Realize Your Product Promise®

Model-Based Design
with
SCADE Suite®

Training Resources

Day 4

ANSYS

AGENDA

Simulation

Generated Code

Integration

Traceability

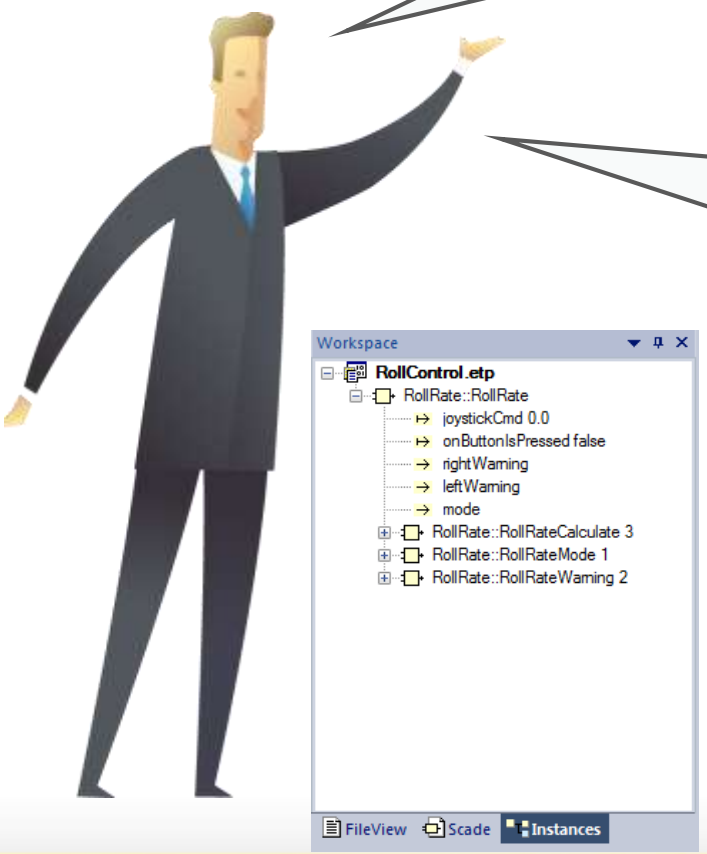
Document Report

Model-Based V&V

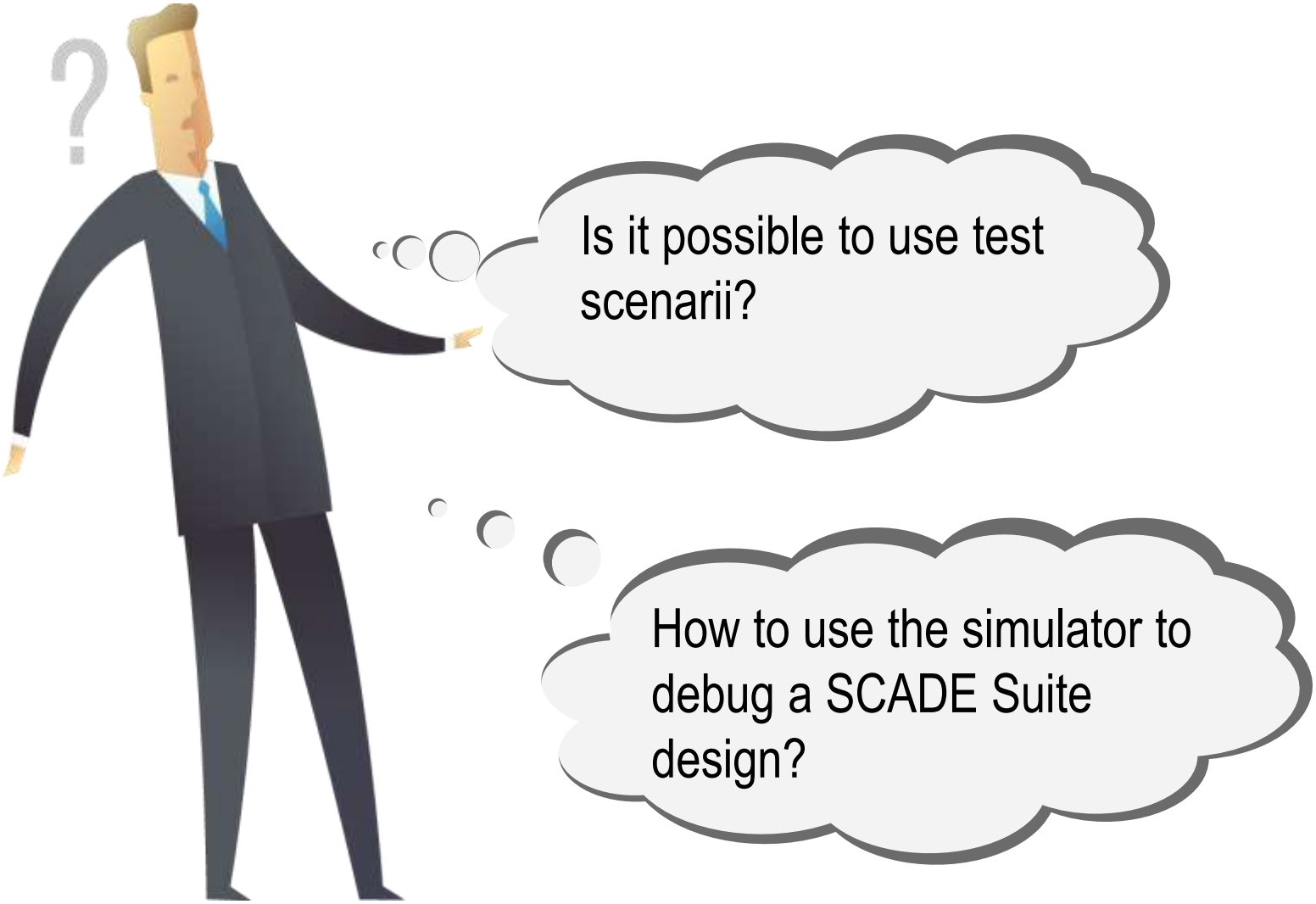
Simulation: I Know!!!

I know how to select and run a Simulation configuration in the **Code Generator** toolbar

I know how to assign values/observe to the model's inputs/outputs and run cycles during the simulation



What Else?



Set Input Values By Scenario Files

4 formats:

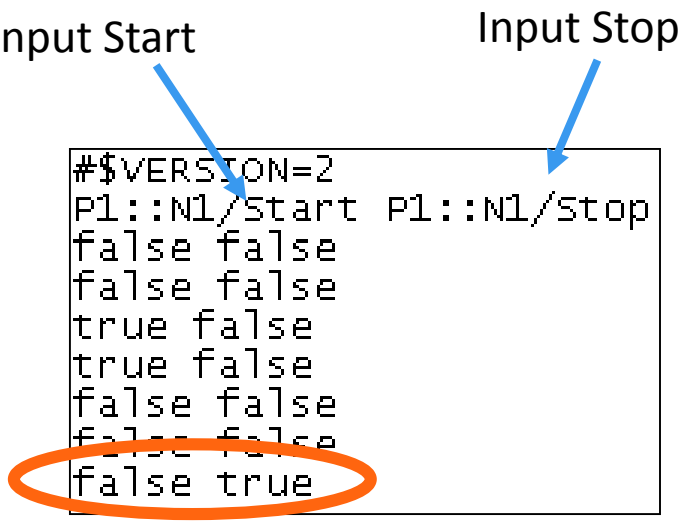
- State scenario (.sts), binary file
- Input vector values file (.in)
- TCL script file (.sss)
- Snapshot scenario (.sns)

State scenario (.sts), binary file

- Store the state of SCADE Suite Simulator at any given step of a simulation session

Set Input Values By Scenario Files

Input vector values file (.in)



One line= one cycle

Simulator script file (.sss)

```
SSM::set P1::N1/Start false
SSM::set P1::N1/Stop false
SSM::cycle 2
SSM::set P1::N1/Start true
SSM::set P1::N1/Stop false
SSM::cycle 2
SSM::set P1::N1/Start false
SSM::set P1::N1/Stop false
SSM::cycle 2
SSM::set P1::N1/Start false
SSM::set P1::N1/Stop true
SSM::cycle 1
```

Producing Code for Testing on External Platforms

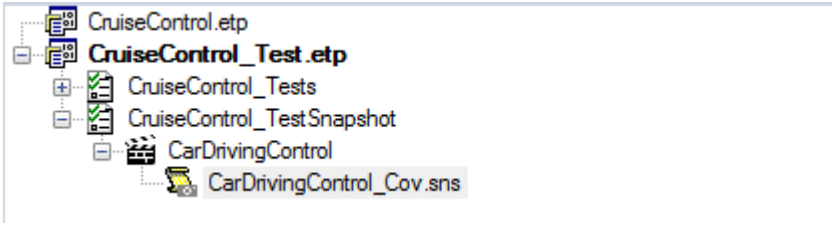
Snapshot scenario (.sns)

- Textual file that stores the SCADE application state: input/output values, memories, sensors, and cycle number
- Outcome of a recorded sequence
- Allows speeding up testing execution
- Save/load application states for external test environments
- Such snapshots can be reused by
  - SCADE Simulator and
  - SCADE Test depending on KCG options and generated code
    - Used as Preamble, initializes the application to a particular state before running the actual test scenarios
    - During the Preamble, neither coverage measurement nor output checks are performed

Producing Code for Testing on External Platforms

Support of System (Closed-loop)

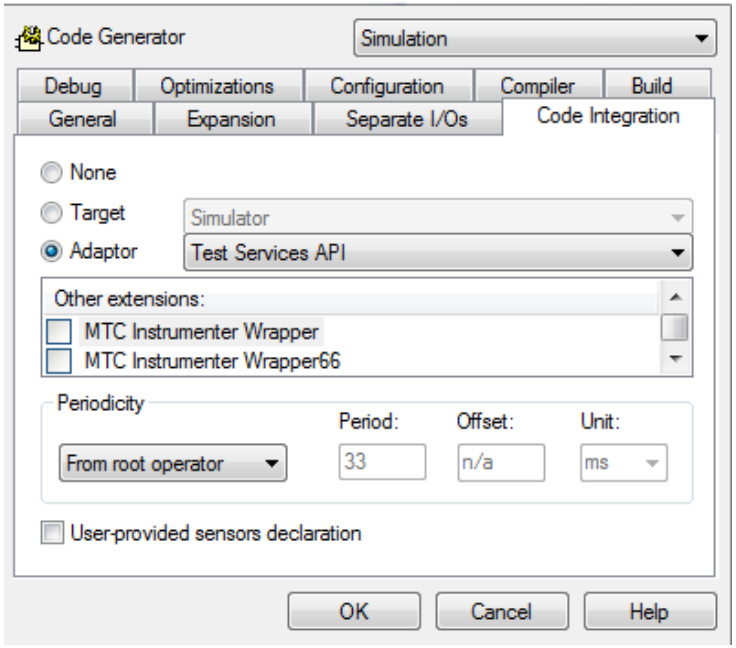
- Coverage measures can be stored from System Testing environment to SCADE Test Model Coverage
- Storage of the Application state in a Snapshot scenario is performed in the external testing environment, using the SCADE Test Services API
- Coverage measurement is loaded and reported in the SCADE Test Environment



Producing Code for Testing on External Platforms

Must generate code depending on model and code generation options to produce an API whose functions can save/load snapshots

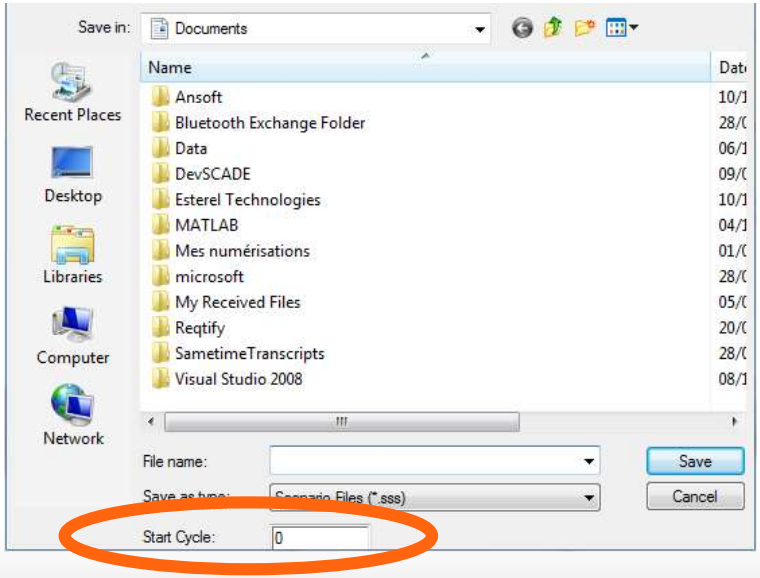
- Set “Test Services API” adaptor in Code Generator settings (Code Integration tab)
- The API is made of the following files: `<ProjectName>_snapshot.h` and `<ProjectName>_snapshot.c`
- These files can be embedded in an external test environment and the main functions of the API allow to save/load application states in snapshots



Managing Scenario Files in Simulation

Save, load, unload files using the toolbar buttons or using the menu: *Simulation > Scenario*

- Save  : save the input history of the current simulation execution in a file (.in, .sss, .sts, .sns)
- Users can select from which cycle you want to save a scenario. By default it is from the first cycle



Managing Scenario Files in Simulation

Load  : unload the previous file (if any), load a file, and simulate from the loaded file:

- When loading a state scenario, the simulator is reset
- Otherwise, the simulation continues from the current cycle

Unload  : unload the current scenario file (.in or .sss).

- When a scenario is loaded, the input values cannot be changed manually

Simulator Script Scenario Files

Simulation-dedicated TCL commands:

- `SSM::<command>`
- With `<command>` = an instruction
- List of main instructions:
  `set`, `get`, `cycle`, `check`, `uncheck`, `set_tolerance`, `default_inputs`, `get_obs_list`, `random_inputs`

A SSS scenario can drive and observe the simulation, by giving values to the inputs and getting back the values of the outputs.

See details in Simulator Technical Document

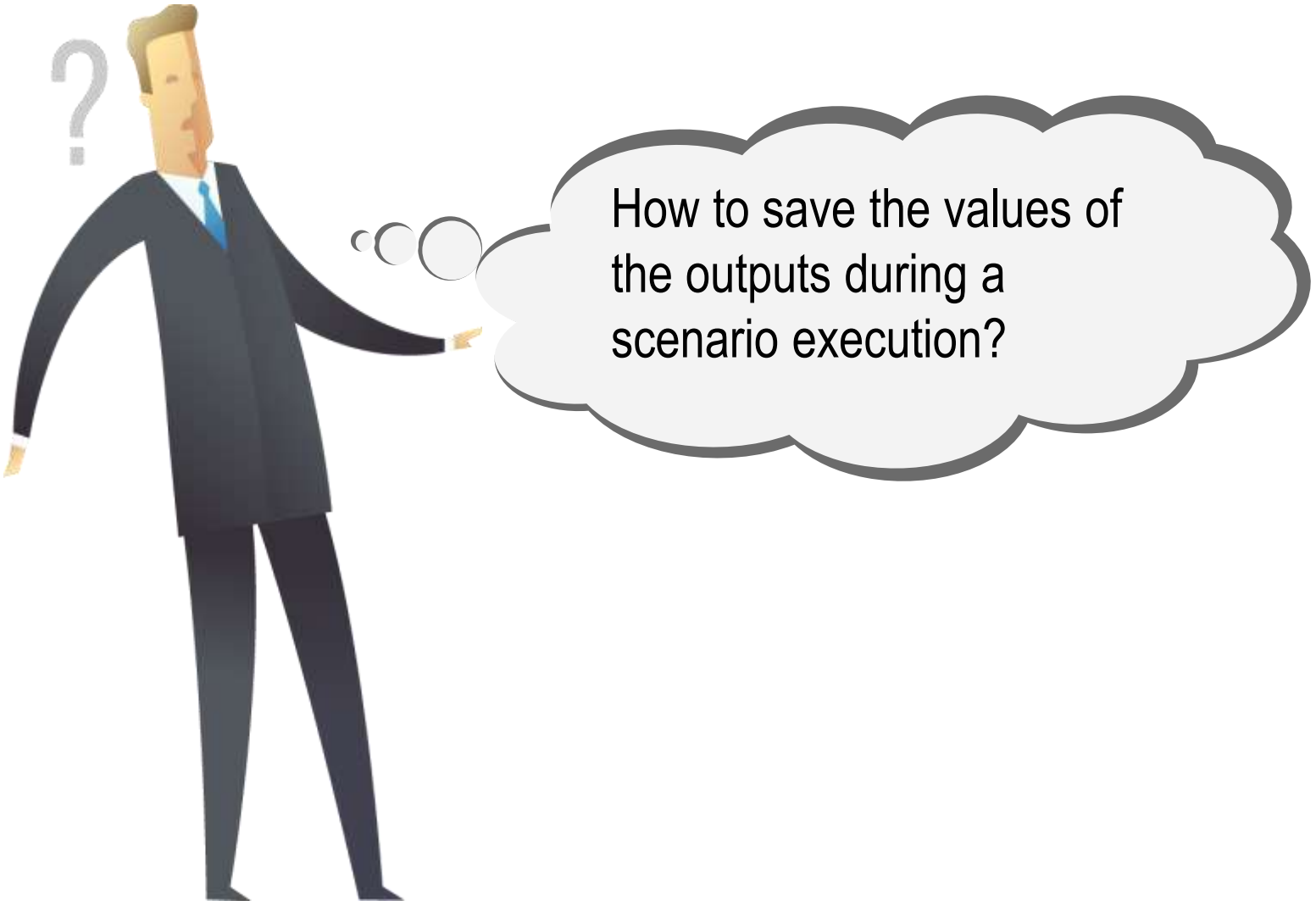
Simulator Script Scenario Files

Example

Scenario.sss

```
set max 100
set i 0

set fd [ open "log.txt" w ]
while { $i < $max } {
   set Accelerator [ expr int(rand() * 100) ]
   set Brake 0

   SSM::set System::SystemSimul/Accelerator $Accelerator
   SSM::set System::SystemSimul/Brake $Brake
   SSM::cycle

   set res1 [ SSM::get System::SystemSimul/VehicleSpeed ]
   set res2 [ SSM::get System::SystemSimul/CruiseSpeed ]

   if { $res1 < 20 } {
      puts $fd "$res1 km/h: Too slow!"
   }
   else {
      if { $res1 < 130 } {
            puts $fd "$res1 km/h: Too fast!"
         }
         else {
            puts $fd "$res1 km/h: Good driver!"
         }
   }
   incr i
}

close $fd
```

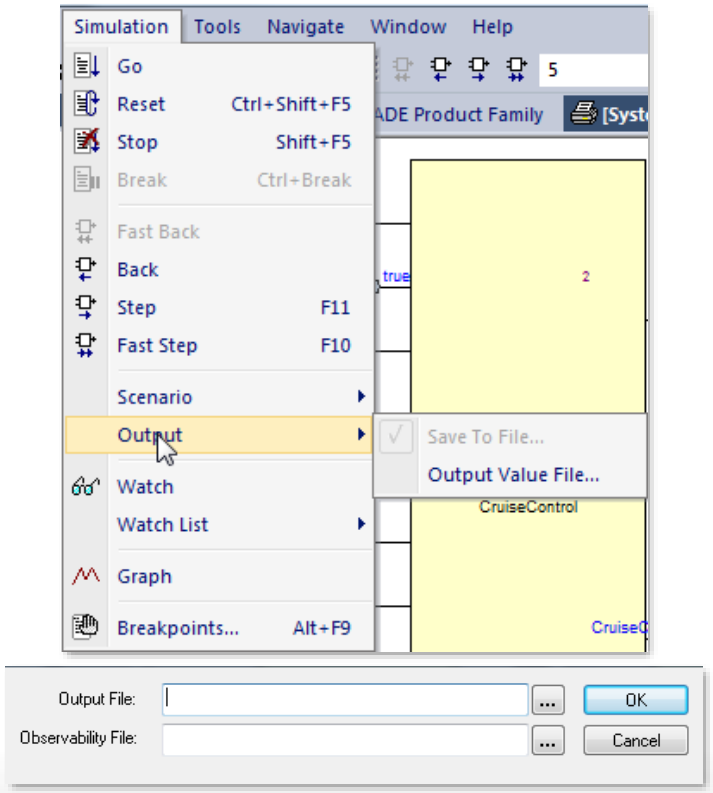
```
# Cruise Control Behavior Tests Cases


# Test Reference: CC_TEST_CCB_01
#
# Test Case Objectives:
#    verify the CC is OFF when on button was never activated.
#
# Covered Requirements: CC_HLR_CCB_01
#
# Test Case Acceptance Criteria
#    check that CruiseState is OFF
#
# Test Case Description
SSM::set Accel 50.0
SSM::set Speed 100.0
SSM::check CruiseState CruiseControl::OFF sustain=22
SSM::cycle 20
SSM::cycle 1

SSM::set Accel 0.0
SSM::cycle 1
```


How to save the values of the outputs during a scenario execution?

Output Files

Save all inputs/outputs and selected observable variables computed during the simulation session.

First, select the output file and the observability file:



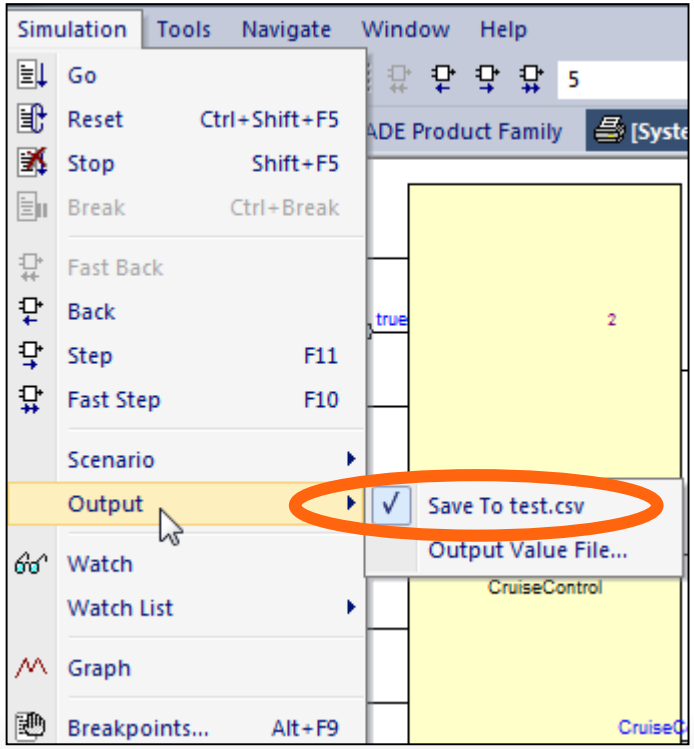
- Output File: This type of file contains input and output values from your simulation sessions.
  File format is “.out“ or “.csv”

- Observability File: This type of file specifies a list of observable variables added in the output file at each simulation cycle.
  File format is “.obs”.

Output Files

From Simulation menu, select “Save To” item and set the output file name

Then, start or stop recording the values in the selected output file from the selected cycle



Output Files

Two output formats are available:

- a textual output file (.out) giving all values for each executed cycle

```
STEP 1
INPUT System::SystemSimul/On = false
INPUT System::SystemSimul/Off = false
INPUT System::SystemSimul/Resume = false
INPUT System::SystemSimul/Set = false
INPUT System::SystemSimul/QuickAccel = false
INPUT System::SystemSimul/QuickDecel = false
INPUT System::SystemSimul/Accel = 0.0
INPUT System::SystemSimul/Brake = 0.0
OUTPUT System::SystemSimul/CruiseSpeed = 0.0
OUTPUT System::SystemSimul/CruiseState = CruiseControl::OFF
OUTPUT System::SystemSimul/CarSpeed = 0.0

STEP 2
INPUT System::SystemSimul/On = false
INPUT System::SystemSimul/Off = false
INPUT System::SystemSimul/Resume = false
INPUT System::SystemSimul/Set = false
```

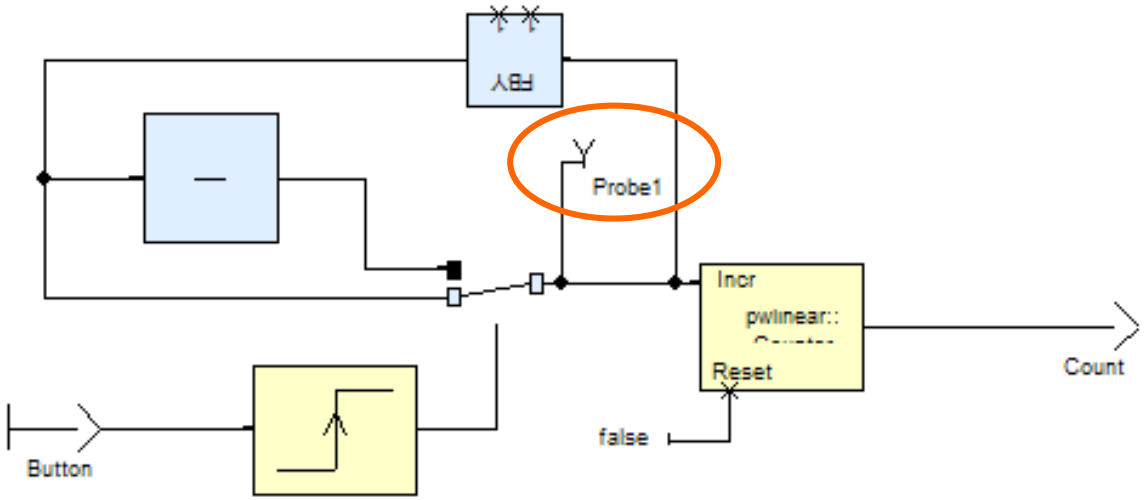
- a csv format file, which can be directly imported into Excel

```
STEP;System::SystemSimul/On;System::SystemSimul/Off;System::SystemSimul/Resume;System::SystemSimul/Set;
11;false;false;false;false;false;false;100.0;0.0;0.0;CruiseControl::OFF;15.914;
12;false;false;false;false;false;false;100.0;0.0;0.0;CruiseControl::OFF;18.552;
13;false;false;false;false;false;false;100.0;0.0;0.0;CruiseControl::OFF;21.185;
```

Probes

Defines a new variable to associate with one flow: 

Enhance observation by allowing the watch of any internal flow during the simulation or execution of the generated code.



