**ͱ HETA cheat sheet**

**Syntax**

**// Component**

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

**{**

**id:** <ID>**,**

**space:** <ID>**,**

**class:** <String>**,**  // class of component

**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 {**

**boundary:** <Boolean>**,**  // if can 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

**p1 []=** <MathExpr>**;**  // the same as .=

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

**// change record values using ODEs**

**pr1 @Process {**

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

**};**

**// correct ProcessExpr**

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

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

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

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

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

**sw1 @TimeSwitcher {**

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

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

**repeatCount:** <Number>, // times to repeat

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

**};**

**// ContinuousSwitcher <= Switcher <= Component**

**// run reassignment of records at condition**

**sw1 @ContinuousSwitcher {**

**condition:** <ID>// required, ref to Record

**};**

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

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

**k1 @Const {**

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

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

**free:** <Boolean> // if true the value is marked for fitting

**};**

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

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

**comp1 @Compartment {**

// no specific properties

**};**

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

**// describes particles in some location**

**s @Species {**

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

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

**};**

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

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

**r1 @Reaction {**

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

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

**};**

**// JSONExport <= Export <= Component**

**// Internal qs3p JSON format**

**json1@JSONExport** **{**

**filepath**: <String> // name of file or directory to export

**};**

**// YAMLExport <= Export <= Component**

**// Internal qs3p YAML format**

**yaml1** **@YAMLExport** **{**

**filepath:** <String> // name of file or directory to export

**};**

**// SLVExport <= Export <= Component**

**// Export to DBSolveOptimum .SLV**

**slv1 @SLVExport** **{**

**filepath**: <String>**,** // name of file or directory to export

**eventsOff:** <Boolean> // of *true* events will not been exported

**};**

**// SBMLExport <= Export <= Component**

**// Export to SBML format**

**sbml1 @SBMLExport** **{**

**filepath**: <String>**,** // name of file or directory to export

**version:** <String> // default: L2V4, currently supports only L2V4

**};**

**// MrgsolveExport <= Export <= Component**

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

**mrg1 @MrgsolveExport** **{**

**filepath**: <String>// name of file or directory to export

**};**

**// SimbioExport <= Export <= Component**

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

**simbio1 @SimbioExport** **{**

**filepath**: <String>**,** // name of file or directory to export

**};**

**// XLSXExport <= Export <= Component**

**// Export to Excel file**

**xlsx1 @XLSXExport** **{**

**filepath**: <String>**,** // name of file or directory to export

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

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

**};**

**// UnitDef <= Component**

**// Add new base unit definition**

**unit1 @UnitDef** **{**

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

**};**

**Actions**

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

**#insert …**

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

**#update …**

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

**#delete …**

**include statement**

**// base syntax “file relative path” / ”module type” / ”options”**

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

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

**// include heta file**

**include** ./addon.heta

**// include xlsx sheet**

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

**sheet:** 2**,**  // number of sheet

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

**waitSec:** 30 // wait before throw an error, only for large xlsx

**}**

**// include JSON notation of components**

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

**// include YAML notation of components**

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