

---

# Goil OIL compiler user manual

*Release 2.0b45*

Jean-Luc Béchenne & Pierre Molinaro



# CONTENTS

<b>1</b>	<b>The Goil templates</b>	<b>1</b>
1.1	The configuration data	2
1.1.1	The <i>PROCESSES</i> , <i>TASKS</i> , <i>BASICTASKS</i> , <i>EXTENDEDTASKS</i> , <i>ISRS1</i> and <i>ISRS2</i> lists	2
1.1.2	The <i>COUNTERS</i> , <i>HARDWARECOUNTERS</i> and <i>SOFTWARECOUNTERS</i> lists	3
1.1.3	The <i>EVENTS</i> list	3
1.1.4	The <i>ALARMS</i> list	4
1.1.5	The <i>REGULARRESOURCES</i> and <i>INTERNALRESOURCES</i> lists	4
1.1.6	The <i>MESSAGES</i> , <i>SENDMESSAGES</i> and <i>RECEIVEMESSAGES</i> lists	5
1.1.7	The <i>SCHEDULETABLES</i> list	6
1.1.8	The <i>OSAPPLICATIONS</i> list	8
1.1.9	The <i>TRUSTEDFUNCTIONS</i> list	9
1.1.10	The <i>READLIST</i> list	9
1.1.11	The <i>SOURCEFILES</i> , <i>CFLAGS</i> , <i>ASFLAGS</i> , <i>LDFLAGS</i> and <i>TRAMPOLINE-SOURCEFILES</i> lists	10
1.1.12	The <i>INTERRUPTSOURCES</i> list	11
1.1.13	Scalar data	11
1.2	The Goil template language (or GTL)	13
1.3	GTL types	13
1.3.1	string readers	14
1.3.2	boolean readers	14
1.3.3	integer readers	15
1.3.4	list readers	15
1.4	GTL operators	15
1.4.1	Unary operators	15
1.4.2	Binary operators	16
1.4.3	Constants	16
1.5	GTL instructions	17
1.5.1	The <i>let</i> instruction	17
1.5.2	The <i>if</i> instruction	17
1.5.3	The <i>foreach</i> instruction	17
1.5.4	The <i>for</i> instruction	18
1.5.5	The <i>loop</i> instruction	19
1.5.6	The <i>!</i> instruction	19

1.5.7	The <i>?</i> instruction . . . . .	19
1.5.8	The <i>template</i> instruction . . . . .	19
1.5.9	The <i>write</i> instruction . . . . .	20
1.5.10	The <i>error</i> and <i>warning</i> instructions . . . . .	20
1.6	Examples . . . . .	21
1.6.1	Computing the list of process ids . . . . .	21
1.6.2	Computing an interrupt table . . . . .	21
1.6.3	Generation of all the files . . . . .	22

# The Goil templates

Goil includes a template interpreter which is used for file generation. Goil generates the structures needed by trampoline to compile the application and may generate other files like a memory mapping file 'MemMap.h', the compiler abstraction files, 'Compiler.h' and 'Compiler\_cfg.h' and a linker script depending on which attributes you set in the OIL file.

A template is a file which is located in the default template directory (set with the environment variable GOIL\_TEMPLATES or with the **--templates** option on the command line) or in the directory of your project. Goil starts by looking for a template in the directory of your project, then, if the template is not found, in the default templates directory.

Four sets of templates are used:

- code generation templates that are located in the 'code' subdirectory of the template directory;
- build system templates that are located in the 'build' subdirectory;
- compiler dependent stuff in the 'compiler' subdirectory and
- linker script templates in the 'linker' subdirectory.

Templates are written using a simple language which allow to access the application configuration data and to mix them with text to produce files.

Files are produced by a template program located in the 'root.goilTemplate' file which is as the root of the template directory. By default the following files are produced:

- 'tpl\_app\_config.c' by using the 'tpl\_app\_config.c.goilTemplate' file
- 'tpl\_app\_config.h' by using the 'tpl\_app\_config.h.goilTemplate' file
- 'Makefile' (if option **-g** or **--generate-makefile** is given) by using the 'Makefile.goilTemplate' file
- 'script.ld' (if memory mapping is used and if the default name is not changed) by using the 'script.goilTemplate' file
- 'MemMap.h' (if memory mapping is used) by using the 'MemMap.h.goilTemplate' file
- 'Compiler.h' (if memory mapping is used) by using the 'Compiler.h.goilTemplate' file
- 'Compiler\_Cfg.h' (if memory mapping is used) by using the 'Compiler\_Cfg.h.goilTemplate' file

## 1.1 The configuration data

The configuration data are computed by Goil from the OIL source files, from the options on the command line and from the 'target.cfg' file. They are available as a set of predefined boolean, string, integer or list variables. All these variables are in capital letters.

