

Telosys – a lightweight and pragmatic code-generator

by Laurent Guerin | May 22, 2018 | code generation, databases, tools | 0 comments



Telosys is a simple and pragmatic code generator. Provided as a [Command Line Interface tool](#) and as an [Eclipse Plugin](#), it uses an original approach based on 2 kinds of “lightweight models” (“database model” and “DSL/text model”). This tool aims to provide an alternative to the classical “UML first” approach that requires a significant workload at the beginning of the project to create a reliable and accurate model.

Unlike conventional tools (tools based on MDA/XMI approach), Telosys is very light and can be considered as a “tactical tool”. It has been designed by developers for developers and focuses on a quick start (its motto is “remain simple”). The typical usage of Telosys can be described in 3 steps:

1. Create your lightweight model

The first step is the setup of a **lightweight model** containing the entities definitions that will be used to generate the code. There are two ways to do that: with a “Database Model” or with a “DSL Model”

- **Database Model**

If you have an **existing relational database** it can be used by Telosys to create automatically a “**database model**”.

Indeed the database schema is by itself a “model”. Telosys will connect to the database and discover all the tables and their relationships in order to create the model for you. This kind of model is stored in XML format in a single file (the “.dbrep” file) and

contains an entity for each table. Once the model has been created, it can be updated and enriched in order to provide exactly what is expected for the code generation (it's the refining step)

The Database model editor in Eclipse :

The screenshot shows the Eclipse Database Model Editor interface. The title bar indicates the file is 'BOOKS.dbrep'. The main window is titled 'Model : Entities attributes and mapping'. It has several input fields: 'Database table name : BOOK', 'Catalog :', 'Schema : ROOT', and 'Entity class name : Book'. Below these are two tabs: 'Mapping table - object' (selected) and 'Foreign keys'. The 'Mapping table - object' tab displays a table with the following data:

	Database Name	Database Type	JDBC Type	Attribute Name	Attribute Type	Further info
<input checked="" type="checkbox"/>	ID	INTEGER	4 : integer	id	Integer	
<input checked="" type="checkbox"/>	PUBLISHER_ID	INTEGER	4 : integer	publisherId	Integer	
<input checked="" type="checkbox"/>	AUTHOR_ID	INTEGER	4 : integer	authorId	Integer	
<input checked="" type="checkbox"/>	ISBN	VARCHAR(13)	12 : varchar	isbn	String	NE,[null;13]
<input checked="" type="checkbox"/>	TITLE	VARCHAR(160)	12 : varchar	title	String	[null;160]
<input checked="" type="checkbox"/>	PRICE	DECIMAL	3 : decimal	price	BigDecimal	
<input checked="" type="checkbox"/>	QUANTITY	INTEGER	4 : integer	quantity	Integer	
<input checked="" type="checkbox"/>	DISCOUNT	INTEGER	4 : integer	discount	Integer	
<input checked="" type="checkbox"/>	AVAILABILITY	SMALLINT	5 : smallint	availability	Short	
<input checked="" type="checkbox"/>	BEST_SELLER	SMALLINT	5 : smallint	bestSeller	Short	

