

# **Mod Code Generator Manual**

Document license (based upon MIT License):

Permission is hereby granted, free of charge, to obtaining a copy of this document file (the "Document"), to deal in the Document without restriction, including without limitation the rights to use, copy, modify, merge, publish and distribute copies of the Document, and to permit persons to whom the Document is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Document.

THE DOCUMENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENT OR THE USE OR OTHER DEALINGS IN THE DOCUMENT.

## Table of Contents

1 Introduction.....	4
2 License.....	4
3 Installation.....	4
3.1 Python Installation.....	4
3.2 MCG Installation.....	5
3.3 Modelio Installation.....	6
4 MCG Overview.....	6
5 Code of practice for modeling in Modelio.....	8
5.1 Model Creation.....	9
Create New Project.....	9
Remove Unused Elements.....	9
Rename Model Package.....	10
5.2 Type Package Creation.....	10
Create Type Package.....	10
Create Model Types.....	10
5.3 Model Component Creation.....	11
Create New Component.....	11
Create Component Interfaces.....	11
Create Component Interface Signals.....	12
Create Component Activity.....	14
Define Component Activity.....	14
Create Next Components.....	16
5.4 Model Package Creation.....	16
Create Package Interfaces.....	16
Create Package Interface Signals and Structures.....	17
Create Package Activity.....	17
Define Package Activity.....	18
5.5 Modeling Rules.....	18
Model Rules.....	18
Component Interface Rules.....	19
Component Activity Rules.....	20
Package Interface Rules.....	21
Package Activity Rules.....	22
Naming Convention Rules.....	22
5.6 Naming Convention.....	23
5.7 List of Actions.....	23
6 How to run MCG.....	23
6.1 Converter Component (MCG CC).....	24
Run Converter Component.....	24
Check Converter Component Outputs.....	25
6.2 Code Generator Component (MCG CGC).....	25
Run Code Generator Component.....	25
Check Code Generator Component Outputs.....	25
7 Configuration File Syntax and Format.....	25

# 1 Introduction

The Mod Code Generator (MCG) program is a code generator tool, written in Python language, which allows to generate code from Modelio environment. You can create a model in Modelio environment following section Code of practice for modeling in Modelio and then generate C code representation of your model as explained in section How to run MCG. A short overview of the MCG program is available in MCG Overview section.

Before you start to use the MCG program please familiarize yourself with license terms and conditions, summarized under below section License.

## 2 License

The MCG program is available under the terms of the GNU General Public License, either version 3 of the license, or (at your option) any later version.

Under Section 7 of GPL version 3, you are granted additional permissions described in the MCG Output Exception, version 1.

You should have received a copy of the GNU General Public License and the MCG Output Exception along with this program, look for **LICENSE** and **MCG OUTPUT EXCEPTION** files available under the MCG release.

**If you wish to use the MCG program, then you shall read and comply with terms and conditions of above license and permission.**

Apart from the above, the MCG program works with:

- Python, available under Python Software Foundation License Version 2,
- Modelio, available under GNU General Public License either version 3 of the license, or (at your option) any later version,

**therefore you shall in addition read and comply with terms and conditions of above licenses in order to use Python and Modelio programs.**

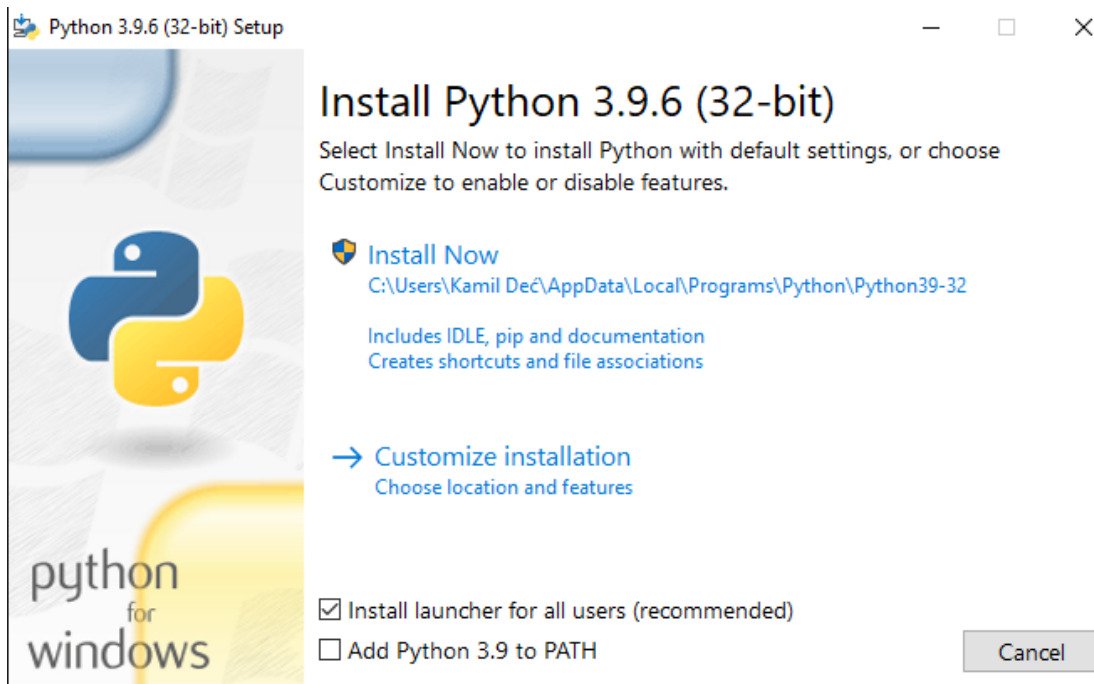
You should have received a copy of above licenses along with their installation packages, which are not enclosed directly with the MCG release. Please follow Installation to see how to obtain installation packages of both Python and Modelio releases.

## 3 Installation

### 3.1 Python Installation

The MCG program requires Python package to run, therefore please install Python release 3.9.6 from <https://www.python.org/>.

During installation make sure to select **Install launcher for all users (recommended)**, because Python launcher will be used to run the MCG program with correct Python version. More information about Python installation on Windows OS and Python Launcher can be found under <https://docs.python.org/3.9/using/windows.html#> and <https://docs.python.org/3.9/using/windows.html#launcher>.



Once required Python release is installed, please open a command line window and type `py -0` to see a list of available Python releases on your computer. The list should contain "-3.9-xx", where "xx" is equal to either 32 or 64 depending whether you installed 32 or 64 bit version of Python release.

```
C:\Users\Kamil Deć>py -0
Installed Pythons found by py Launcher for Windows
-3.9-64 *
-3.9-32
```

Further please check if correct Python version is available from the command line. To do this please type `py -3.9-xx --version` in the command line window. You should receive "Python 3.9.6" in reply.

```
C:\Users\Kamil Deć>py -3.9-64 --version
Python 3.9.6
```

## 3.2 MCG Installation

To install the MCG simply copy MCG folder from this release to your hard drive and create following environment variables with paths to required MCG components (i.e. Converter Component and Code Generation Component):

Table 3.2.1: The MCG Environment Variables

Variable	Value
MCG_CC	py -3.9-xx "<mcg installation directory>\MCG_CC\ mcg_cc_main.py"

MCG_CGC	py -3.9-xx "<mcg installation directory>\MCG_CGC\ mcg_cgc_main.py"
---------	---

, where "xx" is equal to either 32 or 64 depending whether you installed 32 or 64 bit version of Python release. Please open a command line window to check if created environment variables work correctly.