**Warning:** Some configuration data are not listed here because they are dependent on the target. For instance, the `STACKSIZE` data may be an attribute of each item of a `TASKS` list for ppc target but are missing for the c166 target.

### 1.1.1 The `PROCESSES`, `TASKS`, `BASICTASKS`, `EXTENDEDTASKS`, `ISRS1` and `ISRS2` lists

These variables are lists where informations about the processes<sup>1</sup> used in the application are stores:

List	Content
<code>PROCESSES</code>	the list of processes. The items are sorted in the following order: extended tasks, then basic tasks, then ISRs category 2.
<code>TASKS</code>	the list of tasks, basic and extended. The items are sorted in the following order: extended tasks, then basic tasks.
<code>BASICTASKS</code>	the list of basic tasks.
<code>EXTENDEDTASKS</code>	the list of extended tasks.
<code>ISRS1</code>	the list of ISR category 1.
<code>ISRS2</code>	the list of ISR category 2.

Each item of these lists has the following attributes:

Item	Type	Content
<code>NAME</code>	string	the name of the process.
<code>PROCESSKIND</code>	string	the kind of process: "Task" or "ISR".
<code>EXTENDEDTASK</code>	boolean	true if the process is an extended task, false otherwise.
<code>NONPREEMPTABLE</code>	boolean	true if the process is a non-preemptable task, false otherwise.
<code>PRIORITY</code>	integer	the priority of the process.
<code>ACTIVATION</code>	integer	the number of activation of a task. 1 for and extended task or an ISR.
<code>AUTOSTART</code>	boolean	true if the process is an autostart task, false otherwise.
<code>USEINTERNALRESOURCE</code>	boolean	true if the process is a task that uses an internal resource, false otherwise.
<code>INTERNALRESOURCE</code>	string	the name of the internal resource if the process is a task that uses an internal resource, empty string otherwise.
<code>RESOURCES</code>	list	The resources used by the process. Each item has the following attribute: <code>NAME</code>

<sup>1</sup>In Trampoline, a process is a task or an ISR category 2.

### 1.1.2 The *COUNTERS*, *HARDWARECOUNTERS* and *SOFTWARECOUNTERS* lists

This list contains all the informations about the counters used in the application, including the *SystemCounter*.

List	Content
<i>COUNTERS</i>	the list of counters, both hardware and software as long as the <i>SystemCounter</i>
<i>HARDWARECOUNTERS</i>	the list of hardware counters including the <i>SystemCounter</i> .
<i>SOFTWARECOUNTERS</i>	the list of software counters.

Each item of this list has the following attributes:

Item	Type	Content
NAME	string	the name of the counter.
TYPE	string	the type of counter: "HARDWARE_COUNTER" or "SOFTWARE_COUNTER".
MAXALLOWEDVALUE	integer	the maximum allowed value of the counter.
MINCYCLE	integer	the minimum cycle value of the counter.
TICKPERBASE	integer	the number of ticks needed to increment the counter.
SOURCE	string	the interrupt source name of the counter. This can be used to wrap interrupt vector to a counter incrementation function

### 1.1.3 The *EVENTS* list

This list contains the informations about the events of the application. Each item has the following attributes:

Item	Type	Content
NAME	string	the name of the event.
MASK	integer	the mask of the event.

### 1.1.4 The *ALARMS* list

This list contains the informations about the alarms of the application. Each item has the following attributes:

Item	Type	Content
NAME	string	the name of the alarm.
COUNTER	string	the name of the counter that drives the alarm.
ACTION	string	the action to be done when the alarm expire. It can take the following values: “setEvent”, “activateTask” and “callback”. The last action is not available in Autosar mode.
TASK	string	the name of the task on which the action is performed. This attribute is defined for “setEvent” and “activateTask” actions only.
EVENT	string	the name of the event to set on the target task. This attribute is defined for “setEvent” action only.
AUTOSTART	boolean	true if the alarm is autostart, false otherwise
ALARMTIME	integer	the alarm time of the alarm. This attribute is set if AUTOSTART is true
CYCLETIME	integer	the cycle time of the alarm. This attribute is set if AUTOSTART is true
APPMODE	string	the application mode in which the alarm is autostart. This attribute is set if AUTOSTART is true

### 1.1.5 The *REGULARRESOURCES* and *INTERNALRESOURCES* lists

These lists contains the informations about the resources of the application.

List	Content
<i>REGULARRESOURCES</i>	the list of STANDARD and LINKED resources.
<i>INTERNALRESOURCES</i>	the list of INTERNAL resources.

Each item has the following attributes:

Item	Type	Content
NAME	string	the name of the resource.
PRIORITY	integer	the priority of the resource.
TASKUSAGE	list	the list of tasks that use the resource. Each item of this list has an attribute NAME which is the name of the task.