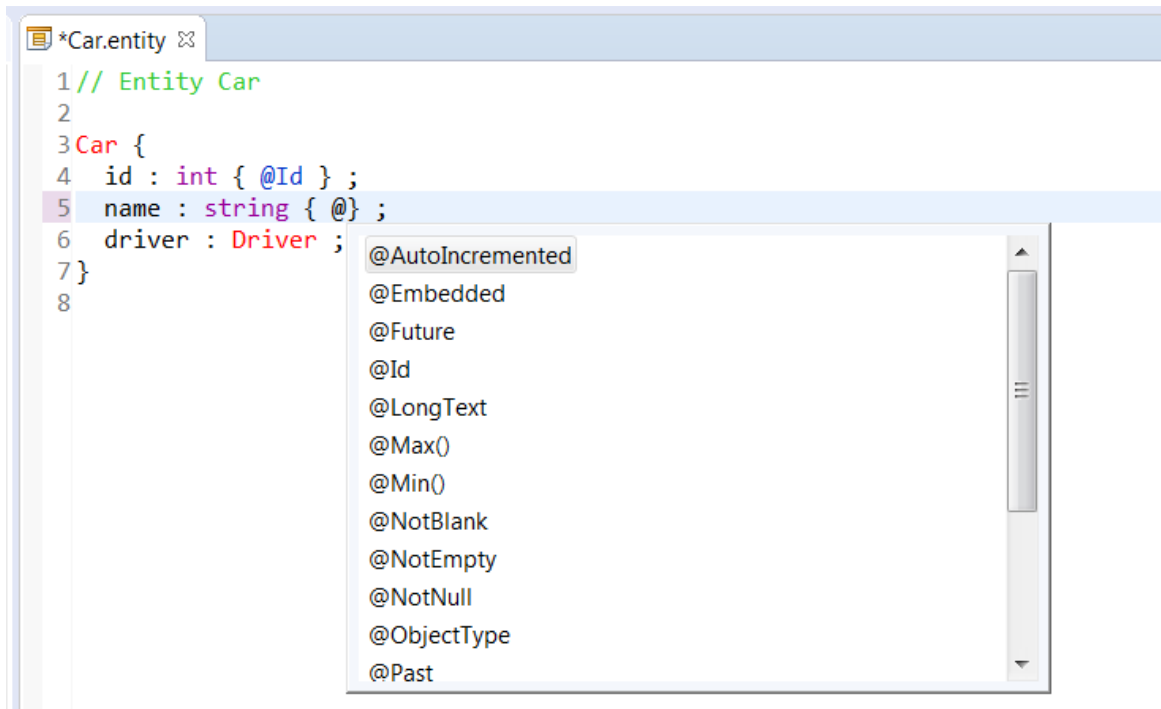
• DSL Model

If you want to start from scratch you can create a **"DSL model"**. This model is defined as a set of text files (one ".entity" file for each entity) with a very simple syntax. This model is a kind of entity-relationship model. Each entity has a set of attributes, each attribute can define :

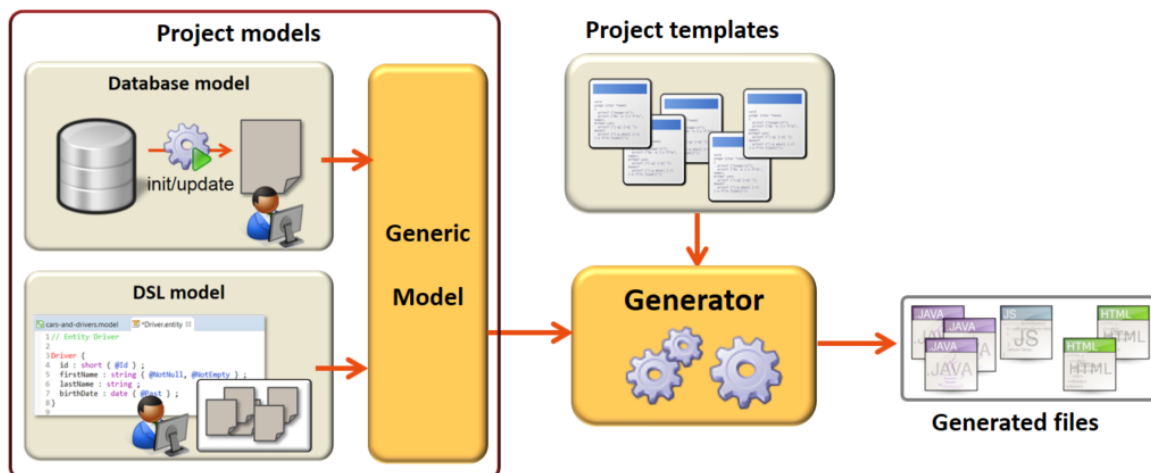
- a typed value (eg: *"id : int"*, *"manager : boolean"*, *"name : string"*, *"birthdate : date"*, *"price : decimal"*, *"comment : text"*, ...)
- a reference to a single entity (eg: *"driver : Driver"* to reference a "Driver" entity)
- a list of entities (eg: *"drivers : Driver []"* to reference 0..N "Driver" entities) .