Type `%MCG_CC%` to check if the Converter Component is invoked:

```
C:\Users\Kamil Deč>%MCG_CC%

Mod Code Generator (MCG)
Copyright (C) 2021 Kamil Deč github.com/deckamil
This is Converter Component (CC) of Mod Code Generator (MCG)

License GPLv3+: GNU GPL version 3 or later.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Incorrect number of command line arguments, MCG CC process cancelled.
Usage: python mcg_cc_main.py "<model_dir_path>" "<output_dir_path>"
Arguments:
    <model_dir_path>      Path to model directory, where all catalogs with .exml files are stored
    <output_dir_path>     Path to output directory, where results from MCG will be saved

Keep specific order of arguments, as pointed in usage above.
See Mod Code Generator Manual for further information.
```

Type `%MCG_CGC%` to check if the Code Generator Component is invoked:

**TBD**

### 3.3 Modelio Installation

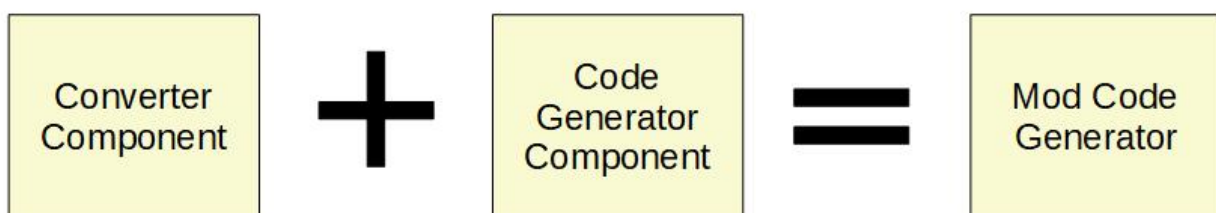
The Modelio environment is optional. If you wish to use only the CGC component of the MCG program and create configuration file on your own, then you can skip installation of Modelio environment.

However, to use full capabilities of the MCG and use both the CC and the CGC components, please install Modelio release 4.1.0 from <https://www.modelio.org/>.

## 4 MCG Overview

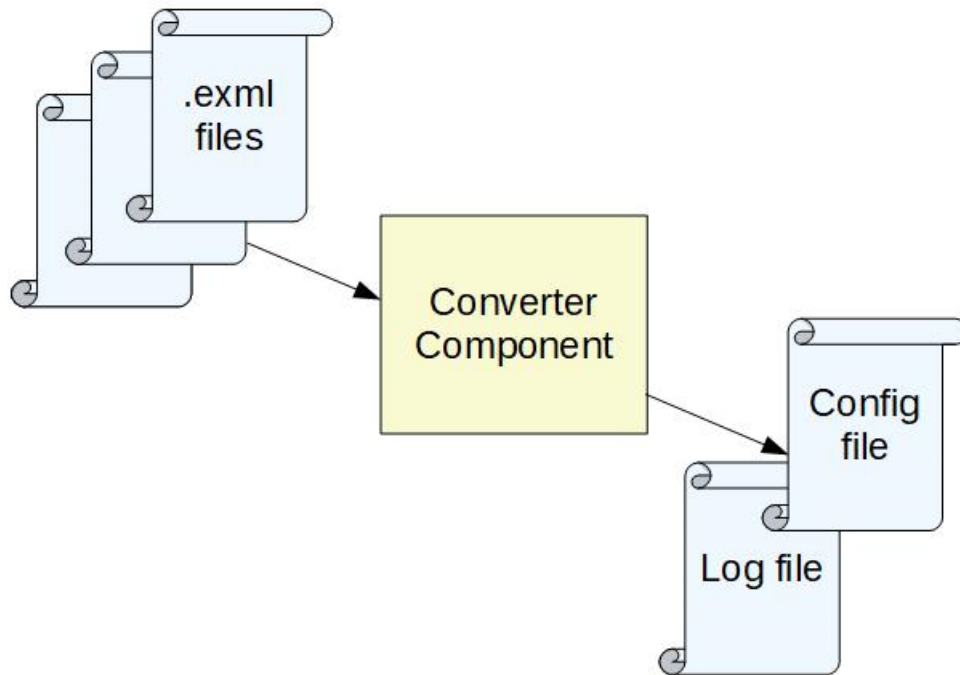
The MCG program consists of two main components (subprograms):

- the Converter Component (CC), which is responsible for conversion of a model created within the Modelio environment into configuration file,
- the Code Generator Component (CGC), which is responsible for code generation from the configuration file.

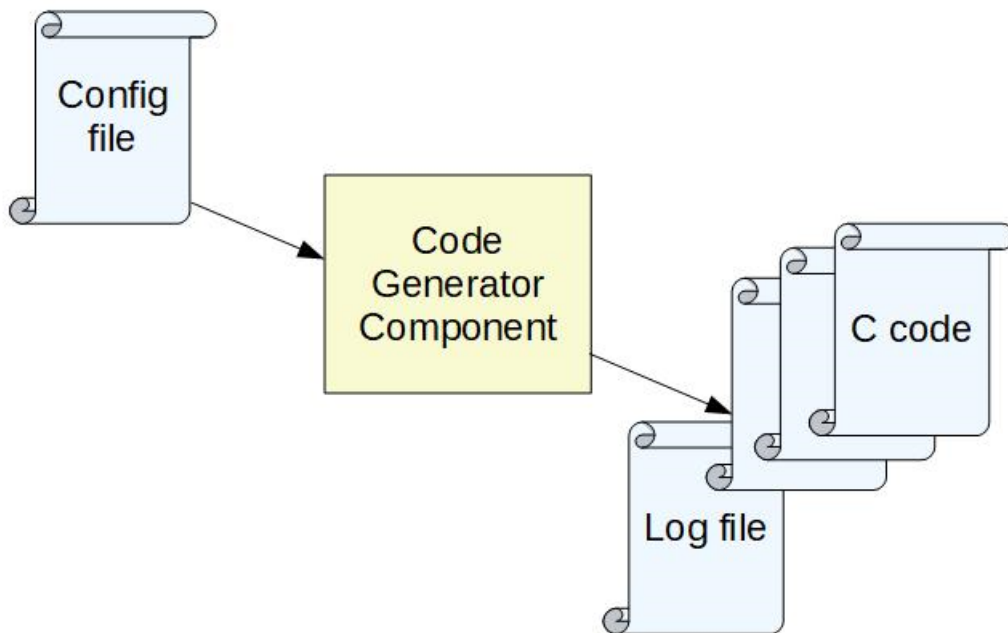


Each component is run independently. In order to run the MCG components please follow How to run MCG section.

When you work on your model in Modelio environment, the model details are saved and stored in set of .exml files. The MCG CC subprogram reads model details from .exml files (model signals, connections between model elements, interactions between signals, etc.) and then generates a configuration file along with the MCG CC log file.