Item	Type	Content
ISRUSAGE	list	the list of ISRs that use the resource. Each item of this list has an attribute NAME which is the name of the ISR.

### 1.1.6 The *MESSAGES*, *SENDMESSAGES* and *RECEIVEMESSAGES* lists

These lists contain the informations about the messages of the application.

List	Content
<i>MESSAGES</i>	the list of messages, both send and receive message.
<i>SENDMESSAGES</i>	the list of send messages.
<i>RECEIVEMESSAGES</i>	the list of receive messages.

Each item has the following attributes

Item	Type	Content
NAME	string	the name of the message.
MESSAGEPROPERTY	string	the type of the message. It can be "RECEIVE_ZERO_INTERNAL", "RECEIVE_UNQUEUED_INTERNAL", "RECEIVE_QUEUED_INTERNAL", "SEND_STATIC_INTERNAL", "SEND_ZERO_INTERNAL" or "RECEIVE_ZERO_SENDERS".
NEXT	string	the name of the next message in a receive message chain. This attribute is defined for receive messages only.
SOURCE	string	the name of the send message which is connected to the receive message. This attribute is defined for receive messages only.
CTYPE	string	the C language type of the message. This attribute is not defined for "RECEIVE_ZERO_INTERNAL" and "SEND_ZERO_INTERNAL" messages.
INITIALVALUE	string	initial value of the receive message. This attribute is defined for "RECEIVE_UNQUEUED_INTERNAL" and "RECEIVE_ZERO_SENDERS" messages only.
QUEUESIZE	integer	queue size of a receive queued message. This attribute is defined for "RECEIVE_QUEUED_INTERNAL" messages only.
TARGET	string	target message of a send message. This is the first message in a receive message chain. This attribute is defined for "SEND_STATIC_INTERNAL" and "SEND_ZERO_INTERNAL" messages only.

Item	Type	Content
FILTER	string	the kind of filter to apply. This attribute may take the following values: “ALWAYS”, “NEVER”, “MASKED-NEWEQUALSX”, “MASKEDNEWDIFFERSX”, “NEWISEQUAL”, “NEWISDIFFERENT”, “MASKED-NEWEQUALSMASKEDOLD”, “MASKEDNEWDIFFERSMASKEDOLD”, “NEWISWITHIN”, “NEWISOUTSIDE”, “NEWISGREATER”, “NEWISLESSOREQUAL”, “NEWISLESS”, “NEWISGREATEROREQUAL” or “ONEEVERYN”.
MASK	integer	Mask of the filter when needed. This attribute is defined for “MASKEDNEWEQUALSX”, “MASKEDNEWDIFFERSX”, “MASKEDNEWEQUALSMASKEDOLD” and “MASKEDNEWDIFFERSMASKEDOLD” filters only.
X	integer	Value of the filter when needed. This attribute is defined for “MASKEDNEWEQUALSMASKEDOLD” and “MASKEDNEWDIFFERSX” filters only.
MIN	integer	Minimum value of the filter when needed. This attribute is defined for “NEWISWITHIN” and “NEWISOUTSIDE” filters only.
MAX	integer	Maximum value of the filter when needed. This attribute is defined for “NEWISWITHIN” and “NEWISOUTSIDE”
PERIOD	integer	Period of the filter. This attribute is defined for “ONEEVERYN” filter only.
OFFSET	integer	Offset of the filter. This attribute is defined for “ONEEVERYN” filter only.
ACTION	string	the action (or notification) to be done when the message is delivered. It can take the following values: “setEvent” or “activateTask”.
TASK	string	the name of the task on which the notification is performed. This attribute is defined for “setEvent” and “activateTask” actions only.
EVENT	string	the name of the event to set on the target task. This attribute is defined for “setEvent” notification only.

### 1.1.7 The *SCHEDULETABLES* list

This list contains the informations about the schedule tables of the application.

Item	Type	Content
NAME	string	the name of the schedule table.
COUNTER	string	the name of the counter which drives the schedule table.

Item	Type	Content
PERIODIC	boolean	true if the schedule table is a periodic one, false otherwise.
SYNCSTRATEGY	string	the synchronization strategy of the schedule table. This attribute may take the following values: "SCHEDTABLE_NO_SYNC", "SCHEDTABLE_IMPLICIT_SYNC" or "SCHEDTABLE_EXPLICIT_SYNC".
PRECISION	integer	the precision of the synchronization. This attribute is define when SYNCSTRATEGY is "SCHEDTABLE_EXPLICIT_SYNC".
STATE	string	the state of the schedule table. This attribute may take the following values: "SCHEDULETABLE_STOPPED", "SCHEDULETABLE_AUTOSTART_SYNCHRON", "SCHEDULETABLE_AUTOSTART_RELATIVE" or "SCHEDULETABLE_AUTOSTART_ABSOLUTE".
DATE	integer	the start date of the schedule table. This attribute has an actual value when STATE is "SCHEDULETABLE_AUTOSTART_RELATIVE" or "SCHEDULETABLE_AUTOSTART_ABSOLUTE", otherwise it is set to 0.
LENGTH	integer	The length of the schedule table.
EXPIRYPOINTS	list	The expiry points of the schedule table. See the following table for items attributes.

