# Documentation for HELM2

## Project Overview:

The whole project consists of four different projects: HELM2NotationToolkit, ChemistryToolkit, HELMNotationParser and the HELM2WebService.

The main core is the HELM2NotationToolkit. The input for the HELM2NotationToolkit will be provided by the HELMNotationParser package. This package’s function is to parse the HELM2 or HELM1 and generate the HELM2Notation object. The section ‘Introduction to HELMNotationParser’ provides more information about the new introduced notation objects. For the HELM2 the whole chemistry was removed from the source code and is now stored in the ChemistryToolkit package. This API contains all abstract classes or interfaces which have to be implemented by a chemical library. Two libraries, Marvin Beans and CDK, are currently implemented and available to the user, see <https://github.com/Chemaxon/pistoia-chemistry-toolkit-marvin> and <https://github.com/MarkusWeisser/ChemistryToolkitCDK>. This structure allows an easy way to change a current implementation of the ChemistryToolkit or extend it with a customizable chemical library, see ‘Introduction to ChemistryToolkit’. The HELM2NotationToolkit provides methods to handle the generated HELM2Notation from the parser package. It generates images and calculates molecule properties for a whole HELM molecule. The project HELM2WebService provides REST API methods. This project builds a war-file, which then can be deployed on a server, for example on Tomcat.

## Introduction to HELMNotationParser

The function of the HELMNotationParser is to parse the HELM string in version one or two and to generate the HELM2Notation object. The parsing process also includes the validation of the input format. It produces only the HELM2Notation object, if the given HELM input is valid. The HELM string contains of four different sections divided by ‘$’. The first section is the polymer section. Here, every polymer is listed with its element. The second section contains every intra- or interconnection, including hydrogen bonds, of all presented polymers in the first section. Group information is represented in the third section. The last section is the annotation section.

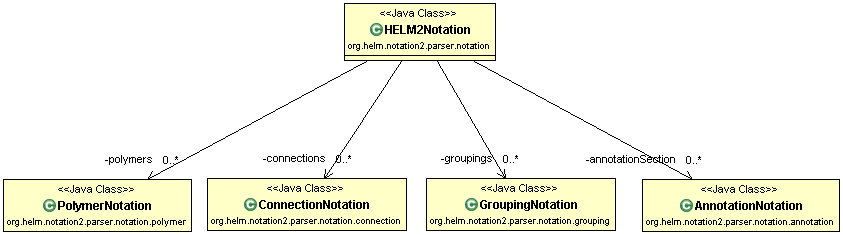
### How to use the HELMNotationParser

The HELMNotationParser can be used as a standalone program. It can be called via command-line or provides a graphical interface to parse HELM input and generates the notation objects.

You can also use the HELM2NotationParser.jar in your own java package. All objects of the HELM input are accessible via the HELM2Notation object. The HELMNotationParser is also capable of converting HELM from version 1.0 to version 2.0 using the ConverterHELM1ToHELM2.class. To generate the HELM2Notation object, call the ParserHELM2 with its method ‘parse’, given a HELM input.

### Main object HELM2Notation

The HELM2Notation contains a list of section objects for every section.



The first section is the polymer section. Here, the single polymer structure will be defined. Every polymer is mapped to the PolymerNotation.class.

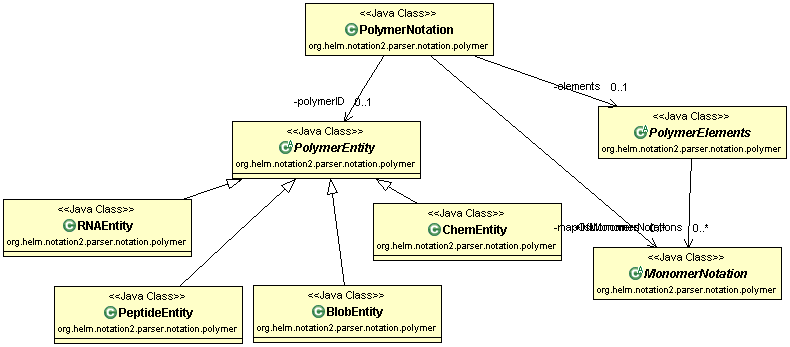
The second section is the connection section. Here, inter- and intra-connections will be defined. Note that also hydrogen bonds are now stored in the section. Every connection is mapped to the ConnectionNotation.class.

The third section is the grouping section. Here, group informations are defined. Every group is mapped to the GroupingNotation.class.

The last section is the annotation section. Here, annotations can be added to the HELM molecule. Every annotation divided by ‘$’ is mapped to an AnnotationNotation.class.

