# Kermeta MDK for Kermeta

## MDK manual

Didier Vojtisek
Cyril Faucher

*Abstract*

This document presents the Model Development Kit (MDK) for Kermeta.

It summarizes the provided services and presents some of its typical uses.

**Published Build date: 3-November-2010**
**$LastChangedDate:: 2010-06-02 10:42:20#$**

CHAPTER

# Preface

Kermeta is a Domain Specific Language dedicated to metamodel engineering. It fills the gap let by MOF which defines only the structure of meta-models, by adding a way to specify static semantic (similar to OCL) and dynamic semantic (using operational semantic in the operation of the metamodel). Kermeta uses the object-oriented paradigm like Java or Eiffel.

A MDK (Model Development Kit) is a set of Kermeta code that goes with a given metamodel. Each of the MDKs provides functionalities dedicated to the metamodel it applies to. (Ex: UML, Ecore, Traceability, yourDomainMetamodel, ...)

In this document we will focus on Kermeta MDK. This MDK applies to Kermeta metamodel itself. Ie. it provides functionalities to manipulate Kermeta programs.

**Important**

Kermeta is an evolving software and despite that we put a lot of attention to this document, it may contain errors (more likely in the code samples). If you find any error or have some information that improves this document, please send it to us using the bug tracker in the forge: **http://gforge.inria.fr/tracker/?group_id=32** or using the developer mailing list (kermeta-developers@lists.gforge.inria.fr) Last check: v1.2.2

**Tip**

The most update version of this document is available on line from http://www.kermeta.org .

# Introduction to the Kermeta MDK

The Model Development Kit for Kermeta contains various kind of codes that work with Kermeta models: some helper classes and some transformations.

**Tip**

To use them you simply have to add the corresponding require statement to your kermeta code. Then, the outline and the KermetaDoc views should help you to use its content.

## 1.1. Short description of the available *require*

***Kermeta MDK available files***

Currently, the MDK provides the following files :

| | |
|---|---|
| **`require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/visitor/KermetaVisitor.kmt"`** | This file adds a visitor to Kermeta metamodel. It helps you traverse a Kermeta model. |
| **`require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/helpers/AllHelpers.kmt"`** | This file adds various helper operations to the Metaclasses of Kermeta. |

**Tip**

If you are interested in only a subset of those helpers, you can directly require the corresponding files in the `helpers` directory of the plugin.

| | |
|---|---|
| **`require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/helpers/StructureHelpers.kmt"`** | Same as `AllHelpers.kmt` except that it provides only helpers on classes of the structure package. |
| **`require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/extension/SingletonSupport.kmt"`** | This files adds a support for getting a kind of singleton in Kermeta. See the presentation done for Kermeta days 09 http://www.kermeta.org/community/workshop/09/slides/Vojtisek-KermetaDay09-Singleton.pdf |
| **`require "platform:/plugin/org.kermeta.language.mdk/src/kmt/math/Math.kmt"`** | Provides a kermeta::Math class that offers some of the classical operation like sin, cos, pi, toDegrees, toRadians, etc |

| | |
|---|---|
| **nguage/visitor/prettyprinters/kmt-PrettyPrinter.kmt"** | This file adds a KMT prettyprinter visitor to Kermeta metamodel. It provides a String output of the kmt surface syntax for your km models. |
| **require "platform:/plugin/org.kermeta.language.mdk/src/kmt/standard/StringUtils.kmt"** | This file adds some String manipulation operations to String. for example trim(), startsWith(), endsWiths(), etc |
| **require "platform:/plugin/org.kermeta.language.mdk/src/kmt/utils/Random.kmt"** | This file offers some operations to get random elements. (Currently, random numbers, but can/should be extended) |
| **require "platform:/plugin/org.kermeta.language.mdk/src/kmt/utils/Date.kmt"** | This file offers some operations to calculate elapsed time. |
| **require "platform:/plugin/org.kermeta.language.mdk/src/kmt/utils/Properties.kmt"** | This file offers an operation to get properties as a Hashtable<String, String> from a Properties file. |

# *CHAPTER 2*

# Cookbook for Kermeta MDK

This chapter presents some of the uses of this MDK.