Each item of the EXPIRYPOINTS list has the following attributes:

Item	Type	Content
ABSOLUTEOFFSET	integer	the absolute offset of the expiry points.
RELATIVEOFFSET	integer	the relative offset of the expiry points from the previous expiry point.
MAXRETARD	integer	maximum retard to keep the schedule table synchronous.
MAXADVANCE	integer	maximum advance to keep the schedule table synchronous.
ACTIONS	list	the actions to perform on the expiry point. See the following table for items attributes.

Each item of the ACTIONS list has the following attributes:

Item	Type	Content
ACTION	string	the action to be done when the alarm expire. It can take the following values: "setEvent", "activateTask", "incrementCounter" and "finalizeScheduleTable".

Item	Type	Content
TASK	string	the name of the task on which the action is performed. This attribute is defined for “setEvent” and “activate-Task” actions only.
EVENT	string	the name of the event to set on the target task. This attribute is defined for “setEvent” action only.
TARGETCOUNTER	string	the name of the counter to increment. This attribute is defined for “incrementCounter” action only.

### 1.1.8 The *OSAPPLICATIONS* list

This list contains the informations about the OS Applications of the application.

Item	Type	Content
NAME	string	the name of the OS Application.
RESTART	string	the name of the restart task. This attribute is not defined if there is no restart task for the OS Application.
PROCESSACCESSVECTOR	string	access right for the processes
PROCESSACCESSITEMS	string	access right for the processes as bytes in a table
PROCESSACCESSNUM	integer	number of elements in the previous table
ALARMACCESSVECTOR	string	access right for the alarms
ALARMACCESSITEMS	string	access right for the alarms as bytes in a table
ALARMACCESSNUM	integer	number of elements in the previous table
RESOURCEACCESSVECTOR	string	access right for the resources
RESOURCEACCESSITEMS	string	access right for the resources as bytes in a table
RESOURCEACCESSNUM	integer	number of elements in the previous table
SCHEDULETABLEACCESSVECTOR	string	access right for the schedule tables
SCHEDULETABLEACCESSITEMS	string	access right for the schedule tables as bytes in a table
SCHEDULETABLEACCESSNUM	integer	number of elements in the previous table
COUNTERACCESSVECTOR	string	access right for the software counters
COUNTERACCESSITEMS	string	access right for the software counters as bytes in a table
COUNTERACCESSNUM	integer	number of elements in the previous table
PROCESSES	list	list of the processes that belong to the OS Application. Each item has an attribute <i>NAME</i> which is the name of the process.
TASKS	list	list of the tasks that belong to the OS Application. Each item has an attribute <i>NAME</i> which is the name of the task.
ISRS	list	list of the ISRs that belong to the OS Application. Each item has an attribute <i>NAME</i> which is the name of the ISR.

Item	Type	Content
ALARMS	list	list of the alarms that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the alarm.
RESOURCES	list	list of the resources that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the resource.
REGULARRESOURCES	list	list of the standard or linked resources that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the resource.
INTERNALRESOURCES	list	list of the internal resources that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the resource.
SCHEDULETABLES	list	list of the schedule tables that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the schedule table.
COUNTERS	list	list of the counters that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the counter.
MESSAGES	list	list of the messages that belong to the OS Application. Each item has an attribute <code>NAME</code> which is the name of the messages.

### 1.1.9 The *TRUSTEDFUNCTIONS* list

This list contains the informations about the trusted functions of the application. Each item contains one attribute only.

Item	Type	Content
NAME	string	the name of the trusted function.

### 1.1.10 The *READLIST* list

This list contains the informations about the ready list. Items are sorted by priority from 0 to the maximum computed priority. The only attribute of each item is the size of the queue.

Item	Type	Content
SIZE	integer	the size of the queue for the corresponding priority.

### 1.1.11 The *SOURCEFILES*, *CFLAGS*, *ASFLAGS*, *LDFLAGS* and *TRAMPOLINESOURCEFILES* lists

The *SOURCEFILES* list contains the source files as found in attributes `APP_SRC` of the OS object in the OIL file.

Item	Type	Content
FILE	string	the source file name.

