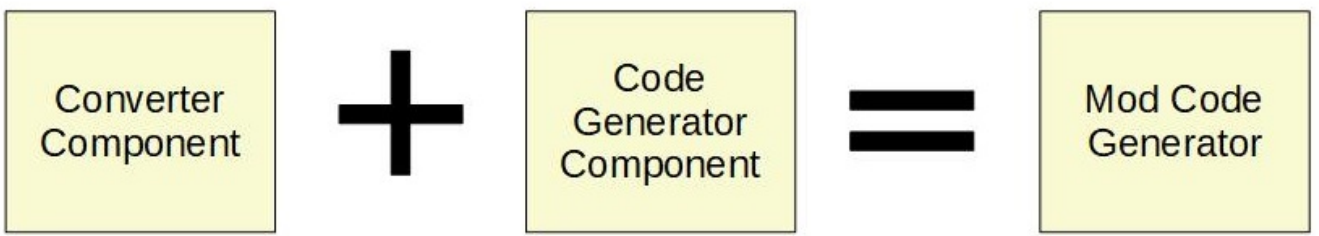
Mod Code Generator Introduction

What is the Mod Code Generator (MCG)?

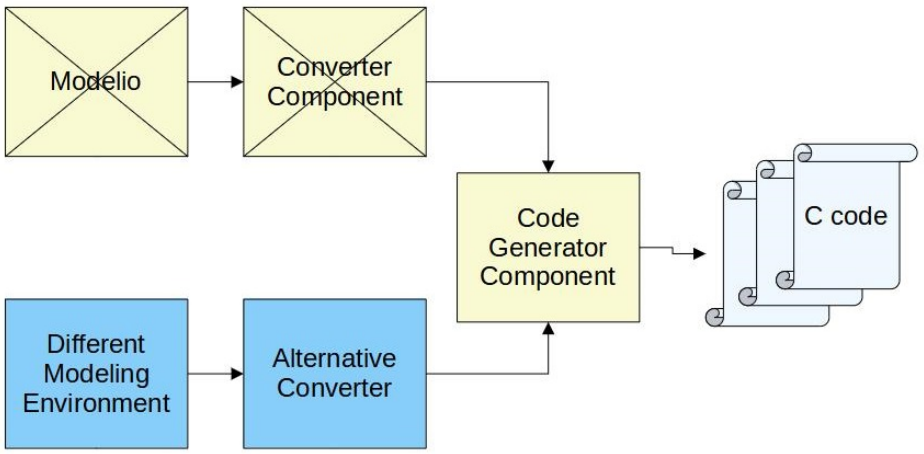
The MCG is a code generator tool. It is written in Python https://www.python.org/. The MCG works in conjunction with Modelio https://www.modelio.org/, that provides modeling environment and graphical user interface used to create a model. The modeling environment is stand-alone program and it is not a part of the MCG, see list of features under https://www.modelio.org/about-modelio/features.html/.

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

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



Why MCG CC and CGC are two separate tools?



The MCG CC is suited to work in conjunction with the Modelio environment and its main task is to generate the configuration file for the MCG CGC.

A tool separation gives possibility to replace the MCG CC with alternative converter tool and generate code from different modeling environment, as long as the alternative converter tool will generate the configuration file in desired format and syntax for the MCG CGC.

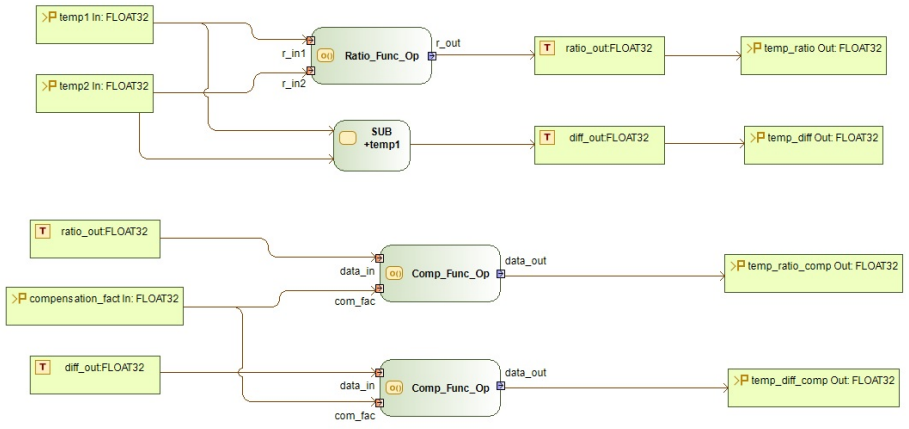
If preferred, the MCG CGC can be used in a stand-alone mode. An end user can write the configuration file in desired format and syntax and input it into the MCG CGC for code generation.