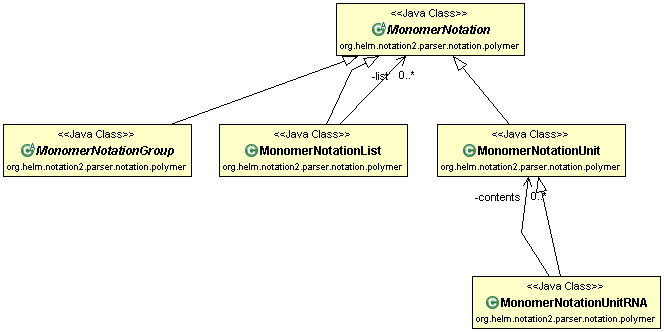
#### Class PolymerNotation: polymer section

Each polymer consists of two important elements: the PolymerEntity.class and the PolymerElements.class, which contains a List of MonomerNotation.class. The PolymerEntity.class describes if the polymer is a peptide, a RNA, a chemical molecule or a blob. The single elements of a polymer divided by ‘.’ are mapped to the superclass MonomerNotation.class.



###### Class MonomerNotation: monomer section

This class represents the single compounds for a polymer. For example a MonomerNotation can now be a single amino acid ‘A’ (= MonomerNotationUnit.class), a list of single amino acids ‘(A.A.G)’ (= MonomerNotationList.class), a group of single amino acids ‘(A+G)’ or a nucleotide ‘R(A)P’ (= MonomerNotationUnitRNA.class).

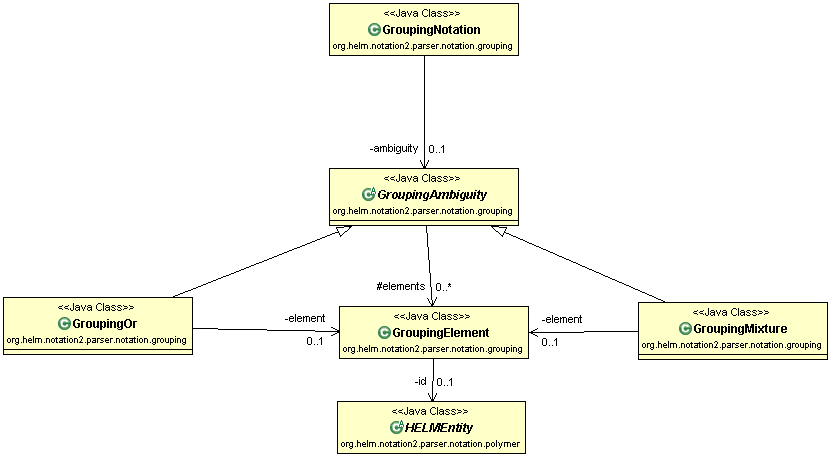


#### Class ConnectionNotation: connection section

The target- and the source-polymer for every connection is mapped to a HELMEntity.class. The details about the connection are saved in variables.

#### Class GroupingNotation: grouping section

This class consists of a GroupingAmiguity. There are two different types of a GroupingNotation.class. A grouping can be G1(CHEM1 + PEPTIDE1) or G1(CHEM2, PEPTIDE1). In the first case the group with the groupID G1 consists of a mixture of the chemical molecule CHEM1 and the peptide PEPTIDE1. In the second case the group consists either of CHEM2 or PEPTIDE1. For every type a new class was implemented, GroupingMixture.class and GroupingOr.class. Both types have a list of grouping elements, which can be a simple polymer or another group (HELMEntity.class).



#### Class AnnotationNotation: annotation section

This class has only on variable to save the given annotation as a string.

## Introduction to ChemistryToolkit

The ChemistryToolkit is an API providing all abstract classes and interfaces which a chemistry library has to implement for the HELM2NotationToolkit. Currently there are two chemical libraries, MARVIN Beans and CDK, implemented. All three java packages are available on github, see <https://github.com/MarkusWeisser/ChemistryToolkit>, <https://github.com/MarkusWeisser/ChemistryToolkitCDK> and <https://github.com/Chemaxon/pistoia-chemistry-toolkit-marvin>.

## Introduction to HELM2WebService

The HELM2WebService provides all applied REST methods. Additionally to this, it also offers a website with the REST methods. The website includes a swagger documentation, describing the given methods in detail. The HELM2WebService can be used as a standalone program.

## Introduction to HELM2NotationToolkit

The new HELM2NotationToolkit is the main part of HELM2. It provides methods to handle the new HELM2 features. In all cases it is using the HELM2 input in form of the HELM2Notation object. This object can be changed by the new HELM2NotationToolkit (ChangeObjects.class). It also calculates molecule properties, such as molecule formula, for the whole HELM2Notation object. The old functionality of the HELMNotationToolkit is also provided by the new version of the HELM2NotationToolkit. The newer version can also serve as a client to get your monomer store via REST API. A short package overview and a short description of some classes are showing which functions are currently available.