Format of OBS Files

Each line must provide the path of variable. If this path contains any space, it has to be surrounded by quotes.

- Variable at top level

```
ABC_N::ABC_N/lockFall
```

- Variable in an instantiated and iterated operator

```
"ABC_N::ABC_N/ABC_N::Button 4[0]/foreground"
```

- Variable in a state machine

```
"ABC_N::ABC_N/ABC_N::Button
4[1]/StateMachine1:Preselected:_L3"
```

Format of OBS Files

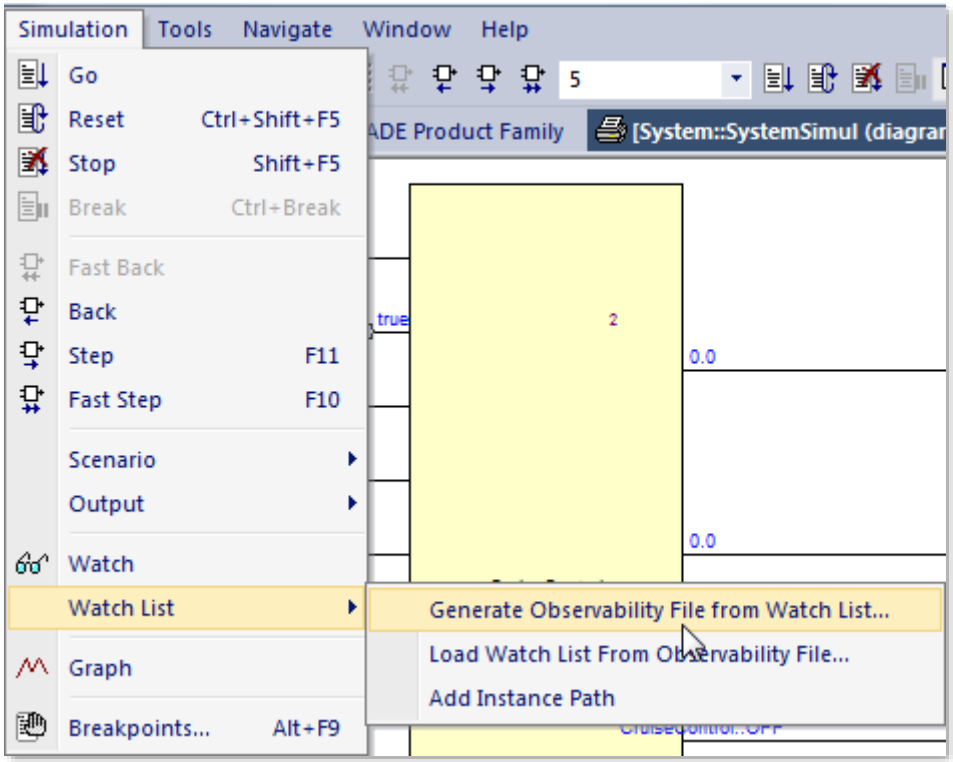
All characters included between the first character following the last quote or the first space, and the end of the line are considered as a comment:

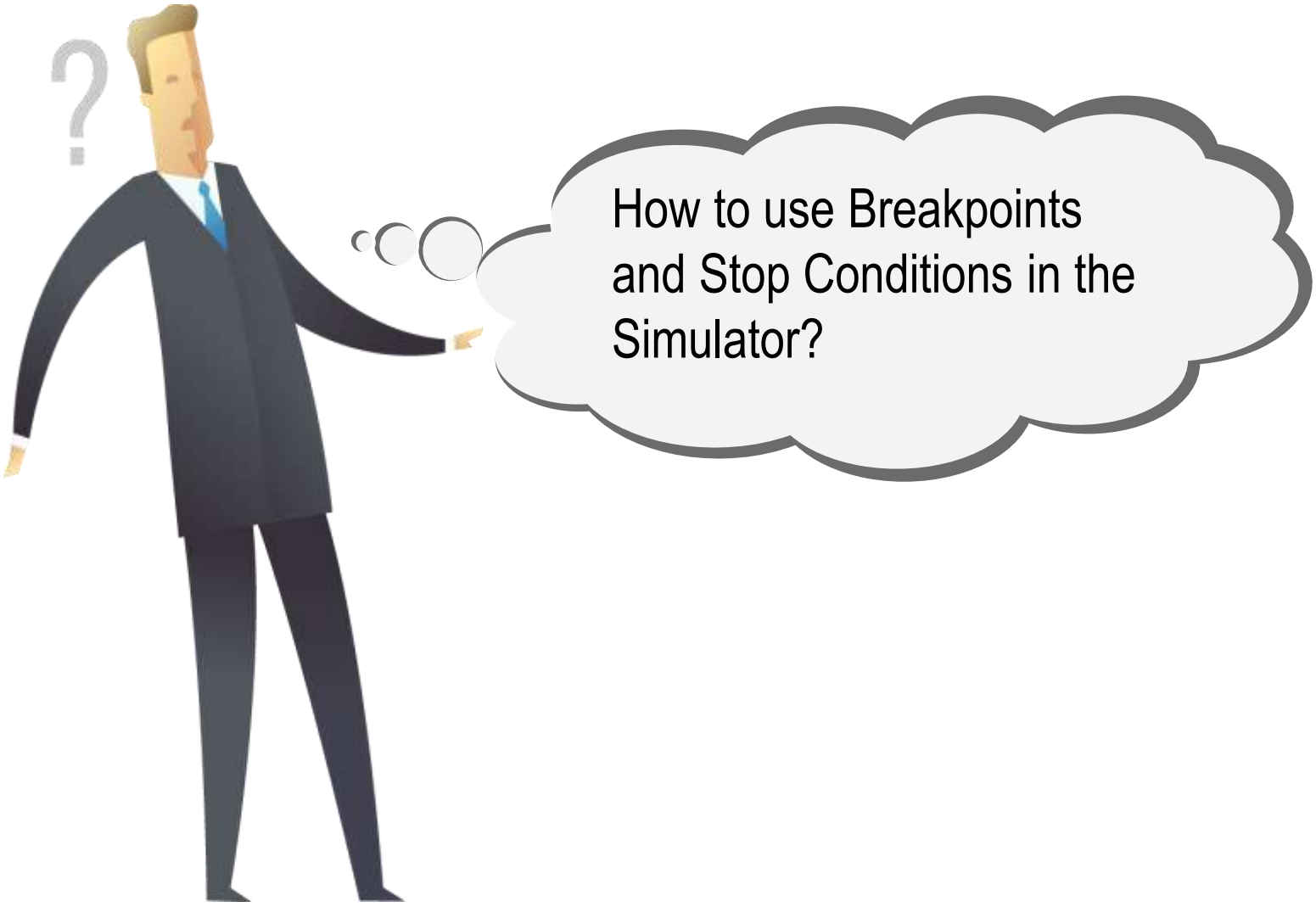
```
ABC_N::ABC_N/Lock this is a comment
```

Format of OBS Files

Automatically generate an OBS file from the Watch window

Import an OBS file to the Watch Window




How to use Breakpoints and Stop Conditions in the Simulator?

Breakpoints and Stop Conditions

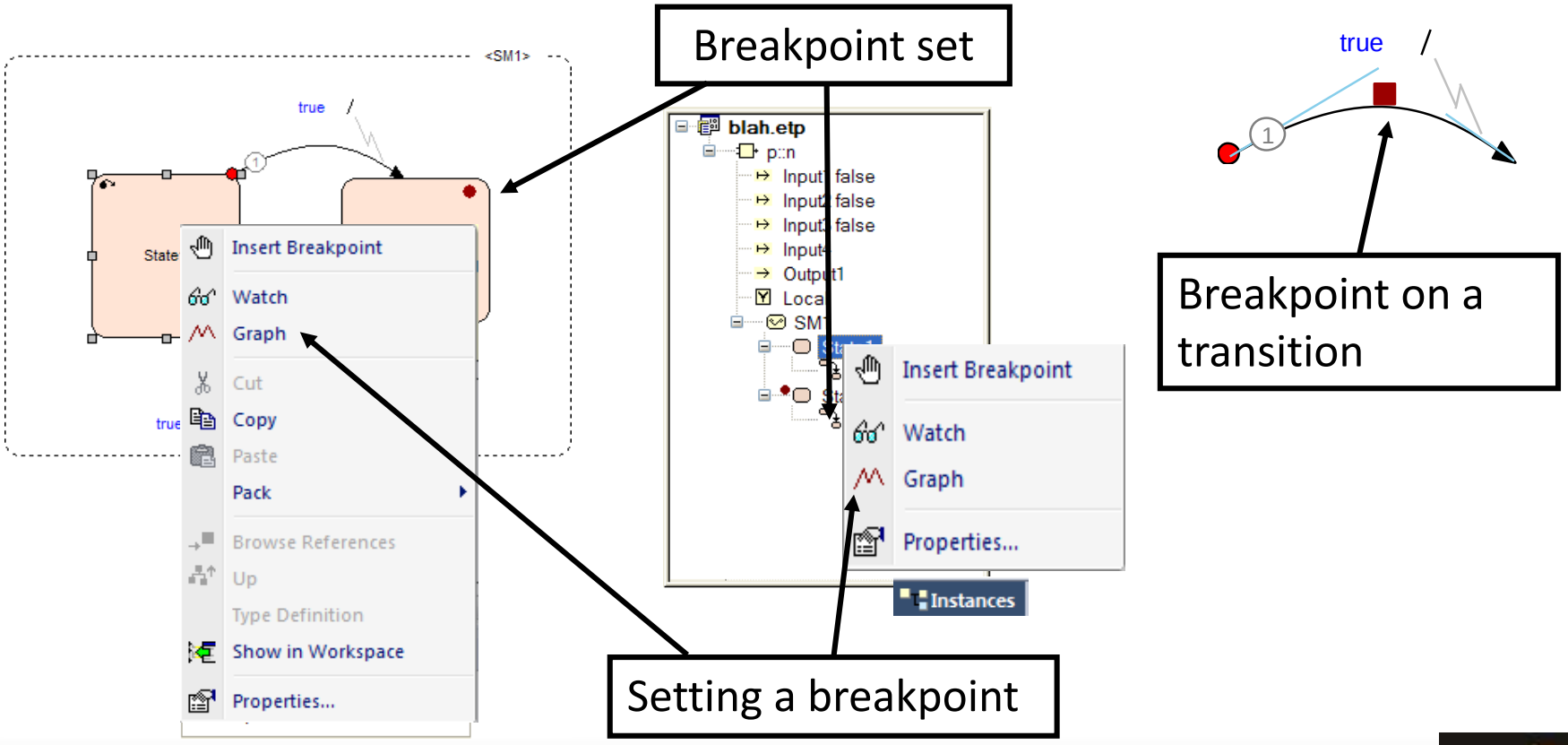
Break points and stop conditions help users to reach specific situations during a simulation.

You have to start the simulation to be able to set breakpoints.

Each time a breakpoint or a stop condition is reached during one cycle, the simulation pauses and the breakpoint owner turns red.

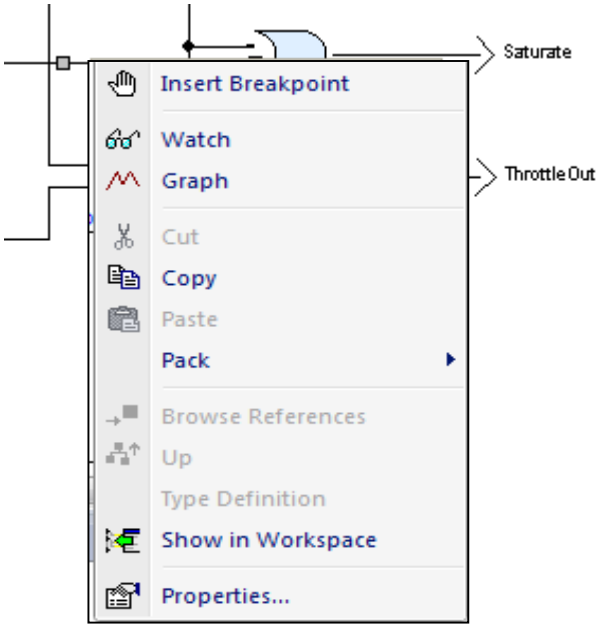
Breakpoints on State Machines

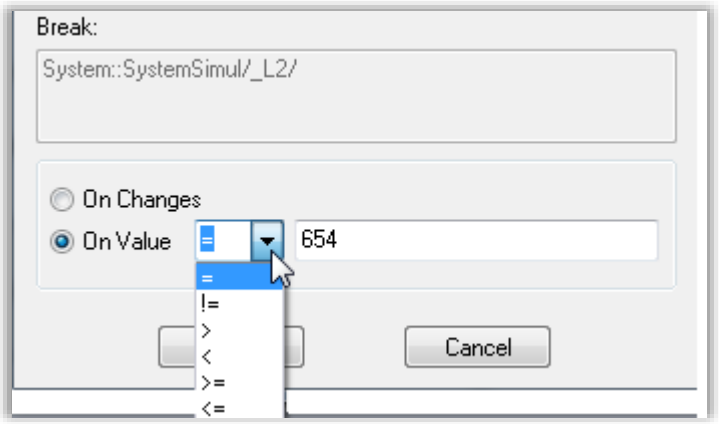
Right click on a state or a transition, on canvas or in the instance tree to set, disable or remove breakpoints.



Breakpoints on Dataflow

Right click on a flow and insert a breakpoint.





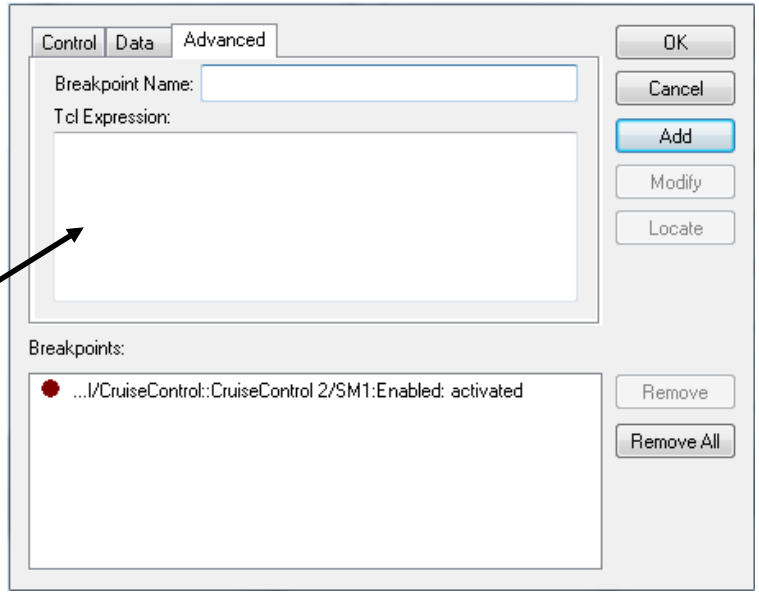
Users can stop:

- When the value of the flow changes
- When the value is =, !=, >, <, <= or >= with a user defined value

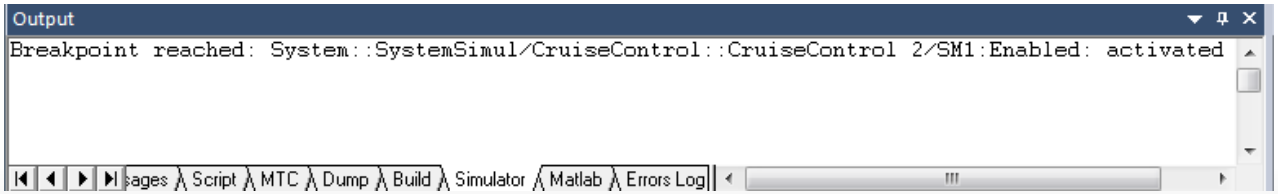
Advanced Breakpoints : Stop Conditions

Stop conditions are used if a more complex expression is needed for a breakpoint:

- During simulation, menu: Simulation – Breakpoints (Alt-F9)
- Click on Advanced tab
- Give a name to your advanced breakpoint
- Type the TCL expression
- Click on *Add*



Each time a breakpoint or a stop condition is reached during one cycle, a message occurs:



Assume & Guarantee

Assume and Guarantee are SCADE Suite constructs defining Boolean expressions that can be attached to operators:

- Expressions used in **assume** can only mention input identifiers or past variables. No such restriction is imposed for **guarantee** expression
- An **assertion** is well typed if the associated expression is of Boolean type

These conditions form the contract of the operator with respect to its environment:

- Safer control during simulation or execution
- Proof objectives for formal verification

Scade Contracts

Characterized by a pair of observers:

- Assume: observes inputs and the past of the outputs

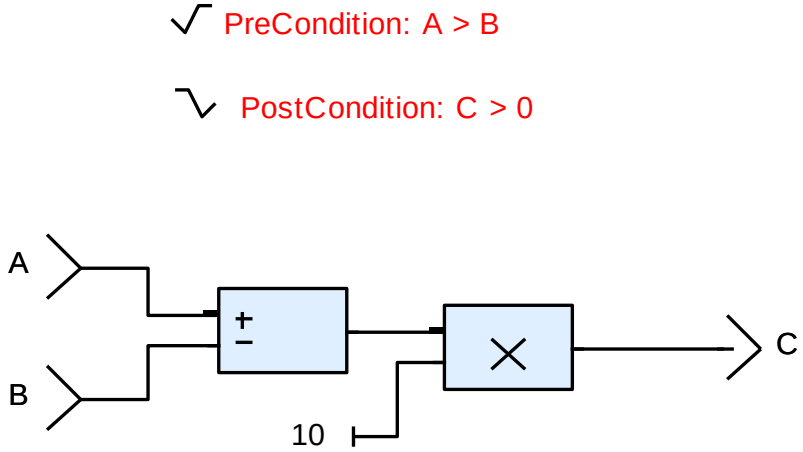
- Guarantee: observes inputs and outputs

Adds information about the environment into the SCADE Suite model

Accelerates tests and proofs by restricting the domains of the inputs

Scade Contracts

Example




```
node N (A:int;B:int)
returns (C:int)
let
  assume PreCond: A > B;
  guarantee PostCond: C > 0;
  C = 10 * ( A - B ) ;
tel ;
```

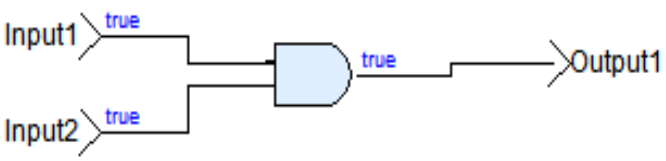
Assume & Guarantee

Assume and Guarantee clauses can be used to stop the simulation when they are violated.

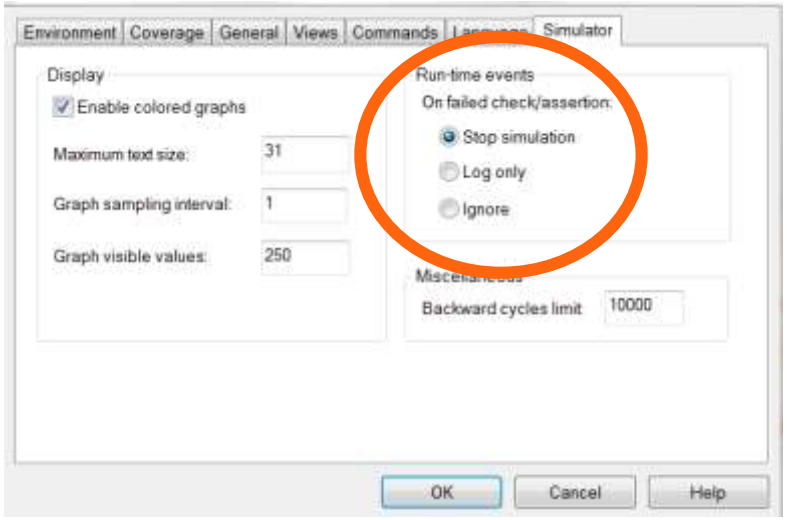
Stop on assertions should be selected before:

Select *Tools > Options*, then on Simulator tab set “Stop simulation” in Run-Time Events

Assertions must be inserted into the equation blocks before the simulation.



A1: Output1 = false



Exercise 1: Cruise Control

Time: 10 min

Open Day4/Prerequisite/Exercise 1/CruiseControl.etp project

Create a new Guarantee on the output cruiseSpeed of CruiseControl node:

$$cruiseSpeed \leq 130.0$$

Configure the simulator to stop the simulation on failed check/assertion

Open the simulation and run Day 4\Solution\Exercise 1\Scenarios\CruiseSpeedTest.sss scenario file

Observe the inputs/outputs value when the assertion has failed

Exercise

Exercise 1: Cruise Control

CC_HLR_CSM_05
The Cruise Speed shall be maintained between SpeedMin and SpeedMax km/h values.

CC_HLR_CCP_01
SpeedMin: shall be 30.0 Km/h

CC_HLR_CCP_02
SpeedMax: shall be 150.0 Km/h

Change the guarantee according to the requirements:

$$cruiseSpeed \leq 150.0$$

Open the simulation and run *Day 4\Solution\Exercise 1\Scenarios\CruiseSpeedTest.sss* scenario file

Exercise

AGENDA

Simulation

Generated Code

Integration

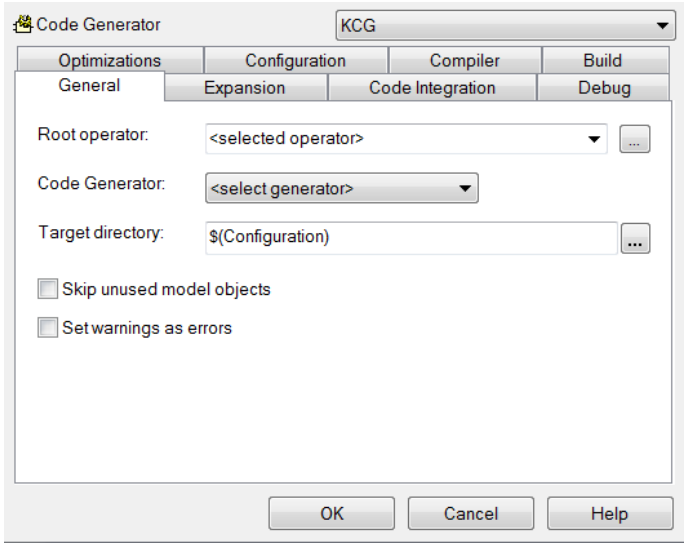
Traceability

Document Report

Model-Based V&V

Generated Code: I Know!!!

I know how to set KCG options



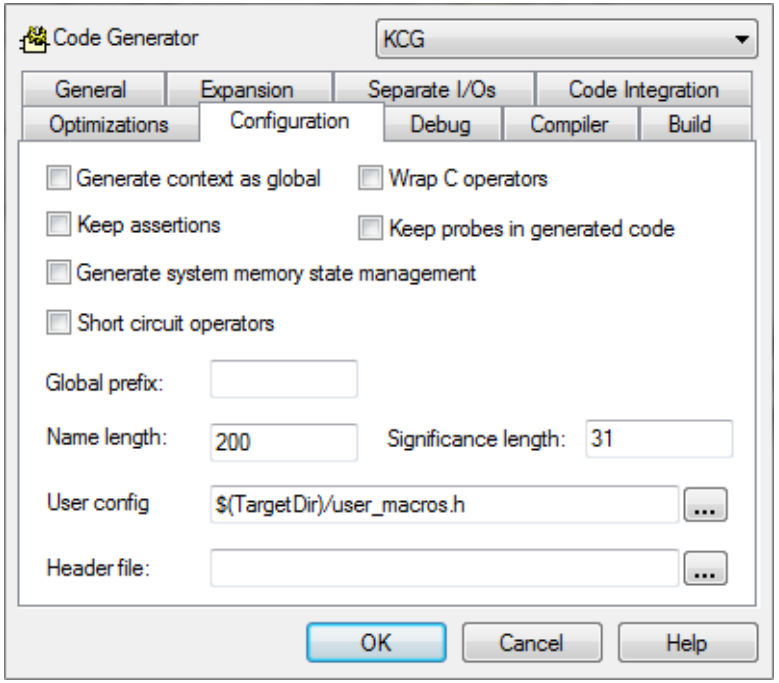
I know how to generate code

Configuration Options (C Code)

Generate context as global

(-global_root_context)

- Modifies the global data types and architecture of execution of the generated C code. Inputs and context variables are not passed as arguments to root functions but as global C variables



Wrap C operators

(-wrap_c_ops)

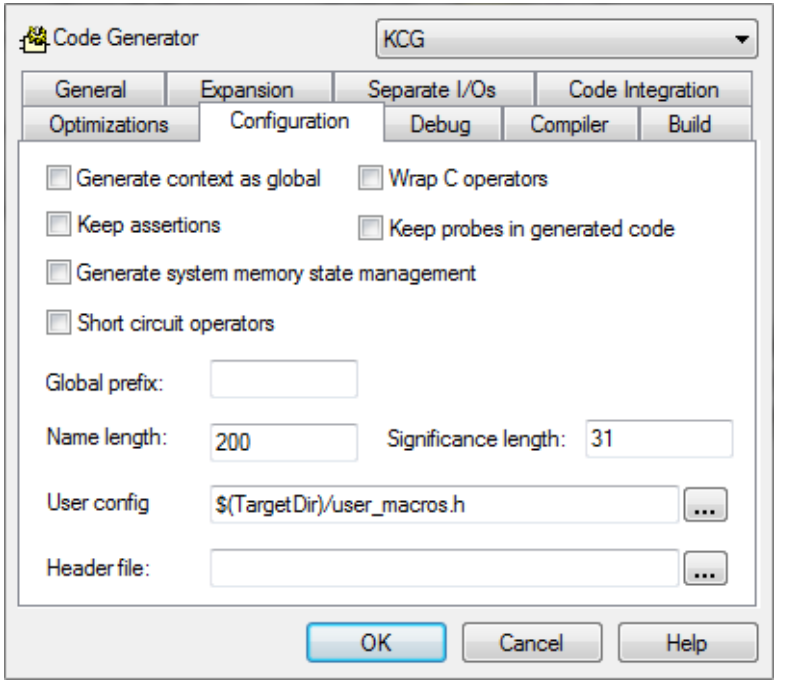
- Encapsulates arithmetic, binary and relational operators into C protected macros (see SCADE Technical Manual for the complete operators list)

Keep assertions

(-keep_asserts)

- If this option is set, a call is performed on user-defined macros named kcg_assume (resp. kcg_guarantee) for each assume (resp. guarantee) equation in the input model