The modeling environment

How the modeling environment is used?

The modeling environment provides graphical user interface standing between an end user and the MCG. The modeling environment allows to build graphical representation of a model and define its data flow and data interactions.

The model data is saved in form of .exml files, which further provide input information about model elements and diagram connections to the MCG CC.



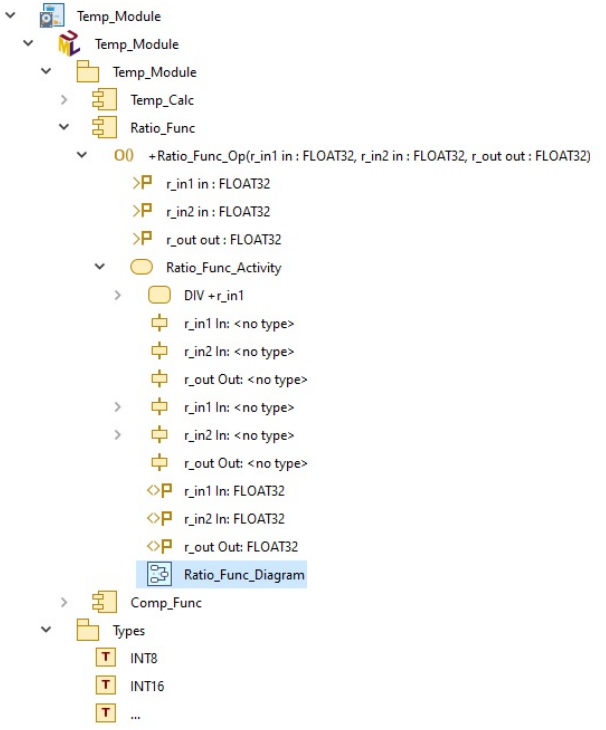

```
</OBJECT>
<OBJECT>
  <ID name="temp1" mc="Standard.ActivityParameterNode" uid="df43fb75-8a6f-4022-9b27-abfa26c26df9"/>
  <ATTRIBUTES>
    <ATT name="IsControlType">false</ATT>
    <ATT name="Ordering">FIFO</ATT>
    <ATT name="SelectionBehavior"></ATT>
    <ATT name="UpperBound"><![CDATA[1]]></ATT>
    <ATT name="Name"><![CDATA[temp1]]></ATT>
    <ATT name="status">1970354901745664</ATT>
  </ATTRIBUTES>
  <DEPENDENCIES>
    <LINK relation="RepresentedRealParameter">
      <REFOBJ>
        <ID name="temp1" mc="Standard.BehaviorParameter" uid="01d22598-31dc-4bc3-ae83-c7de4aaf9d05"/>
      </REFOBJ>
    </LINK>
    <COMP relation="Outgoing">
      <OBJECT>
        <ID name="ObjectFlow" mc="Standard.ObjectFlow" uid="5ac562bb-d7e5-482e-b329-66a9036255bb"/>
        <ATTRIBUTES>
          <ATT name="TransformationBehavior"></ATT>
          <ATT name="SelectionBehavior"></ATT>
          <ATT name="IsMultiCast">false</ATT>
          <ATT name="IsMultiReceive">false</ATT>
          <ATT name="Effect">ReadFlow</ATT>
          <ATT name="Guard"></ATT>
          <ATT name="Weight"><![CDATA[1]]></ATT>
          <ATT name="Name"><![CDATA[ObjectFlow]]></ATT>
          <ATT name="status">1970354901745664</ATT>
        </ATTRIBUTES>
        <DEPENDENCIES>
          <LINK relation="Target">
            <REFOBJ>
              <ID name="r_in1" mc="Standard.InputPin" uid="4852c79f-3e0b-49c9-815c-f7fd2cf72b77"/>
            </REFOBJ>
          </LINK>
        </DEPENDENCIES>
      </OBJECT>
      <OBJECT>
```

Model package and component elements

A model should be composed from at least one package and one component element with operation and activity diagram defined under the component element.