### Package – overview:

org.helm.notation2

- Chemistry.class: The chemistry engine of the package. It is defined by the config-file.

- InterConnections.class: Representing intra- and interconnections of molecules.

- MoleculeInfo.class: The properties of a molecule.

- RgroupStructure.class: Class to save all Rgroups of a molecule.

org.helm.notation2.calculation

- ExtincitonCoefficient.java: Class to calculate the extinction coefficient for a HELM or a single

polymer.

- MoleculeInformation.java: Class to calculate all properties for a HELM molecule.

org.helm.notation2.tools

- Images.class: Class to generate an image for a HELM molecule or a monomer.

- BuilderMolecule: Class to build molecule/s for a HELM molecule, polymer and monomer.

- SMILES.class: Class to get SMILES (standard or canonical) for a HELM molecule and polymer.

- Validation: Class to check, if the HELM2Notation is valid. It is also checking the validation of monomers.

- WebService.class: Class to provide every method, which should be called by the WebServer project.

- XHELM.class: Class to read or write XHELM.

- RNAUtils.class: Class providing methods for RNA molecules, for example to get the natural sequence for a RNA polymer.

- PeptideUtils.class: Class providing methods for peptide molecules, for example to get the sequence for a peptide polymer.

- PolymerUtils.class: Class to get the total monomer count for a polymer.

- ChangeObjects.class: Class to provide methods, to add, delete and change parts of a HELM2Notation object.

- FastaFormat.class: Class to read or write FASTA-files.

- MDLUtils.class: Class to write a molfile for a HELM molecule.

- SequenceConverter.class: Class to write or read peptide/RNA sequences.

- HELM1Utils.class: Class to produce standard or canonical HELM2Notation.

- HELM2NotationUtils.class: Class to get additional information from a HELM2Notation object.

- MethodsMonomerUtils.class: Class to get monomers for the HELM2Notation object.

## Tips for Setup

### How to use another monomer store in the HELM2NotationToolkit?

You can use a monomer store different from local file in ./helm directory. Therefore your monomer store has to be accessible via REST API. Hereby the HELM2NotationToolkit serves as a client. In your own ./helm (“user.home”) directory you can find the MonomerStoreConfig.properties file. Here, you have to set use.webservice to true and define where your monomers and nucleotides are coming from. There is a really short JAVA example showing how the REST API for a monomer database has to look like, see <https://github.com/MarkusWeisser/HELM2WebService/releases/tag/1.1.3 DemoWebserver.zip>. Your webserver has to implement four different REST methods; GET monomers, PUT monomers, GET nucleotides and PUT nucleotides. Note that all necessary values of a monomer and nucleotide, defined in the HELM2NotationToolkit, have to be there.

GET Monomers:

The request contains a query-parameter (“polymerType”) to allow filtering monomers according to their type. Three types have to be there: PEPTIDE, RNA and CHEM.

A list of monomers in JSON will be returned.

PUT Monomers:

The client (HELM2NotationToolkit) sends a PUT request with the monomer in its JSON-format to the server. An example for the Monomer A of the polymer-type PEPTIDE is given below:

{

"id" : 0,

"alternateId" : "A",

"naturalAnalog" : "A",

"name" : "Alanine",

"canSMILES" : "C[C@H](N[\*])C([\*])=O |$;;;\_R1;;\_R2;$|",

"molfile" : "\n Marvin 06250814262D \n\n 7 6 0 0 0 0 999 V2000\n 5.4886 -3.0482 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0\n 6.2031 -3.4608 0.0000 C 0 0 1 0 0 0 0 0 0 0 0 0\n 6.9176 -3.0483 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0\n 6.2030 -4.2858 0.0000 N 0 0 0 0 0 0 0 0 0 0 0 0\n 6.9177 -2.2233 0.0000 O 0 0 0 0 0 0 0 0 0 0 0 0\n 7.6321 -3.4609 0.0000 R# 0 0 0 0 0 0 0 0 0 0 0 0\n 5.4886 -4.6983 0.0000 R# 0 0 0 0 0 0 0 0 0 0 0 0\n 2 1 1 1 0 0 0\n 4 2 1 0 0 0 0\n 2 3 1 0 0 0 0\n 3 5 2 0 0 0 0\n 3 6 1 0 0 0 0\n 4 7 1 0 0 0 0\nM RGP 2 6 2 7 1\nM END\n",

"monomerType" : "Backbone",

"polymerType" : "PEPTIDE",

"attachmentList" : [ {

"id" : 0,

"alternateId" : "R2-OH",

"label" : "R2",

"capGroupName" : "OH",

"capGroupSMILES" : "O[\*] |$;\_R2$|"

}, {

"id" : 0,

"alternateId" : "R1-H",

"label" : "R1",

"capGroupName" : "H",

"capGroupSMILES" : "[\*][H] |$\_R1;$|"

} ],

"newMonomer" : false,

"adHocMonomer" : false,

"modified" : false,

"attachmentListString" : "R1-H$R2-OH"

}

GET Nucleotides:

This method is returning a list of all nucleotides in the database, using JSON.

