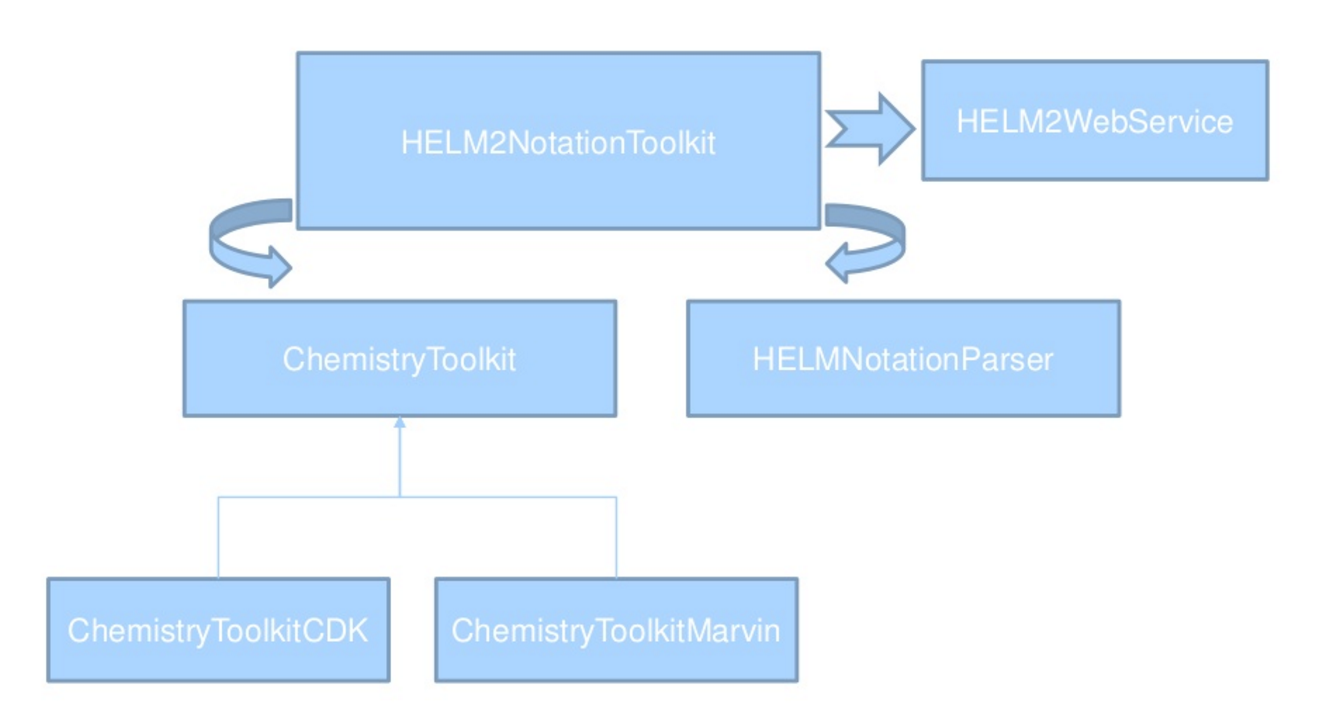
# HELM2NotationToolkit

The new HELM2NotationToolkit is a top-level package for the HELM2 project. It contains the core HELM data model and all necessary functionality for reading, writing and managing HELM structures. These include:

* A chemical engine for calculating simple chemical properties: molecular formula, extinction coefficient and molecular weight.
* Connection manager for managing the connections between simple polymers and monomers within a polymer.
* Utilities for generating HELM images, SMILEs, FASTA, MDL Mol, and xHELM files.
* Polymer-specific utilities for accessing generating sequences from RNA or Peptides
* WebServices features for accessing remote monomer libraries and reading and writing JSON structured monomers.



The diagram above illustrates the HELM API data model and two important frameworks: ChemistryToolkit and the HELMNotationParser. The ChemistryToolkit deals with all atomistic details of HELM macromolecules while the HELMNotationParser provides functionality specific to parsing HELM strings.