## 2.1. Saving a km model with references to ClassDefinitions in framework.km

When creating a km model, you may need to create references to ClassDefinition that are defined in the framework.km. Typically, when you create an Operation, you'll also have to create a Class whose reference `typeDefinition` points to the ClassDefinition in the framework.

A first but wrong approach would be to use the reflectivity :

```
var anOperation : kermeta::language::structure::Operation init kermeta::language::structure::Operation.new
anOperation.name := "Op1"
var aClass : kermeta::language::structure::Class init kermeta::language::structure::Class.new
// set the return type of the Operation to Boolean
aClass.typeDefinition := kermeta::standard::Boolean.typeDefinition      // will typecheck, but cannot be saved !!!
anOperation.type := aClass
anOperation.containedType.add(aClass)
```

This code will correctly typecheck, but cannot be saved. This is because it mixes model element and program definitions. Unfortunatly, it is not possible to ensure that those definitions comes from a concrete model file. For example, the definition can come from an ecore or a kmt file (which are NOT km models).

The solution consists in explicitly loading the km file that contains the ClassDefinition and then, retrieve the ClassDefinition in it. In our sample we probably want to load `framework.km` and find the ClassDefinition Boolean in it.

The MDK will help us in this task by providing a function `getTypeDefinitionByQualifiedName` to ModelingUnit. The code will then look like :

```
...
// get the helper
require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/helpers/structure/ModelingUnitHelper.kmt"
...

var repository : EMFRepository init EMFRepository.new
var resFramework : kermeta::persistence::EMFResource

resFramework ?= repository.getResource("platform:/plugin/fr.irisa.triskell.kermeta.io/src/kermeta/framework.km")
// find Boolean in the resource
var booleanTypeDef : kermeta::language::structure::GenericTypeDefinition
var mainModelingUnit : kermeta::language::structure::ModelingUnit
mainModelingUnit ?= resFramework.detect{ m | m.isInstanceOf(kermeta::language::structure::ModelingUnit)}
        // use the helper to find Boolean in the ModelingUnit
booleanTypeDef ?= mainModelingUnit.getTypeDefinitionByQualifiedName("kermeta::standard::Boolean")

var anOperation : kermeta::language::structure::Operation init kermeta::language::structure::Operation.new
anOperation.name := "Op1"
var aClass : kermeta::language::structure::Class init kermeta::language::structure::Class.new
// set the return type of the Operation to Boolean
aClass.typeDefinition := booleanTypeDef      // correct, can be saved (using the same repository that was used to load framework.km)
anOperation.type := aClass
anOperation.containedType.add(aClass)
```

Obviously, you can load any km file you want.

**Note**

If you want to point to a definition that is in a kmt (textual) or in an ecore file, you'll need to convert it into a km file first.

## 2.2. Sample use of the kmt prettyprinter : prettyprinting a failed Constraint

**Note**

The base of this prettyprinter was written before the introduction of aspect in Kermeta. For a better design, we would have now weaved the prettyprint operation rather than weaved a visitor.

Here is a sample use of this printer (prettyprints the body of a failed invariant). Its main class is `kermeta::utils::BasicPrettyPrinter`

```
    ...
    require "platform:/plugin/org.kermeta.language.mdk/src/kmt/language/visitor/prettyprinters/kmtPrettyPrinter.kmt"

    ...
    do
        // checking invariants
        inputResource.instances.each{e|
            e.checkAllInvariants
        }
        stdio.writeln("model is valid")

  rescue (myError : ConstraintViolatedInv)
        // note that this locator helper work only for UMl model and is provided by UML MDK
        // you
        var locatorhelper : UMLUtils::UMLElementLocatorHelper init UMLUtils::UMLElementLocatorHelper.new
        stdio.errorln(myError.message)
        stdio.errorln("Faulty object is : " + locatorhelper.getLocationString(myError.constraintAppliedTo))

        // Do not forget to initialize the BasicPrettyPrinter !
        var basicPP : kermeta::utils::BasicPrettyPrinter init kermeta::utils::BasicPrettyPrinter.new.initialize
        stdio.errorln("Violated constraint is : " +basicPP.accept(myError.failedConstraint.body, 0))
        stdio.errorln(myError.stackTrace)
     end
```

**Warning**

Do not forget to initialize the BasicPrettyPrinter !