Configuration Options (C Code)



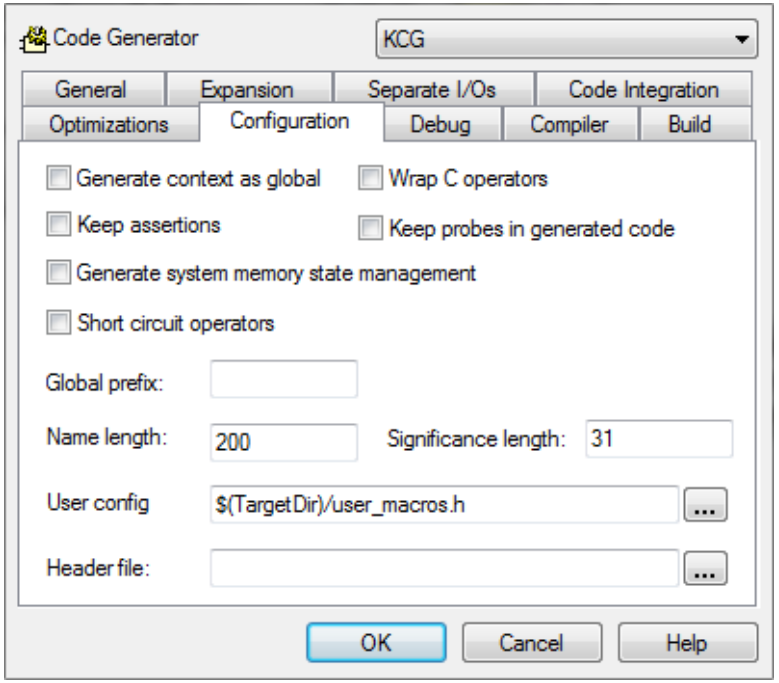
Keep probes in generated code
(-probes)

- All C variables corresponding to probes in the input model are kept in the contexts corresponding to node calls

Generate system memory state management
(-state_vector)

- Generates the state vector structure and function prototypes in the header files of the root operator and of dependent operators

Configuration Options (C Code)



Short circuit operators
(-short_circuit)

- Forces the use of short circuit operators (&& and ||) instead of bitwise operators

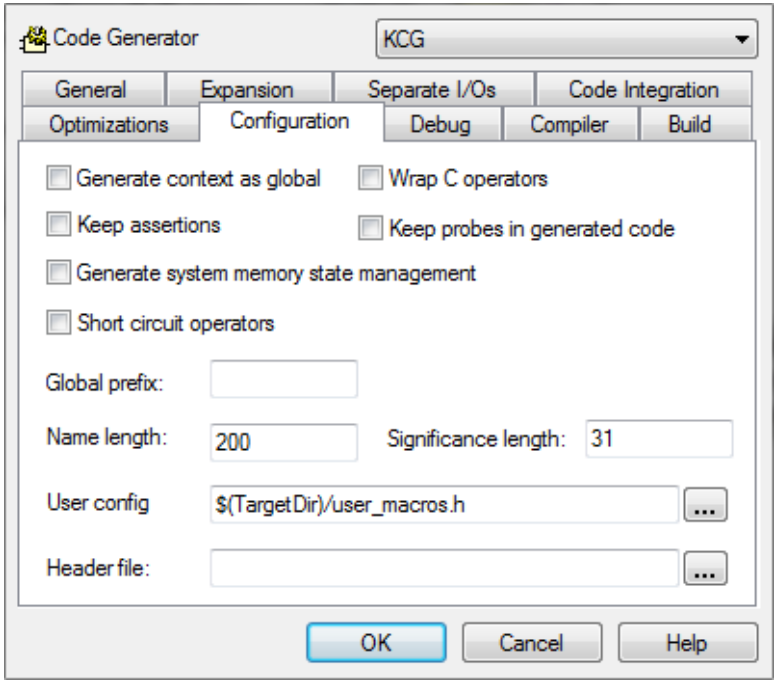
Name length
(-name_length <nlength>)

- Limits the length of the name of generated identifiers to <nlength> characters

Significance length
(-significance_length <slength>)

- Allows to differentiate two generated identifiers on their first <slength> characters

Configuration Options (C Code)



Global prefix
(-globals_prefix <prefix>)

- All global names are prefixed by the <prefix> string

User config
(-user_config <filename>)

- Includes the header file <filename> at the beginning of the generated file kcg_types.h

Header file
(-header <filename>)

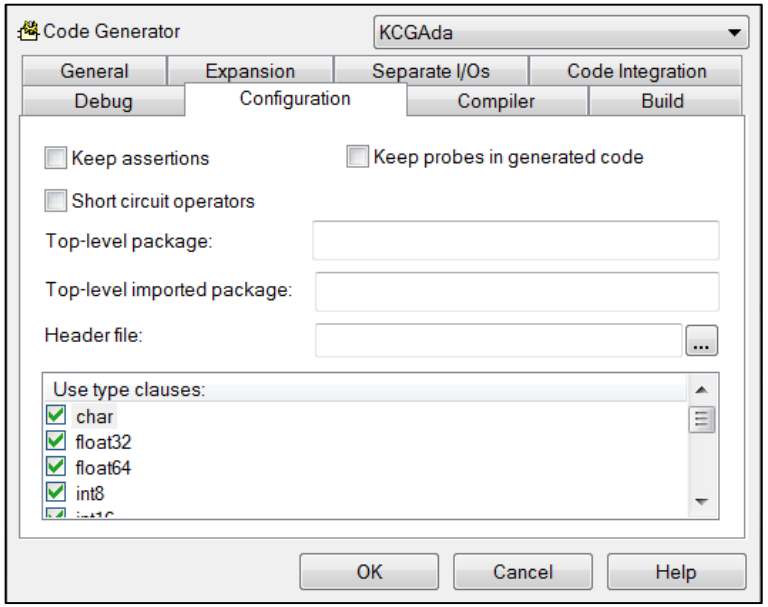
- Adds the content of the file <filename> at the beginning of each generated file. <filename> should only contain valid C comments

Configuration Options (Ada)

Ada

Keep assertions (-keep_asserts)

- Translates assume and guarantee constructs using procedure calls
- Users must provide all procedure definitions in kcg_config.ads/adb files



Short circuit operators (-short_circuit)

- Forces the use of short circuit operators (&& and ||) instead of bitwise operators

Keep probes in generated code (-probes)

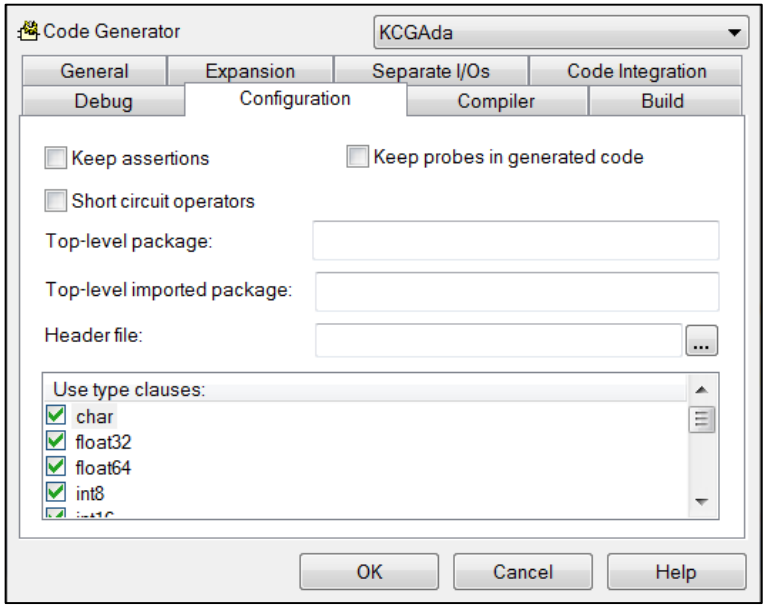
- All variables corresponding to probes in the input model are kept in the contexts corresponding to node calls

Configuration Options (Ada)

Ada

Top-level package
(-root_package <path>)

- Specifies an Ada valid path where all Ada code is generated



Top-level imported package
(-imported_root_package <path>)

- Specifies an Ada valid path where all Ada imported code must be provided by users

Header file
(-header <filename>)

- Browses for files and specifies a file whose content is added at the beginning of each generated file before the banner generated by Code Generator

Use type clauses (-use_types <list>)

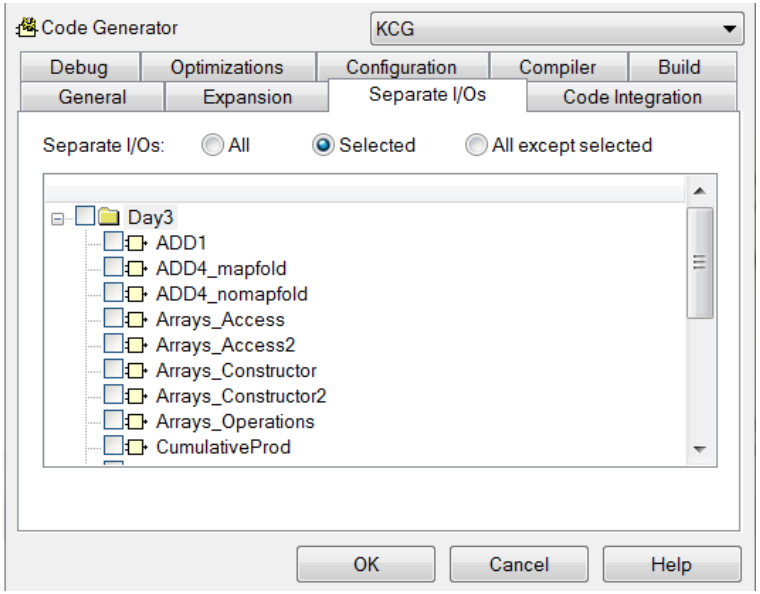
- For each checked predefined type, a “use type” clause is generated in the Ada code when needed

Separate I/Os Options

All

(-separate_io_all)

- To mark all operators for separate I/Os



Selected

(-separate_io <pathlist>)

- To define the list of operators (checked) marked for separate I/Os with respect to the usage of pragmas or not

All except selected

(-no_separate_io <pathlist>)

- To define the list of operators (checked) not marked for separate I/Os with respect to the usage of pragmas or not

List of operators marked for separate I/Os:

Generated cyclic function of selected operator is different for I/Os:

- Outputs are passed as separate parameters
- Inputs are no more passed in input context for the root operator

Exercise 2: Cruise Control

Time: 10 min

Generate KCG code with different options and observe the generated files:

C code:

- A global prefix,
- Generate context as global,
- …

Ada code:

- Top-level package
- Use Type clause
- ….

Exercise

Fixed-Point Representation

AGENDA

Simulation

Generated Code

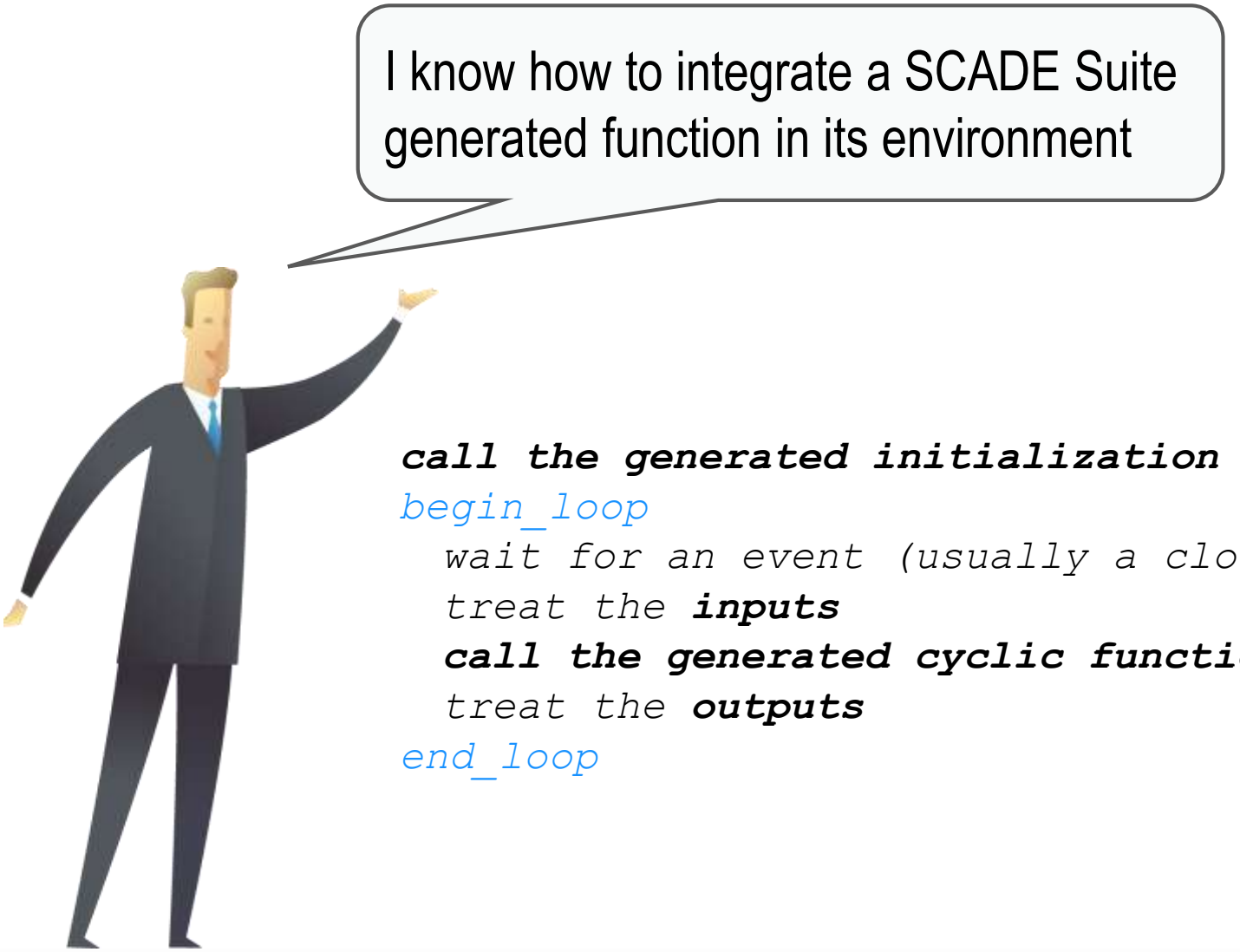
Integration

Traceability

Document Report

Model-Based V&V

Integration: I Know!!!




```
call the generated initialization function
begin_loop
  wait for an event (usually a clock signal)
  treat the inputs
  call the generated cyclic function
  treat the outputs
end_loop
```

What Else?

Where are the SCADE Suite types defined?

How to modify the SCADE Suite predefined types?

What are the integration options proposed by SCADE Suite?

C Code: kcg_types.c/h

Contains C code related to the types defined in the input model:

kcg_types.h contains:

- Definition of all predefined types
- Definition of boolean constants
- Definition of assignment macros
- Definition of enumerated types
- Definition of structured types, and copy/compare functions
- …

kcg_types.c contains:

- Definition for each structure comparison function
- Definition for each array comparison function
- …

C Generated Code: kcg_types.h

Example

Predefined types

```
#ifndef _KCG_TYPES_H_
#define _KCG_TYPES_H_

#include "stddef.h"

#define KCG_MAPW_CPY

#include "./user_macros.h"

#ifndef kcg_char
#define kcg_char kcg_char
typedef char kcg_char;
#endif /* kcg_char */

#ifndef kcg_bool
#define kcg_bool kcg_bool
typedef unsigned char kcg_bool;
#endif /* kcg_bool */
```

Assignment macros

```
#ifndef kcg_assign
#include "kcg_assign.h"
#endif /* kcg_assign */

#ifndef kcg_assign_struct
#define kcg_assign_struct kcg_assign
#endif /* kcg_assign_struct */

#ifndef kcg_assign_array
#define kcg_assign_array kcg_assign
#endif /* kcg_assign_array */
```

Enum types

```
/* CruiseControl::teCruiseState */
typedef enum kcg_tag_teCruiseState_CruiseControl {
  OFF_CruiseControl,
  INT_CruiseControl,
  STDBY_CruiseControl,
  ON_CruiseControl
} teCruiseState_CruiseControl;
/* CruiseControl::CruiseControl::SM1 */
typedef enum kcg_tag_SSM_TR_SM1 {
  SSM_TR_no_trans_SM1,
  SSM_TR_Off_1_SM1,
  SSM_TR_Enabled_1_SM1
} SSM_TR_SM1;
/* CruiseControl::CruiseControl::SM1 */
typedef enum kcg_tag_SSM_ST_SM1 {
  SSM_st_Off_SM1,
  SSM_st_Enabled_SM1
} SSM_ST_SM1;
/* CruiseControl::CruiseControl::SM1::Enabled::SM2 */
typedef enum kcg_tag_SSM_TR_SM2_SM1_Enabled {
  SSM_TR_no_trans_SM2_SM1_Enabled,
  SSM_TR_Active_1_SM2_SM1_Enabled,
  SSM_TR_Interrupt_1_SM2_SM1_Enabled
} SSM_TR_SM2_SM1_Enabled;
```

...

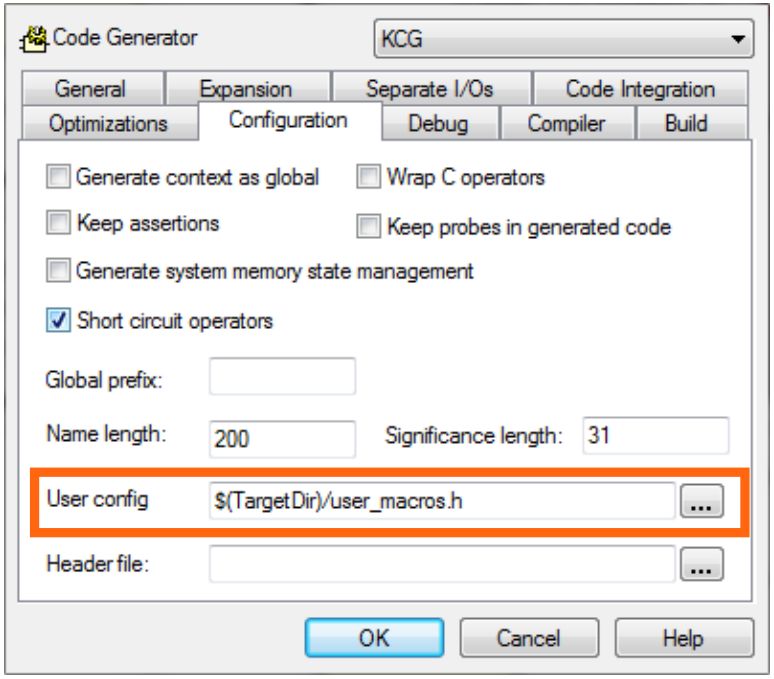
How to Modify Predefined SCADE Suite Types (C Code)

Use the –user_config KCG option to set a user configuration file containing macro definition.

Example

User_macro.h define int16 type:

```
#ifndef _USER_MACRO_H_
#define _USER_MACRO_H_

#ifndef kcg_int16
#define kcg_int16 kcg_int16
typedef signed int kcg_int16;
#endif /* kcg_int16 */


#endif /* _USER_MACRO_H_ */
```



How to Modify Assignment Macros? (C Code)

kcg_assign macro determines “memcpy”:

The kcg_assign macro must be provided by one of the following means:

- -user_config option mechanism
- kcg_assign.h file

kcg_assign.h header file must be provided by the user

C Code: kcg_assign.h

Example

Find an example in <install_dir>\SCADE\include

```
#ifndef kcg_assign
#include "string.h"
#define kcg_assign(kcg_C1, kcg_C2, kcg_size) (memcpy((kcg_C1), (kcg_C2), (kcg_size)))
#endif /* kcg_assign */
```

KCG_COPY Macro (C Code)

A copy function is a two-parameters macro generated to copy an object of a structured type (structure and array) into another object of the same type

Both parameters are given by reference

- The first parameter is the target and the second is the source

KCG_COPY Macro (C Code)

kcg_types.h file contains a copy function:

- For each structure type equivalence class if the input model contains structure type definitions
  - Named *kcg_copy_structname*
- For each array type equivalence class if the input model contains array type definitions
  - Named *kcg_copy_arrayname*

KCG_COPY Macro (C Code)

kcg_types.h:

- If the type is a structure named *structname* then *kcg_copy* is generated as follows:

```
#ifndef kcg_copy_structname
#define kcg_copy_structname(kcg_C1 , kcg_C2 ) \
( kcg_assign_struct ( ( kcg_C1 ) , (kcg_C2 ) , sizeof (structname ) ) )
#endif
```

- If the type is an array named *arrayname* then *kcg_copy* is generated as follows:

```
#ifndef kcg_copy_arrayname
#define kcg_copy_arrayname(kcg_C1 , kcg_C2 ) \
( kcg_assign_array ( ( kcg_C1 ) , (kcg_C2 ) , sizeof (arrayname ) ) )
#endif
```

KCG_COPY Macro (C Code)

KCG optimizes the generation of kcg_copy macros

A copy function is mainly generated in the target code when

- An output crosses bottom-up one or several nested nodes (so with memories)
- The design assigns a new value from a structured input to a structured output

Context Structure (C Code): reminder

An input context is a structure in the generated C code that gathers the inputs of an operator:

- Defined by: `typedef struct { inputs } structname;`
  where *structname* is the canonical name of the operator with prefix inC_

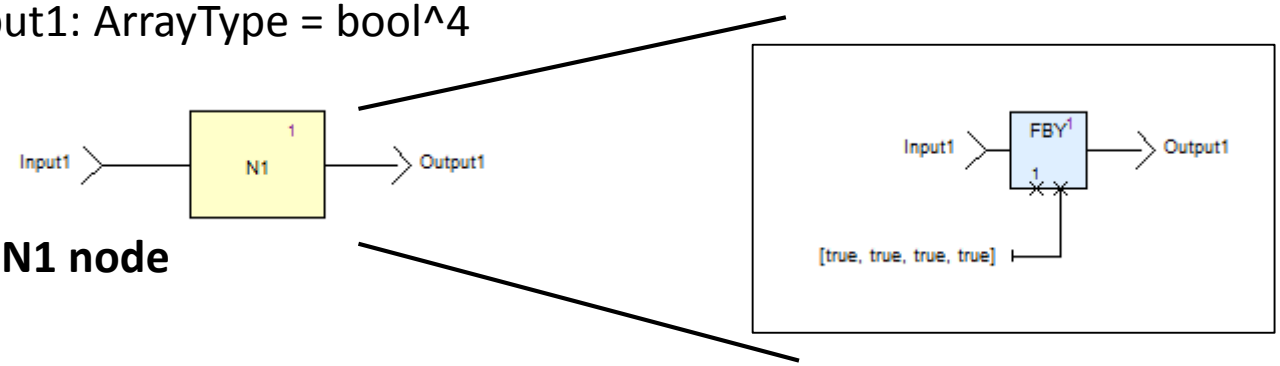
The operator output context is the variables set composed of

- All its memories, initialization variables and recursively output contexts of sub-operators instances
- Depending on code generation options: all outputs, all probes, some local variables, all assertions, and variables containing the clock value of observable variables
- Defined by: `typedef struct { … } structname;`
  where *structname* is the canonical name of the operator with prefix outC_

KCG_COPY Macro (C Code)

Example

Input1 and Output1: ArrayType = bool^4



Root Node calls N1 node

Root.c:

```
void Root(inC_Root *inC, outC_Root *outC)
{
  N1(&inC->Input1, &outC->Context_1);
  kcg_copy_ArrayType(&outC->Output1, &outC->Context_1.Output1);
}
```

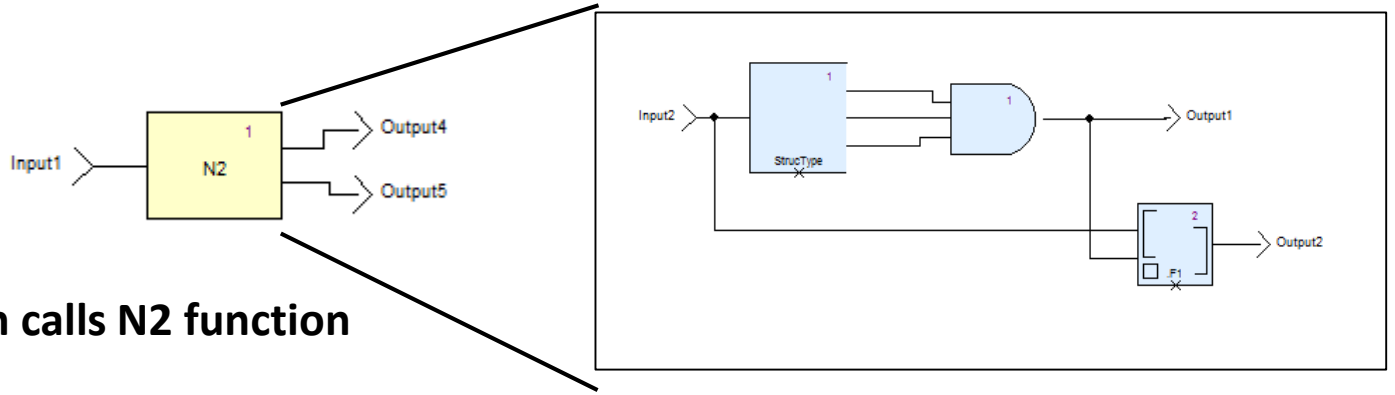
N1.c:

```
void N1(ArrayType *Input1, outC_N1_FakeData *outC)
{
  /* fby_1_init_1 */ if (outC->init) {
    outC->init = kcg_false;
    outC->Output1[0] = kcg_true;
    outC->Output1[1] = kcg_true;
    outC->Output1[2] = kcg_true;
    outC->Output1[3] = kcg_true;
  }
  else {
    kcg_copy_ArrayType(&outC->Output1, &outC>rem_Input1);
  }
  kcg_copy_ArrayType(&outC->rem_Input1, Input1);
}
```

Without expansion option

KCG_COPY Macro (C Code)

Input1 and Output2: StrucType = {F1:bool, F2:bool, F3:bool}
Output1: bool



Root function calls N2 function

Root.c:

```
void Root(inC_Root *inC, outC_Root *outC)
{
  N2(&inC->Input1, &outC->Output4, &outC->Output5);
}
```