The MCG CGC subprogram takes the configuration file and base on it generates C code representation of the model and as well the MCG CGC log file. The configuration file contains details about model interfaces, data flow and data interactions in specific syntax and format, described in section Configuration File Syntax and Format.



There is a reason why the MCG CC and CGC are separate, standalone subprograms. If you wish, you can create configuration file on your own without engaging the MCG CC subprogram. You have also possibility to replace the MCG CC with your own converter subprogram, which could convert model data from different modeling environment than Modelio. As long as that alternative converter subprogram will deliver configuration file in required syntax and format, the MCG CGC subprogram will be able to generate C code.

## 5 Code of practice for modeling in Modelio

Although Modelio itself is used mainly as UML modeling tool, the modeling environment will be used in different way to allow code generation with the MCG, i.e. modeling will be focused more on data flow through model elements and data interaction (called also data processing), than on representation of model structure and its elements.

The main objective of model package is to define couplings between different model components (data flow from one component to another). The model package is used also to define main input and output interface of entire model.

Each model component is treated as a separate module and MCG will generate separate C source file for each component. Components are used to define operations between signals (also known as actions).

Type package is used only to define allowed types of signals, which will be used later to define interface signals.

Please see below code of practice for modeling in Modelio environment, which will give you some introduction over modeling rules and as well please see Modeling Rules where you can find list of rules that you have to follow during modeling.



## 5.1 Model Creation

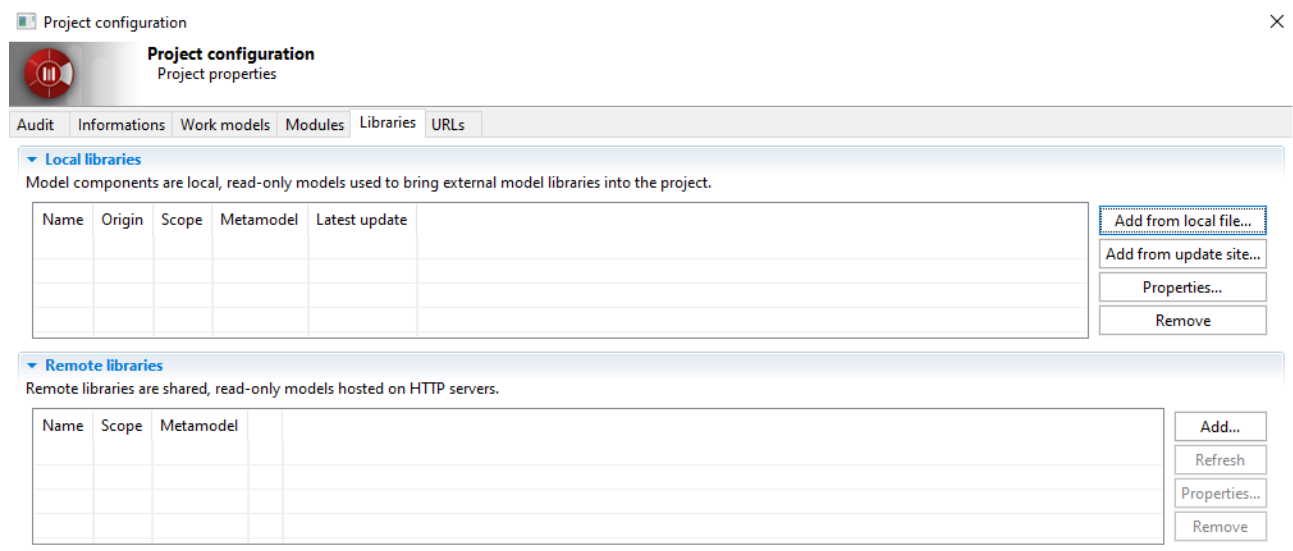
### Create New Project

Open Modelio environment, then select **File → Switch** workspace and select directory, where you would like to place new model.

After that select **File → Create a project**. In **Project name** property type desired project name, then click **Create the project**.

### Remove Unused Elements

Select **Configuration → Libraries**, then remove all **Local libraries** and **Remote libraries**. After that go to **Modules** tab and disable all modules, then close **Project configuration** window.



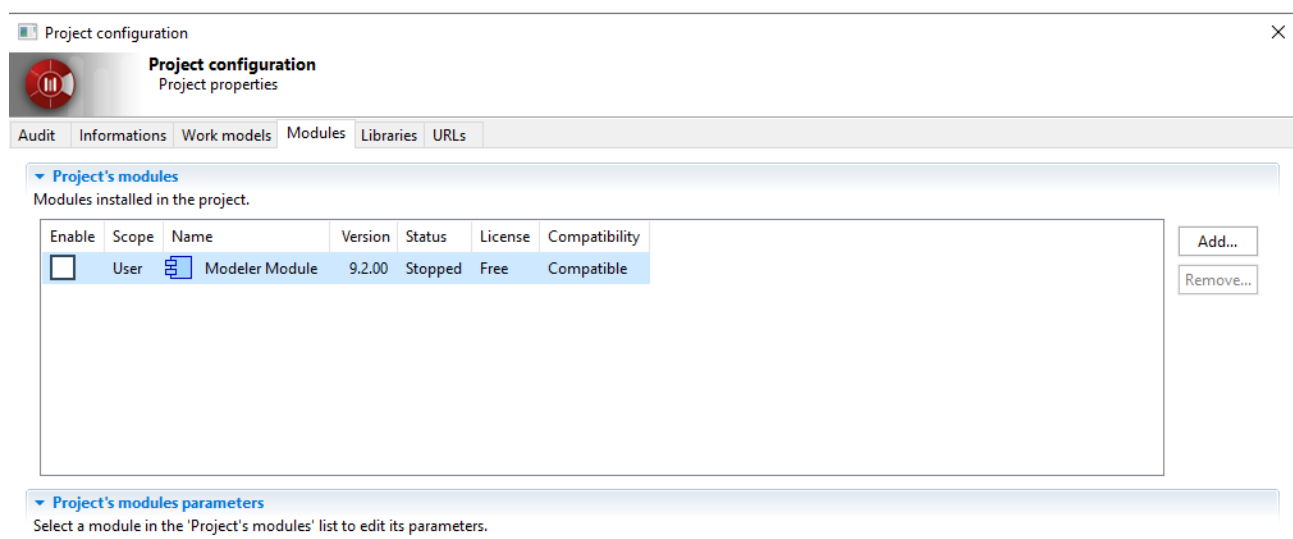
Project configuration window, Libraries tab. The window shows two sections: Local libraries and Remote libraries. Local libraries are described as model components that are local, read-only models used to bring external model libraries into the project. Remote libraries are described as shared, read-only models hosted on HTTP servers.