# HELM2NotationParser

The HELM2NotationParser is the core package for low-level HELM2 notation parsing, exporting and validation. This package includes methods for converting between HELM1.0 and HELM2.0, and basic calculations of molecular properties such as molecular formula, weight and extinction coefficient.

Convert from HELM1 to HELM2:

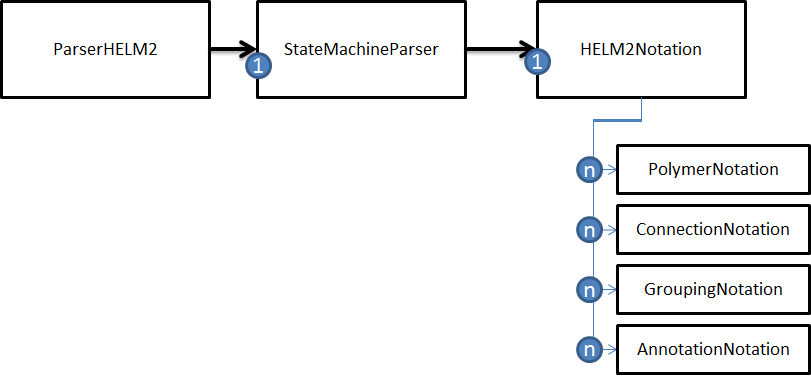
|  |
| --- |
| // convert a HELM1 to HELM2 notation.  String helm1 = "RNA1{[am6]P.R(C)P.R(U)P… etc”...  conv = new ConverterHELM1ToHELM2();  String helm2 = conv.doConvert( **helm1** ); |

**Convert to a ParserHELM2 object for exporting to JSON or HELM2 string.**

|  |
| --- |
| ParserHELM2 parser = new ParserHELM2();  parser.parse(helm2);  String json\_helm2 = parser.getJSON(); |

## API Structure: Polymers

The ParserHELM2 class serves as the central object for all HELM and HELM2 parsing needs. As of HELM2 supported parsing operations are broken into four main packages: annotations, connections, groupings and polymer.



One of the most important operations HELM users require is programmatic iteration over all polymers and monomers in a particular HELM2 notation string. This can be accomplished by using the HELM2Notation object via the ParserHELM2 object as illustrated in the diagram above. The HELM2Notation object contains methods for accessing lists for all four notation subgroups.

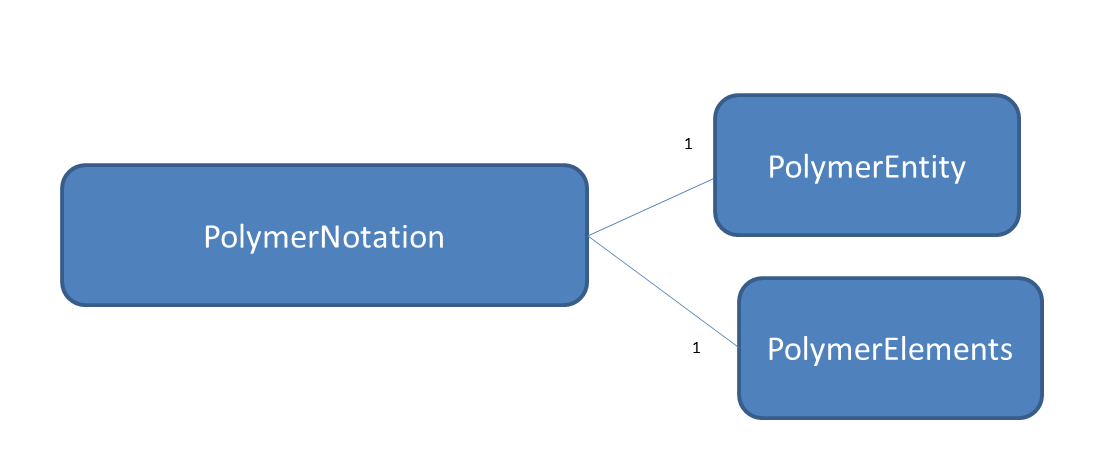
The following illustrates how to loop over all PolymerNotation objects in a HELM2 string:

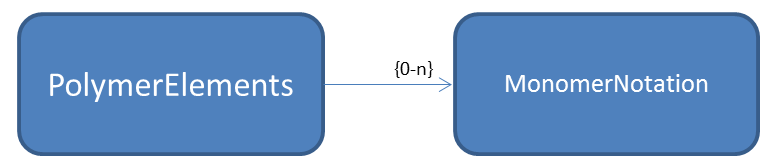
|  |
| --- |
| List<PolymerNotation> polymer\_notation\_list = helm2\_notationstr.getListOfPolymers();  for ( PolymerNotation polymer\_notation : polymer\_notation\_list )  {  String id = polymer\_notation.getPolymerID().getId();  String type = polymer\_notation.getPolymerID().getType(); |

In the above example the “id” and “type” would be PEPTIDE1 and PEPTIDE respectively for the following HELM string:

*PEPTIDE1{M.A.L.W.M.R.L.L.P.L.L.A.L.L.A.L.W.G.P.D.P.A.A.A.F.V.N.Q.H.L.C.G.S.H.L.V.E.A.L.Y.L.V.C.G.E.R.G.F.F.Y.T.P.K.T.R.R.E.A.E.D.L.Q.G.S.L.Q.P.L.A.L.E.G.S.L.Q.K.R.G.I.V.E.Q.C.C.T.S.I.C.S.L.Y.Q.L.E.N.Y.C.N}$$$$V2.0*

The PolymerNotation object contains a reference to two important polymer objects: the PolymerEntity and the PolymerElements. The Entity object is the HELMEntity that manages the type and id of the polymer wherease the PolymerElements object contains a reference to all the MonomerNotation elements.





The PolymerElements object contains a list of its monomers as *MonomerNotation* objects. The code for this is below shows how to access these monomers units:

|  |
| --- |
| PolymerElements polymer\_element = polymer\_notation\_object.getPolymerElements();  List<MonomerNotation> monomer\_notation\_list = polymer\_element.getListOfElements();  for ( MonomerNotation monomer : monomer\_notation\_list){  ...etc.... |

## API Structure: Monomers

## 

The *Monomer* object contains several attributes for managing the monomer identity, polymer type (e.g. , polymer attachment bonds, and chemical structure. In the HELM2NotationParser the monomer unit is parsed into a MonomerNotation object where the “unit” of the monomer is the monomer ID.

To get the monomer ID for each monomer in a polymer:

|  |
| --- |
| PolymerNotation polymerNode = ..etc...  String polymerType = polymerNode.getPolymerID().getType();  if (polymerType.equals(Monomer.**NUCLIEC\_ACID\_POLYMER\_TYPE**)) {  **List<MonomerNotation> monomers = polymerNode.getListMonomers();**  **for(MonomerNotation monomer : monomers ){**  **monomer.getUnit();** |

## HELM2Notation with Complex polymers

The ParserHELM2 class is primarily used to construct HELM2Notation, acore HELM object that encapsulate the four main subcomponents of any complex polymer. These are groups include: (1) simple polymer (2) connection (3) group and (4) annotation, and are separated by the ‘**$**’ character:

i.e. ListOfSimplePolymers$ListOfConnections$ListOfSimplePolymerGroups$ExtendedAnnotations$

### Simple Polymers

The HELM2Notation object contains methods for iterating over each subgroup. For example, the following shows how to parse and iterate over all monomers within a helm string:

|  |
| --- |
| List<PolymerNotation> polymerNodes = helm2notation.getListOfPolymers();  for (PolymerNotation polymerNode : polymerNodes) {  String polymerType = polymerNode.getPolymerID().getType();  if (polymerType.equals(Monomer.**NUCLIEC\_ACID\_POLYMER\_TYPE**)) {  **List<MonomerNotation> monomers = polymerNode.getListMonomers();**  ..etc….  }  } |

### Connections

The “connections” section defines the covalent bonds between any two simple polymers. The method HELM2Notation.**getListOfConnections**() returns a list of ConnectionNotation objects with references to the source and target HELM entities, i.e. polymers or group objects.