Without expansion option

N2.c:

```
void N2(StrucType *Input2, kcg_bool *Output1, StrucType *Output2)
{
  *Output1 = (*Input2).F1 & (*Input2).F2 & (*Input2).F3;
  kcg_copy_StrucType(Output2, Input2);
  (*Output2).F1 = *Output1;
}
```

Example

Ada Code Generation Files

Ada

The default definitions (predefined types, …) are not generated but are contained in

- kcg_config.ads located under *<installation>\SCADE\include\Ada* and
- Kcg_config.adb located under *<installation>\SCADE\lib\Ada*

```
-- Predefined types definitions
subtype Kcg_Char is Character;
subtype Kcg_Int8 is Interfaces.Integer_8;
subtype Kcg_Int16 is Interfaces.Integer_16;
subtype Kcg_Int32  is Interfaces.Integer_32;
subtype Kcg_Int64  is Interfaces.Integer_64;
subtype Kcg_Uint8  is Interfaces.Unsigned_8;
subtype Kcg_Uint16 is Interfaces.Unsigned_16;
subtype Kcg_Uint32 is Interfaces.Unsigned_32;
subtype Kcg_Uint64 is Interfaces.Unsigned_64;
subtype Kcg_Float32 is Interfaces.IEEE_Float_32;
subtype Kcg_Float64 is Interfaces.IEEE_Float_64;
subtype Kcg_Size is Integer;
```

ANSYS

Ada Code Generation Files

Ada

path *corresponding to the package hierarchy leading to the package or operator*

`kcg_types.ads`: contains

- manifest user-defined types as *types*
- structures for the State Machines as *types*

`<package-path>.ads`: Ada specification for non-imported elements (non-expanded operators)

- Non-manifest Scade types as *types*
- Constants
- I/Os root operator and sub-operators structures (output contexts) as *types*
- init, reset and cyclic procedures for root and sub-nodes
- Function if the non expanded Scade function is not the root operator and returns a single output otherwise a procedure

ANSYS

Ada Generated Code: kcg_types.ads

Ada

Example

```
package Kcg_Types
is

  -- CruiseControl::teCruiseState/
  type teCruiseState is (OFF, INT, STDBY, ON);

  -- CruiseControl::CruiseControl/SM1:
  type SSM_TR_SM1 is
    (SSM_TR_no_trans_SM1,
    SSM_TR_Off_Enabled_1_Off_SM1,
    SSM_TR_Enabled_Off_1_Enabled_SM1);

  -- CruiseControl::CruiseControl/SM1:
  type SSM_ST_SM1 is (SSM_st_Off, SSM_st_Enabled);

  -- CruiseControl::CruiseControl/SM1:Enabled:SM2:
  type SSM_TR_SM2 is
    (SSM_TR_no_trans_SM2,
    SSM_TR_Active_Interrupt_1_Active_SM2_Enabled_SM1,
    SSM_TR_Interrupt_Active_1_Interrupt_SM2_Enabled_SM1);

  -- CruiseControl::CruiseControl/SM1:Enabled:SM2:
  type SSM_ST_SM2 is (SSM_st_Active, SSM_st_Interrupt);

  -- CruiseControl::CruiseControl/SM1:Enabled:SM2:Active:SM3:
  type SSM_TR_SM3 is
    (SSM_TR_no_trans_SM3,
    SSM_TR_On_StandBy_1_On_SM3_Active_SM2_Enabled_SM1,
    SSM_TR_StandBy_On_1_StandBy_SM3_Active_SM2_Enabled_SM1);

  -- CruiseControl::CruiseControl/SM1:Enabled:SM2:Active:SM3:
  type SSM_ST_SM3 is (SSM_st_On, SSM_st_StandBy);

  -- truthtables::TruthTableValues/
  type TruthTableValues is (T, F, X);

  type Array_Float64 is
    array (Kcg_Config.Kcg_Size range <>) of Kcg_Config.Kcg_Float64;

  subtype Range_0_4 is Kcg_Config.Kcg_Size range 0 .. 4;

  subtype Float64_Range_0_4 is Array_Float64 (Range_0_4);
```

Code Integration Options

You can generate additional wrapping code or build executable code dedicated to the environment

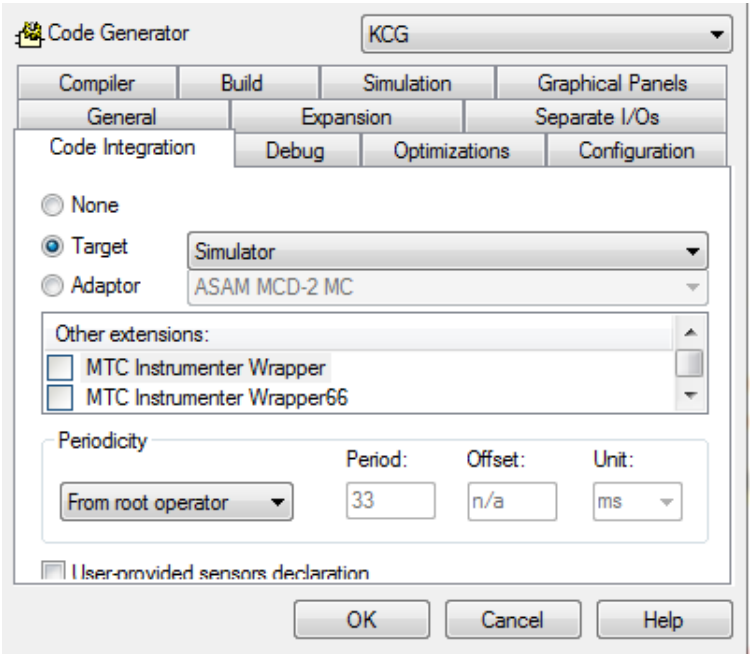
Code Integration Options (C Code)

None:

- Embedded code generation

Target:

- Code generation extends code for the selected target:
  - Simulator, FMU, Mobile, NI Veristand
  - Simplorer, Simulink, Standalone executable
  - Timing and Stack Optimizer or Verifiers



Adaptor:

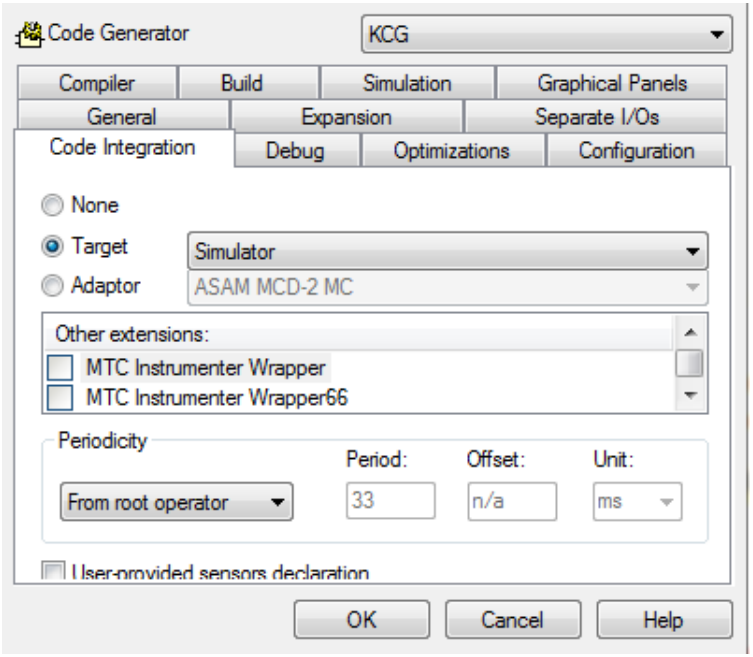
- Code generation extends code to the selected adaptor (RTOS and external tools):
  - ASAM MCD-2MC, AUTOSAR, INTEGRITY-178B , MicroC_OS-II, OSEK, PikeOS, SCADE Display Co-generation, Test Services API, VxWorks 653, VxWorks CERT

Not KCG options but SCADE Suite Editor features

Code Integration Options (C Code)

Other extensions:

- **MTC Instrumenter Wrapper:**
  - Generate KCG 6.4 MTC wrapper code
- **MTC Instrumenter Wrapper66:**
  - Generate KCG 6.6 MTC wrapper code



Periodicity:

- Set the periodicity of code execution
  - From the drop-down list
  - Directly from the entered values

Not KCG options but SCADE Suite Editor features

Code Integration Options (Ada)

Ada

None:

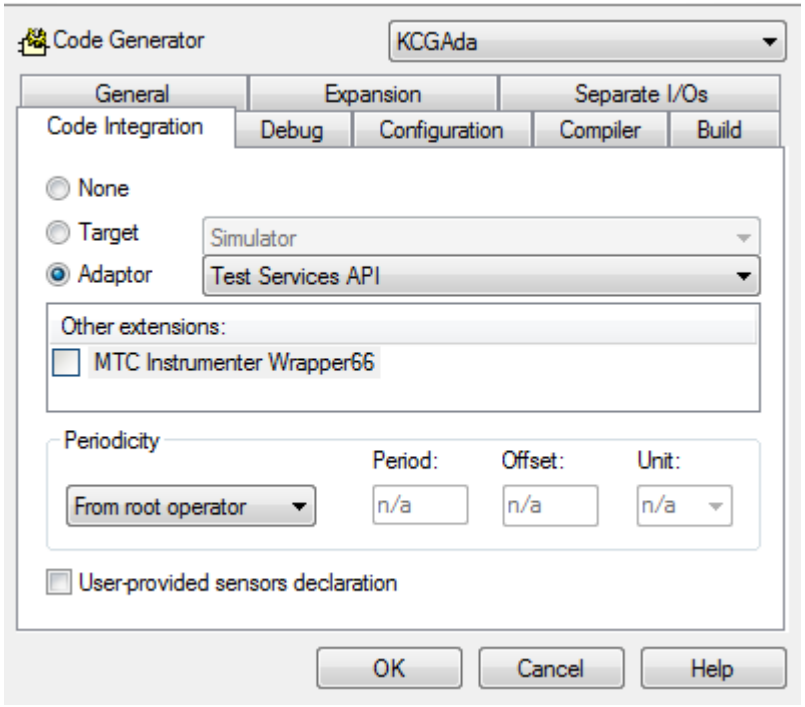
- No code extension is produced

Target:

- Code generation extends code for the selected target:
  - Simulator

Adaptor

- Code generation extends code to the selected adaptor:
  - Test Services API



Other extensions:

- MTC Instrumenter Wrapper66:
  - Generate KCG 6.6 MTC wrapper code

AGENDA

Simulation

Generated Code

Integration

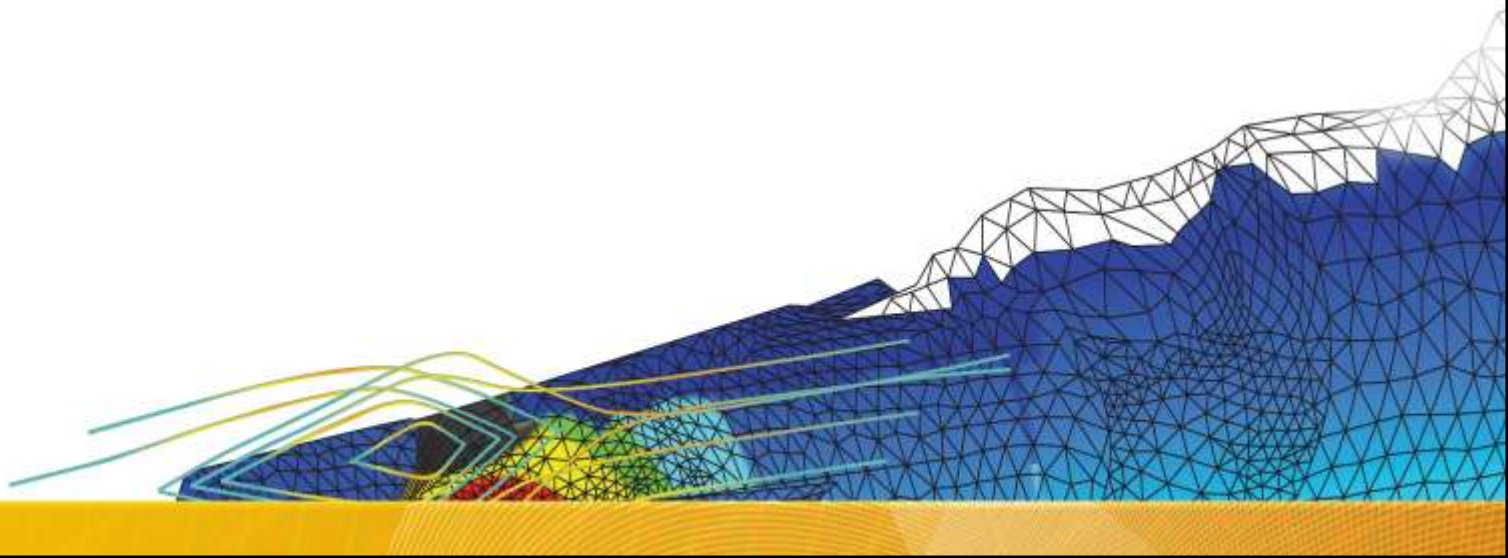
Traceability

Document Report

Model-Based V&V

SCADE Suite Basics

Traceability



Exercise

Brainstorming

Objective:

Define what requirements traceability is

Requirements:

In a group, share and discuss your opinion and then present it to the other groups

Preparation Time: 5 min

What is requirements traceability?

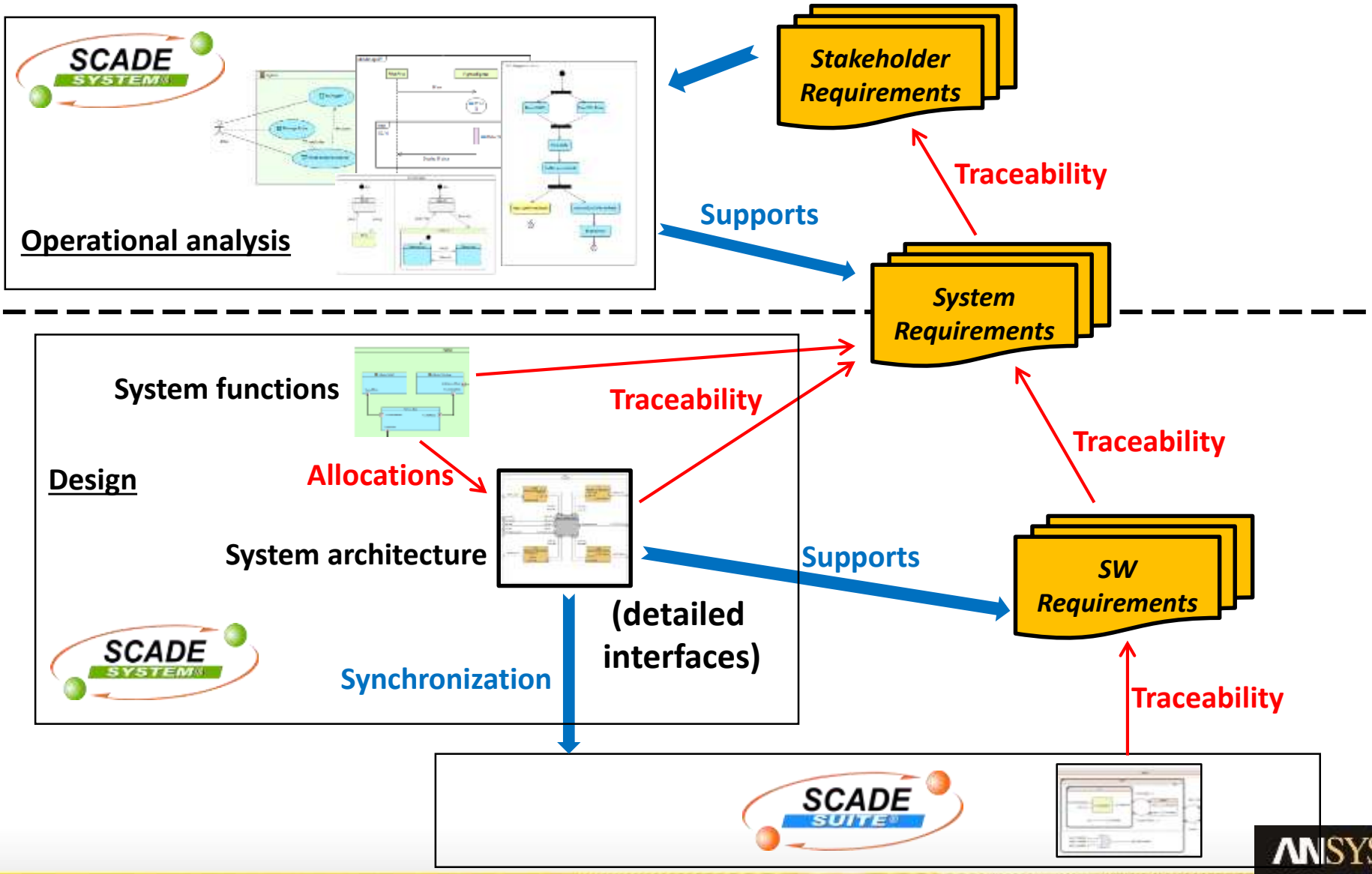
Requirements…

- Requirements are explicitly defined and identified at different levels of the system development

… Traceability

- Requirements are tracked across the entire design cycle, including the iterative phases (evolutions, derivatives, etc.)

What is requirements traceability?



What is requirements traceability?

Initial investment…

- Requirements standards definition
- Information added to files according to standards
- Requirements management tool

… with great benefits

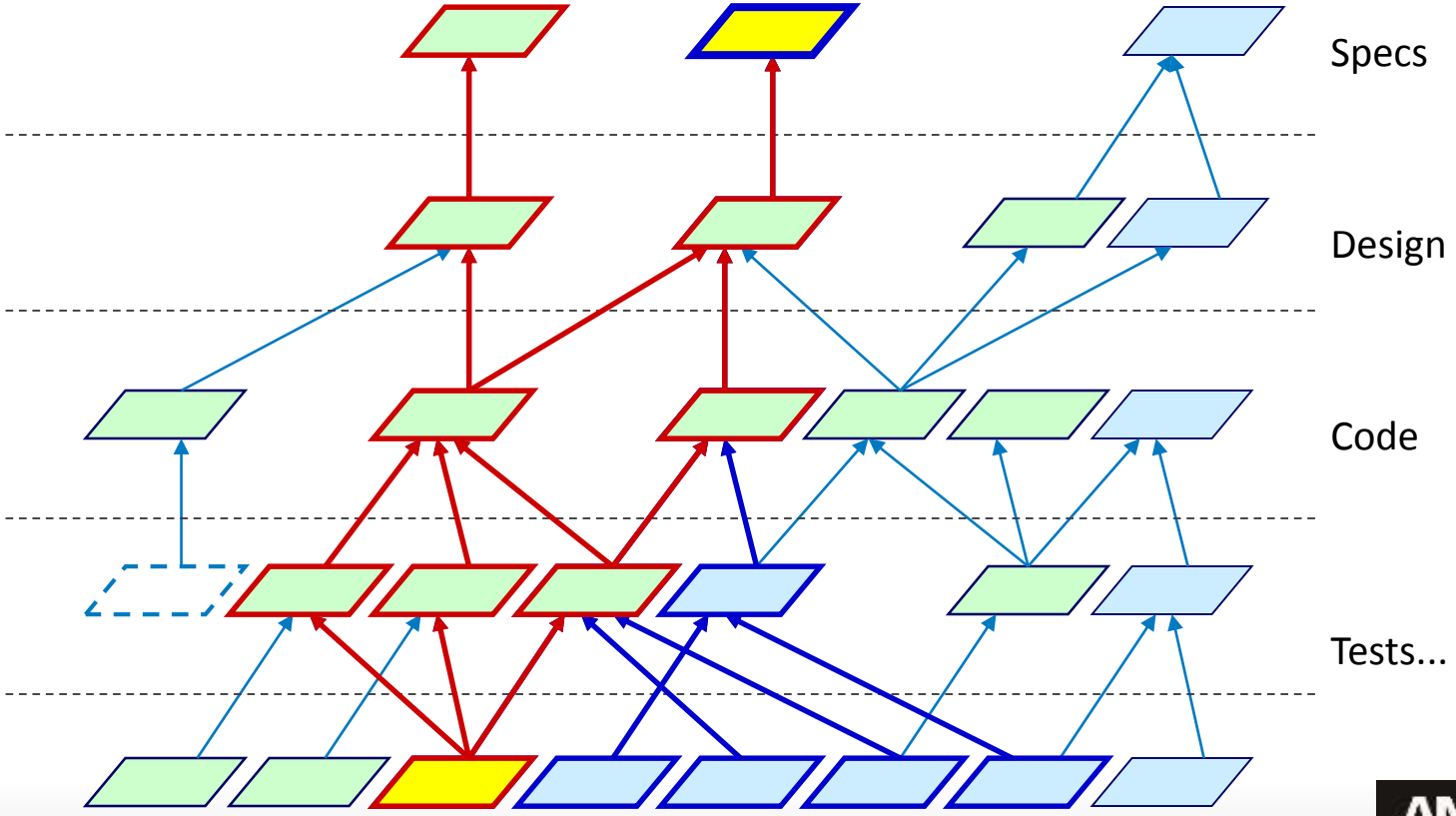
- Improves project members and stakeholders relationship
- Provides management visibility and support for decision making
- Enables fast maturity of specifications
- Saves time to evaluate impact of changes

What is requirements traceability?

« How do you implement this requirement…? »

« If I change something in my design, what's the impact…? »

« If I change something in my implementation, what about the regression…? »



What is Requirements Traceability?

Example

Requirements Traceability is mandatory in Aeronautics (DO-178C for Software, DO-254 for Hardware):

Table A-3	Objective
6	High-level requirements are traceable to system requirements

Table A-4	Objective
6	Low-level-requirements are traceable to high-level requirements

Table A-5	Objective
5	Source code is traceable to low-level requirements

Requirements Traceability is required as well by the EN 50128 standard for Rail Transportation software:

6.5.4.14 Traceability to requirements shall be an important consideration in the validation of a safety-related system and means shall be provided to allow this to be demonstrated throughout all phases of the lifecycle.

6.5.4.15 Within the context of this European Standard, and to a degree appropriate to the specified software safety integrity level, traceability shall particularly address

a) traceability of requirements to the design or other objects which fulfil them,

b) traceability of design objects to the implementation objects which instantiate them,

c) traceability of requirements and design objects to the tests (component, integration, overall test) and analyses that verify them.

Whatever is the solution used for requirements, the need for traceability with project workbenches and authoring tools is always present.

What is Requirements Traceability?

Example

For the automotive industry, ISO 26262 indicates:

Part 6, 7.4.2 During the development of the software architectural design the following shall be considered:

a) the verifiability of the software architectural design

=> This implies bi-directional traceability between the software architectural design and the software safety requirements

Application Lifecycle Management Gateway

SCADE LifeCycle Application Lifecycle Management Gateway (ALM) extends SCADE Suite with ALM tool connection capabilities:

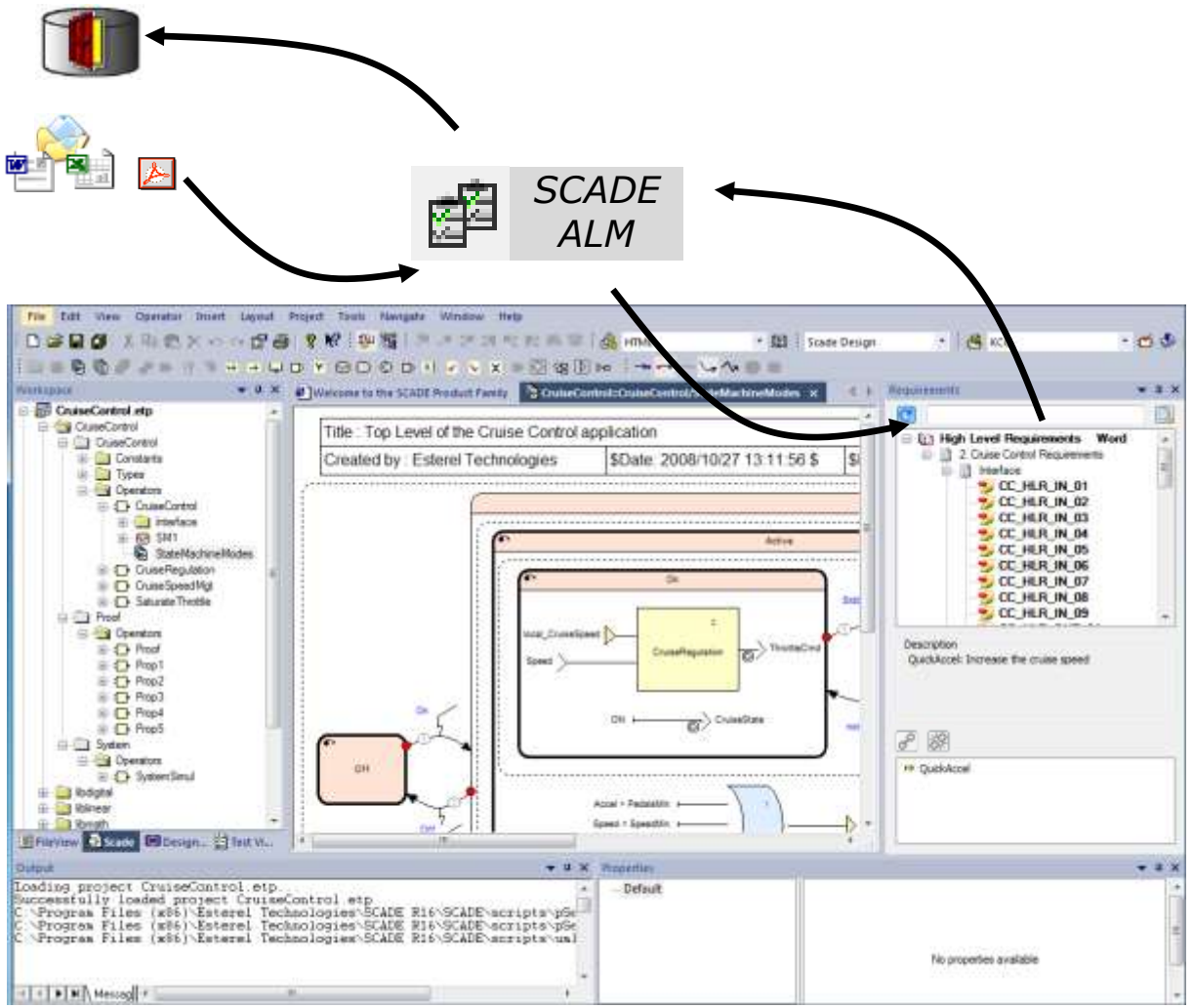
- Connects to ALM tools like DOORS or Reqtify
- Provides a complete traceability solution for the model design process in ALM tool environments

Traceability applied to a SCADE Model

Requirements automatically captured from DOORS or textual documents,…

Traceability is done while doing the SW development work.