Name	Origin	Scope	Metamodel	Latest update

Buttons: Add from local file..., Add from update site..., Properties..., Remove

Name	Scope	Metamodel

Buttons: Add..., Refresh, Properties..., Remove



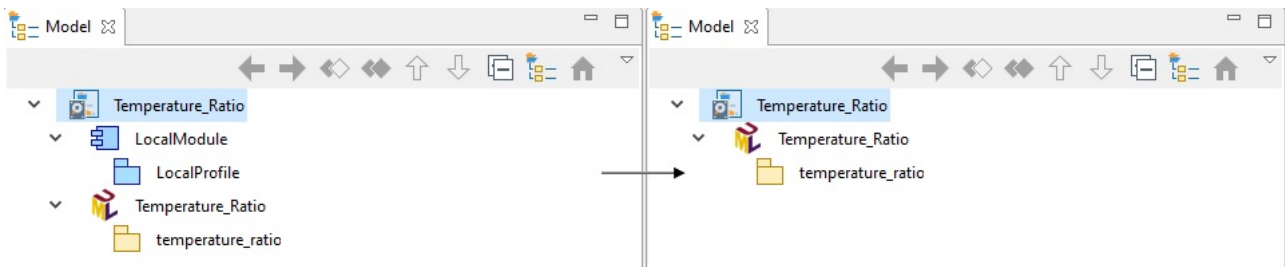
Project configuration window, Modules tab. The window shows the Project's modules section, which lists modules installed in the project.

Enable	Scope	Name	Version	Status	License	Compatibility
<input type="checkbox"/>	User	Modeler Module	9.2.00	Stopped	Free	Compatible

Buttons: Add..., Remove...

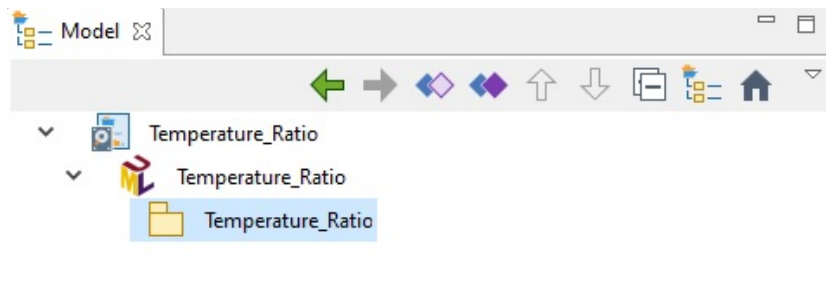
Project's modules parameters section: Select a module in the 'Project's modules' list to edit its parameters.

Expand all model elements in model tree. Find and remove **LocalModule** element.



## Rename Model Package.

Locate model package under UML sub-project element, click on it and press F2 on keyboard, then type desired model package name.



## 5.2 Type Package Creation

### Create Type Package

Right-click on UML sub-project element and select **Create element → Package**. Click on new package, press F2 on keyboard and name it as Types.

### Create Model Types

Define all data types listed under column **Types** in Table 5.2.1: Applicable Data Types.

Right-click on type package and select **Create element → Data Type**. Click on new data type, press F2 on keyboard and name new data type as per column **Type**. Repeat the process for the rest of required data types.

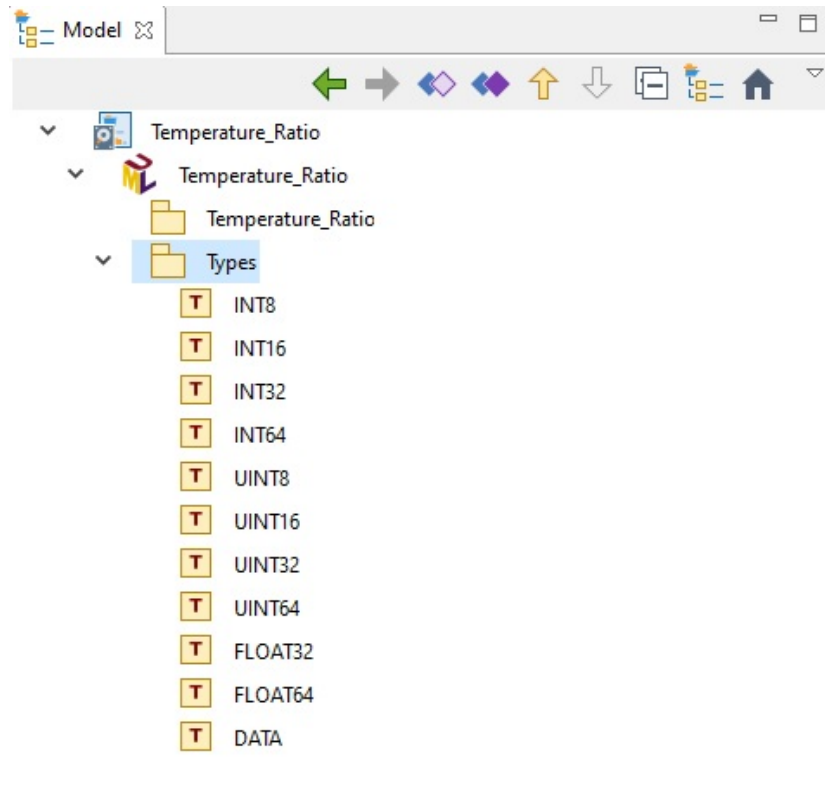
The table Table 5.2.1: Applicable Data Types points which data types are applicable for specific interface type of Component or Package model element:

Table 5.2.1: Applicable Data Types

Type	Component Interfaces			Package Interfaces		
	Input Interface	Output Interface	Local Data	Input Interface	Output Interface	Local Data
INT8	✓	✓	✓	✓	✓	

INT16	✓	✓	✓	✓	✓	
INT32	✓	✓	✓	✓	✓	
INT64	✓	✓	✓	✓	✓	
UINT8	✓	✓	✓	✓	✓	
UINT16	✓	✓	✓	✓	✓	
UINT32	✓	✓	✓	✓	✓	
UINT64	✓	✓	✓	✓	✓	
FLOAT32	✓	✓	✓	✓	✓	
FLOAT64	✓	✓	✓	✓	✓	
DATA						✓

✓ means that given data type can be used for specific interface type.



## 5.3 Model Component Creation

### Create New Component

Right-click on model package and select **Create element → Component**. Click on new component element, press F2 on keyboard and then give desired name to it.

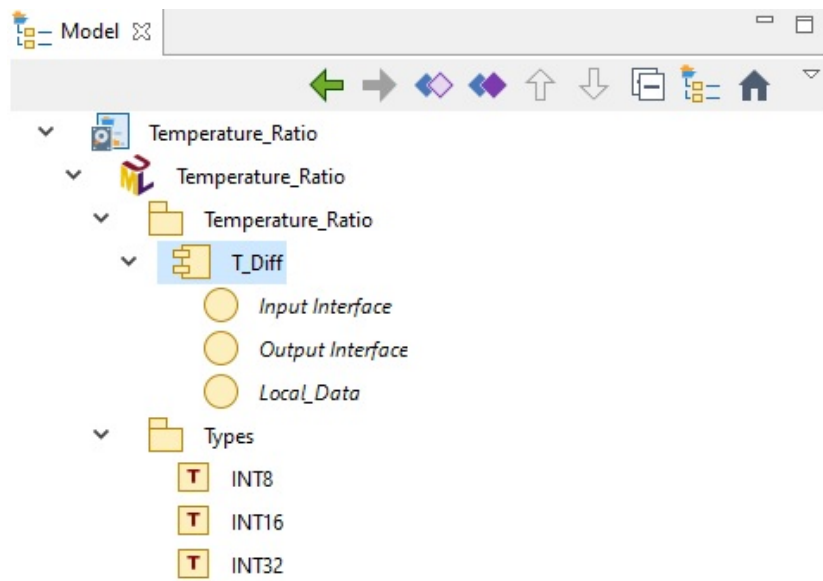
### Create Component Interfaces

Create all interfaces listed under **Interface Type** column in Table 5.3.1: Component Interface Types.