For example if two polymers of type RNA1 and CHEM1 are connected by monomers at position 21 and 1 respectively the following HELM2 connection notation would apply:

“RNA1,CHEM1,21:R2‐1:R1”

You may programmatically access this by doing the following:

|  |
| --- |
| List<ConnectionNotation> polymerConnections = helm2notation.getListOfConnections();  for (ConnectionNotation polymerConnection : polymerConnections) {  HELMEntity helmSource\_rna1 = polymerConnections.getSourceId()  HELMEntity helmTarget\_chem1 = polymerConnections.getTargetId()  etc… |

### Groupings

The “Grouping” section of a complex polymer provides a way to group simple polymers and assign relationships to other objects within the HELM string. For example the recently approved drug Kadcyla is an antibody called Trastuzumab that contains a small molecule conjugate (emtansine) that binds to lysine residues with an average frequency of 3.5 per antibody. The grouping notation describes the pseudo-specific nature of this molecule:

**NOTE**: peptide sequences are removed in order to highlight the grouping structure

PEPTIDE1{...light\_chain\_sequence...}|PEPTIDE2{...heavy\_chain\_sequence..}|PEPTIDE3{...light\_chain\_sequence....}|PEPTIDE4{..heavy\_chain\_sequence...}|CHEM1{[MCC]}$PEPTIDE1,PEPTIDE1,134:R3-194:R3|PEPTIDE2,PEPTIDE3,229:R3-229:R3|PEPTIDE4,PEPTIDE4,134:R3-194:R3|PEPTIDE1,PEPTIDE1,23:R3-88:R3|PEPTIDE3,PEPTIDE4,223:R3-214:R3|PEPTIDE3,PEPTIDE3,370:R3-428:R3|PEPTIDE2,PEPTIDE2,264:R3-324:R3|PEPTIDE3,PEPTIDE3,147:R3-203:R3|PEPTIDE4,PEPTIDE4,23:R3-88:R3|PEPTIDE2,PEPTIDE1,223:R3-214:R3|PEPTIDE2,PEPTIDE2,22:R3-96:R3|PEPTIDE3,PEPTIDE3,22:R3-96:R3|PEPTIDE2,PEPTIDE2,147:R3-203:R3|PEPTIDE2,PEPTIDE2,370:R3-428:R3|PEPTIDE3,PEPTIDE3,264:R3-324:R3|PEPTIDE2,PEPTIDE3,232:R3-232:R3$G1(PEPTIDE1+PEPTIDE2+PEPTIDE3+PEPTIDE4)|G2(CHEM1+G1:3.4)$$V2.0

In this example two groups, G1 and G2, are defined at the end of the HELM string:

….G1(PEPTIDE1+PEPTIDE2+PEPTIDE3+PEPTIDE4)|G2(CHEM1+G1:3.4)$$V2.0

G1 declares a group that contains the entire antibody structure while G2 describes a mixture of elements indicated by the ‘+’ notation where CHEM1 ‘and’ G1 exist together with a respective ratio of 3.4.