Traceability analysis is always up to date.

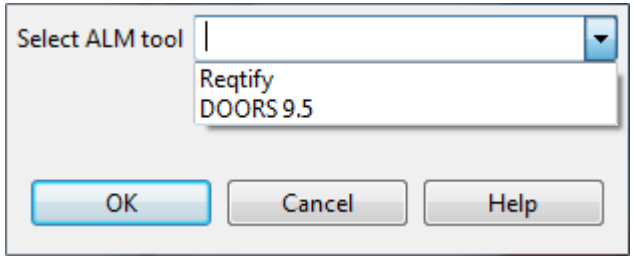


ALM Tool Selection

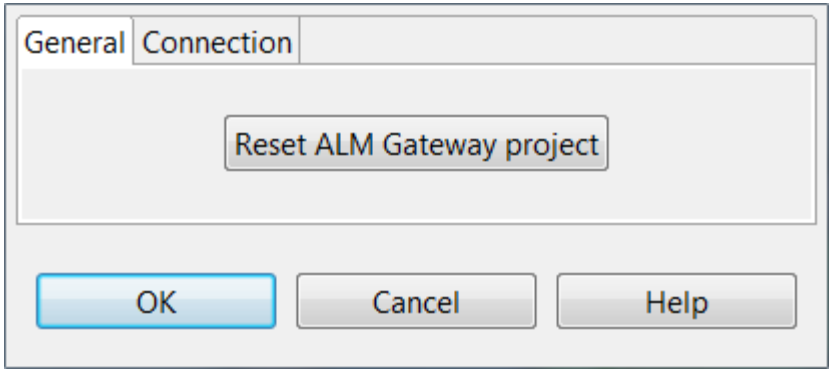
Launch the ALM tool:

- Select the SCADE menu *Project > ALM Gateway* or click on the ALM tool toolbar:

- Select ALM tool (if need be)



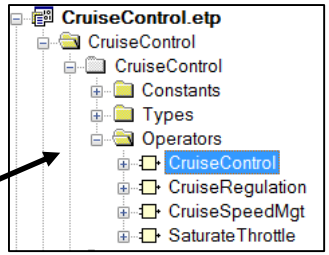
ALM Tool General Settings



Option	Description
Reset ALM Gateway Project	Remove the ALM Gateway project from the SCADE project to allow defining another ALM connection

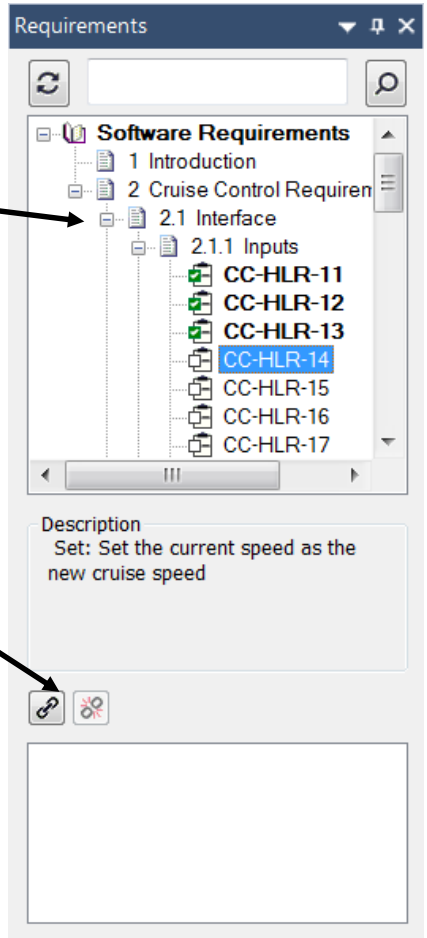
Tip To switch to another ALM tool, remove the ALM Gateway project from the SCADE project (the ALM Gateway project file is not deleted from the disk) and create a new ALM connection

Perform Requirements Traceability



Add the coverage of requirements:

- Select a Scade object in the Scade view
- From the Requirements Docking Window
  - Select a requirement in the available list
  - Click on Create Link

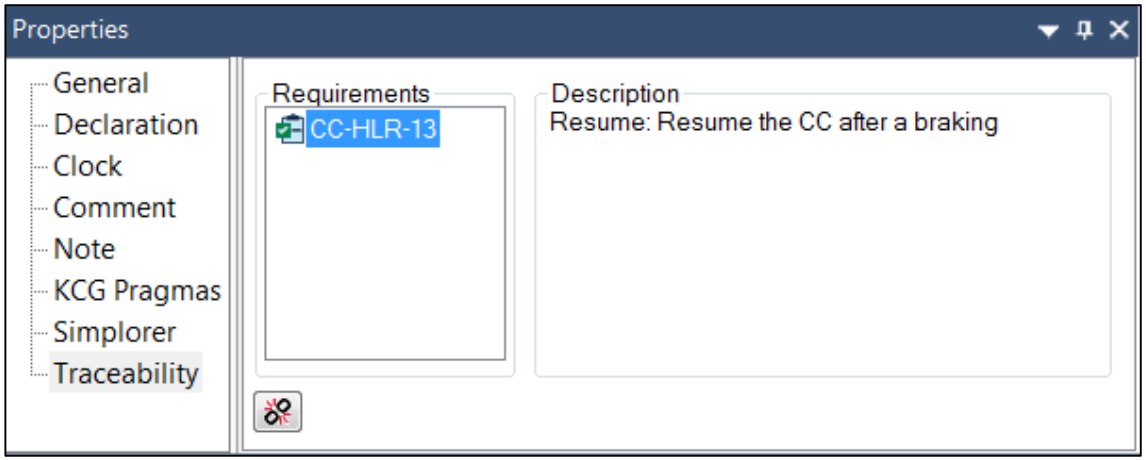


Remove the coverage of requirements:

- From the Requirements Docking Window
  - Select an element in the covered objects list
  - Click on Remove Link 

Create Requirements Traceability

Traceability links of the selected Scade object are displayed in the Traceability tab of the Properties window (empty if no requirements coverage)



From the Properties window

- Remove a covered requirement
  - Select it in the Requirements list
  - Click on Remove Link

AGENDA

Simulation

Generated Code

Integration

Traceability

Document Report

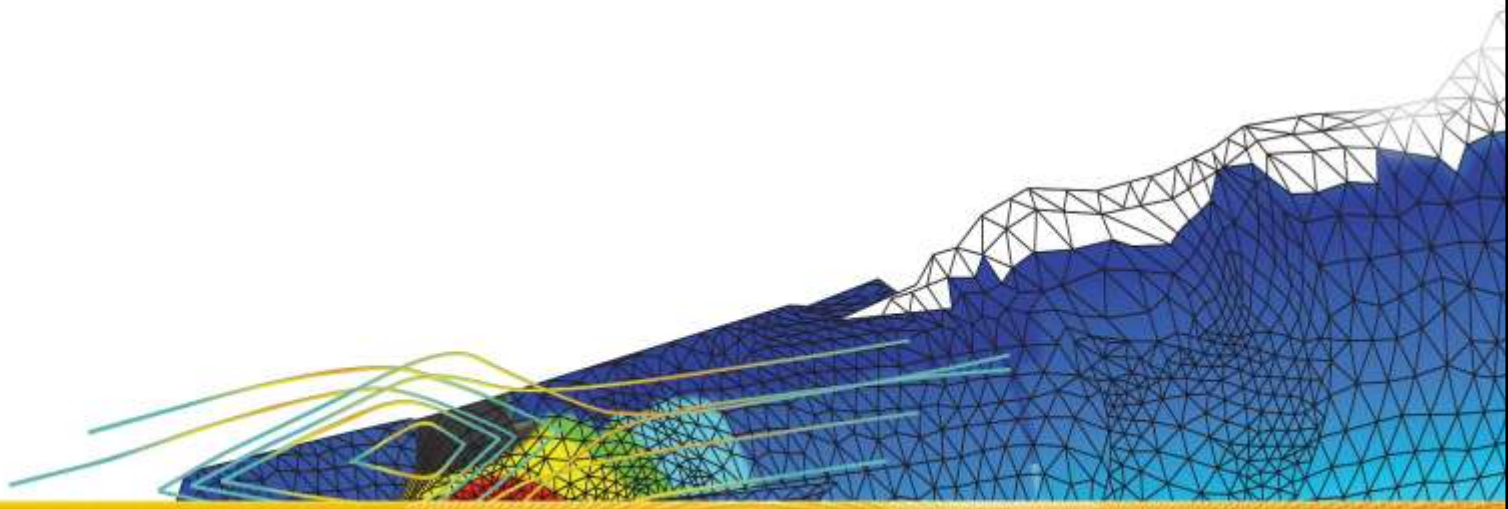
Model-Based V&V

Realize Your Product Promise®

ANSYS

SCADE Suite Basics

Automated Documentation Generation



SCADE LifeCycle Reporter

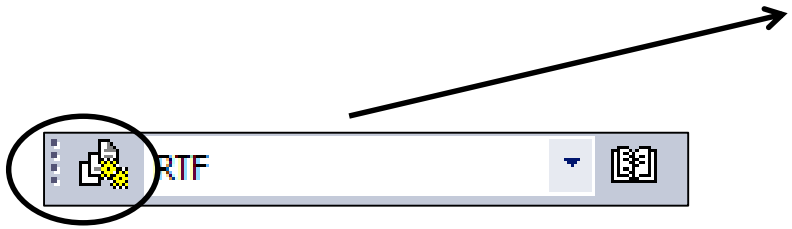
Automated documentation generation enables:

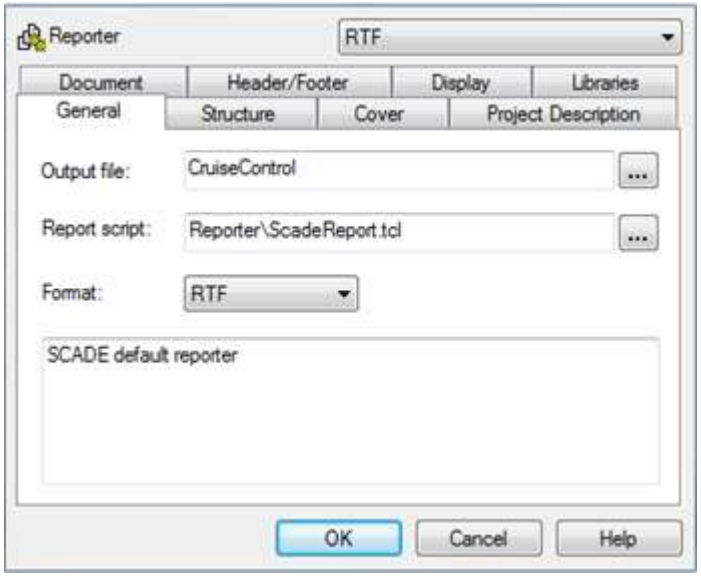
- Review of the SCADE Suite model
  - Verification that the SCADE design conforms to project standards
  - Verification that the SCADE design complies with requirements (in addition to simulation results)
- Communication to other teams
- Constitution of the dossier to be build up for certification

Reporter: General Settings

Generate project reports using predefined configurations (HTML or RTF) or user configurations

Use the toolbar to open the Settings dialog:





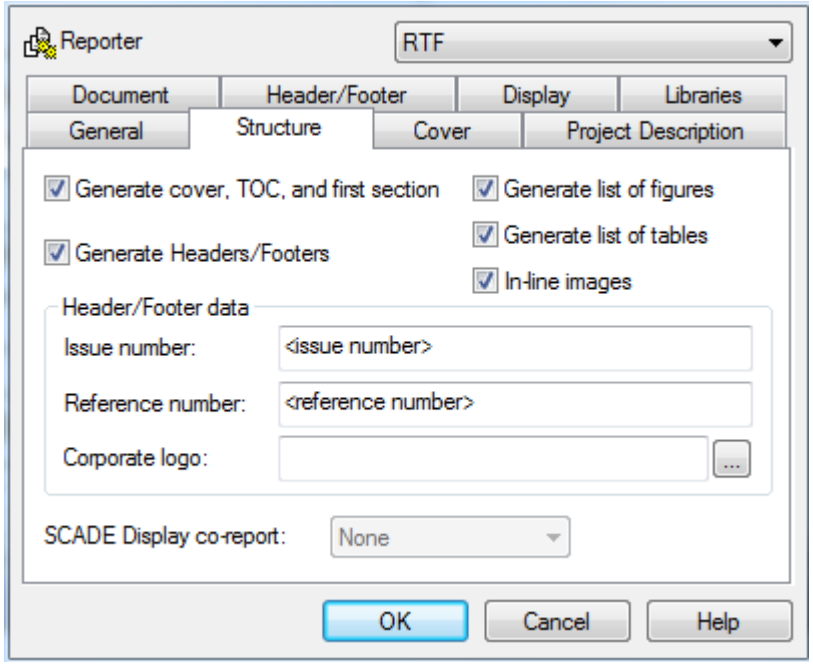
Content of the report

- Two formats : HTML or RTF
- Choose a TCL script to generate a content fully customizable
  - A default one is available: *ScadeReport.tcl*

Reporter: Structure Settings

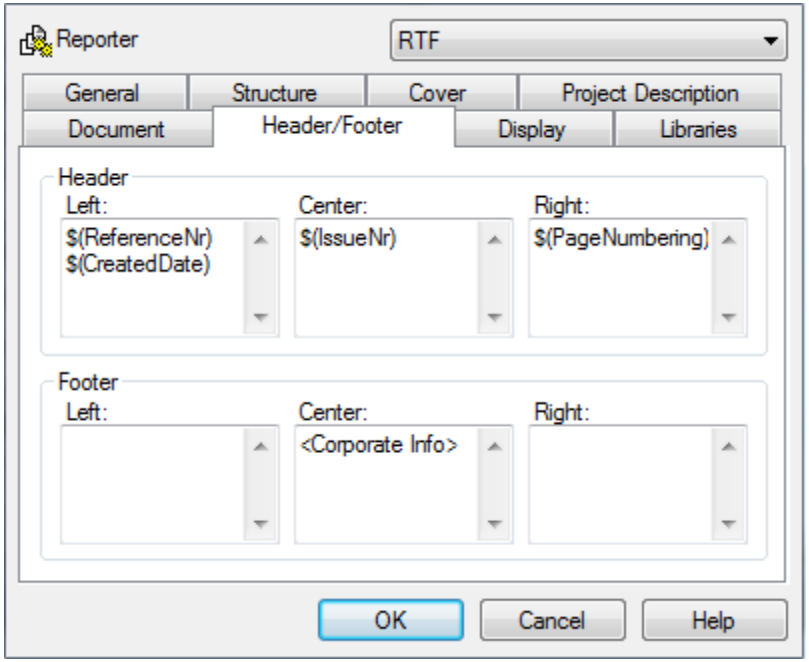
Customization of the structure:

- Call graph, table of content, etc.



Reporter: Structure Settings

Easy management of the headers/footers

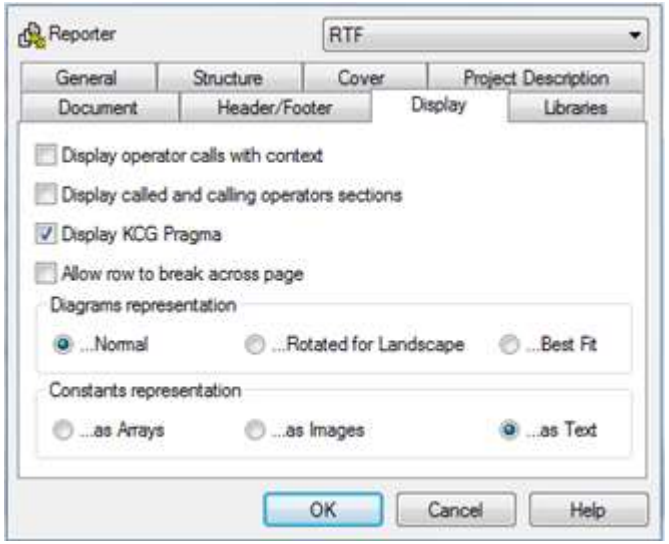


Display Options

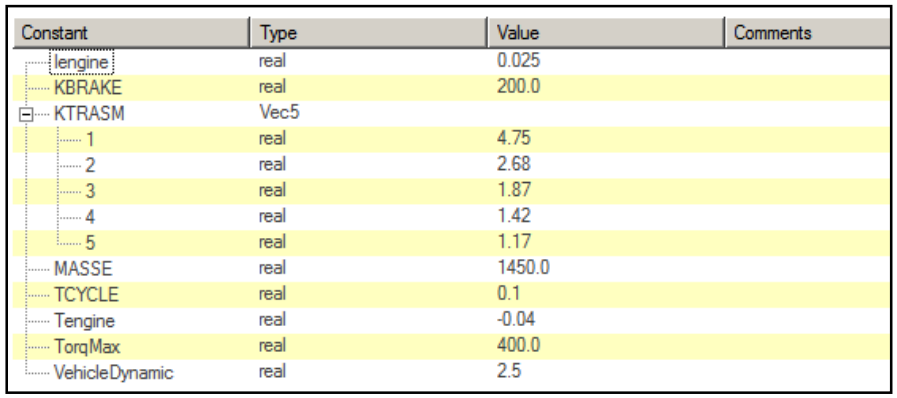
Constants:

- Three possible representations :

Name	Type	Value	Comments/Annotations
Iengine	real	0.025	
KBRAKE	real	200.0	
KTRASM	Vec5	1: 4.75; 2: 2.68; 3: 1.87; 4: 1.42; 5: 1.17	
MASSE	real	1450.0	
TCYCLE	real	0.1	
Tengine	real	-0.04	
TorqMax	real	400.0	
VehicleDynamic	real	2.5	



As Arrays



Constant	Type	Value	Comments
Iengine	real	0.025	
KBRAKE	real	200.0	
KTRASM	Vec5		
1	real	4.75	
2	real	2.68	
3	real	1.87	
4	real	1.42	
5	real	1.17	
MASSE	real	1450.0	
TCYCLE	real	0.1	
Tengine	real	-0.04	
TorqMax	real	400.0	
VehicleDynamic	real	2.5	

As Images

Name	Type	Value	Comments/Annotations
Iengine	real	0.025	
KBRAKE	real	200.0	
KTRASM	Vec5	[4.75 , 2.68 , 1.87 , 1.42 , 1.17]	
MASSE	real	1450.0	
TCYCLE	real	0.1	
Tengine	real	-0.04	
TorqMax	real	400.0	
VehicleDynamic	real	2.5	

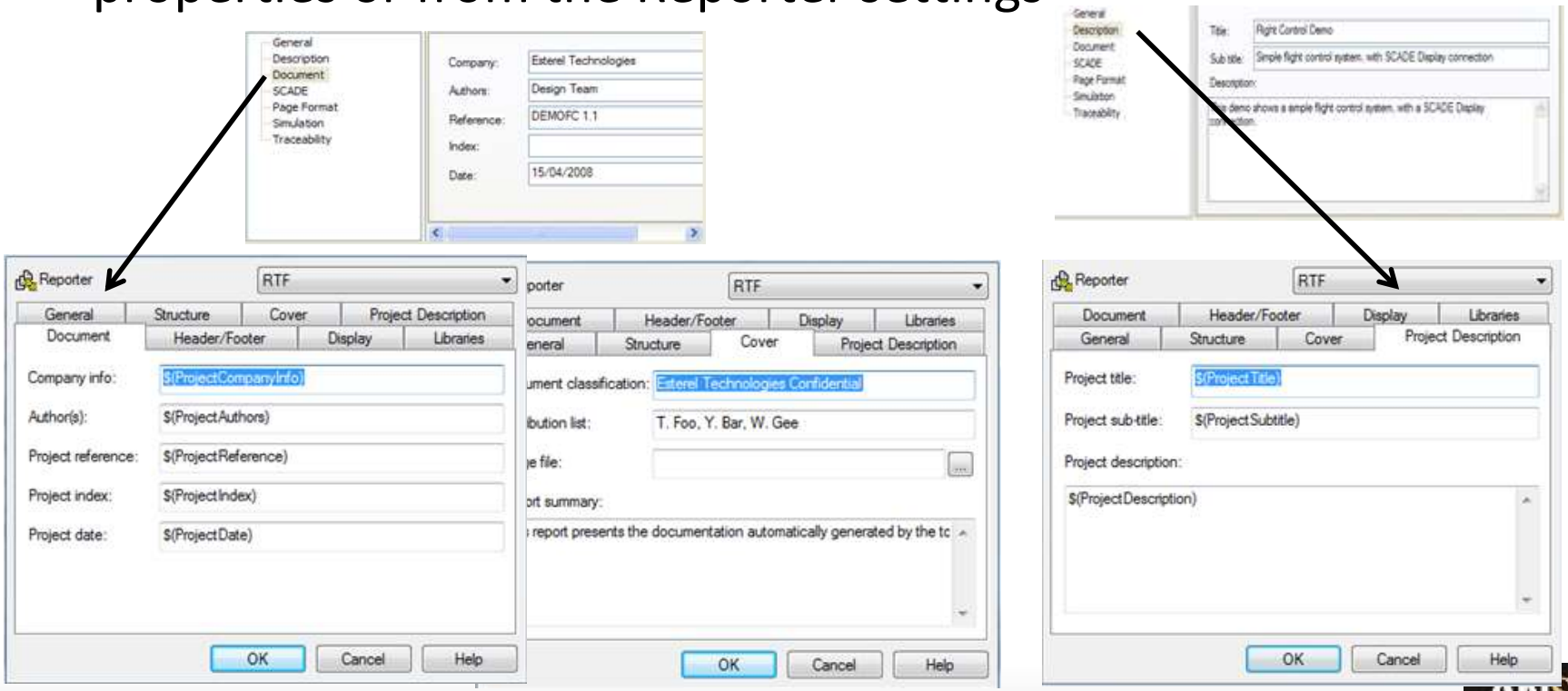
As Text

Custom Attributes

Manage document attributes:

- Classification, distribution list, etc.

Customize cover, structure, description from project properties or from the Reporter settings

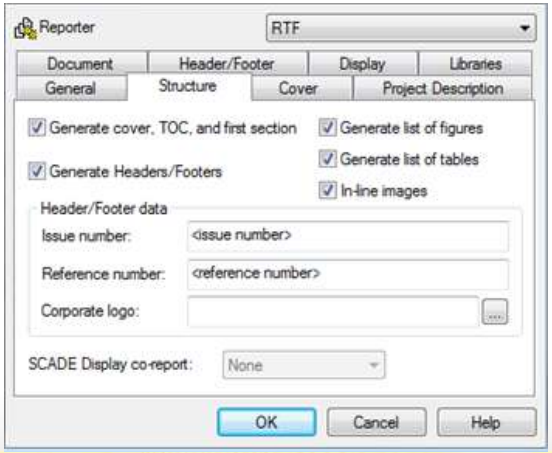


SCADE Display Integration

When there is at least one SCADE Display specification in the project or one operator connected to a specification

SCADE display co-report:

- *Complete*: generate the "SCADE Display Integration" section in the "Software Architecture" chapter, the connection table for each connected operator and SCADE Display reports
- *Connection Tables*: same as "Complete", without the generation of the SCADE Display reports
- *None*: no report of SCADE Display elements



Report generation

After having selected a configuration, to produce a report, click on:

A HTLM or RTF document is displayed, according to the selected format

- Contains the current SCADE design loaded in SCADE Suite

SCADE Architect Model

In the same way, report the system design (in SCADE Architect) that is synchronized with the SCADE Suite model such as the software architecture model

Report Document

Example

<document classification>

Pilot Example

Auto Pilot

Summary:
<summary>

Company: Esterel Technologies
Authors: S.SABATHIER
Reference: 1.4
Index: 2
Date: 19/10/2007

Distribution List: <distribution list>

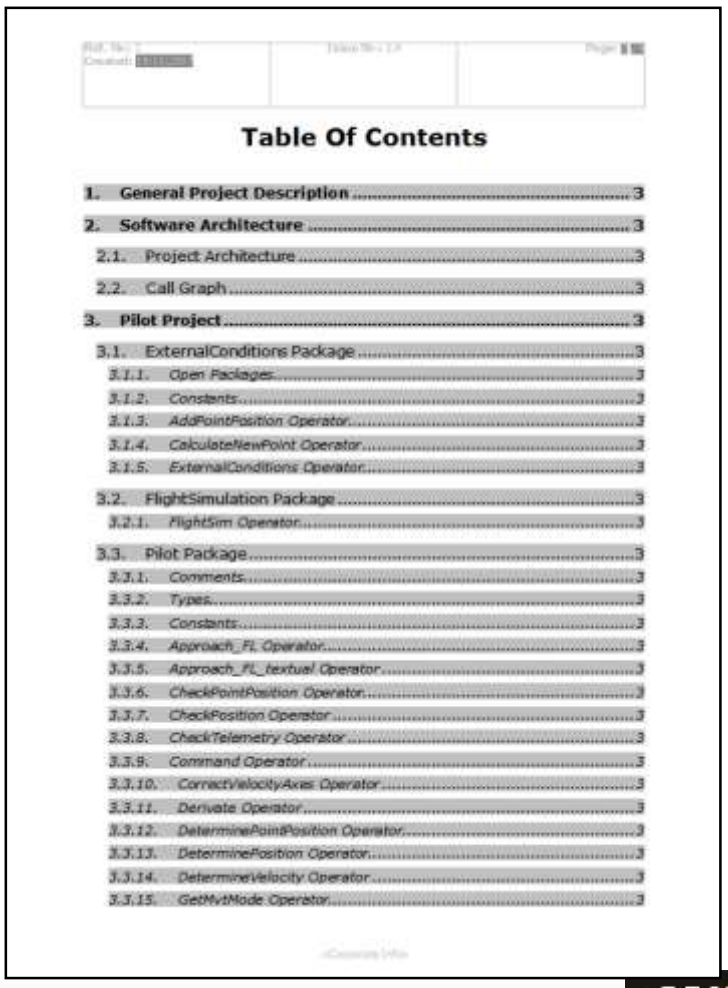


Table Of Contents

Report Document

2. Software Architecture

2.1. Project Architecture

This section displays the package hierarchy of projects.