The *CFLAGS* list contains the flags for the C compiler as found in attributes `CFLAGS` of the OS object in the OIL file.

Item	Type	Content
CFLAG	string	the C compiler flag.

The *ASFLAGS* list contains the flags for the assembler as found in attributes `ASFLAGS` of the OS object in the OIL file.

Item	Type	Content
ASFLAG	string	the assembler flag.

The *LDFLAGS* list contains the flags for the linker as found in attributes `LDFLAGS` of the OS object in the OIL file.

Item	Type	Content
LDFLAG	string	the linker flag.

The *TRAMPOLINESOURCEFILES* list contains the trampoline source files used by the application.

Item	Type	Content
DIRECTORY	string	the directory of the source file relative to the Trampoline root directory ('os', 'com' or 'autosar').

Item	Type	Content
FILE	string	the source file name.

### 1.1.12 The *INTERRUPTSOURCES* list

This list is extracted from the ‘target.cfg’ file. Each item has the following attribute:

Item	Type	Content
NAME	string	the name of the interrupt source. This is one of the name used in the OIL file as value for the SOURCE attribute
NUMBER	string	the id of the interrupt source.

### 1.1.13 Scalar data

The following scalar data are defined:

Data	Type	Content
APPNAME	string	name of executable as given in the APP_NAME attribute in the OS object
ARCH	string	name of the architecture. This is the first item in the target.
ASSEMBLEREXE	string	name of the assembler executable used. This is the ASSEMBLER attribute in the OS object. It is set to <i>as</i> by default. It is used for build dependent templates.
ASSEMBLER	string	name of the assembler used. This is the ASSEMBLER attribute in the MEMMAP attribute of the OS object. It is used for assembler dependent templates.
AUTOSAR	boolean	true if Trampoline is compiled with the Autosar extension.
BOARD	string	name of the board. This is the third item (if any) in the target.
CHIP	string	name of the chip. This is the second item (if any) in the target.
COMPILEREXE	string	name of the compiler executable used. This is the COMPILER attribute in the OS object. It is set to <i>gcc</i> by default. It is used for build dependent templates. Do not confuse with the COMPILER data.
COMPILER	string	name of the compiler used. This is the COMPILER attribute in the MEMMAP attribute of the OS object. It is used for compiler dependent templates.

Data	Type	Content
CPUNAME	string	name given to the OIL CPU object
EXTENDED	boolean	true if Trampoline is compiled in extended error handling mode.
FILENAME	string	the name of the file which will be written as the result of the computation of the current template.
FILEPATH	string	the full absolute path of the file which will be written as the result of the computation of the current template.
NATIVEFILEPATH	string	the full absolute path of the file which will be written as the result of the computation of the current template in native OS format.
ITSOURCESLENGTH	integer	number of interrupt sources as defined in the 'target.cfg' file.
LINKEREXE	string	name of the linker executable used. This is the LINKER attribute in the OS object. It is set to <i>gcc</i> by default. It is used for build dependent templates. Do not confuse with the LINKER data.
LINKER	string	name of the linker used. This is the LINKER attribute in the MEMMAP attribute of the OS object. It is used for linker dependent templates.
LINKSCRIPT	string	name of the link script file as given in the MEMMAP attribute of the OS object.
MAXTASKPRIORITY	integer	the highest computed priority among the tasks.
OILFILENAME	string	name of the root OIL source file
PROJECT	string	name of the project. The name of the project is the <b>-p</b> (or <b>--project</b> ) value if it is set or the name of the oil file without the extension.
SCALABILITYCLASS	integer	the Autosar scalability class used by the application. If Autosar is not enabled, SCALABILITYCLASS is set to 0.
TARGET	string	name of the target. This is the <b>-t</b> (or <b>--target</b> ) option value of goil.
TEMPLATEPATH	string	path to the template root directory. This is the <b>--templates</b> option value of goil or the value of the GOIL_TEMPLATES environment variable.
TIMESTAMP	string	current date
TRAMPOLINEPATH	string	path to the trampoline root directory. This is the TRAMPOLINE_BASE_PATH attribute of the OS object. It defaults to ".".
USECOMPILERSETTINGS	boolean	true if memory mapping is enabled (Goil generates the 'Compiler.h' and 'Compiler_Cfg.h' files and Trampoline includes them).
USEBUILDFILE	boolean	true if a build file is used for the project ie option <b>-g</b> or <b>--generate-makefile</b> is given.
USECOM	boolean	true if the application uses OSEK COM.
USEERRORHOOK	boolean	true if Trampoline uses the Error Hook.
USEGETSERVICEID	boolean	true if Trampoline uses the service ids access macros.