The following code shows how to get the groupings and ambiguity values from the Kadcyla HELM string:

|  |
| --- |
| HELM2Notation kadcyla\_helm2\_notation = helm2.getHELM2Notation();  List<GroupingNotation> groupings = kadcyla\_helm2\_notation.getListOfGroupings();  for ( GroupingNotation gn : groupings )  {  GroupingAmbiguity ambiguity = gn.getAmbiguity();  System.out.println( " Ambiguity = "+ gn.getAmbiguity().toHELM2() );  List<GroupingElement> grouping\_element\_list = ambiguity.getListOfElements();  for ( GroupingElement ge : grouping\_element\_list )  {  HELMEntity entity = ge.getID();  String type = entity.getType ();  String id = entity.getId();  List<Double> value\_list = ge.getValue();  for ( Double value : value\_list )  {  System.out.println ("\t"+ id + " Type:" + type + " Value:" + value ); |

The above code fragment produces the following output:

*Ambiguity = PEPTIDE1+PEPTIDE2+PEPTIDE3+PEPTIDE4*

*PEPTIDE1 Type:PEPTIDE Value:1.0*

*PEPTIDE2 Type:PEPTIDE Value:1.0*

*PEPTIDE3 Type:PEPTIDE Value:1.0*

*PEPTIDE4 Type:PEPTIDE Value:1.0*

*Ambiguity = CHEM1+G1:3.5*

*CHEM1 Type:CHEM Value:1.0*

*G1 Type:G Value:3.5*

### Annotations

The final section of the complex polymer HELM string contains the annotations. This section is a free-form, name-value representation for more unstructured features. For example, Mipomersen is an antisense oligonucleotide that was approved by the FDA for Homozygous Familial Hypercholesterolemia. If you wanted to register this drug with these annotations you could do the following:

RNA1{[moe](G)[sp].[moe]([m5C])[sp].[moe]([m5C])[sp].[moe](T)[sp].[moe]([m5C])[sp].[dr](A)[sp].[dr](G)[sp].[dr](T)[sp].[dr]([m5C])[sp].[dr](T)[sp].[dr](G)[sp].[dr]([m5C])[sp].[dr](T)[sp].[dr](T)[sp].[dr]([m5C])[sp].[moe](G)[sp].[moe]([m5C])[sp].[moe](A)[sp].[moe]([m5C])[sp].[moe]([m5C])[sp]}$$$**{“Approved Indication”:”Homozygous Familial Hypercholesterolemia (HoFH)”}**$V2.0

Accessing these annotations programmatically is the same as the previous examples:

|  |
| --- |
| List<AnnotationNotation> annotations = mipmersen\_helm2\_notation.getListOfAnnotations();  for( AnnotationNotation annotation : annotations )  {  String annotation\_value = annotation.getAnnotation();  System.out.println( " Annotation value : " + annotation\_value );  } |

The output produced by the code above is:

*Annotation value : {“Approved Indication”:”Homozygous Familial Hypercholesterolemia (HoFH)”}*

## Command-line utility

This package contains a command line tool for quickly converting HELMv1.0 to HELMv2.0 and to export HELM structures to JSON format.

To convert HELM to HELM2 use the “translate” option.

|  |
| --- |
| Example:  **>** java -jar helm2parser.jar -inputHELM ./filename.txt -output JSON  *To convert HELM1 to HELM2 use the “–translate” option.*  **>** java –jar helm2parser.jar –inputHELM [file] -output [HELM2/JSON] -translate |

# 

# ChemistryToolkit

The ChemistryToolkit provides a general framework for specific implementations of the chemical nature of monomers. For example the AbstractChemistryManipulator class provides methods for implementing SMILES validation and bitmap image renderings. Depending on the implementation that is specified in the configuration of the HELM2NotationToolkit the ChemistryManipulator will have different renderings of the same monomer. This allows users to “plug-in” third-party chemistry software at “run-time”.

For example two different venders might represent the same monomer differently:

|  |  |
| --- | --- |
| OpenScience Project | ChemAxon |

The HELM2Notation toolkit may be configured to point to either implementation without a code compilation step, thus greatly simplifying the integration of third party solutions.

Currently there are two implementations of the ChemistryToolkit released to the Pistoia Alliance source repository (<https://github.com/PistoiaHELM>) 1) ChemistryToolkitMarvin from ChemAxon (<https://www.chemaxon.com/>) and 2) ChemistryToolkitCDK (from the OpenScience project (<http://www.openscience.org/blog/>)