Project Pilot
- ExternalConditions
- FlightSimulation
- Pilot

2.2. Call Graph

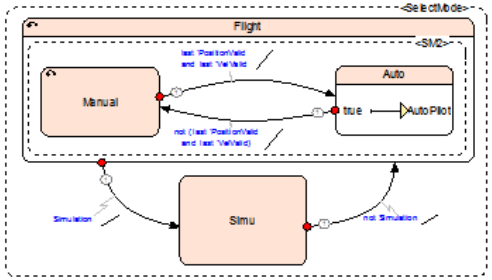
This Call Graph displays the dependency tree of model operators.

1. FlightSimulation::FlightSim
   - 1.1. ExternalConditions::ExternalConditions (L59=)
     - 1.1.1. ExternalConditions::CalculateNewPoint (L69=)
       - 1.1.1.1. ExternalConditions::AddPointPosition (L9=)
   - 1.2. Pilot::Pilot (L56=)
     - 1.2.1. Pilot::CheckPosition (L37=)
       - 1.2.1.1. Pilot::CheckTelemetry (L13=)
         - 1.2.1.1.1. Pilot::CheckPointPosition (L25=)
         - 1.2.1.1.2. pwlinear::ClockCounter (L31=)
       - 1.2.1.2. Pilot::DeterminePosition (L47=)
         - 1.2.1.2.1. Pilot::DeterminePointPosition (L3=)
       - 1.2.1.3. Pilot::DetermineVelocity (L19=)
         - 1.2.1.3.1. Pilot::Derivate (L52=)
         - 1.2.1.3.2. Pilot::Module (L55=)
           - 1.2.1.3.2.1. mathext::SqrtR (L10=)
           - 1.2.1.3.2.2. Pilot::SquareSum (L12=)
     - 1.2.2. Pilot::Command (L24=)
       - 1.2.2.1. Pilot::MvtCalculus (TheoricVel=)
         - 1.2.2.1.1. Pilot::Approach_FL_textual (L48=)
         - 1.2.2.1.2. Pilot::Rising_FL_imported (L46=)
         - 1.2.2.1.3. Pilot::Waiting_FL (L45=)
       - 1.2.2.2. Pilot::VelocityRegulation (L21=)
         - 1.2.2.2.1. Pilot::CorrectVelocityAxes (L144=)
     - 1.2.3. Pilot::GetMvtMode (L30=)
2. Pilot::Approach_FL
3. Pilot::Rising_FL

3.2.1.1. Node Hierarchy

state-machine : SelectMode
- state : Simu
- state : Flight
  - state-machine : SM2
    - state : Manual
    - state : Auto

3.2.1.2. Graphical Views



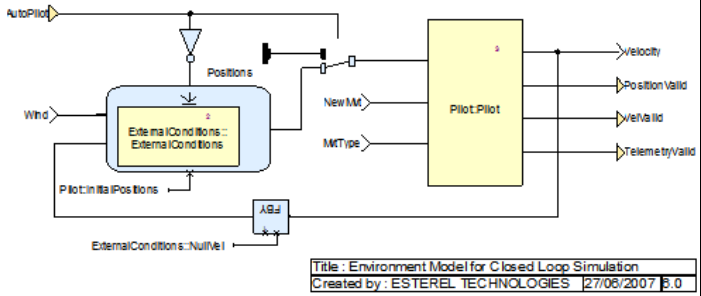


Figure 4: FlightSim

Table 13: Transitions of FlightSim

Source/Target	#	Conditions/Actions	Comments/Annotations
Source: SelectMode:Flight:SM2:Manual **Target:** SelectMode:Flight:SM2:Auto	1	**Condition:** last 'PositionValid and last 'VelValid	

Qualified Verification Tool

SCADE LifeCycle Reporter is qualified as a verification tool, to ensure that , in any cases, what is printed represents the model:

- Certification Kit DO-178C Level A (Criteria 3 / TQL-5)
- Qualified for SCADE Suite and SCADE Display
- Qualified report generation only in batch mode
- Qualified tool for the RTF format (forced option) and a defined environment:
  - Use the required “Reporter\ScadeQualifiedReport.tcl” script
  - A dedicated reporter configuration named “Qualified Reporter” is available

Qualified Verification Tool

In the scope of the qualification, several “standard” reporter properties values are forced, as follows:

Reporter property	Forced value	Reporter GUI tab	Reporter GUI field name
ImagesInLineWithText	true (default)	Structure	In-line images
ReportHeaderAndFooter	true (default)	Structure	Generate Headers/Footers
ReportStructure	true (default)	Structure	Generate cover, TOC, and first section
AllowRowToBreak	true	Display	Allow row to break across page
cstDisplayType	Flat	Display	Constants representation: …as Text
DisplayCalledAndCalling	true	Display	Display called and calling operators sections
DisplayKCGPragma	true	Display	Display KCG Pragma

Batch Generation

```
SCADE –report <project> -configuration <configuration>
```

Notes:

- <project> = *“project name”.etp*
- To call the qualified reporter, the “reporter script” property must be set to “*Reporter\ScadeQualifiedReport.tcl*” in the configuration

Advanced TCL Customization

SCADE LifeCycle Reporter can be customized with TCL scripts:

- Use the Reporter-related TCL commands to set the content and display of model documentation and reports (refer to details in the User Manual document)

The customization script files must be registered in the SCADE Suite environment for a proper evaluation

Use this tool with the user customized TCL procedures (or options outside the qualification context)

Additional verification activities are required

Exercise 3: Cruise Control

Time: 10 min

Generate report documents with different options and observe the generated files:

First format : HTML

Constant representation: As Arrays

Generate a table of content

Second format: RTF

Constant representation: As Images

Do not generate a table of content

Exercise

AGENDA

Simulation

Generated Code

Integration

Traceability

Document Report

Model-Based V&V

SCADE Combined Testing Process

What is SCADE Combined Testing Process?

SCADE Combined Testing Process is a SCADE Model-Based approach for combining efficient and rigorous testing activities

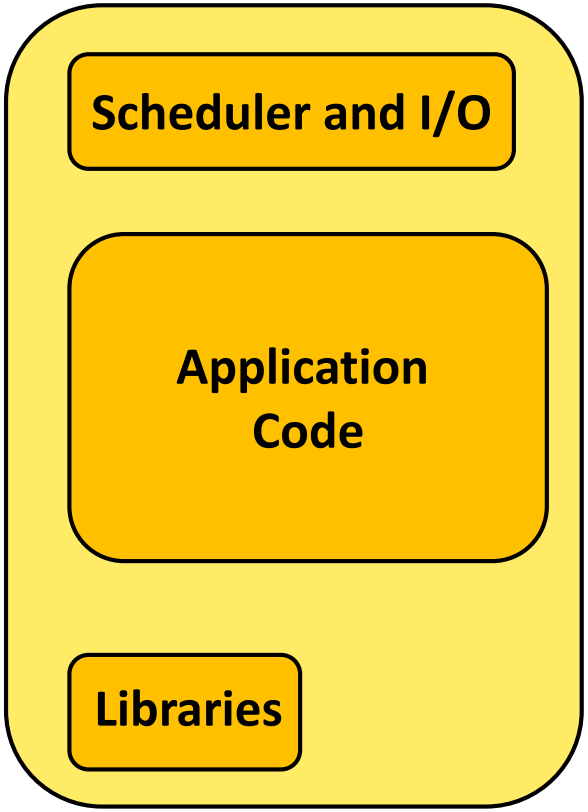
Objectives are the following:

- Optimize the Testing effort
- Maximize the benefit from KCG qualification/certification

Combination is the following:

- Software Requirements-Based Testing
- Low-level Testing on a sample
- Testing on host and on target

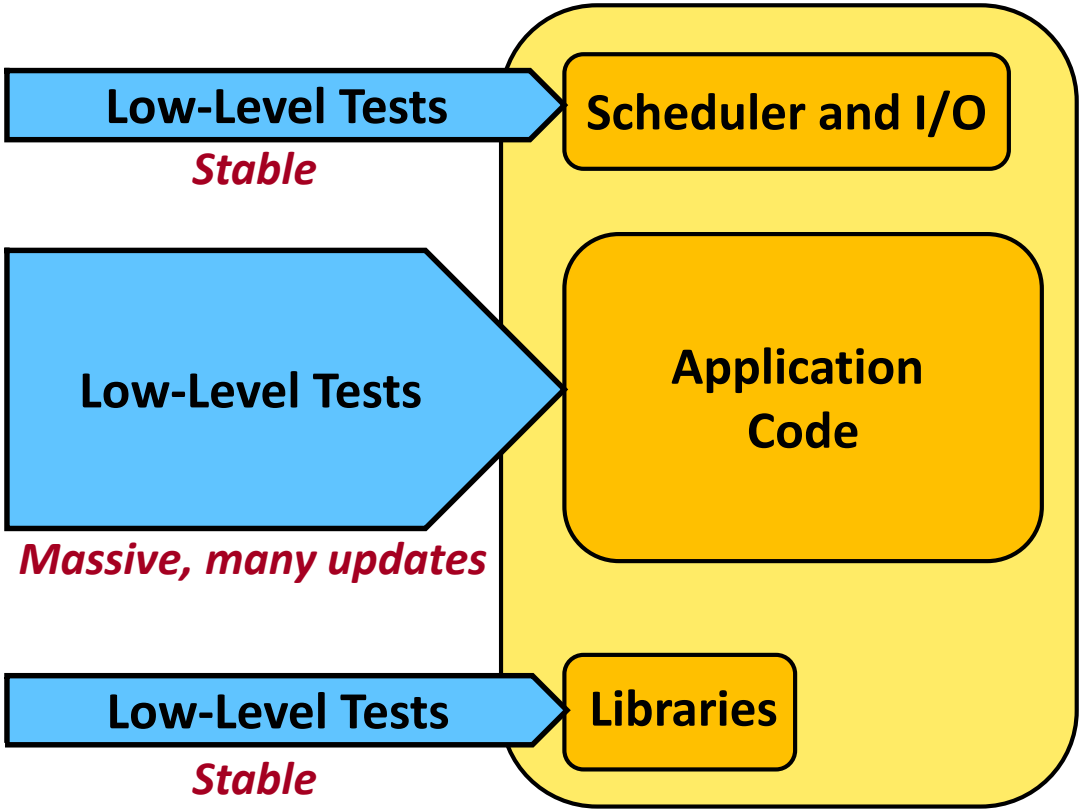
“Traditional” Testing Process



An Embedded Application is typically made of:

- A scheduler or an Operating System + Drivers for Inputs and Outputs
- The code of the Application
- Libraries of some elementary operations (C, and Ada code) that are repeatedly used in the application

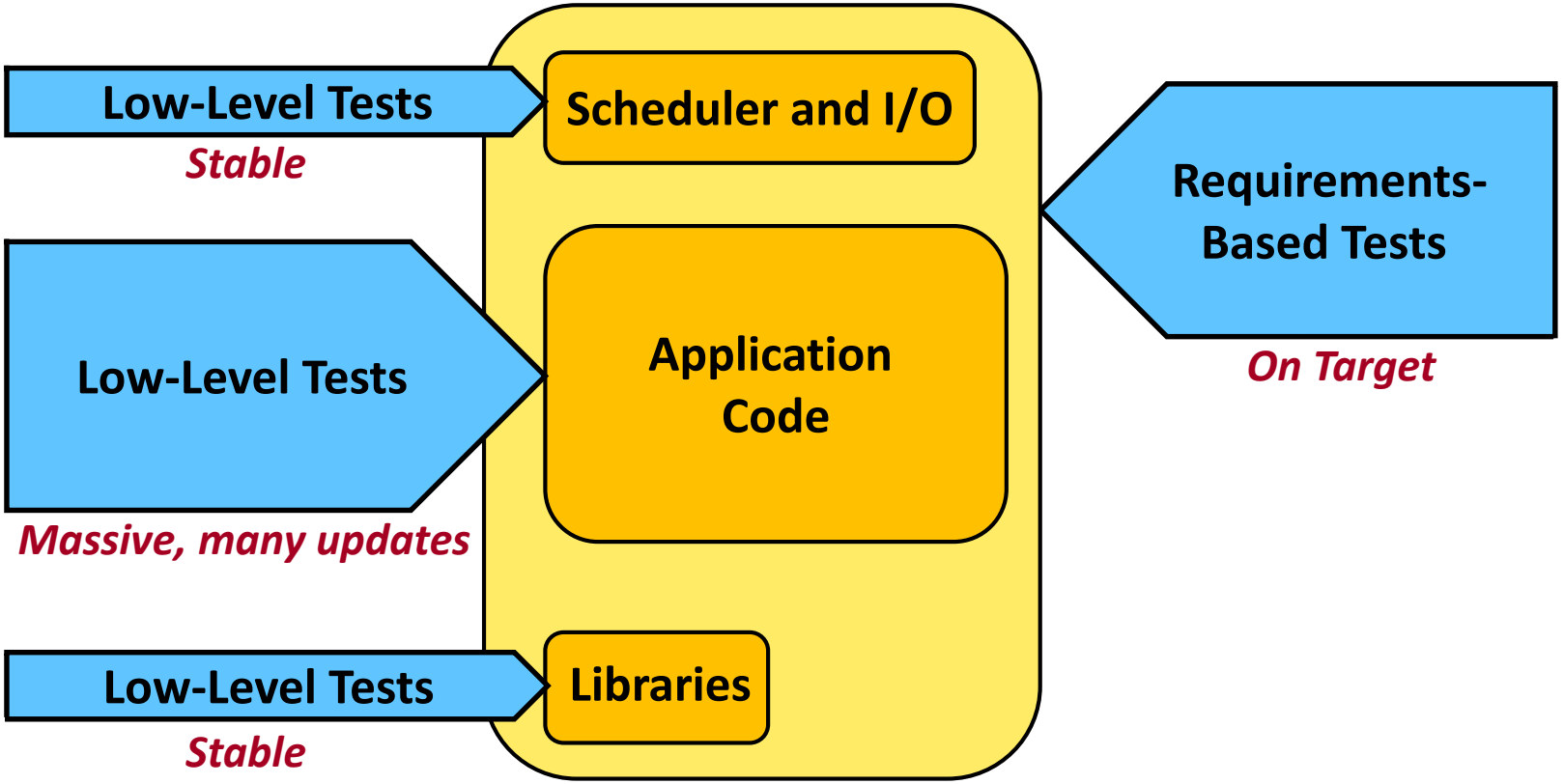
“Traditional” Testing Process



Low-Level Testing (a.k.a. Unit Testing) is applied to the 3 components of the application

Low-level Testing of the Application Code is time consuming, requires updates each time there is a code update (which is a standard situation), and the stopping criterion is not easy to establish

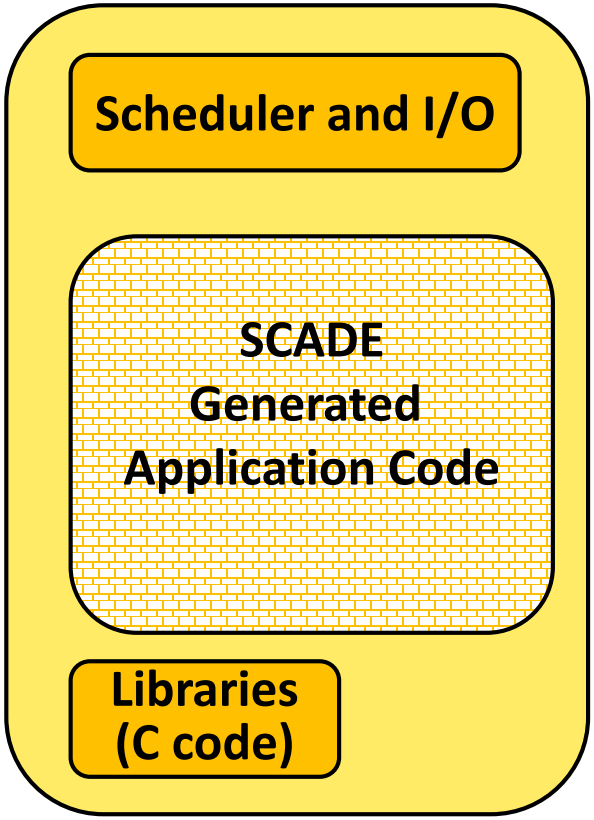
“Traditional” Testing Process



The Requirements-Based Tests are coded in C, Ada and/or in a specific Target Test Tool format, and run on target; thus, they are debugged on target

Target Test Tool Structural Coverage measurement can be the stopping criterion, generally mixed with low-level test coverage scores, sometimes not easy to justify

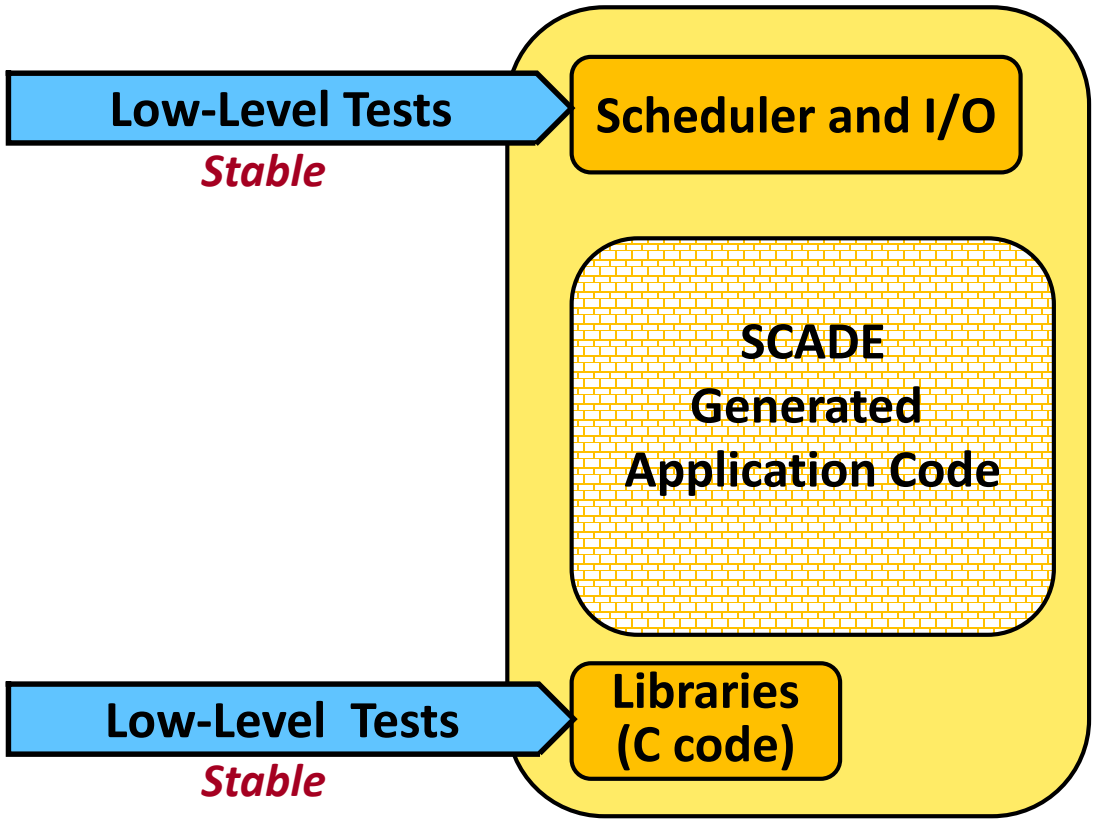
SCADE Combined Testing Process (CTP)



An Embedded Application designed with SCADE Suite is typically made of:

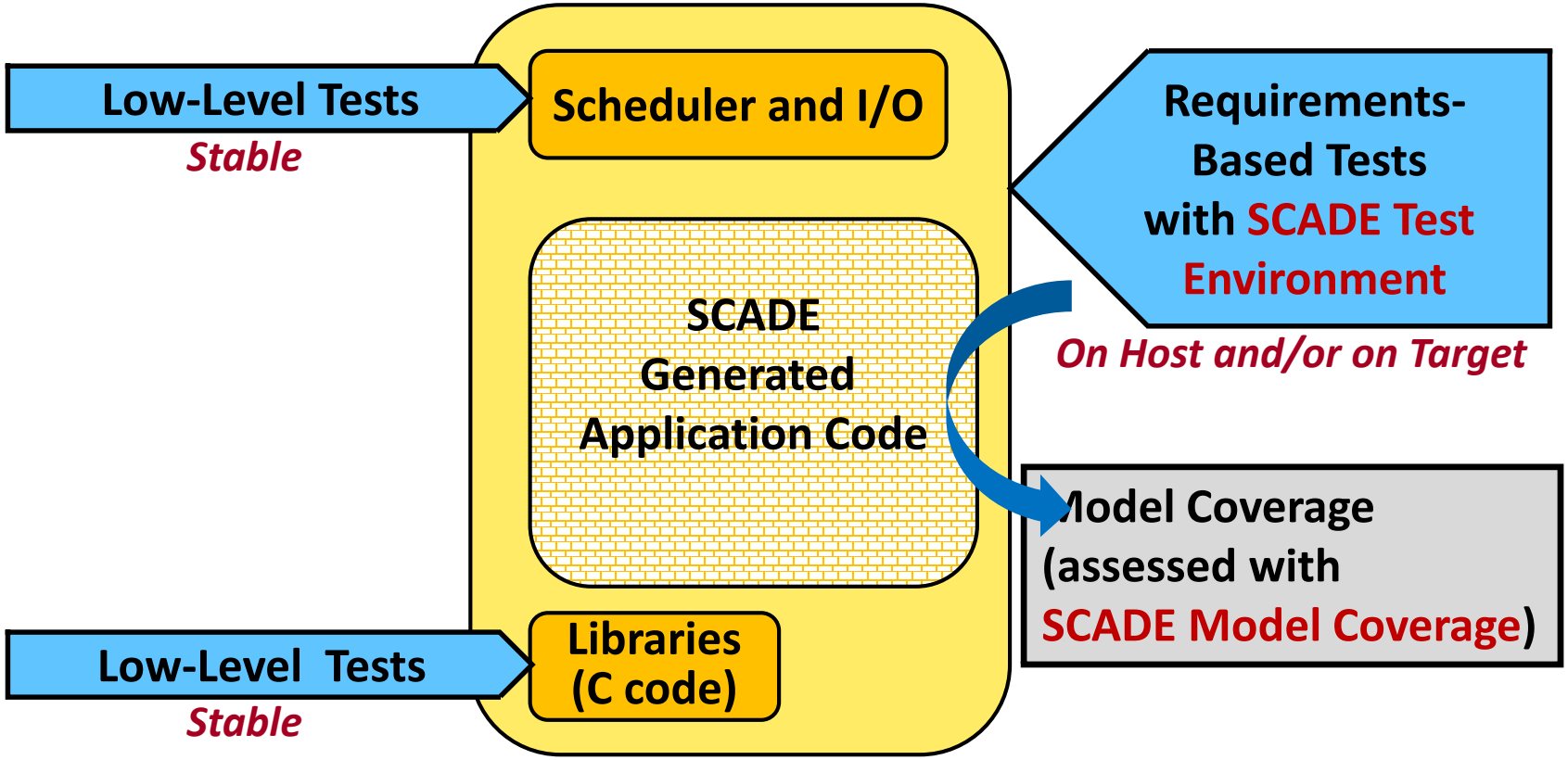
- A scheduler or an Operating System + Drivers for Inputs and Outputs
- The code of the Application, generated by SCADE Suite KCG
- Libraries of some elementary imported operators that are repeatedly called in the SCADE model (SCADE libraries are part of the generated code)

SCADE Combined Testing Process (CTP)



No change in Low-level Testing against the Scheduler and I/O (or OS) and against the Libraries of C Code

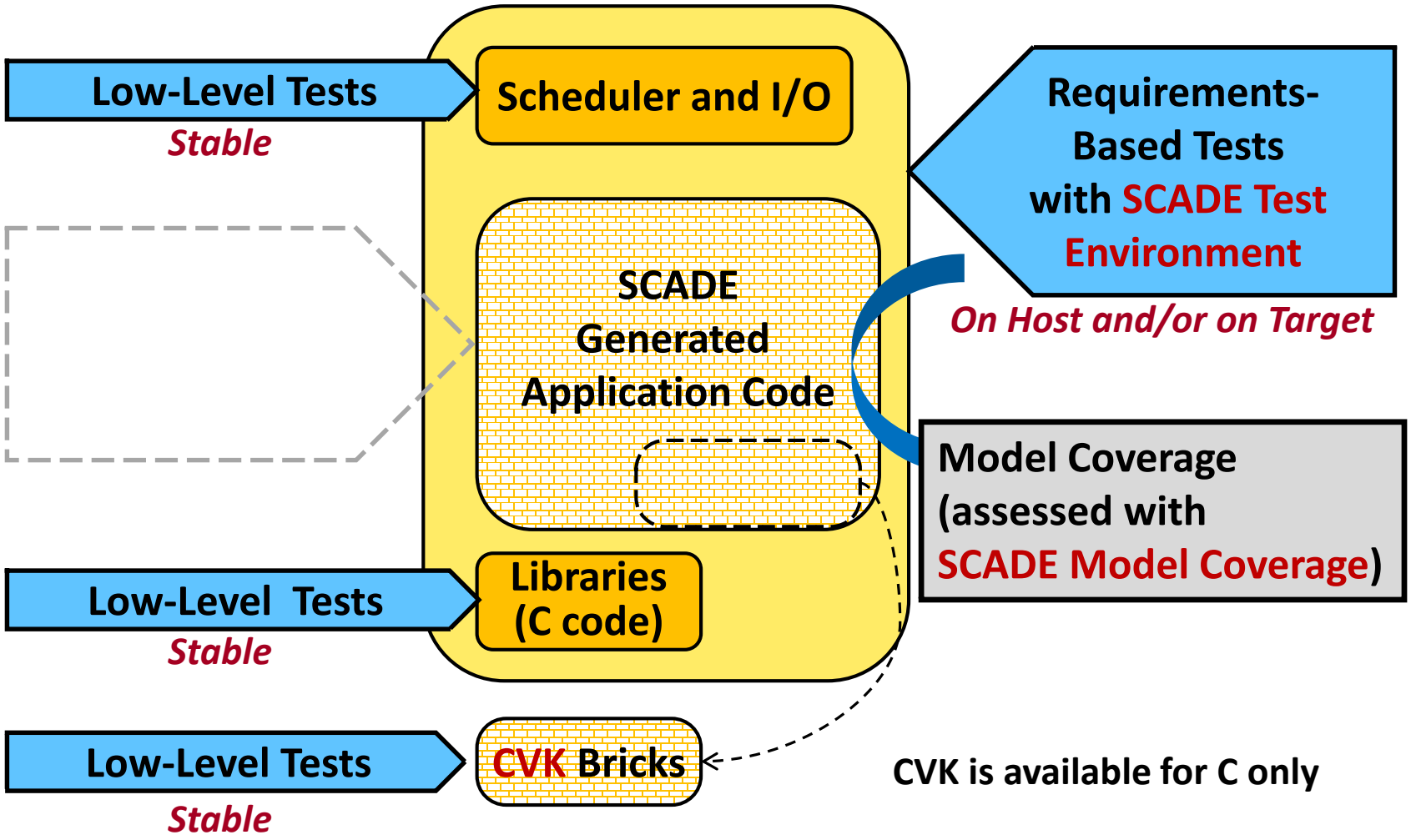
SCADE Combined Testing Process (CTP)