Otherwise, complementary information can be added to each attribute in order to characterize them more precisely (eg: @Id, @Min(12), @NotNull, etc). The model can be improved gradually over time just by editing the entities' files.

The DSL model editor in Eclipse (extensions are also available for VSCode and Atom) :



These two types of models (database model and DSL model) are different but are used in the same way by the generator thanks to a “**generic model**” interface which exposes an abstract view of the model

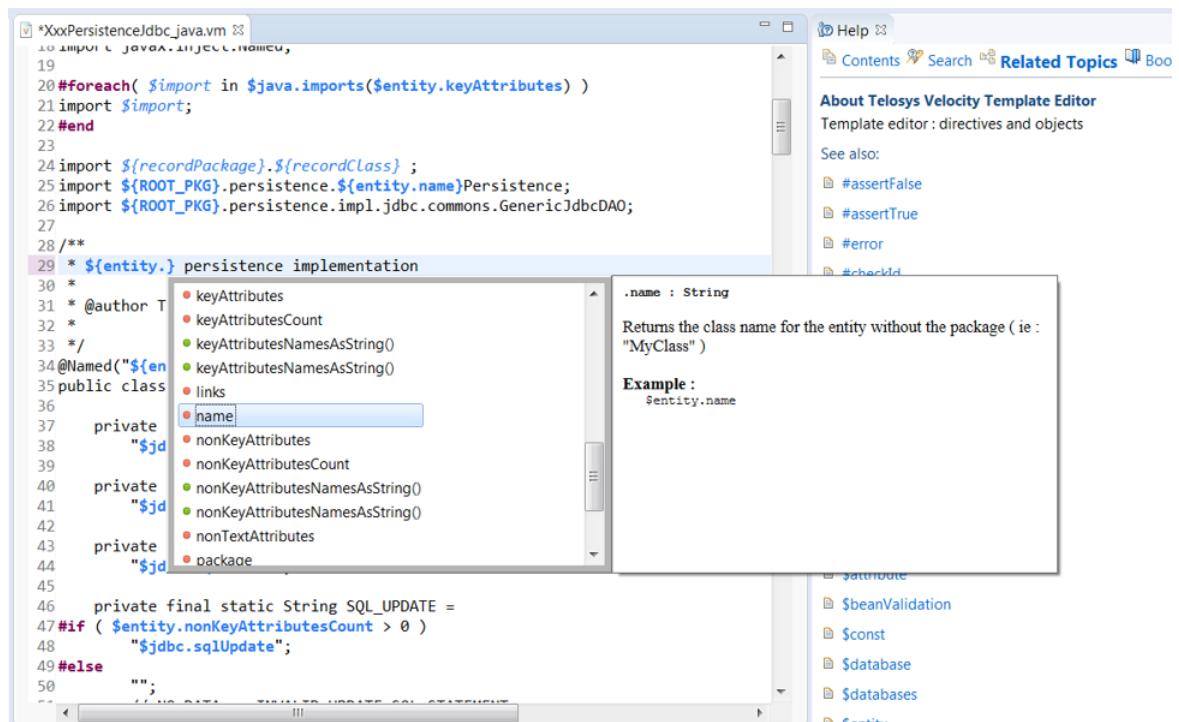


2. Choose the code-generation template

Once the model is ready, you need a set of **templates** in order to define the way the code will be generated. The generator is based on the “**Velocity**” engine, hence the well-known Velocity syntax is used to define the templates. Templates are organized in “**bundles**”, a bundle is a set of templates focusing a specific goal (for example web interface, persistence layer, etc)