Data	Type	Content
USEINTERRUPTTABLE	boolean	true if the wrapping of interrupt vector to glue functions used to increment a counter or to activate an ISR2 (for instance) should be generated. The actual code generation is up to the port.
USEMEMORYMAPPING	boolean	true if memory mapping is enabled (Goil generates the 'MemMap.h' file and Trampoline includes it).
USEMEMORYPROTECTION	boolean	true if Trampoline uses the Memory Protection.
USEOSAPPLICATION	boolean	true if Trampoline uses OS Applications.
USEPARAMETERACCESS	boolean	true if Trampoline uses the parameters access macros.
USEPOSTTASKHOOK	boolean	true if Trampoline uses the Post-Task Hook.
USEPRETASKHOOK	boolean	true if Trampoline uses the Pre-Task Hook.
USEPROTECTIONHOOK	boolean	true if Trampoline uses the Protection Hook.
USERESSCHEDULER	boolean	true if Trampoline uses the RES_SCHEDULER resource.
USESHUTDOWNHOOK	boolean	true if Trampoline uses the Shutdown Hook.
USESTACKMONITORING	boolean	true if Trampoline uses the Stack Monitoring.
USESTARTUPHOOK	boolean	true if Trampoline uses the Startup Hook.
USESYSTEMCALL	boolean	true if services are called using a System Call (i.e. a software interrupt).
USETIMINGPROTECTION	boolean	true if Trampoline uses Timing Protection.
USETRACE	boolean	true if tracing is enabled.

## 1.2 The Goil template language (or GTL)

A template is a text file with file extension `'.goilTemplate'`. This kind of file mixes literal text with an embedded program. Some instructions (see section 1.5.6) in the embedded program outputs text as a result of the program execution and this text is put in place of the instructions. The resulting file is then stored.

The template interpreter starts in literal text mode. Switching from literal text mode to program mode and back to text mode is done when a `'%'` is encountered. A literal `'%'` and a literal `'\'` may be used by escaping them with a `'\'`.

## 1.3 GTL types

GTL supports 4 types: **string**, **integer**, **boolean** and **list**. These types have readers to get informations about a variable. A reader is invoked with the following syntax:

```
[expression reader]
```

### 1.3.1 string readers

The following readers are available for string variables:

Item	Type	Meaning
<b>HTMLRepresentation</b>	string	this reader returns a representation of the string suitable for an HTML encoded representation. ‘&’ is encoded by &amp; , ‘”’ by &quot; , ‘<’ by &lt; and ‘>’ by &gt; .
<b>identifierRepresentation</b>	string	this reader returns an unique representation of the string conforming to a C identifier. Any Unicode character that is not a latin letter is transformed into its hexadecimal code point value, enclosed by ‘_’ characters. This representation is unique: two different strings are transformed into different C identifiers. For example: value3 is transformed to value_33_ ; += is transformed to _2B__3D; An_Identifier is transformed to An_5F_Identifier.
<b>lowercaseString</b>	string	this reader returns lowercased representation of the string.
<b>length</b>	integer	this reader returns the number of characters in the string
<b>stringByCapitalizingFirstCharacter</b>	string	if the string is empty, this reader returns the empty string; otherwise, it returns the string, the first character being replaced with the corresponding upper case character.
<b>uppercaseString</b>	string	this reader returns uppercased representation of the receiver

### 1.3.2 boolean readers

The following readers are available for boolean variables:

Item	Type	Meaning
<b>trueOrFalse</b>	string	this reader returns “true” or “false” according to the boolean value
<b>yesOrNo</b>	string	this reader returns “yes” or “no” according to the boolean value
<b>unsigned</b>	integer	this reader returns 0 or 1 according to the boolean value

### 1.3.3 integer readers

The following readers are available for integer variables:

Item	Type	Meaning
<b>string</b>	string	This reader returns the integer value as a character string.
<b>hexString</b>	string	this reader returns an hexadecimal string representation of the integer value.

### 1.3.4 list readers

The following reader is available for list variables:

Item	Type	Meaning
<b>length</b>	integer	this reader returns the number of objects currently in the list.

## 1.4 GTL operators

### 1.4.1 Unary operators

Operator	Operand Type	Result Type	Meaning
<b>+</b>	integer	integer	no operation.
<b>~</b>	integer	integer	bitwise not.
<b>not</b>	boolean	boolean	boolean not.
<b>exists</b>	<i>any variable</i>	boolean	true if the variable is defined, false otherwise. But see below

A second form of **exists** is:

```
exists var default (expression)
```

*var* and *expression* should have the same type. If *var* exists, the returned value is the content of *var*. If it does not exist, *expression* is returned.