Right-click on component and select **Create element → Interface**. Click on new interface type, press F2 on keyboard and name new interface type as per column **Interface Type**. Repeat the process for the rest of required interface types.

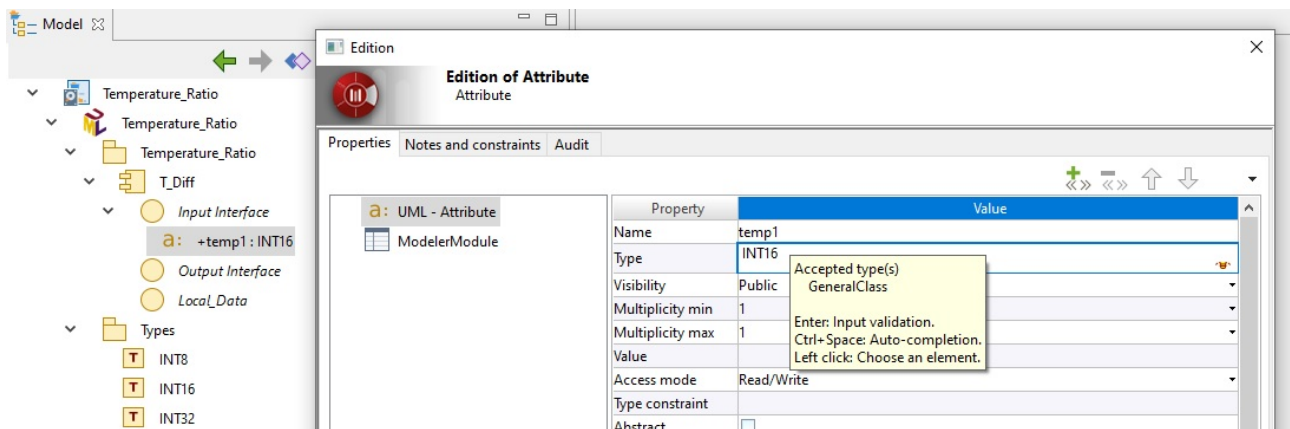
Table 5.3.1: Component Interface Types

Interface Type	Attribute Meaning	Interface Objectives
Input Interface	Signal	Defines input interface of the component, i.e. list of signals, which are inputs to the component.
Output Interface	Signal	Defines output interface of the component, i.e. list of signals, which are outputs from the component.
Local Data	Signal	Defines local signals within the component, i.e. list of signals, which are used internally within the component to exchange data between actions.



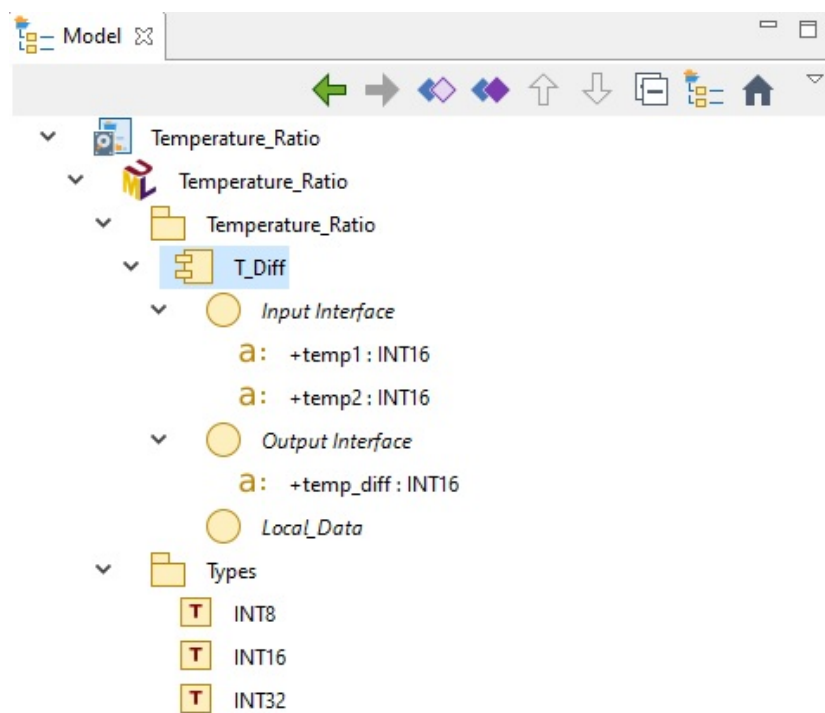
## Create Component Interface Signals

Right-click on desired interface type and select **Create element → Attribute**. Double-click on new attribute. In **Name** property type desired name and start typing desired type in **Type** property then press “Ctrl + Space” combination on keyboard to see auto-completion list.



**Attribute** meaning for each interface type is defined by **Attribute Meaning** column in Table 5.3.1: Component Interface Types.

Please refer to Table 5.2.1: Applicable Data Types for list of data types, which could be used under each component interface and please follow Component Interface Rules.



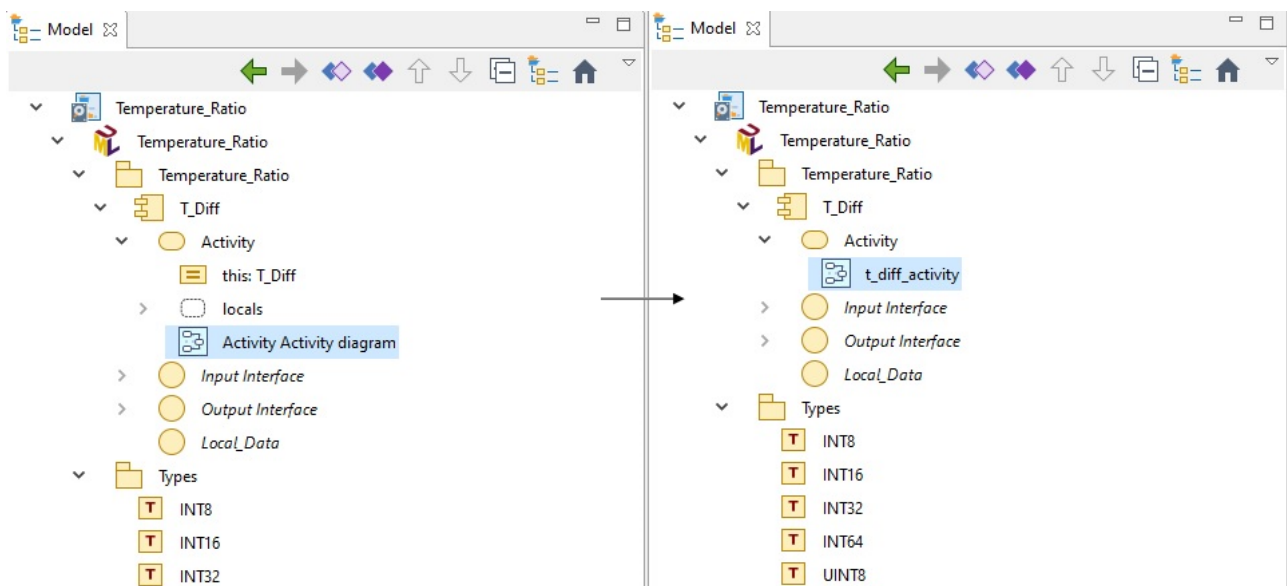
If you wish to pass signal between two components, then source component and target component need to have signal with same name and data type under appropriate interface, i.e. under output interface of source component and input interface of target component.

