A quick guide to FRESH MARKER

What is FreshMarker?

FRESH MARKER is a simple and embedded Java 21 template engine, which is inspired by *FreeMarker*. Most of the basic concepts are the same.

Integration

Maven Dependency

```
<dependency>
  <groupId>de.schegge</groupId>
  <artifactId>freshmarker</artifactId>
  <version>1.9.0</version>
</dependency>
```

Template Evaluation

```
String content = "Hello ${first} ${last}!";
Map<string, Object> model = Map.of(
    "first", "Jens", "last", "Kaiser");
Configuration configuration = new Configuration();
TemplateBuilder builder = configuration.builder();
Template template = builder.getTemplate("tst", content);
String output = template.process(model);
```

This template produces the output `"Hello Jens Kaiser!"`.

Template Reduction

Templates can be partially evaluated. The newly created template saves you from having to evaluate static values again later.

```
Map<string, Object> model = Map.of("first", "Jens");
Template reduced = template.reduce(model);
```

This new template works internally on `"Hello Jens ${last}!"`.

Model

The model contains all the data, that can be used in the template. A large number of types are automatically supported so that no conversions need to take place when using your own data.
Unless otherwise stated, the names refer to the corresponding classes in **JDK**.

Primitives

Primitives are all **Java** types whose value can be output directly in the template.

`String`, `StringBuilder`, `StringBuffer`, `Boolean`, `AtomicBoolean`, `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `BigInteger`, `BigDecimal`, `AtomicInteger`, `AtomicLong`, `Locale`, `URI`, `URL`, `UUID`, `Date`, `Time`, `LocalDate`, `LocalTime`, `LocalDateTime`, `ZonedDateTime`, `Instant`, `Period`, `Duration`, `Year`, `YearMonth`, `MonthDay`

Literals of `Boolean`, `String`, `Integer` and `Double` can be used in the template.

Boolean	`true`, `false`	String	`'Gonzo'`, `''`, `"Gonzo"`, `""`
Integer	`42`	Double	`23.5`

Enums

All **Java** *Enum* types can be used as *primitive* types. To display them, the `toString` method is used. This can be overridden for an *Enum*.

Null

Processing `NULL` usually leads to errors. There are three ways to avoid these errors in FRESH MARKER. The *Default Operator*, the *Existence Operator* and the comparison with the `null` literal.
The default operator (`!`) replaces a null value with its optional parameter or the empty string.
The *Existence Operator* (`??`) returns `false`, if the value is `null`, otherwise `true`.
A comparison with the `null` literal can be used to check whether a value is `null`.

Sequences

Everything that implements `List` interface can be used as a *sequence*. Literal Sequences can be used in the template

```
[1, 2, 3, 4, 5, 6, 7]
['eins', 'zwei', 'drei']
```

Hashes

Everything that implements the `Map` interface with `String` keys and also *Beans* and *Records* can be used as *Hash*.

Ranges

A *range* is an numeric interval with a lower limit.

Right-unlimited Range

A *right-unlimited range* has no upper limit. It has the form `x..` where *x* can be any numerical expression.

`-10..`	lower bound −10
`a..`	lower bound in the numeric variable *a*

Right-limited Range

A *right-limited range* has a lower and an upper limit. It has the form `x..y` or `x..<y` where *x* and *y* can be any numeric expression. The second form describes a range with an exclusive upper limit.

`-10..<0`	lower bound −10 and upper bound −1
`a..b`	lower bound in variable a and upper bound in *b*
`10..0`	lower bound 1 and upper bound 10
`a..<b`	lower bound in variable *a* and an exclusive upper bound in *b*

Length-limited Range

A *length-limited range* is a right-limited range with a different syntax `a..*c`. Here *a* again corresponds to the lower limit of the range and *c* indicates the size of the range.

`4..*-5`	lower bound 4 and upper bound 0
`a..*b`	lower bound in variable *a* and an range size in *c*

Slices

Slices define slice operators on other data types. These operators can be applied to *Range*, *List*, and *String* values. The *Slices* are described by range expressions in square brackets. The range specifies the lower and upper limits of the *Slice*. With inverted ranges, the slice operation produces an inverted result.

Optionals

Optionals of type `java.util.Optional` are also supported in the model. The empty optionals are interpreted as `NULL` and all other values are interpreted as instances of generic type.

Lazys

Not all values in the model are used and if their provision is costly, then they can be loaded lazily. FRESH MARKER offers the `TemplateObjectSupplier` class for this purpose.

```
Map<String, Object> model = Map.of(
  "tree", TemplateObjectSupplier.of(() -> fromDB(name))
);
template.process(model);
```

Comments

Comments can be placed anywhere inside the template. They can be used to clarify the details of the template.

```
<!-- Comment -->
<#-- Comment -->
```

Expression

Expressions can be used in *Interpolations* and in many other places. A simplified overview shows the possibilities.

```
Expression : OrExpr;
OrExpr : AndExpr ( ('|' | '||' | '^') AndExpr )*;
AndExpr : EqualityExpr ( ('&'|'&&') EqualityExpr)*;
EqualityExpr : RelExpr [ ('='|'=='|'!=') RelExpr];
RelExpr : RangeExpr [ ('>'|'>='|'<'| '<=') RangeExpr ];
RangeExpr : AddExpr [ ('..'|'..<'|'..*) [ AddExpr ] ];
AddExpr : MultExpr ( ('+'|'-') MultExp)*;
MultExpr : UnaryExpr ( ('*'|'/'|'%') UnaryExpr )*;
UnaryExpr : PlusMinusExpr | NotExpr | DefaultExpr;
PlusMinusExpr : ('+'|'-') DefaultExpr;
NotExpr : "!" DefaultExpr;
DefaultExpr : PrimaryExpr '!' [ PrimaryExpr ];
PrimaryExpr : BaseExpr ( DotKey | DynamicKey
     | MethodInvoke | BuiltIn | Exists);
BaseExpr : Identifier | Literal | Parenthesis
     | BuiltinVar;
Parenthesis : '(' Expression ')';
DotKey : '.' Identifier;
DynamicKey : '[' Expression ']';
MethodInvoke : '(' [ArgsList] ')';
BuiltinVariable : '.' Identifier!
```

FRESH MARKER

Directives

Directives give the template structure. Depending on conditions, their contents are output once, multiple times, or not at all.