The Requirements-Based Tests are created using SCADE Test Environment and run on Host (SCADE Simulation) and/or on Target (Target Testing) according to the Test Strategy that has been established

SCADE Model Coverage is the unique means to monitor and stop the Requirements-Based Testing activities

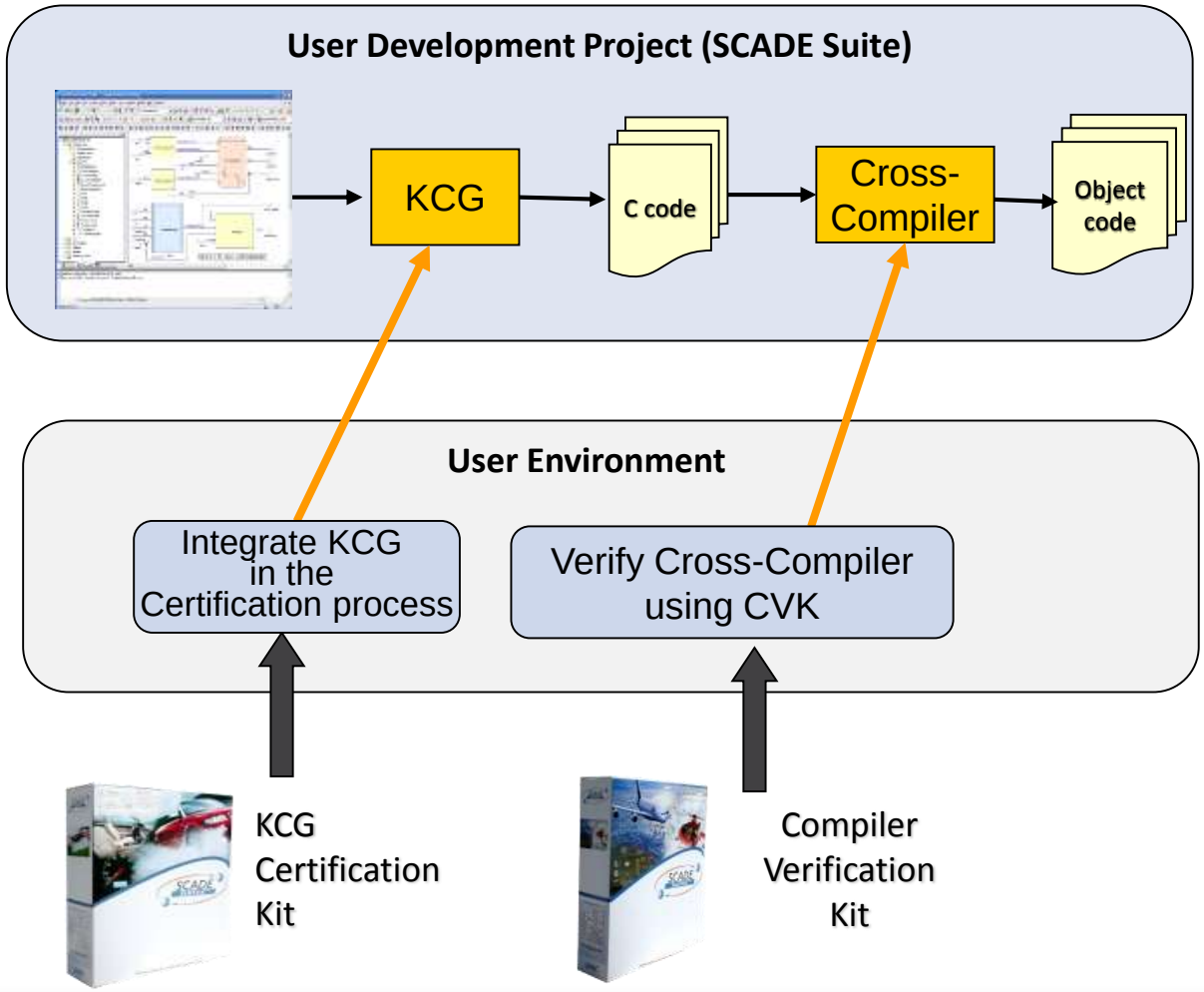
SCADE Combined Testing Process (CTP – C Code)



SCADE Suite CVK is a Low-Level Test Suite used to perform unit testing on target of all the C constructs that may be used in the SCADE Generated Application Code

SCADE Compiler Verification Kit (Overview)

Combined Testing Process & CVK



Low-level Testing on Target is cancelled only if:

- The Code Generator is qualified/certified
  → *KCG is qualified/certified*
- A formal sample of KCG generated code is cross-compiled and tested on target
  → *CVK is a suite of tests dedicated to compiler verification*

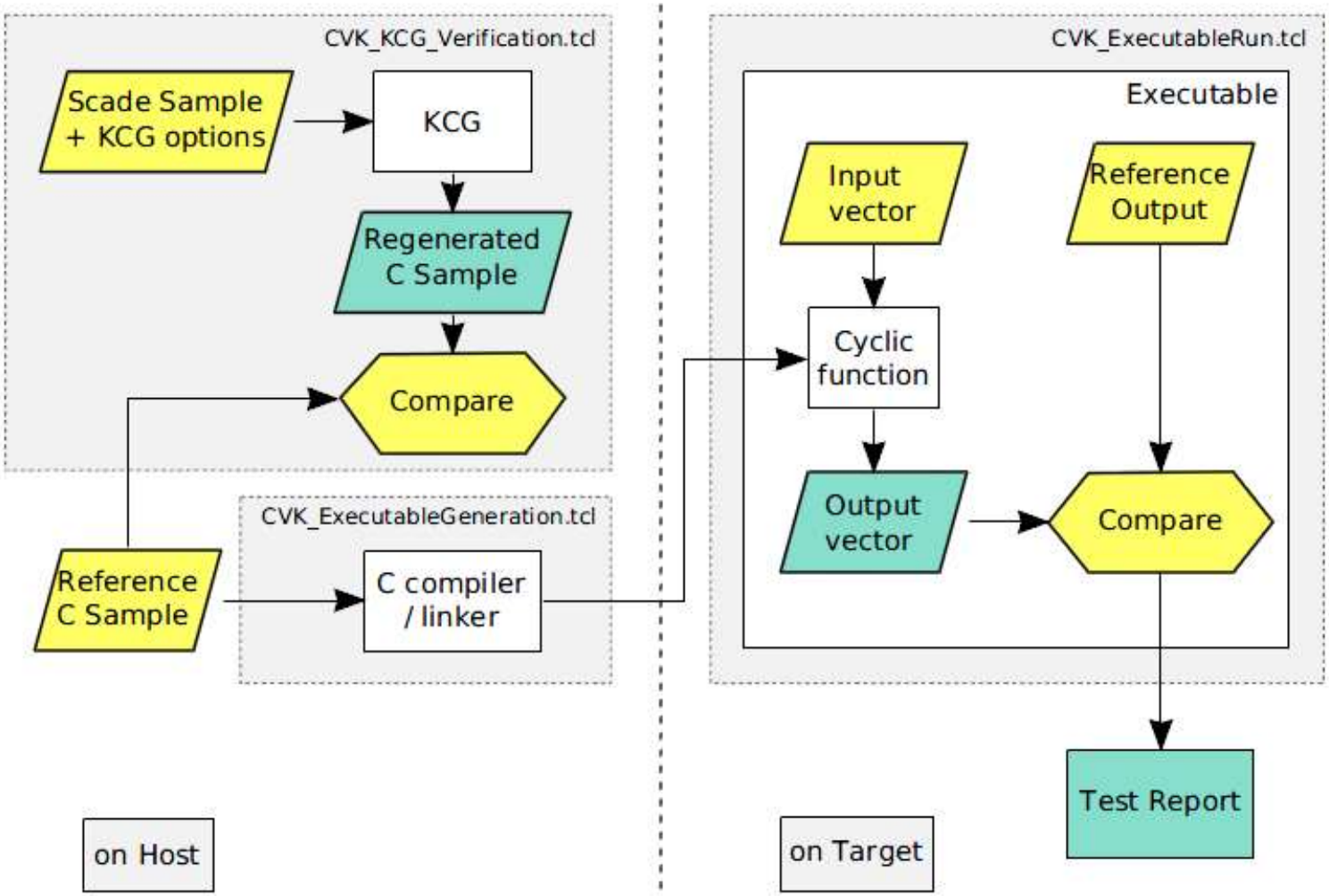
Compiler Verification Kit (CVK)

CVK is a test suite whose purpose is to verify that a compiler correctly compiles code generated by SCADE Suite KCG

CVK has been developed for the verification of a development environment containing KCG

CVK does not validate the C compiler or processor
CVK is not an executable software and not a tool

The CVK Workflow



Annexes

Simulator

AGENDA

Integrated Development Environment

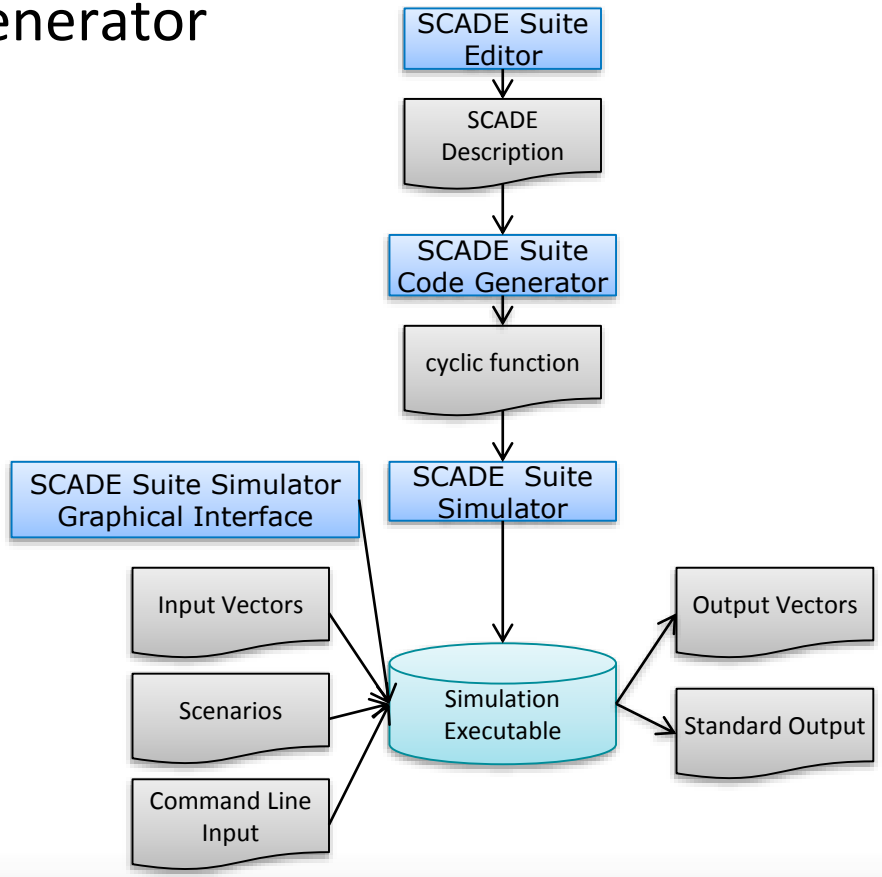
Batch Mode

How do Software engineers debug a SCADE Suite design to resolve a defect?

How do Software engineers simulate a SCADE Suite design to check its behavior?

Simulator Main Features

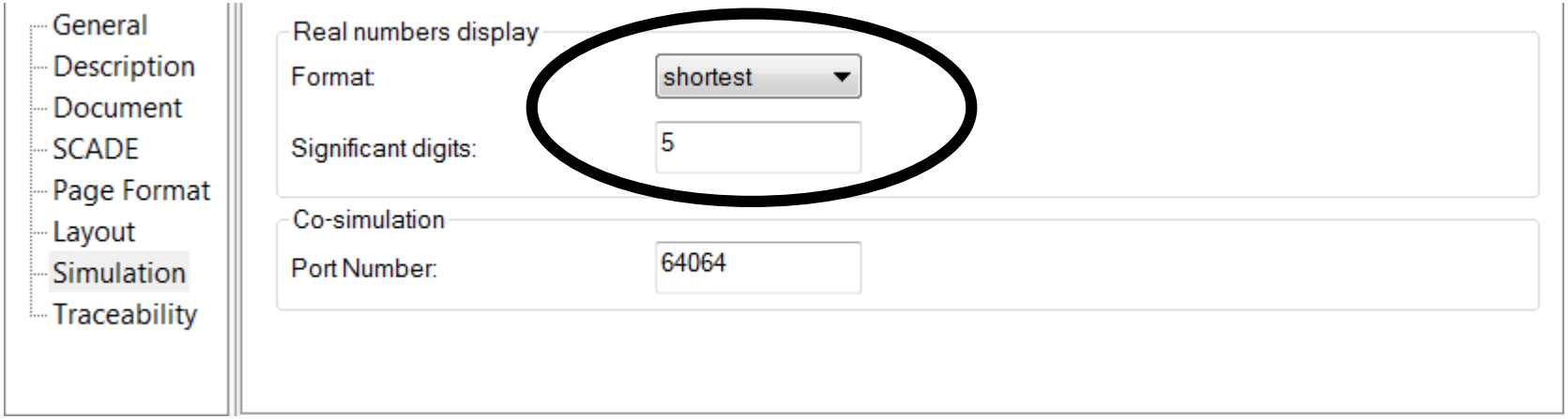
SCADE Suite Simulator enables the simulation and debugging of SCADE Suite models by executing code which is generated by Code Generator



Simulation Precision: Real Values

Possible to modify the format to display and store real values:

- Right-click on the project file in Scade View and select Properties Simulation tab
- From the **Format** field: select scientific, decimal, or shortest to set the real numbers display
- From the **Significant digits** field: set the number of significant digits
  - An integer between 1 and 17 (by default this number is 5)



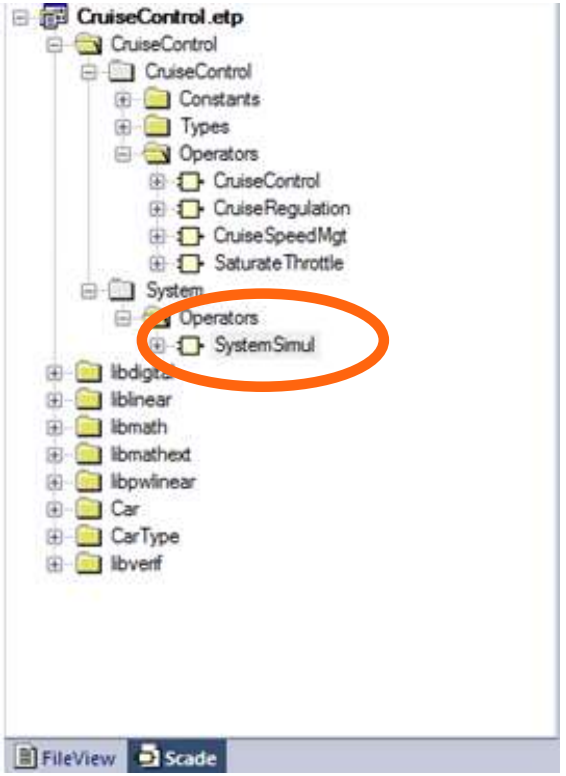
Note: In the Port Number field, set the port for the co-simulation mode with an external application (by default 64064)

Simulator Interface

Select the operator to simulate from Scade View:

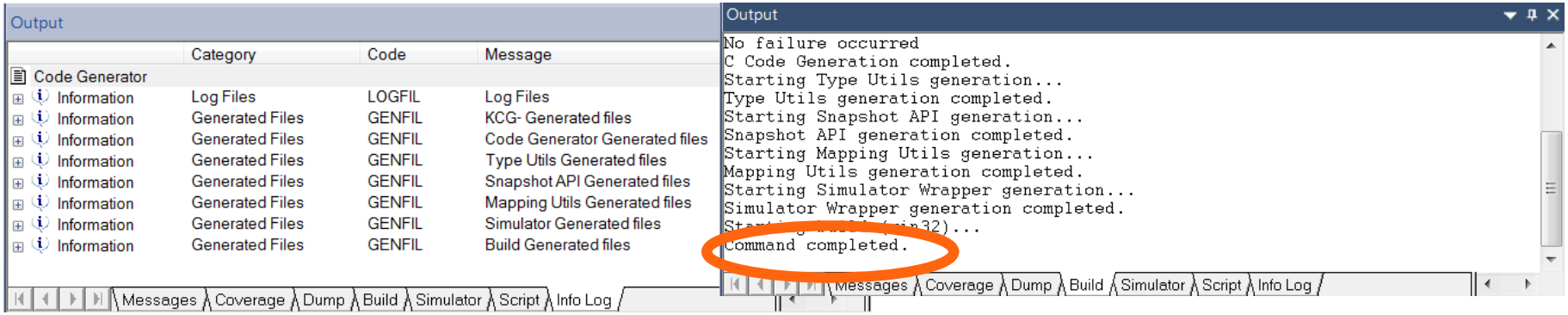
From Code Generator toolbar:

- Select the “Simulation” configuration
- Build the simulation binary



Simulator Interface

Look under the “Info Log” and “Build” tab if the simulation build is completed:



Launch the simulation:

Simulator Interface

Play buttons

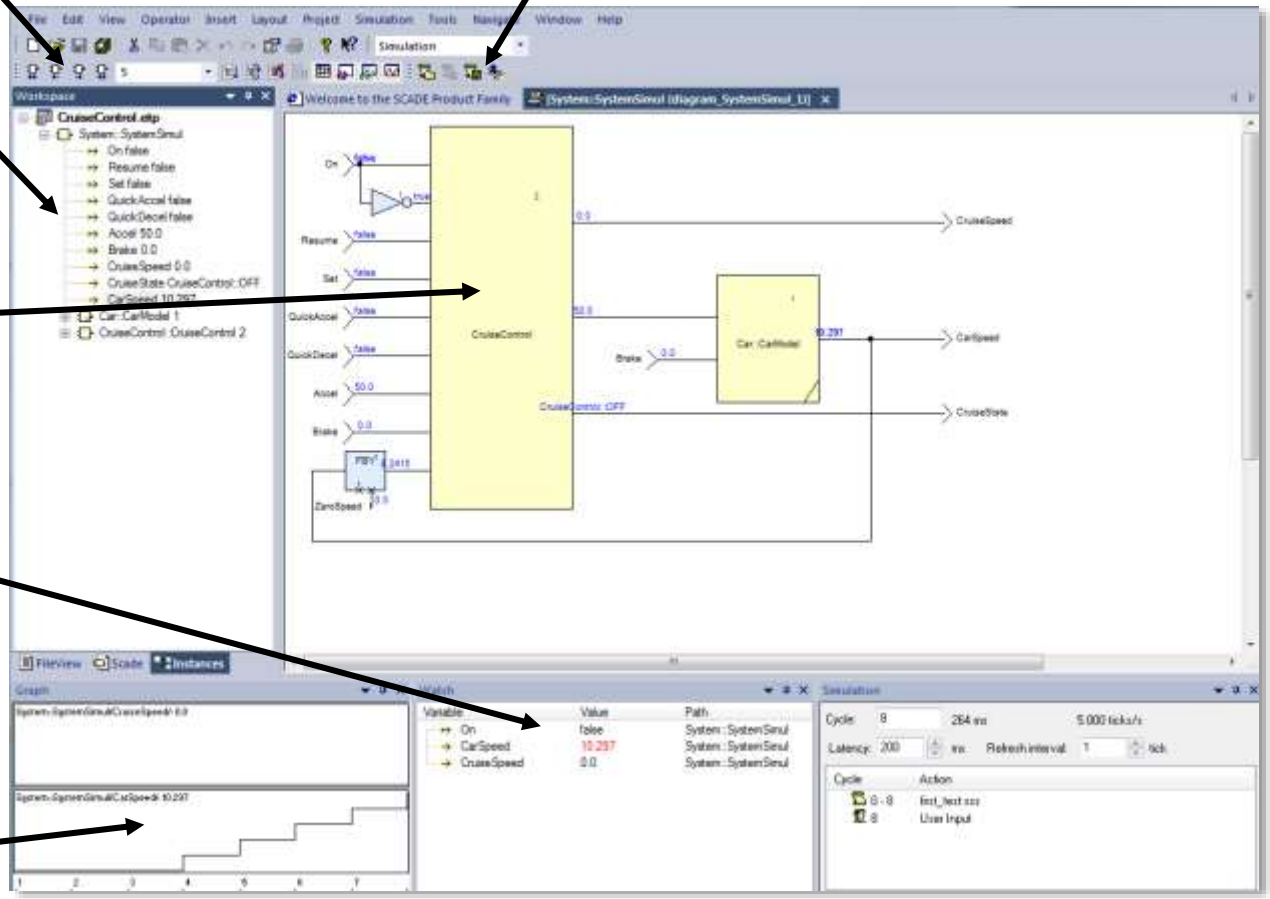
Scenario file handling

User inputs

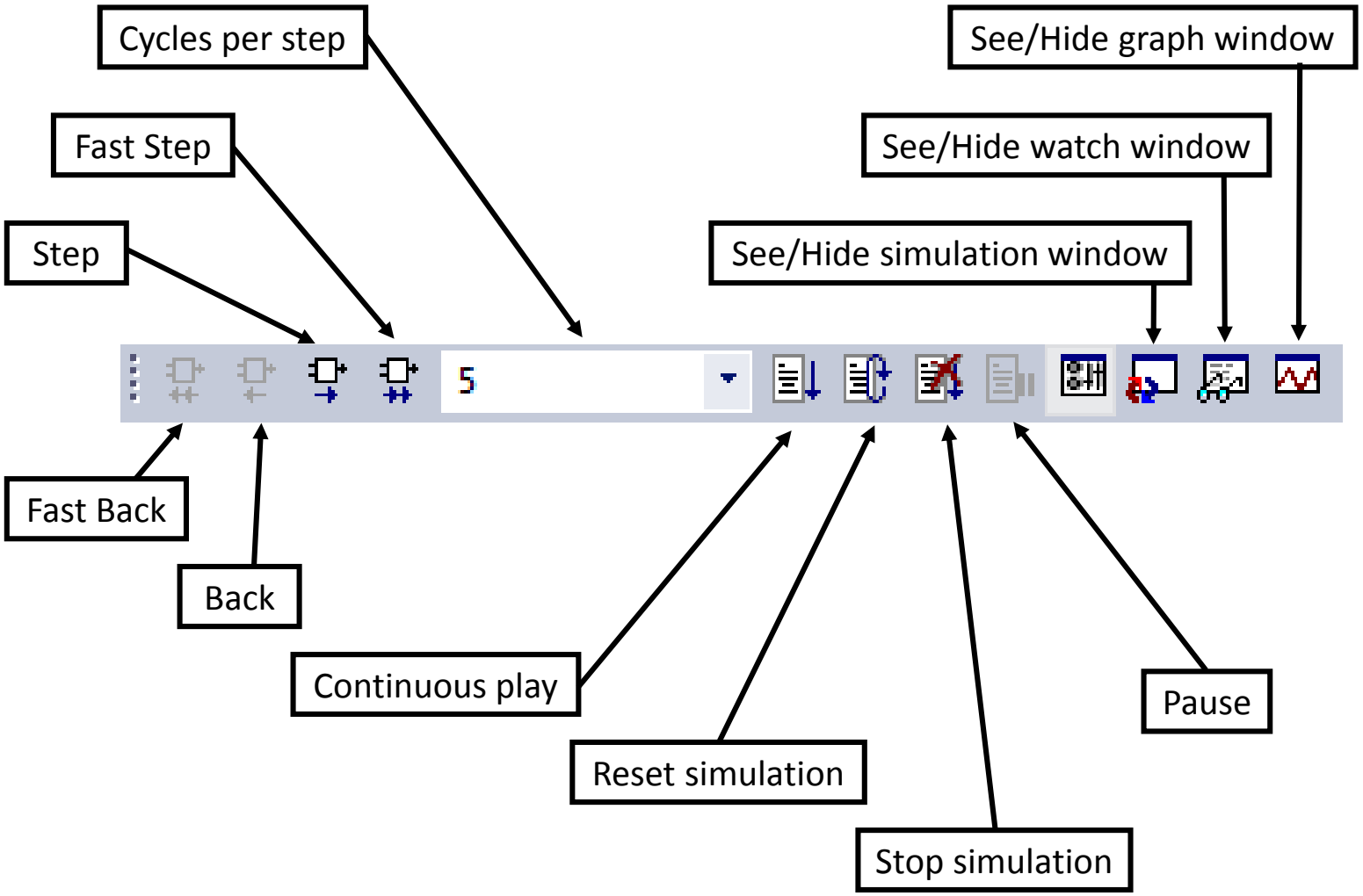
Updated variables

User inputs & Watch list variables

Graph: Timing Diagrams

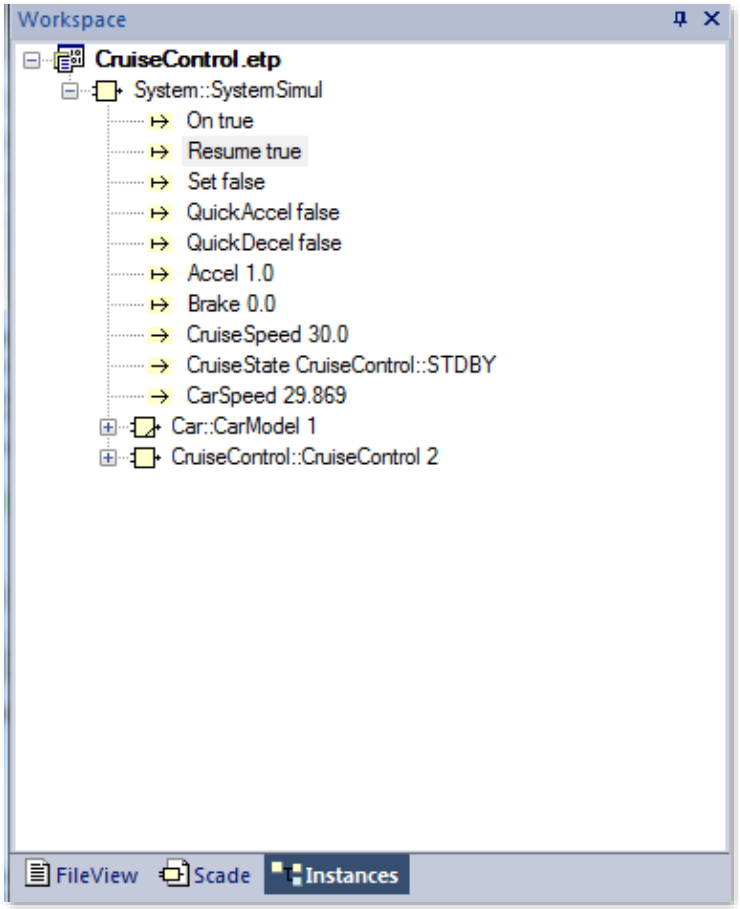


Simulator Toolbar



Set Input Values

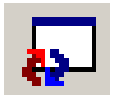
To assign values to the model's inputs for each cycle :



Click on the input (or press F2) into the Instances tab and set the new value.