## Create Component Activity

Right-click on component and select **Create diagram**, then select **Activity diagram** from available list of diagram. You should now find **Activity** element under your component. Remove from **Activity** following elements:

1. **this: <component name>**
2. **locals**

Once **Activity** is created, please find **Activity diagram** under **Activity**, click on it and press F2 on keyboard, then enter desired name for **Activity diagram**.



## Define Component Activity

This process is composed from following basic steps:

1. Move interface signals into diagram space
2. Define component actions
3. Define first input signal if needed
4. Connect signals and action

During this process please follow Component Activity Rules.

### Move interface signals into diagram space

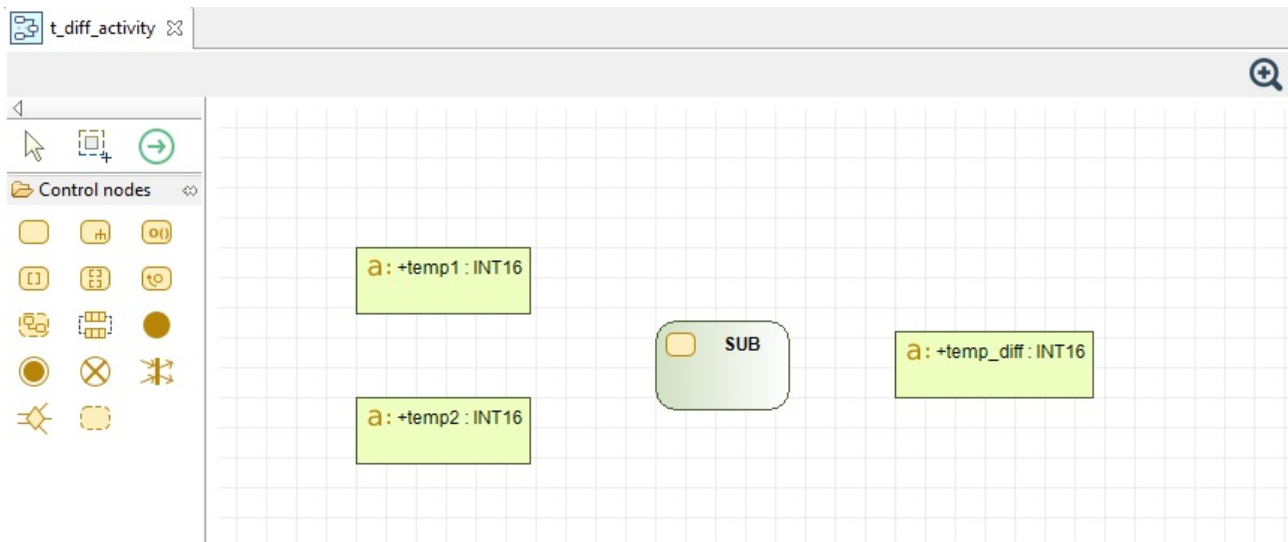
Open **Activity diagram**, then drag and drop interface signals into diagram space. You can drag and drop signal again if you need to use it more than one time.

### Define component actions

Actions define interactions between interface signals. Please see list of allowed actions between signals and their description in Table 5.7.1: Applicable Action Types.

Within **Control nodes** box select **Action – Create an Opaque Action**, then click on diagram space where you wish to place new action. Double-click on new action.

In **Name** property type desired action type from Table 5.7.1: Applicable Action Types.



#### Define first input signal if needed

Some kind of actions require to distinguish which input signal is first in action equation in order to compute correct result. Please see column **\$FIRST\$ marker needed** in Table 5.7.1: Applicable Action Types to find out which actions require to point first input signal.

#### **As example:**

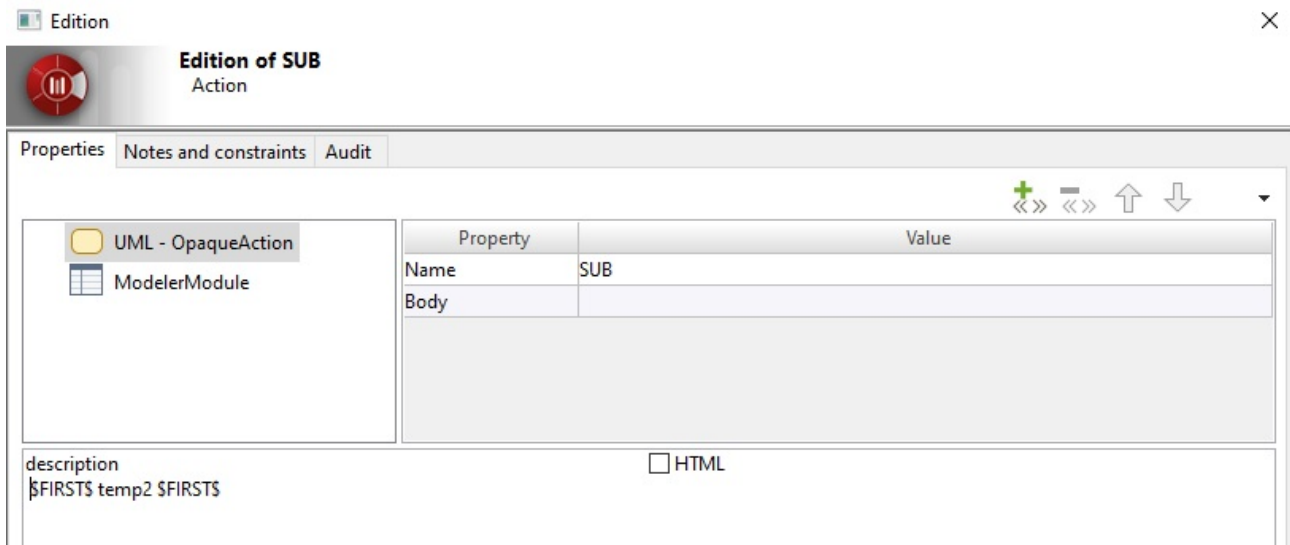
*Subtraction action  $temp\_diff=temp1-temp2$  will give different result than  $temp\_diff=temp2-temp1$ .*

If you are using action, which require to point first input signal, then type in **Description** field (you can open it by double-click on action):

\$FIRST\$ marker, then name of first input signal, then \$FIRST\$ marker again, keeping one white space separation between markers and signal name.