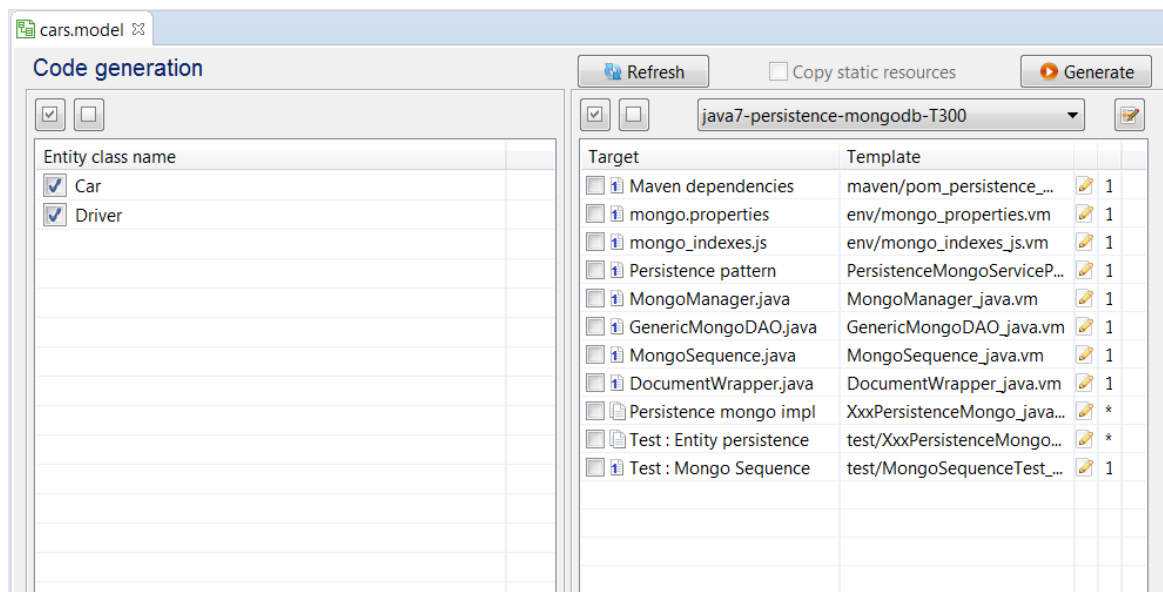
Some [ready-to-use templates are available on GitHub](#) (each Git repository is a bundle of templates, predefined templates for Java, JavaScript, Python, etc are available in the repository). All the templates can be adapted to conform with specific project requirements, and new templates can be created if necessary.

The templates editor :



3. Generate your application code

Finally, the last and simplest step is the **generation**. The generation will apply the selected templates on the selected entities as shown below (entities on the left part, templates on the right part).