List Directive

A *List Directive* directive prints its contents for each element of a *Sequence* or *Hash*. The optional looper variable gives access to the loop metadata. The list can be reduced by the optional attributes `filter`, `offset` and `limit`.

```
<#list sequence as item with looper>
${looper}. ${item}
</#list>
```

The hash keys can be sorted by the optional `sorted` attribute.

```
<#list hash as key sorted asc, values with looper>
${looper}. ${key} $value}
</#list>
```

If Directive

An *If Directive* prints the content for which the first expression in the *If* or one of the optional *ElseIf* parts results in `true`. If no expression applies and there is an optional *Else* part, then its content is printed.

```
<#if number <= 0>zero or less!
<#elseif number == 1>one!
<#else>two or biggerQ
</#if>
```

Switch Directive

A *Switch Directive* prints the content for which the expression in the optional *Case* parts first match the expression in the *Switch* part. If no expression applies and there is an optional *Default* part, then its content is printed.

```
<#switch number>
<#case 0>zero
<#case 1>one
<#default>two or bigger
</#case>
```

A second variant uses *On* parts instead of *Case* parts. These parts can match multiple expressions.

```
<#switch number>
<#on 0, 1>zero or one
<#on 2, 3>two or three
<#default>four or bigger
</#case>
```

...and much more

Further directives (*Brick*, *Setting*, *Outputformat*, *Include*, *Import*, *Var*, *Set*, *Macro*) can be found in the documentation.

Interpolation

Interpolations produce output for the values in the model. The syntax of an interpolation is `${expression}`, with an *Expression* described previously. An *Interpolation* must evaluate to a FRESH MARKER *Primitive* type. For example, if the result is a *Sequence*, an exception is thrown. If the result of the evaluation is the special value `NULL`, then an exception is also thrown. An interpolation must always return a result.

Built-Ins

Built-Ins are interpolation operations, which are called on the current value of the interpolation.
The following list is not complete and only shows the more popular built-ins.

Boolean Built-Ins

name	*command*	*output*
computer	`true?c`	true
human	`false?h`	falsch
then	`false?then(23, 42)`	42

String Built-Ins

name	*command*	*output*
upper case	`'TesT'?upper_case`	TEST
lower case	`'TesT'?lower_case`	test
capitalize	`'tesT'?capitalize`	Test
camel case	`'camel-case'?camel_case`	camelCase
snake case	`'snakeCase'?snake_case`	snake_case
kebab case	`'kebapCase'?kebap_case`	kebap-case
slugify	`'One two three'?slugify`	one-two-three
trim	`' trim '?trim`	trim
contains	`'test'?contains('t')`	true
starts with	`'Test'?starts_with('t')`	false
ends with	`'Test'?ends_with('t')`	true
length	`'test'?length`	4
esc	`'<>'?esc('HTML')`	`<>`
left padding	`'test'?left_pad(6,'#')`	##test
right padding	`'test'?right_pad(6,'#')`	test##
center padding	`'test'?center_pad(6,'#')`	#test#
mask	`'one two'?mask(2)`	*** *to
mask full	`'one two'?mask_full('?')`	???????

Number Built-Ins

name	*command*	*output*
computer	`6?c`	6
human	`6?h`	six
format	`2.5??format("%.2f")`	2.50
absolute	`-5?abs`	5
sign	`-5?sign`	-1
min	`23?min(42)`	23
max	`23?min(42)`	42
roman	`2012?roman`	MMXII

Looper Built-Ins

These are *Built-Ins* on the Looper Variable in a List-Directive. The values change with every loop.

name	*command*	*output*
is first	`looper?is_first`	`true`
is last	`looper?is_last`	`false`
has next	`looper?has_next`	`false`
index	`looper?index`	`0`
item parity	`looper?item_parity`	even
item cycle	`looper?item_cycle('A', 'B', 'C')`	A

Temporal Built-Ins

Number Built-Ins

name	*command*	*output*
computer	`dateTime?c`	1968-08-24T12:34:56
human	`yesterday?h`	yeasterday
since	`yesterday?since`	`P1D`
date	`dateTime?date`	`1968-08-24`
time	`dateTime?time`	`12:34:56`
string	`date?string('dd. MMMM')}`	24. August

...and much more

Further *Built-Ins* for (*Enum*, *Sequence*, *Locale*, *Range*, *Duration*, *Period*, *Version*) can be found in the documentation.

Built-In Variables

Some information is available as *Built-In Variables*. These include the following.

name	*description*
`.now`	the current date and time
`.locale`	the current locale
`.version`	the current **FreshMarker** version

Extensions

FreshMarker File `File` and `Path` typ support.

FreshMarker Money `Money` typ support based on Moneta https://javamoney.github.io/ri.html.

FreshMarker Random `Random` typ support.

Getting started with FRESH MARKER

Project: https://gitlab.com/schegge/freshmarker
Documentation: https://schegge.gitlab.io/freshmarker

Based on a Overleaf cheat sheet template https://overleaf.com.