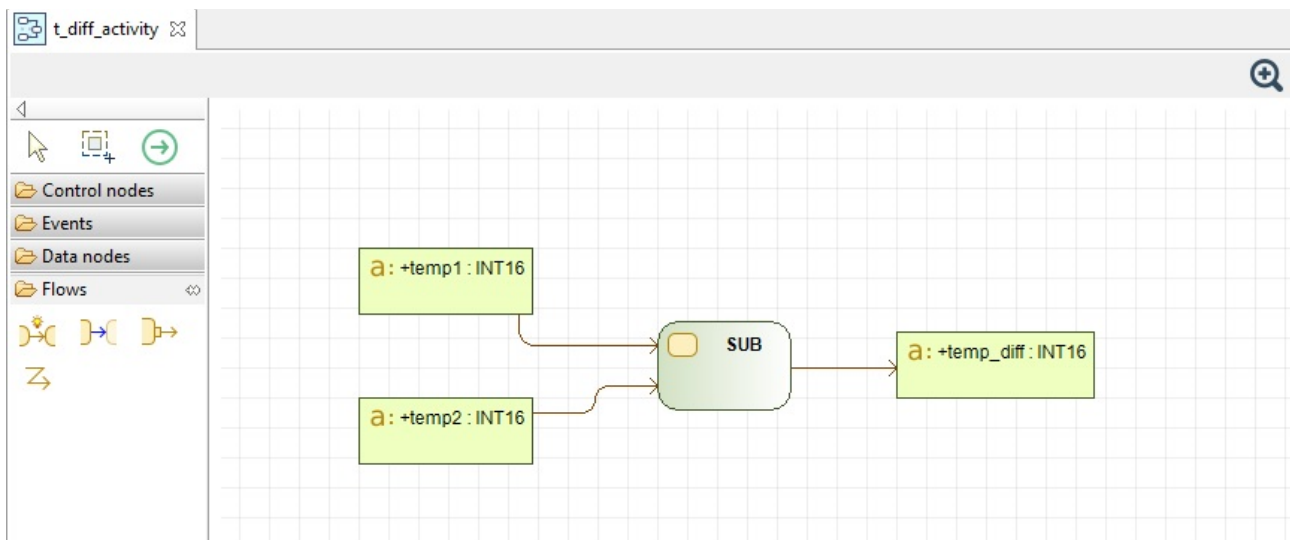
#### **As example:**

**\$FIRST\$ temp2 \$FIRST\$**



### Connect signals and actions

Within **Flows** box select **Object Flow – Create an Object Flow**, then connect component elements (signals and actions) on diagram space.



### Create Next Components

Once component is defined you can create further components following steps since Create New Component.

## 5.4 Model Package Creation

### Create Package Interfaces

Create all interfaces listed under **Interface Type** column in Table 5.4.1: Package Interface Types.



Right-click on package and select **Create element → Interface**. Click on new interface type, press F2 on keyboard and name interface type as per column **Interface Type**. Repeat the process for the rest of required interface types.

Table 5.4.1: Package Interface Types

Interface Type	Attribute Meaning	Interface Objectives
Input Interface	Signal	Defines input interface of the package, i.e. list of signals, which are inputs to the package.
Output Interface	Signal	Defines output interface of the package, i.e. list of signals, which are outputs from the package.
Local Data	Structure	Defines local structures within the package, i.e. list of structures, which are used internally within the package to exchange data between components.

## Create Package Interface Signals and Structures

Right-click on desired interface type and select **Create element → Attribute**. Double-click on new attribute. In **Name** property type desired name and start typing desired type in **Type** property then press “Ctrl + Space” combination on keyboard to see auto-completion list.

**Attribute** meaning for each interface type is defined by **Attribute Meaning** column in Table 5.4.1: Package Interface Types.

Please refer to Table 5.2.1: Applicable Data Types for list of data types, which could be used under each package interface and please follow Package Interface Rules.

Please be aware that meaning of attributes under Local Data interface of model package is different than in case of other interfaces. These attributes are representing names of structures, which will be used to exchange data between model components on activity diagram of model package.

Each component occurrence on activity diagram of model package need to have separate structure, which represents outputs from that component.

At minimum create two attributes under Local Data interface and name them **Input Interface** and **Output Interface**. These attributes will represent Input and Output Interface of model package.

## Create Package Activity

Right-click on package and select **Create diagram**, then select **Activity diagram** from available list of diagram. You should now find **Activity** element under your package. Remove from **Activity** following elements:

1. **this: <component name>**

## 2. locals

Once **Activity** is created, please find **Activity diagram** under **Activity**, click on it and press F2 on keyboard, then enter desired name for **Activity diagram**.

## Define Package Activity

This process is composed from following basic steps:

1. Move structures and components into diagram space
2. Connect structures and components

During this process please follow Package Activity Rules.

### Move structures and components into diagram space

Open **Activity diagram**, then drag and drop interface structures into diagram space, then drag and drop all components into diagram space. You can drag and drop structure or component again if you need to use it more than one time.

### Connect structures and components

Within **Flows** box select **Object Flow – Create an Object Flow**, then connect package elements (structures and components) on diagram space.

Please be aware that connection on activity diagram between two components through structure is not enough to pass data from one component to another. To pass signal from one component to another, apart from connection on activity diagram of model package, the output interface of source component and input interface of target component need to have signal with same name and data type, as pointed in Create Component Interface Signals.

## 5.5 Modeling Rules

### Model Rules

#### **MR#1 RULE**

Each model created within Modelio environment **shall** have following generic layout in order to allow proper code generation with MCG:

1. <model package>
  - a) <model component 1>
  - b) <model component 2>
  - c) ...
  - d) <model component n>
2. <type package>

#### **MR#2 RULE**

Each model **shall** have exactly one model package.

#### MR#3 RULE

Each model **shall** have exactly one type package.

#### MR#4 RULE

Each model package **shall** have at least one model component.

#### MR#5 RULE

<type package> **shall** be named Types.

#### MR#6 RULE

<type package> **shall** have following types:

- INT8
- INT16
- INT32
- INT64
- UINT8
- UINT16
- UINT32
- UINT64
- FLOAT32
- FLOAT64
- DATA

### Component Interface Rules

#### CIR#1 RULE

Model component **shall** have all interface types listed in Table 5.3.1: Component Interface Types.

#### CIR#2 RULE

Output interface of model component **shall** have at least one signal.

#### CIR#3 RULE

Signals of model component **shall** be created only under interface types pointed in Table 5.3.1: Component Interface Types.

#### CIR#4 RULE

Signals of model component **shall** use only applicable data types pointed in Table 5.2.1: Applicable Data Types.

#### CIR#5 RULE

Each input interface signal of model component **shall** have unique name in comparison to other interface signals under the same component.

#### CIR#6 RULE

Each output interface signal of model component **shall** have unique name in comparison to other interface signals under the same component and output interface signals of other components.

#### **CIR#7 RULE**

Each local data interface signal of model component **shall** have unique name in comparison to other interface signals under the same component.

### **Component Activity Rules**

#### **CAR#1 RULE**

Actions of model component **shall** use only applicable action types listed in Table 5.7.1: Applicable Action Types.

#### **CAR#2 RULE**

Actions of model component **shall** be connected only with interface signals of model component.

#### **CAR#3 RULE**

Each action of model component **shall** have required number of input signals and required number of output signals as per its definition under **Definition** column in Table 5.7.1: Applicable Action Types.