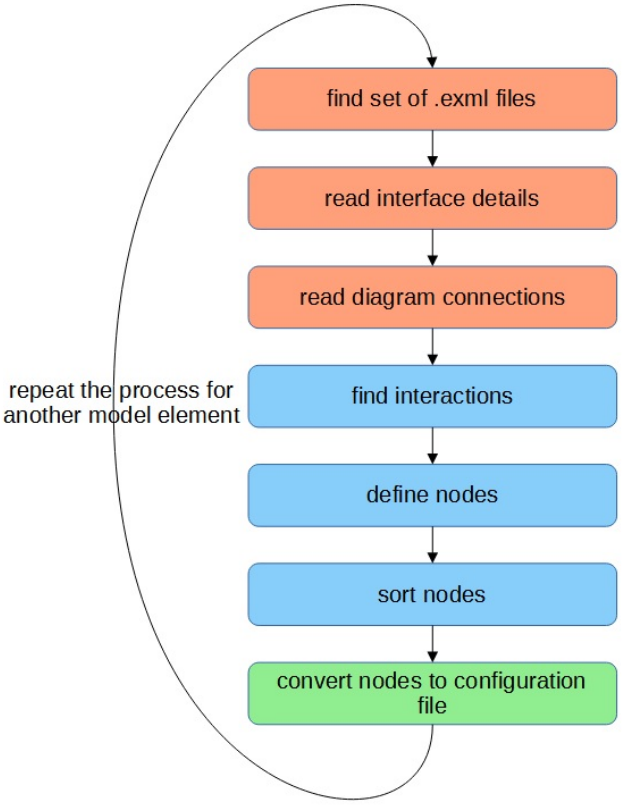
Components are used to define interactions between model data under activity diagram of component operation. Each component is treated as a separate C source file module.

Packets in general are used to segregate components. The Type package is used only to define allowed data types for model interfaces.



The MCG Converter Component (MCG CC)

The MCG CC workflow

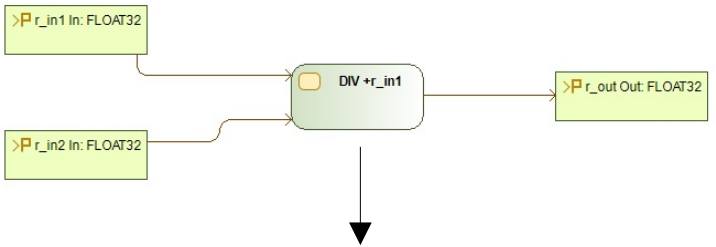


The entire MCG CC workflow is a 3-step process:

1. READING – find set of .exml files, that describe one model element, then read from them interface details and information about connections between diagram elements.
2. SORTING – use connections to find all interactions on activity diagram and to define diagram nodes, then sort nodes at the end of that process step.
3. CONVERSION – convert sorted nodes into the configuration file.

The entire process is repeated for each model element that defines an operation and have appended activity diagram.

Model connections and nodes



Connections:

- $SOURCE$: r_in1 $TARGET$: DIV +r_in1 0210d51d-dfc5-40b7-b690-3cb158284153 connection
- $SOURCE$: r_in2 $TARGET$: DIV +r_in1 0210d51d-dfc5-40b7-b690-3cb158284153 connection
- $SOURCE$: DIV +r_in1 0210d51d-dfc5-40b7-b690-3cb158284153 $TARGET$: r_out connection

Nodes:

- $INPUTS$: r_in1 r_in2 $INTERACTION$: DIV +r_in1 0210d51d-dfc5-40b7-b690-3cb158284153 $OUTPUT$: r_out node

The MCG CC looks for set of .exmls files, that describe one model element and then reads from these files information about model element interfaces and connections between diagram elements.

Connections are used to find all data interactions from given activity diagram. Diagram connection and interaction data is used finally to define all diagram nodes.

A single diagram node describes one, specific instance of data interaction (like DIVide arithmetic interaction) that appears on activity diagram, along with all interaction input and output connections.

How activity diagram nodes are sorted?