### 1.4.2 Binary operators

Operator	Operands Type	Result Type	Meaning
+	integer	integer	add.
-	integer	integer	subtract.
*	integer	integer	multiply.
/	integer	integer	divide.
&	integer	integer	Bitwise and.
&	boolean	boolean	boolean and.
	integer	integer	Bitwise or.
	boolean	boolean	boolean or.
^	integer	integer	Bitwise xor.
^	boolean	boolean	boolean xor.
.	string	string	string concatenation.
<<	integer	integer	shift left.
>>	integer	integer	shift right.
!=	any	boolean	comparison (different).
==	any	boolean	comparison (equal).
<	integer <i>or</i> boolean	boolean	comparison (lower than).
<=	integer <i>or</i> boolean	boolean	comparison (lower or equal).
>	integer <i>or</i> boolean	boolean	comparison (greater).
>=	integer <i>or</i> boolean	boolean	comparison (greater or equal).

### 1.4.3 Constants

Constant	Type	Meaning
<b>emptyList</b>	list	this constant is an empty list
<b>true</b>	boolean	true boolean
<b>false</b>	boolean	false boolean
<b>yes</b>	boolean	true boolean
<b>no</b>	boolean	false boolean

## 1.5 GTL instructions

### 1.5.1 The *let* instruction

Data assignment instruction. The general form is:

```
let var := expression
```

A second form allows to add a string to a list (only, this should be extended in the future)

```
let var += expression
```

*var* is a list and *expression* is a string.

The scope of a variable depends on the location where the variable is assigned the first time. For instance, in the following code:

```
let a := 1
foreach TASKS do
  let b := INDEX
  let a := INDEX
end foreach
!a !b
```

Because *a* is assigned outside the **foreach** loop, it contains the value of the last INDEX after the **foreach**. Because *b* is assigned inside the **foreach** loop, it does not exist after the loop anymore and **!b** will trigger an error.

### 1.5.2 The *if* instruction

Conditional execution. The forms are:

```
if expression then ... end if
if expression then ... else ... end if
if expression then ... elsif expression then ... end if
if expression then ... elsif expression then ... else ... end if
```

The *expression* must be boolean. In the following example, the blue text (within the %) is produced only if the USECOM boolean variable is true:

```
if USECOM then %
#include "tpl_com.h" %
end if
```

### 1.5.3 The *foreach* instruction

This instruction iterates on the elements of a list. Each element may have many attributes that are available as variables within the **do** section of the **foreach** loop. The simplest form is the following one

```
foreach expression do ... end foreach
```

In the following example, for each element in the ALARMS list, the text between the **do** and the **end** **foreach** is produced with the NAME attribute of the current element of the ALARMS list inserted at the specified location. INDEX is not an attribute of the current element. It is generated for each element and ranges from 0 to the number of elements in the list minus 1.

```
foreach ALARMS do
%
/* Alarm % !NAME % identifier */
#define % !NAME %_id % !INDEX %
CONST(AlarmType, AUTOMATIC) % !NAME % = % !NAME %_id;
%
end foreach
```

A more general form of the **foreach** instruction is:

```
foreach expression prefixedby string
  before ...
  do ...
  between ...
  after ...
end foreach
```

**prefixedby** is optional and allows to prefix the attribute names by *string*. If the list is not empty, the **before** section are executed once before the first execution of the **do** section. The **between** section is executed between the execution of the **do** section. If the list is not empty, the **after** section is executed once after the last execution of the **do** section.

In the following example, a table of pointers to alarm descriptors is generated:

```
foreach ALARMS
  before %
tpl_time_obj *tpl_alarm_table[ALARM_COUNT] = {
%
  do % &% !NAME %_alarm_desc%
  between %,
%
  after %
};
%
end foreach
```

#### 1.5.4 The *for* instruction

The **for** instruction iterates along a literal list of elements.

```
for var in expression, ... , expression do
  ...
end for
```

At each iteration, *var* gets the value of the current *expression*. As in the **foreach** instruction, INDEX is generated and ranges from 0 to the number of elements in the list minus 1.

### 1.5.5 The *loop* instruction

The **loop** instruction is the classical integer loop. Its simplest form is:

```
loop var from expression to expression do
  ...
end loop
```

Like in the foreach instruction, **before**, **between** and **after** sections may be used:

```
loop var from expression to expression
  before ...
  do ...
  between ...
  after ...
end loop
```

### 1.5.6 The *!* instruction

**!** emits an expression. The form is:

```
! expression
```

### 1.5.7 The *?* instruction

**?** stores in a variable a number of spaces equal to the current column in the output. The form is:

```
? var
```

### 1.5.8 The *template* instruction

The **template** instruction includes the output of another template in the output of the current template. Its simplest form is the following one:

```
template template_file_name
```

If the file *template\_file\_name.goilTemplate* does not exist, an error occurs. To include the output of a template without generating an error, use the following form:

```
template if exists template_file_name
```

A third form allows to execute instructions when the included template file is not found:

```
template if exists template_file_name or ... end template
```

At last, it is possible to search templates in a hierarchy (code, linker, compiler, build) different from the current one. For instance to include a template located in the linker hierarchy, use one of the following forms:

```

template template_file_name in hierarchy
template if exists template_file_name in hierarchy
template if exists template_file_name in hierarchy or ... end template

```

In all cases, the included template inherits from the current variables table but works on its own local copy.