#### **CAR#4 RULE**

Each action of model component, which require to distinguish first input signal as per **\$FIRST\$ marker needed** column in Table 5.7.1: Applicable Action Types **shall** point first input signal in its **Description** field between \$FIRST\$ markers in following way:

\$FIRST\$ first\_input\_signal\_name \$FIRST\$

#### **CAR#5 RULE**

There **shall** be exactly one white space separation between \$FIRST\$ markers in action **Description** and name of first input signal.

#### **CAR#6 RULE**

All interface signals of model component **shall** be connected with other elements on activity diagram of model component.

#### **CAR#7 RULE**

Each input interface signal of model component **shall** be connected only as input to another signal or action.

#### **CAR#8 RULE**

Each output interface signal of model component **shall** be connected only as output from another signal or action.

#### **CAR#9 RULE**

Each local data interface signal of model component **shall** be connected as:

- input to another signal or action, and
- output from another signal or action.

#### **CAR#10 RULE**

Any interface signal of model component **shall** have only one input in form of another signal or action, apart from input interface signals, which **shall not** have any input connection.

## Package Interface Rules

### PIR#1 RULE

Model package **shall** have all interface types listed in Table 5.4.1: Package Interface Types.

### PIR#2 RULE

Input and output interface of model package **shall** have at least one signal.

### PIR#3 RULE

Local data interface of model package **shall** have at least **Input Interface** and **Output Interface** structures.

### PIR#4 RULE

Signals of model package **shall** be created only under input and output interface types pointed in Table 5.4.1: Package Interface Types.

### PIR#5 RULE

Structures of model package **shall** be created only under local data interface type pointed in Table 5.4.1: Package Interface Types.

### PIR#6 RULE

Signals and structures of model package **shall** use only applicable data types pointed in Table 5.2.1: Applicable Data Types.

### PIR#7 RULE

Each input interface signal of model package **shall** have unique name in comparison to other interface signals and structures under the same package.

### PIR#8 RULE

Each output interface signal of model package **shall** have unique name in comparison to other interface signals and structures under the same package.

### PIR#9 RULE

Each local data interface structure of model package **shall** have unique name in comparison to other interface signals and interface structures under the same package.

### PIR#10 RULE

Each local data interface structure of model package **shall** have different name than source component element of the structure.

### PIR#11 RULE

Each component occurrence on activity diagram of model package **shall** have individual structure, which represents output from that component.

## Package Activity Rules

### PAR#1 RULE

Components of model package **shall** be connected only with local data interface structures of model package.

### PAR#2 RULE

All local data interface structures of model package **shall** be connected with other elements on activity diagram of model package.

### PAR#3 RULE

**Input Interface** local data interface structure of model package **shall** be connected only as input to component.

### PAR#4 RULE

**Output Interface** local data interface structure of model package **shall** be connected only as output from another structure of component.

### PAR#5 RULE

Each other local data interface structure (other than **Input Interface** and **Output Interface** structures) of model package **shall** be connected as:

- input to component or **Output Interface** structure, and
- output from component.

### PAR#6 RULE

Any local data interface structure of model package **shall** have only one input in form of component, apart from:

- Input Interface structure, which **shall not** have any input connection, and
- Output Interface structure, which **shall** have at least one input in form of other structure or component.

### PAR#7 RULE

To pass signal from one component to another, apart connection between two components via local data interface structure on activity diagram of model package, the output interface of source component and input interface of target component **shall** have signal of same name and data type.

## Naming Convention Rules

### NCR#1 RULE

Names of model elements **shall** contain only allowed characters listed in Table 5.6.1: Applicable Characters.

### NCR#2 RULE

Names of model elements **shall** start with upper or lower case letters only.

### NCR#3 RULE

Names of model elements **shall not** contain white spaces.

## NCR#4 RULE

Names of model elements shall be different than action types

## 5.6 Naming Convention

Table 5.6.1: Applicable Characters defines list of applicable characters within name of any model element.

Table 5.6.1: Applicable Characters

Character type	Allowed characters
Upper case letters	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Lower case letters	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits	1 2 3 4 5 6 7 8 9 0
Special characters	_ -

## 5.7 List of Actions

Table 5.7.1: Applicable Action Types defines list of applicable interactions (actions) between signals on diagram space of component element.

Table 5.7.1: Applicable Action Types

Action type	Definition	\$FIRST\$ marker needed
ADD	Addition arithmetic operation. Requires at least two input signals <sup>1)</sup> . Requires exactly one output signal <sup>2)</sup> .	
SUB	Subtraction arithmetic operation. Requires at least two input signals <sup>1)</sup> . Requires exactly one output signal <sup>2)</sup> .	Yes
MUL	Multiplication arithmetic operation. Requires at least two input signals <sup>1)</sup> . Requires exactly one output signal <sup>2)</sup> .	
DIV	Division arithmetic operation. Requires at least two input signals <sup>1)</sup> . Requires exactly one output signal <sup>2)</sup> .	Yes

1) Input Interface or Local Data signal could be connected as input signal to that action.

2) Local Data or Output Interface signal could be connected as output signal from that action.

## 6 How to run MCG

Please make sure that installation steps described in Installation sections were completed and that two environment variables, i.e. "MCG\_CC" and "MCG\_CGC", were created.

## 6.1 Converter Component (MCG CC)

### Run Converter Component

The MCG CC requires two command line arguments to run:

- `<model_dir_path>` Path to model directory, where all catalogs with .exml files are stored
- `<output_dir_path>` Path to output directory, where results from MCG will be saved

```
C:\Users\Kamil Deć>%MCG_CC%

Mod Code Generator (MCG)
Copyright (C) 2021 Kamil Deć github.com/deckamil
This is Converter Component (CC) of Mod Code Generator (MCG)

License GPLv3+: GNU GPL version 3 or later.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Incorrect number of command line arguments, MCG CC process cancelled.
Usage: python mcg_cc_main.py "<model_dir_path>" "<output_dir_path>"
Arguments:
  <model_dir_path>      Path to model directory, where all catalogs with .exml files are stored
  <output_dir_path>     Path to output directory, where results from MCG will be saved

Keep specific order of arguments, as pointed in usage above.
See Mod Code Generator Manual for further information.
```

To run the MCG CC please type in a command line window, keeping the specific order of arguments:

```
%MCG_CC% "<model_dir_path>" "<output_dir_path>"
```

The "`<model_dir_path>`" is usually equal to "`<modelio_project_path>\data\framgents\  
<modelio_model_name>\model`".

Please consider example where:

- the "`<model_dir_path>`" is equal to "`C:\Example\Model_Project\Simple_Calc\data\fragments\Simple_Calc\model`", and
- the "`<output_dir_path>`" is equal to "`C:\Example\MCG_CC_Output`"

In the above example the MCG CC should be invoked as:

```
%MCG_CC% "C:\Example\Model_Project\Simple_Calc\data\fragments\Simple_Calc\model" "C:\Example\MCG_CC_Output"
```



[illegible]

## Check Converter Component Outputs

## 6.2 Code Generator Component (MCG CGC)

## Run Code Generator Component

## Check Code Generator Component Outputs

## 7 Configuration File Syntax and Format