Nodes:

- $INPUTS$: temp1->r_in1 temp2->r_in2 $INTERACTION$: Ratio_Func_Op() a9538338-83f8-45a5-81df-0d16f4915ade $OUTPUT$: r_out->ratio_out node
- $INPUTS$: diff_out $INTERACTION$: ASSIGNMENT $OUTPUT$: temp_diff node
- $INPUTS$: compensation_fact->com_fac ratio_out->data_in $INTERACTION$: Comp_Func_Op() aa9cf965-96d7-4eb6-9b21-e4e8469ddc18 $OUTPUT$: data_out->temp_ratio_comp node
- $INPUTS$: compensation_fact->com_fac diff_out->data_in $INTERACTION$: Comp_Func_Op() bf9bb8a7-b942-4893-97cb-37a45ef5b332 $OUTPUT$: data_out->temp_diff_comp node
- $INPUTS$: ratio_out $INTERACTION$: ASSIGNMENT $OUTPUT$: temp_ratio node
- $INPUTS$: temp1 temp2 $INTERACTION$: SUB +temp1 4640e233-4085-424c-91c3-483cdfe4d763 $OUTPUT$: diff_out node

Sorted Nodes:

- $INPUTS$: temp1->r_in1 temp2->r_in2 $INTERACTION$: Ratio_Func_Op() a9538338-83f8-45a5-81df-0d16f4915ade $OUTPUT$: r_out->ratio_out node
- $INPUTS$: temp1 temp2 $INTERACTION$: SUB +temp1 4640e233-4085-424c-91c3-483cdfe4d763 $OUTPUT$: diff_out node
- $INPUTS$: compensation_fact->com_fac ratio_out->data_in $INTERACTION$: Comp_Func_Op() aa9cf965-96d7-4eb6-9b21-e4e8469ddc18 $OUTPUT$: data_out->temp_ratio_comp node
- $INPUTS$: compensation_fact->com_fac diff_out->data_in $INTERACTION$: Comp_Func_Op() bf9bb8a7-b942-4893-97cb-37a45ef5b332 $OUTPUT$: data_out->temp_diff_comp node
- $INPUTS$: ratio_out $INTERACTION$: ASSIGNMENT $OUTPUT$: temp_ratio node
- $INPUTS$: diff_out $INTERACTION$: ASSIGNMENT $OUTPUT$: temp_diff node

Once nodes are defined, the MCG CC will sort them:

1. At beginning the MCG CC looks for nodes, which use only input interface data (no dependency from local data) and then it adds these nodes at beginning of sorted nodes list.
2. After that the MCG CC looks for nodes, which use either input interface data or local data computed by previously sorted nodes, then appends them to the list.
3. The MCG CC repeats the process from (2) for the rest of unsorted nodes till all are sorted.

Conversion into configuration file

An example of configuration file part (beginning):

```
$MODULE START$
$MODULE NAME START$
Temp_Calc
$MODULE NAME END$
$OPERATION NAME START$
Temp_Calc_Op
$OPERATION NAME END$
$INPUT INTERFACE START$
type FLOAT32 name temp1
type FLOAT32 name temp2
type FLOAT32 name compensation_fact
$INPUT INTERFACE END$
$OUTPUT INTERFACE START$
type FLOAT32 name temp_ratio
type FLOAT32 name temp_diff
type FLOAT32 name temp_ratio_comp
type FLOAT32 name temp_diff_comp
$OUTPUT INTERFACE END$
$LOCAL INTERFACE START$
type FLOAT32 name ratio_out
type FLOAT32 name diff_out
$LOCAL INTERFACE END$
$OPERATION BODY START$
$OPE Ratio_Func_Op
$INP temp1->r_in1
...
```

An example of configuration file part (continuation):