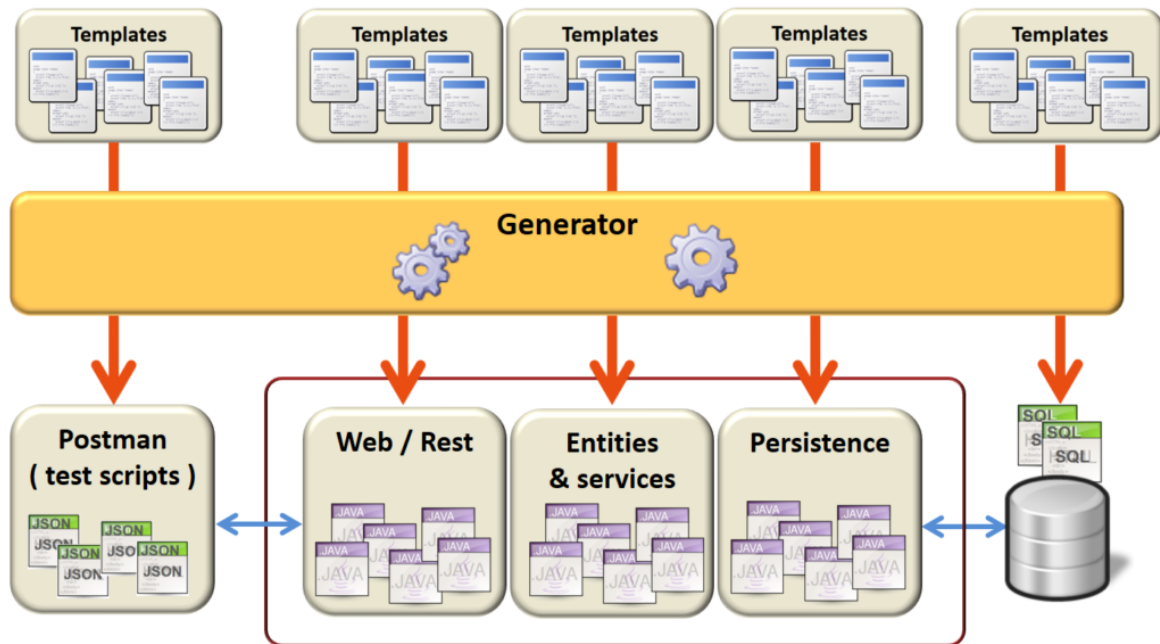


It's possible to select all or only part of the entities and templates. All kinds of text files can be generated, so the tool can be used to generate any kind of languages (Java, PHP, Python, HTML, JavaScript, TypeScript, Go, C#, etc) with any kind of framework (Spring, Bottle, Angular, VueJS, etc)

Project bootstrapping

Very often Telosys is used to bootstrap a project. It provides scaffolding capabilities that can be used to accelerate the development of all the layers of the architecture, including unit tests and integration tests.

Example for a Java web/REST application with Postman for REST tests :



Summary

Thanks to its pragmatic approach Telosys is generally used as a “development accelerator”, it allows a quick start for different kinds of projects. The “lightweight model” can be more or less refined depending on the context.

For more information, see :

- <http://www.telosys.org/>
- <http://marketplace.eclipse.org/content/telosys-tools>
- <https://twitter.com/telosys>


Or check this presentation

Telosys

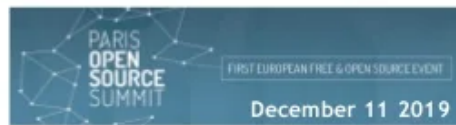
project booster



<http://www.telosys.org/>

 @telosys

 telosys@team@gmail.com



Who's speaking ?

Telosys project booster Paris Open Source Summit 2019 from **Laurent Guérin**

(the first version of this post was published in 2013, it has been updated since then to reflect the evolution of Telosys)

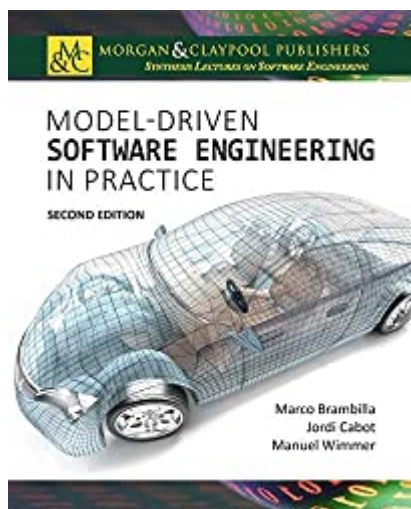


<input type="text"/>	Search
----------------------	--------

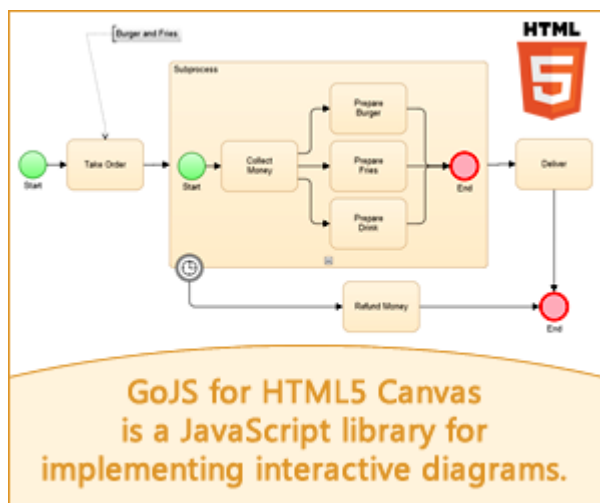
Showcase your modeling / low-code tool

Advertise Here

Modeling: all you need to know



GoJS - Interactive Diagrams with JS



Show your tool to hundreds of modeling experts

Advertise Here

Modeling Languages Copyright © 2020.

Tags

action language Alf atl bot bpmn chatbots Eclipse embedded EMF ER
executable UML fUML GitHub graphical ifml Java JavaScript json low-code
miotope modelia modelsconf modisco NoSQL OData OpenAPI open data
open source papyrus php python rest shlaer-mellor sql STAF SYSML
systems engineering testing textual tutorial UML Profile verification wordpress
XMI Xtext

