file ending "`.properties`" (e.g. `Messages_en.properties` for english). This file must reside in the mathlet's *files directory*.
A message in the "en" set could be:
    myApplet.title = This is the title in english!
whereas in the "de" set:
    myApplet.title = Dies ist der Titel auf deutsch!

6.2.2 Messages Properties

Sets of messages can also be defined via XML in a mathlet's property file.
Inside a "i18n" section a single message is defined by three attributes: the key, the language and the text/message.
A `messages` (plural!) section defines a set or sub-set of messages with at least

one common attribute. By this way sets of messages can be organized e.g. by the language or the title in a hierarchical structure.
The "leaf" nodes must be concretemessages defined each in a `message` (singular!) node.
The following examples define three identical sets of messages:
Example 1:

```
<mf:i18n>
   <mf:message key="title" language="en" value="Example Title"/>
   <mf:message key="title" language="de" value="Beispiel-Titel"/>
</mf:i18n>
```

Example 2:

```
<mf:i18n>
   <mf:messages key="title">
      <mf:message language="en" value="Example Title"/>
      <mf:message language="de" value="Beispiel-Titel"/>
   </mf:messages>
</mf:i18n>
```

Example 3:

```
<mf:i18n>
   <mf:messages language="en">
      <mf:message key="title" value="Example Title"/>
   </mf:messages>
   <mf:messages language="de">
      <mf:message key="title" value="Beispiel-Titel"/>
   </mf:messages>
</mf:i18n>
```

6.2.3 Messages File Definitions

Inside a "i18n" section it is possible to include additional messages from custom messages files. In the following example, a messages file for each language will be defined:
Example:

```
<mf:i18n>
   <mf:message-file language="de" file="my_messages_de.properties"/>
   <mf:message-file language="en" file="language_en.ini"/>
</mf:i18n>
```

Note that the filenames here must not confirm to the common filename convention mentioned above.
If a defined file is not found during the runtime start, an exception will be thrown. This can be omitted with the additional attribute `mustExist` with the value `false`. By this way, naming conventions for additional messages files can be declared (as for the common filename convention).

Example:

```
<mf:i18n>
   <mf:message-file language="en" file="Messages_en.properties"
      mustExist="false"/>
</mf:i18n>
```

6.3 Parametrised Messages

Sometimes a message does not only contain a static text but may also include *variable content*. Instead of splitting the text around this content, placeholders can be used inside the text definition. The values they represent are defined in a list which must be specified while retrieving the message. Their name corresponds to the respective *position* of their value in the list, prefixed with a "$" character. The numbering starts as usual at "1". It is possible to use the same placeholder multiple times in a message.
If the list does not contain the appropriate number of items, an exception will be thrown.

The following example demonstrates a message with two variables:
Example:

```
my_text_key = The value is $1 but must be $2.
```

This message must be retrieved with a list of two values. The list could be created as follows:
Example:

```
Object[] list = new Object[] { "my value 1", "my value 2" };
```

See the next section for more information about retrieving messages.

6.4 Retrieving Messages

Messages of the current set can be retrieved through the two `getMessage(...)` methods defined in the `BaseApplet`-class.

The first method only takes a `String` argument which represents the key:

```
public String getMessage(String key)
```

Note that only non-parametrised messages can be retrieved with it.

The second method has an additional argument for the list of parameters:

```
public String getMessage(String key, Object[] params)
```

It allows to retrieve parametrised messages with the given parameters.

Examples:

```
setTitle(getMessage("title"));
setTitle(getMessage("my_text_key",
      new Object[] { "my value 1", "my value 2" }));
```

Both methods return `null` if no message is found in the current set.

7 Deploying Mathlets

There are some different possibilities to deploy own mathlets. Common to all of them is that the MathletFactory library must be deployed with.

7.1 Applets

The browser plugin (or the *appletviewer*) needs a website (i.e. a HTML or XHTML page) with proper ”applet” or ”object” tag to start the applet. This tag must include the location of the compiled classes, the applet size inside the page and may define other Plug-In settings and user-specified parameters.

7.1.1 Applet Tag

Applets may be embedded in websites with an `<applet>` tag.
Note that this tag is deprecated by the W3C in favor of the `<object>` tag (see next section) but is also preferred by some browser plugins.
The following example shows the usage of the `<applet>` tag:

```
<applet code="MyApplet.class" codebase="."
        archive="MyApplet.jar" width="400" height="300">
  <param name="myParameter1" value="a value">
  <param name="myParameter2" value="a value">
</applet>
```

Attributes:

- `code` specifies the name of the applet with or without suffix ”.class”. Note that no path must be specified!
- `codebase` specifies the path to the class-file relative to the website's location
- `archive` specifies a comma-separated list of needed archive files (i.e. ”jar” or ”zip” files where also the applet class can be located in)

7.1.2 Object Tag

Applets may also be embedded in websites with an `<object>` tag.
The usage of this tag is encouraged by the W3C. Below is an example for an `<object>`-tag:

```
<object classid="java:MyApplet.class" codetyte="application/java"
      codebase="./" archive="MyApplet.jar" width="400" height="300">
  <param name="myParameter1" value="a value">
  <param name="myParameter2" value="a value">
```

```
</object>
```

Attributes:

- `classid` specifies the name of the applet with prefix "java:" and suffix ".class". Note that no path must be specified!
- `codebase` specifies the path to the class-file relative to the website's location
- `archive` specifies a comma-separated list of needed archive files (i.e. "jar" or "zip" files where also the applet class can be located in)

7.2 Applications

Applications runs in an own virtual machine outside the browser (e.g. from the command line). Hence some functionality must be added to satisfy the specific applet requirements. Starting an applet as an application means basically to perform these steps:

- create a `main` method; this is the entry point for applications; the applet class must be created and initialized here
- call the applet instance methods `init` and `start` in order to initialize the applet correctly
- create a frame where the applet should be put into and show that frame

These steps can be summarised as in the following example: `import net.mumie.mathletfactor`

```
import net.mumie.mathletfactory.util.*;

public class MyApplet extends SingleG2DCanvasApplet {
   int width = 400, height = 300;
   public static void main(String[] args) {
      MyApplet applet = new MyApplet();
      applet.init();
      applet.start();
      BasicApplicationFrame frame =
                  new BasicApplicationFrame(applet, width, height);
      frame.show();
   }
}
```