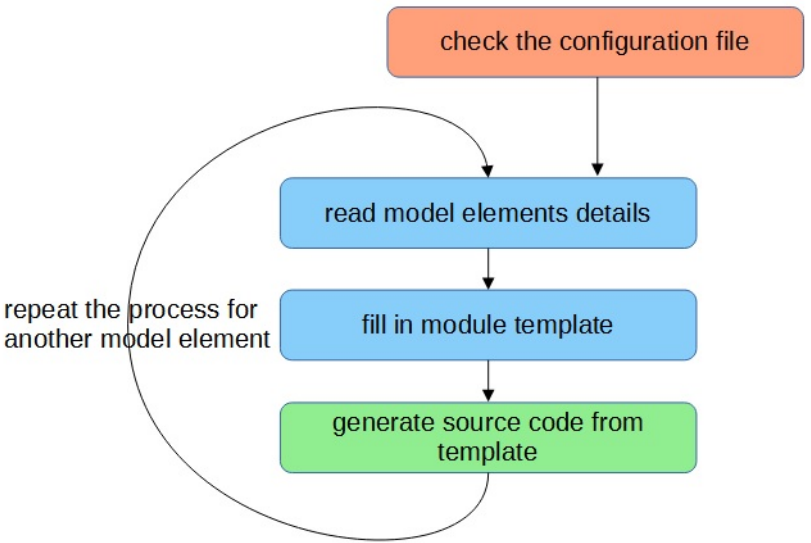
```
...
$INP temp2->r_in2
$OUT r_out->ratio_out
$INS diff_out = temp1 - temp2
$OPE Comp_Func_Op
$INP compensation_fact->com_fac
$INP ratio_out->data_in
$OUT data_out->temp_ratio_comp
$OPE Comp_Func_Op
$INP compensation_fact->com_fac
$INP diff_out->data_in
$OUT data_out->temp_diff_comp
$INS temp_ratio = ratio_out
$INS temp_diff = diff_out
$OPERATION BODY END$
$MODULE END$
```

At the end the interface data and sorted nodes, which were sourced from .exml files are converted into configuration file format by the MCG CC.

The configuration file is treated as input data to the MCG CGC process and it contains information about interfaces of model element and set of instructions which represent data interactions from activity diagram.

The MCG Code Generator Component (MCG CGC)

The MCG CGC workflow



The entire MCG CGC workflow is a 3-step process:

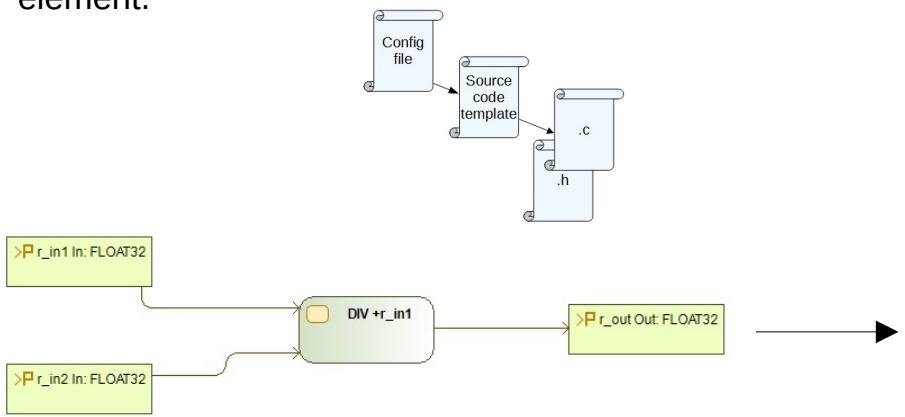
1. CHECKING – check the configuration file that contains information about interfaces of model element and set of instructions which represent data interactions
2. READING – read details of model element from the configuration file and fill in a template of source code item
3. GENERATION – generate source code from the template of source code item

The reading & generation processes are repeated for each model element until end of the configuration file.

The template and code generation

The MCG CGC reads configuration data of given model element and then fills in the template of source code item with data acquired from the configuration file.

Once template is filled with data, the MCG CGC generates from the template both the source code .c and header .h files that represent a single model element.




```c
/*
 *   Generated with Mod Code Generator (MCG) Code Generator Component (CGC)
 *   on 31 May 2023, 18:02:18
 *
 *   This is source file of Ratio_Func module.
 */

#include "Ratio_Func.h"
#include "basic_data_types.h"

// This is definition of module function
void Ratio_Func_Op(Ratio_Func_Op_input_type *Ratio_Func_Op_input,Ratio_Func_Op_output_type *Ratio_Func_Op_output) {

    // Input interface
    FLOAT32 r_in1 = Ratio_Func_Op_input->r_in1;
    FLOAT32 r_in2 = Ratio_Func_Op_input->r_in2;

    // Local interface

    // Output interface
    FLOAT32 r_out;

    // Function body
    r_out = r_in1 / r_in2;

    // Collect output data
    Ratio_Func_Op_output->r_out = r_out;

}

/*
 * END OF MODULE
 */
```