**ͱ HETA cheat sheet**

**Syntax**

**// Component**

**// Base statement and annotation, semicolon is required**

**{**

**title:** <String>**,**  // Human readable name of component

**notes:** <String>**,** // any notes, supports Markdown

**tags:** <String[]>**,** // tags for component

**aux:** <Dict> // Any user defined properties

**};**

*''' Notes '''*

sp1::**cmd @Component ‘Title’ {**

**tags:** **[**a**,** b**,** c**],**

**aux: { }**

**};**

**// Record <= Size <= Component**

**// describes value which can change its value in time**

**p1 @Record {**

**output:** <Boolean>**,**  // display on plot

**boundary:** <Boolean>**,**  // if true it can’t be changed by @Process

**units:** <UnitExpr>**,**  // units describing the value

**assignments:** **{**

**[<ID>]:** <MathExpr>**,** // describes value changes

...

**}**

**};**

// record assignments

**p1 .=** <MathExpr>**;** // calculated at start\_ switcher

**p1 :=** <MathExpr>**;**  // calculated at ode\_ switcher

**p1 [sw1]=** <MathExpr>**;** // calculated at sw1 switcher

**// Process <= Record <= Size <= Component**

**// change record values using ODEs**

**pr1 @Process {**

**actors:** <ProcessExpr>/<Actor[]> // records to change

**};**

**// ProcessExpr format**

1\*A = 2\*B + 3\*C

A => 2B + 3C // same as >, mark as irreversible

A <=> B + B + 3C // same as <>, mark as reversible

**// TimeSwitcher <= Switcher <= Component**

**// run reassignment of records at specific time points**

**sw1 @TimeSwitcher {**

**start:** <Number>/<ID>**,** // required, when switcher is called

**period:** <Number>/<ID>**,** // >0, if set, the switcher period

**stop:** < Number>/<ID>, // time when stop the repeat

**active:** <Boolean> // if false the switcher is ignored

**};**

**// СSwitcher <= Switcher <= Component**

**// run reassignment of records at numeric trigger**

**sw1 @CSwitcher {**

**trigger:** <MathExpr>**,** // required, numeric result

**active:** <Boolean> // if false the switcher is ignored

**};**

**// DSwitcher <= Switcher <= Component**

**// run reassignment of records at boolean trigger**

**sw2 @DSwitcher {**

**trigger:** <MathExpr>**,** // required, boolean result

**active:** <Boolean> // if false the switcher is ignored

**};**

**// Const <= Size <= Component**

**// numerical value which does not change in time**

**k1 @Const {**

**units:** <UnitExpr>**,**  // units describing the value

**num**: <Number> // required, constant value

**};**

// example

**k1 @Const** = 1.1**;** // = symbol describes num value

**// Compartment <= Record <= Size <= Component**

**// describes volumes where Species instances are located**

**comp1 @Compartment {**

// no specific properties

**};**

**// Species <= Record<= Size <= Component**

**// describes particles in some location**

**S1 @Species {**

**isAmount:** <Boolean>**,** // if not concentration

**compartment:** <ID> // required, ref to Compartment

**};**

**// Reaction <= Process <= Record <= Size <= Component**

**// As Process, but all target references should be Species**

**r1 @Reaction {**

**actors:** <ProcessExpr>/<Reactant[]>**,** // ref to Species

**modifiers:** <Modifier[]>/<Id[]> // ref to Species

**};**

**Actions**

**// Add new unit definition**

**unit1 #defineUnit {**

**units**: <UnitsExpr>/<UnitDefComponent[]>**,** // unit components

**};**

// creates a new component. **Default** if class presents.

**#insert {**

**id:** <ID>**,** // identifier inside namespace

**space:** <ID>**,** // identifier of parent namespace

**class:** <String> // class name

**};**

// updates the component. **Default** if class does not present.

**#update {**

**id:** <ID>**,** // identifier inside namespace

**space:** <ID>// identifier of parent namespace

**};**

// removes the component. Error if it doesn’t exist.

**#delete {**

**id:** <ID>**,** // identifier inside namespace

**space:** <ID>// identifier of parent namespace

**};**

// Create namespace “one”.