PUT Nucleotide:

The client (HELM2NotationToolkit) sends a PUT request with the nucleotide in its JSON-format to the server. An example for this format is given below:

{"positionType":2,"symbol":"test","modified":false,"notation":"test","naturalAnalog":"X","nucleosideNotation":"tes","linkerNotation":"t","baseMonomer":null,"phosphateMonomer":null,"sugarMonomer":null}

{

"positionType" : 2,

"symbol" : "test",

"modified" : false,

"notation" : "test",

"naturalAnalog" : "X",

"nucleosideNotation" : "tes",

"linkerNotation" : "t",

"baseMonomer" : null,

"phosphateMonomer" : null,

"sugarMonomer" : null

}

### How to define which Chemistry Plugin you want to use?

In your own ./helm directory there is the chemistry.property file. Here you can define which Chemistry Engine you want to use. Currrently there are only two different chemistries available: MARVIN Beans and CDK. Note that the default chemistry engine is CDK. To run a specific implementation of the ChemistryToolkit, the class which extends the AbstractChemistryManipulator class has to be given in the chemistry.property file. For example for Marvin, this file has to look like this: *chemistry=org.helm.chemtoolkit.chemaxon.ChemaxonManipulator.*

### How to implement a new Chemistry Plugin in HELM2NotationToolkit?

The chemistry in the HELM2NotationToolkit is now transferred to a single java project, called ChemistryToolkit. Currently two different chemistry engines are available: MARVIN Beans and CDK. You can use these java packages as templates to implement your own chemistry. To run your own chemistry engine, just add your new package to the pom.xml of HELM2NotationToolkit. The chemistry property is your class, which extends the AbstractChemistryManipulator.class.

### How can I use the HELM2NotationToolkit’s REST API?

All accessible REST API methods are included in the HELM2WebService package. A built war-file is available on github for deploying this on a server, for example on tomcat7 or tomcat8. The ./helm directory in the user.home directory has to be added to the server. The address of the HELM2WebService will be {your own IP-address}:8080/WebService/FirstPage.html. The swagger documentation gives an overview of all REST API methods, see {your own IP-address}:8080/WebService/HowToUse.html.

### Which jar-dependencies are needed to use HELM2NotationToolkit.jar in another java-project?

The ChemistryToolkit.jar, ChemistryToolkitMarvin, ChemistryToolkitCDK, HELMNotationParser.jar have to be included. Also the library ‘commons-configuration’ and its dependent jars ‘commons-lang’ and ‘commons-logging’ are necessary to run the HELM2NotationToolkit.jar.

# Some small examples how to use HELM2Notationtoolkit

## How can I check if a HELM input is valid?

There are two possibilities to validate a HELM input in the old or new format.

First option:

*WebService.validateHELM(input);*

If the input is not a valid HELM, it throws an exception. Note that you can also call this method with an xHELM as input.

Second option:

*HELM2Notation helm2notation = HELM2NotationUtils.read(input);*

*Validation.validateNotationObjects(helm2notation);*

If the input isn’t a valid HELM, an exception will be thrown.

## How can I calculate something for a HELM molecule?

Before you can calculate something for a HELM molecule, you have to generate a HELM2Notation object. After that you can use the MoleculeInformation.class doing some calculations or use ExtinctionCoefficient.class to calculate the extinction coefficient of the HELM molecule.

*HELM2Notation helm2notation = HELM2NotationUtils.read(input);*

*MoleculeInformation.getMolecularWeight(helm2notation);*

*Moleculeinformation.getExactMass(helm2notation);*

*MoleculeInformation.getMolecularFormular(helm2notation);*

*ExtinctionCoefficient.getInstance().calculate(helm2notation);*

## How can I generate a standard or canonical HELM?

Before you can generate a standard or canonical HELM, you have to generate a HELM2Notation object. In the next step you can use the HELM1Utils class to generate standard or canonical HELM.

*HELM2Notation helm2notation = HELM2NotationUtils.read(input);*

*HELM1Utils.getStandard (helm2notation);*

*HELM1Utils.getCanonical(helm2notation);*