### 1.5.9 The *write* instruction

The write instruction defines a block where the template processing output is captured to be written to a file. The general form is:

```

write to expression :
    ...
end write

```

Where *expression* is a string expression.

In the following example, the result of the 'script' template is written to the link script file.

```

if exists LINKER then
    write to PROJECT."/".LINKSCRIPT:
        template script in linker
    end write
end if

```

### 1.5.10 The *error* and *warning* instructions

It can be useful to generate an error or a warning if a data is not defined or if it looks strange. For instance if a target needs a STACKSIZE for a task or if the STACKSIZE is too large for a 16bit target. **error** and **warning** have 2 forms:

```

error var expression
warning var expression

```

and

```

error here expression
warning here expression

```

*expression* must be of type string. In the first form, *var* is a configuration data. The file location of this configuration may be a location in the OIL file or in the template file if the variable was assigned in the template. In the second form, **here** means the current location in the template file.

In the following example an error is generated for each task with not STACKSIZE attribute in the OIL file:

```

foreach TASKS do
    if not exists STACKSIZE then
        error NAME "STACKSIZE of Task " . NAME . " is not defined"
    end if
end foreach

```



```

    end if
end foreach

```

In this second example, an error is generated if a template is not found:

```

template if exists interrupt_wrapping or
    error here "interrupt_wrapping.goilTemplate not found"
end template

```

## 1.6 Examples

Here are examples of code generation using GTL.

### 1.6.1 Computing the list of process ids

```

foreach PROCESSES do
    if PROCESSKIND == "Task" then
%
/* Task % !NAME % identifier */
#define % !NAME %_id % !INDEX %
CONST(TaskType, AUTOMATIC) % !NAME % = % !NAME %_id;
%
    else
%
/* ISR % !NAME % identifier */
#define % !NAME %_id % !INDEX
        if AUTOSAR then
            #
            # ISR ids constants are only available for AUTOSAR
            #
%
CONST(ISRType, AUTOMATIC) % !NAME % = % !NAME %_id;
%
        end if
    end if
end foreach

```

### 1.6.2 Computing an interrupt table

```

if USEINTERRUPTTABLE then
    loop ENTRY from 0 to ITSOURCESLENGTH - 1
        before
%
#define OS_START_SEC_CONST_UNSPECIFIED
#include "tpl_memmap.h"
CONST(tpl_it_vector_entry, OS_CONST)
tpl_it_table[% !ITSOURCESLENGTH %] = {

```

```

%
do
  let entryFound := false
  foreach INTERRUPTSOURCES prefixedby interrupt_ do
    if ENTRY == interrupt_NUMBER then
      # check first for counters
      foreach HARDWARECOUNTERS prefixedby counter_ do
        if counter_SOURCE == interrupt_NAME & not entryFound then
          % { tpl_tick_% !interrupt_NAME %, (void *)NULL }%
          let entryFound := true
        end if
      end foreach
    end if
    if not entryFound then
      foreach ISRS2 prefixedby isr2_ do
        if isr2_SOURCE == interrupt_NAME & not entryFound then
          % { tpl_central_interrupt_handler_2, (void*)%
            !([TASKS length] + INDEX) % }%
          let entryFound := true
        end if
      end foreach
    end if
  end foreach
  if not entryFound then
    % { tpl_null_it, (void *)NULL }%
  end if
between %,
%
after
%
};
#define OS_STOP_SEC_CONST_UNSPECIFIED
#include "tpl_memmap.h"
%
end loop
end if

```

### 1.6.3 Generation of all the files

This is the default 'root.goilTemplate' file

```

write to PROJECT."/tpl_app_config.c":
  template tpl_app_config_c in code
end write

write to PROJECT."/tpl_app_config.h":
  template tpl_app_config_h in code
end write

write to PROJECT."/tpl_app_define.h":
  template tpl_app_define_h in code
end write

```

```
if exists COMPILER then
  write to PROJECT."/MemMap.h":
    template MemMap_h in compiler
  end write
  write to PROJECT."/Compiler.h":
    template Compiler_h in compiler
  end write
  write to PROJECT."/Compiler_Cfg.h":
    template Compiler_Cfg_h in compiler
  end write
end if

if exists LINKER then
  write to PROJECT."/".LINKSCRIPT:
    template script in linker
  end write
end if
```