**#setNS** one::**\*;**

// clone all components from namespace “source” to “one”.

**#importNS** one::\* **{**

**fromSpace: source,**

**prefix: “”,**

**suffix: “”,**

**rename:** <Dict> // renaming rules

**};**

// clone component “k1” from namespace “source” to “one”

**#import** one::**k1 { fromId: k1, fromSpace: source };**

// include the content from external file

**#include { source: ./model.heta, type: heta };**

**include statement**

**// semicolon at the end is not required**

**include** <String> **type** <String> **with** <Dictionary>

**// include heta file**

**include** ./addon.heta

**// include table sheet**

**include** ./table.xlsx **type** table **with** **{**

**sheet:** 2**,**  // number of sheet starting from 0

**omitRows:** 3 // empty rows between header and components

**}**

**// include JSON notation of components**

**include** ./addon.json **type** json

**// include YAML notation of components**

**include** ./addon.yml **type** yaml

**// include SBML**

**include** ./model.xml **type** sbml

**QSP units (loaded from qsp-units.heta)**

fmole , pmole, nmole, umole, mmole

fM, pM, nM, uM, mM, M, kM

fL, pL, nL, uL, mL, dL, L

fs, ps, ns, us, ms, s

h, week

fg, pg, ng, ug, mg, g, kg

kat

cell, kcell

cal, kcal

fm, pm, nm, um, mm, cm, m

UL

percent

**Base units**

ampere, gram, katal, metre, watt

becquerel, gray, kelvin, mole, siemens, weber

candela, henry, kilogram, newton, sievert

coulomb, hertz, litre, ohm, steradian

dimensionless, item, lumen, pascal, tesla

farad, joule, lux, radian, volt

second, minute, hour, day, year

**#export action in Heta compiler**

**// General form of export**

**#export {**

**format**: <String>, // one of the supported formats

**filepath**: <String>, // directory path where to export

**spaceFilter**: <String> // regular expression to filter namespaces

**};**

**// Internal qs3p JSON format**

**#export {**

**format**: JSON,

**filepath**: <String>, // directory path where to export

**omit**: <String>[] // array of component properties to skip

**};**

**// Internal qs3p YAML format**

**#export**  **{**

**format**: YAML,

**filepath:** <String>, // directory path where to export

**omit**: <String>[] // array of component properties to skip

**};**

**// Export to DBSolveOptimum .SLV**

**#export {**

**format**: DBSolve,

**filepath**: <String>**,** // directory path where to export

**powTransform:** keep/operator/function**,** // x^y or pow(x, y)

**groupConstBy:** <String>**,** // sort parameters, default tags[0]

**version:** 25/26 // version of SLV file

**};**

**// Export to SBML format**

**#export {**

**format**: SBML,

**filepath**: <String>**,** // directory path where to export

**version:** <String> // SBML version, default: L2V4

**};**

**// Export to Metrum mrgsolve .CPP model format**

**#export** **{**

**format**: Mrgsolve,

**filepath**: <String>// directory path where to export

**};**

**// Export to Matlab/Simbiology .M file**

**#export** **{**

**format**: Simbio,

**filepath**: <String>// directory path where to export

**};**

**// Export to table**

**#export {**

**format**: Table,

**filepath**: <String>**,** // directory path where to export

**omitRows:** <Number>**,** // empty rows

**omit**: <String>[] // array of component properties to skip

**splitByClass:** <Boolean>**,** // split to several sheets

**bookType:** <String> // xlsx, csv, etc

**};**

**// Export to Matlab.M file**

**#export {**

**format**: Matlab,

**filepath**: <String>// directory path where to export

**};**

**// Export to Julia file for usage in HetaSimulator**

**#export** **{**

**format**: Julia,

**filepath**: <String>// directory path where to export

**};**

**// Export to DOT scheme format**

**#export** **{**

**format**: Dot,

**filepath**: <String>// directory path where to export

**};**

**// Export to Heta code format**

**#export** **{**

**format**: HetaCode,

**filepath**: <String>// directory path where to export

**};**

<https://hetalang.github.io/>