Simulation Window

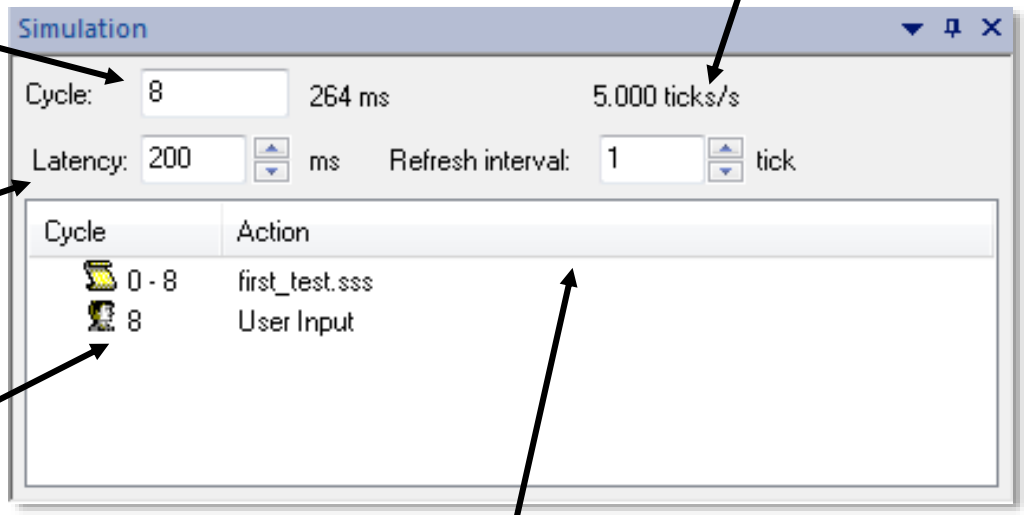
Menu: View >Docking Windows >Simulation

Simulation toolbar button: 

Current cycle. Change = Go to

Cycle delay in continuous mode

Simulation actions history



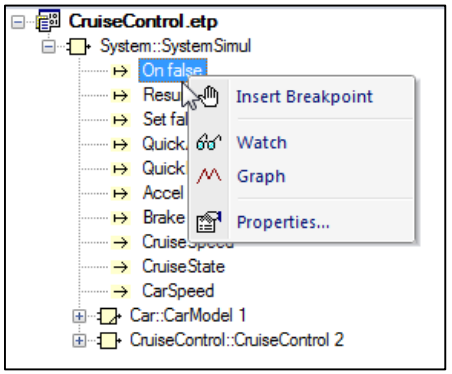
Number of ticks/s in continuous mode

Customizable refresh rate in numbers of ticks to speed up simulation

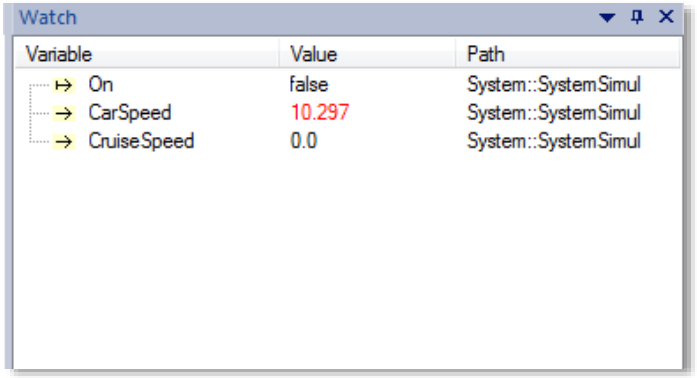
Watch Window

Menu: *View >Docking Windows >Watch*

Simulation toolbar button: 



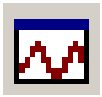
- Drag & drop the variables to watch from the Instances browser or
- Use the variable contextual menu or
- Use the command Simulation – Watch List Add - Instance Path

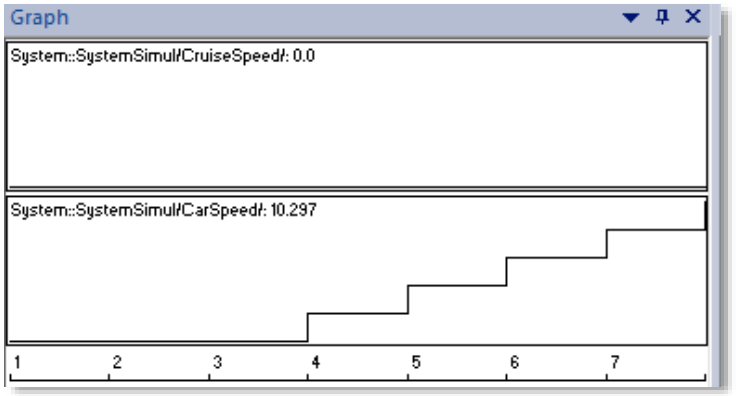


Useful to assign values to the model's inputs

Use DEL to remove a variable from the view

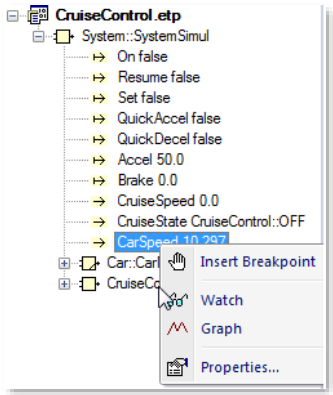
Graph Window

Menu: *View >Docking Windows >Graph* 



Simulation toolbar button:

- Drag & drop variables to view from the:
  - Instance view / instance net view or
  - Watch window (work also on a selected field a structured value) or
  - Use the variable contextual menu.



Use DEL to remove a variable from the view

Graph settings in menu *Tools > Options*, Simulator tab

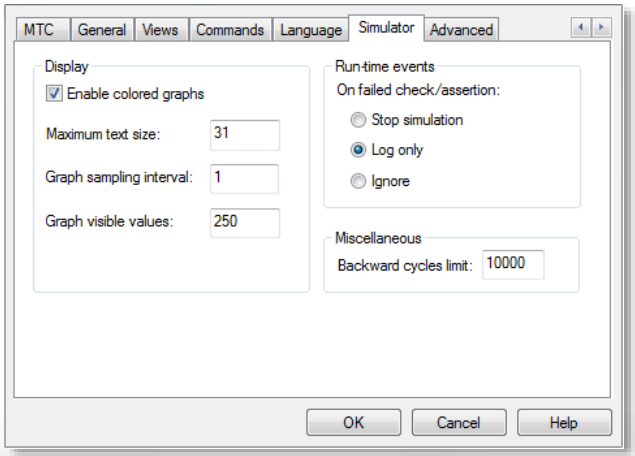
Copy graphs into the clipboard to paste into others applications

Graph Window

Menu: *Tools >Options >Simulator*

Display options:

- Check Enable colored graphs to display colored lines in the Graph window
- In the Graph sampling interval field, specify the sampling interval (per tick) of the graphs during continuous or multiple step simulation
- In Graph visible values field, set the number of values displayable in the Graph window



AGENDA

Integrated Development Environment

Batch Mode

Simulation in Batch Mode

Simulation can be used in batch mode:

- This can be useful to launch several tests in a row and get back the output files later

The first step is to generate the simulation executable for the node to simulate:

- Using the “classic” way, by pressing build  from the editor
- Using the batch mode, with the following command:
  SCADE -code <project.etp> -sim -conf <configuration>

Both actions generate the simulation executable and dynamic library named after the simulated node (i.e. respectively <root_node>.exe and <root_node>.dll) in the <target_dir> directory specified in <configuration>

Simulation in Batch Mode

Second step is to launch the batch execution of the simulation executable:

<SCADE installdir>/bin/scssmlnc.exe <root_node>.dll [-scenario <filename.(sss|in|sts)>] [-out <filename.(out|csv)>] [-obs <filename.obs>]

Depending on the options used when launching simulation, the behavior will be different

Simulation in Batch Mode

Full Batch Mode

- This mode runs simulation sessions driven by the SCADE Suite Simulator inputs' scenarios (input vector scenarios, TCL scenarios, or state scenarios) without any human intervention
- The outputs for successive simulation cycles are produced in a file by the –out option
- Simulation cycles are computed from the values of the input scenario file that must be specified by the –scenario option

Simulation in Batch Mode

Interactive Command Line Mode:

- This mode allows to launch any SCADE Suite Simulator feature using simulation TCL command lines
- The mode lets you interact directly with the simulation executable: you enter commands on the prompt and the executable displays computed results
- This is the default mode when the –out option is not used

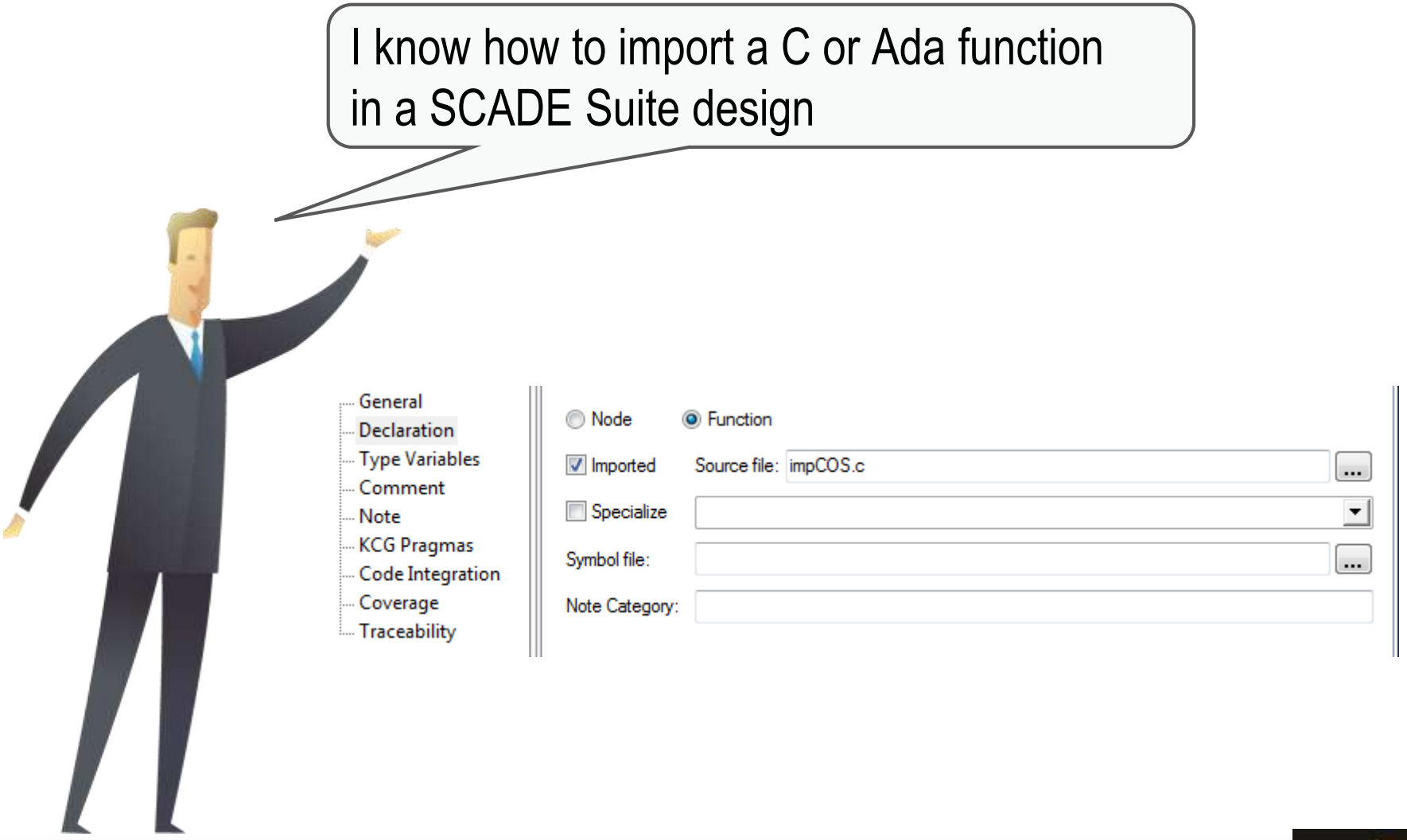
Simulation in Batch Mode

Options:

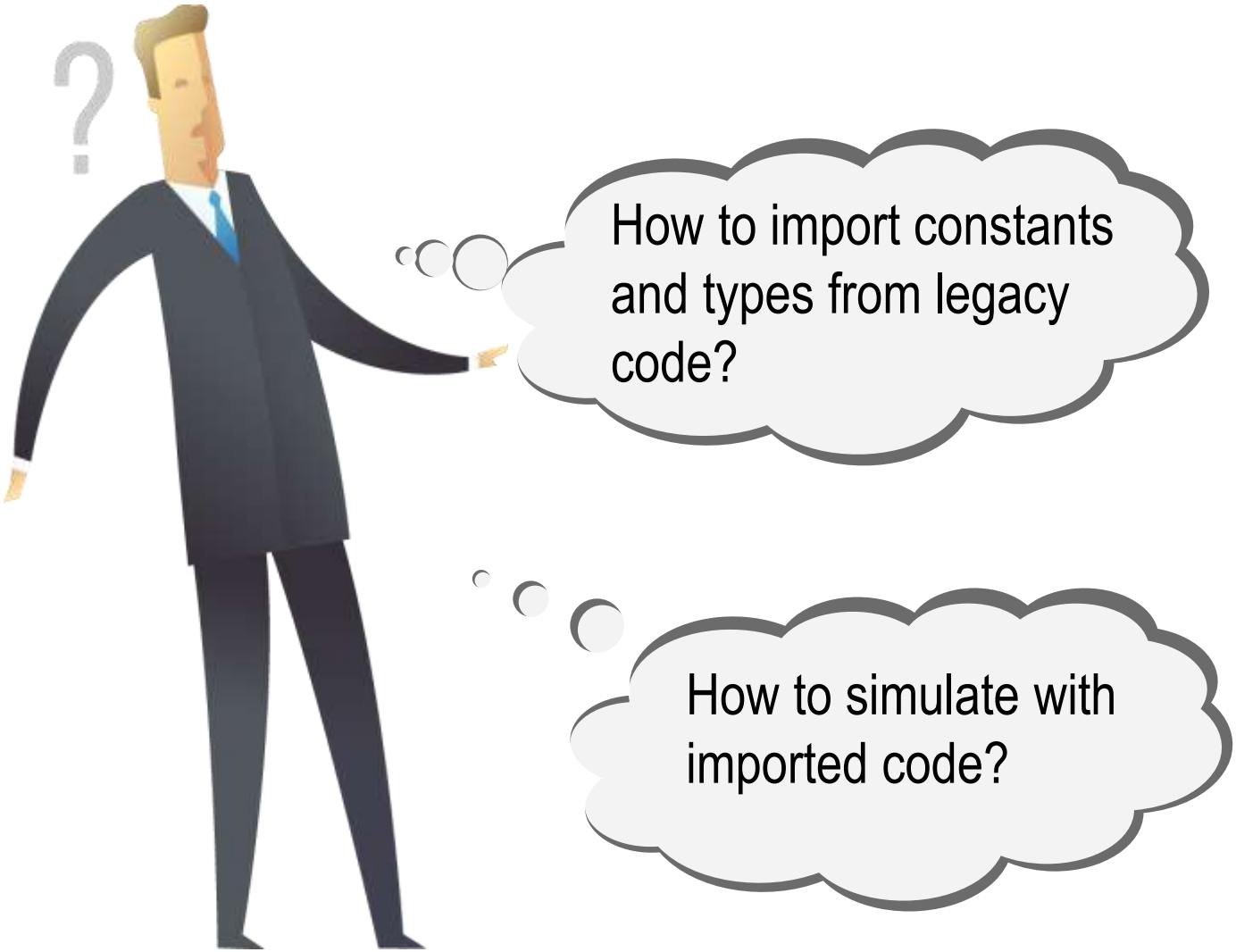
- -scenario:
  Load all kind of scenarios as discriminated by their file extension (.sts, .sss, or .in) Can specify a list of paths to heterogeneous scenario files separated by a space The specified scenarios are played consecutively in the order of the list
- -out:
  Specify the output file where the simulation results are saved. Use this option to run the simulation in “Full Batch Mode”
- -obs:
  This option specifies the name of a file containing a list of variables that must be added to the variables saved in the output file

Imported Code

Imported Code: I Know!!!



What Else?


How to import constants and types from legacy code?
How to simulate with imported code?

AGENDA

Imported Types

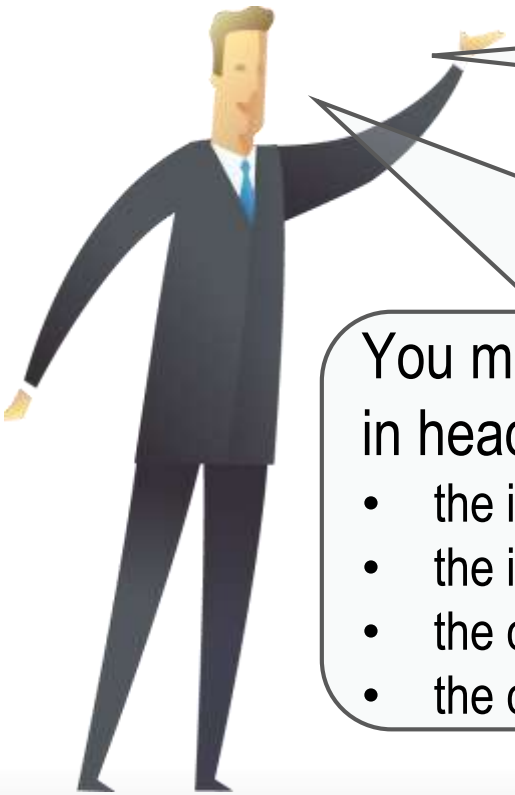
Imported Constants

Imported Functions

Imported Nodes

Imported Types (C Code)

Externally defined user types can be introduced in SCADE Suite models as imported types



Only imported operators can modify the data of imported typed flows

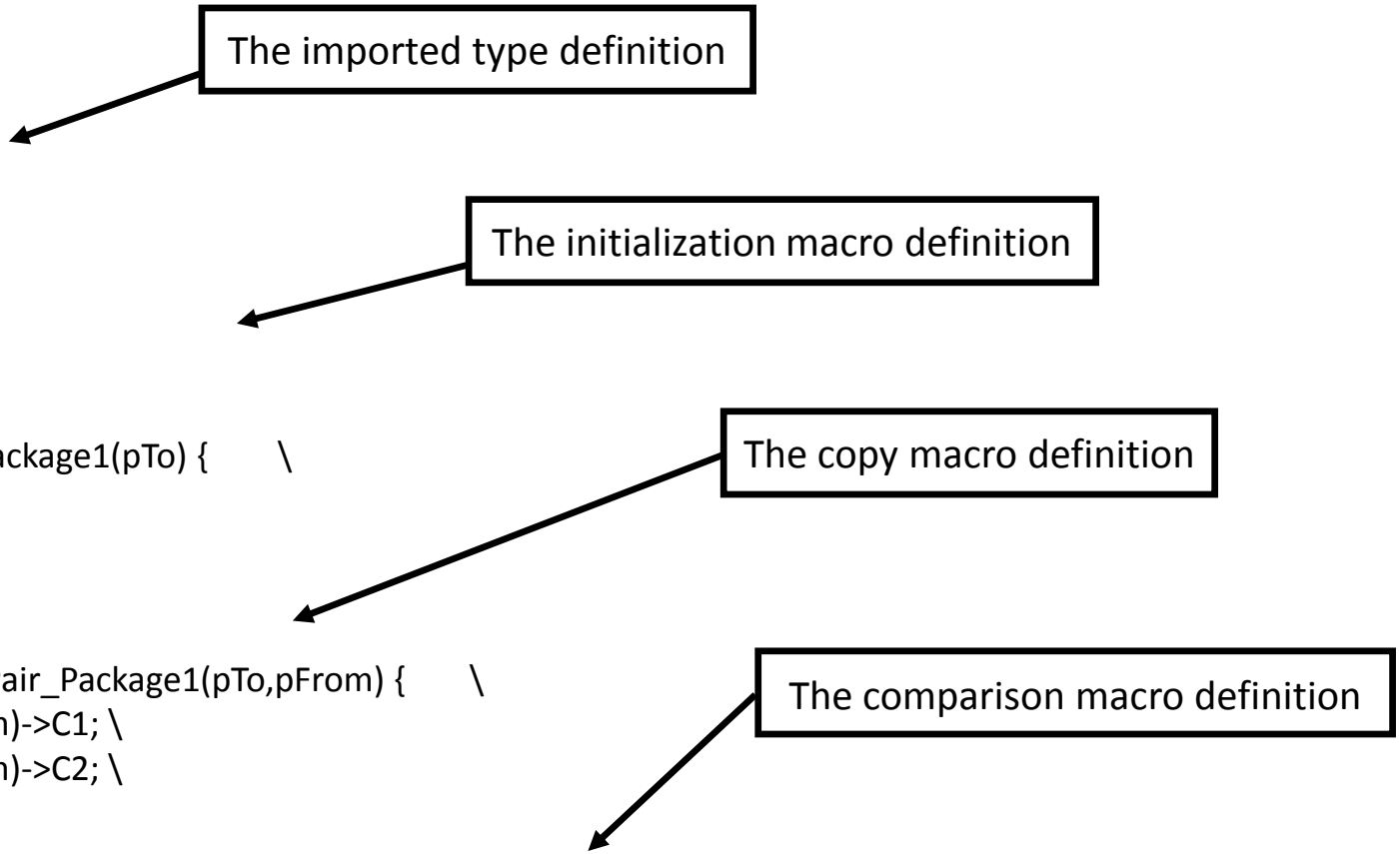
You must provide the definition of imported types in header files that contain definitions relative to:

- the imported type
- the initialization macro named kcg_init_<type name>
- the copy macro named kcg_copy_<type name>
- the comparison macro named kcg_comp_<type name>

C Imported Types

Example

Pair imported type defined in Pair.h into the Package1 package:



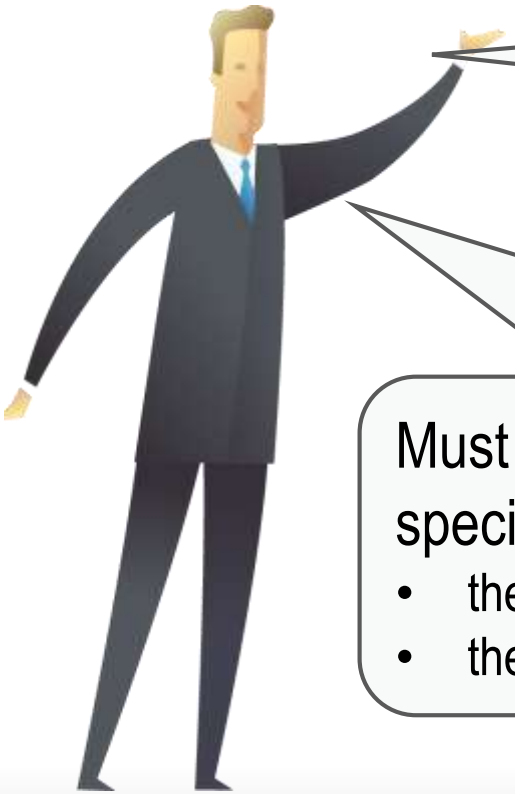

```
#ifndef PAIR_H
#define PAIR_H

typedef struct{
  int8 C1;
  int8 C2;
} Pair_Package1;

#define Pair_init_Package1(pTo) {       \
  (pTo)->C1=0; \
  (pTo)->C2=0; \
}

#define kcg_copy_Pair_Package1(pTo,pFrom) {      \
  (pTo)->C1 = (pFrom)->C1; \
  (pTo)->C2 = (pFrom)->C2; \
}

#define kcg_comp_Pair_Package1(x, y)   ( ( (x)->C1==(y)->C1 &&  (x)->C2==(y)->C2)?kcg_true:kcg_false)
#endif
```

Imported Types (Ada)

Ada

Externally defined user types can be introduced in SCADE Suite models as imported types

Only imported operators can modify the data of imported typed flows



Must provide the definition of imported types in specification files containing definitions relative to:

- the imported type
- the initialization value

ANSYS

Ada Imported Types

Ada

Example

Pair, imported type defined into the Pimp package:

```
package Pimp
-- Pimp::
is

 -- Pimp::Pair/
 type Pair is record;
      C1: Kcg_Config.Kcg_Int8;
      C2: Kcg_Config.Kcg_Int8:
  end record;

 --- initial value to be provided
 Pair_init : constant Pair := Pair' (C1 => 28, C2 => 28);

end Pimp;
```

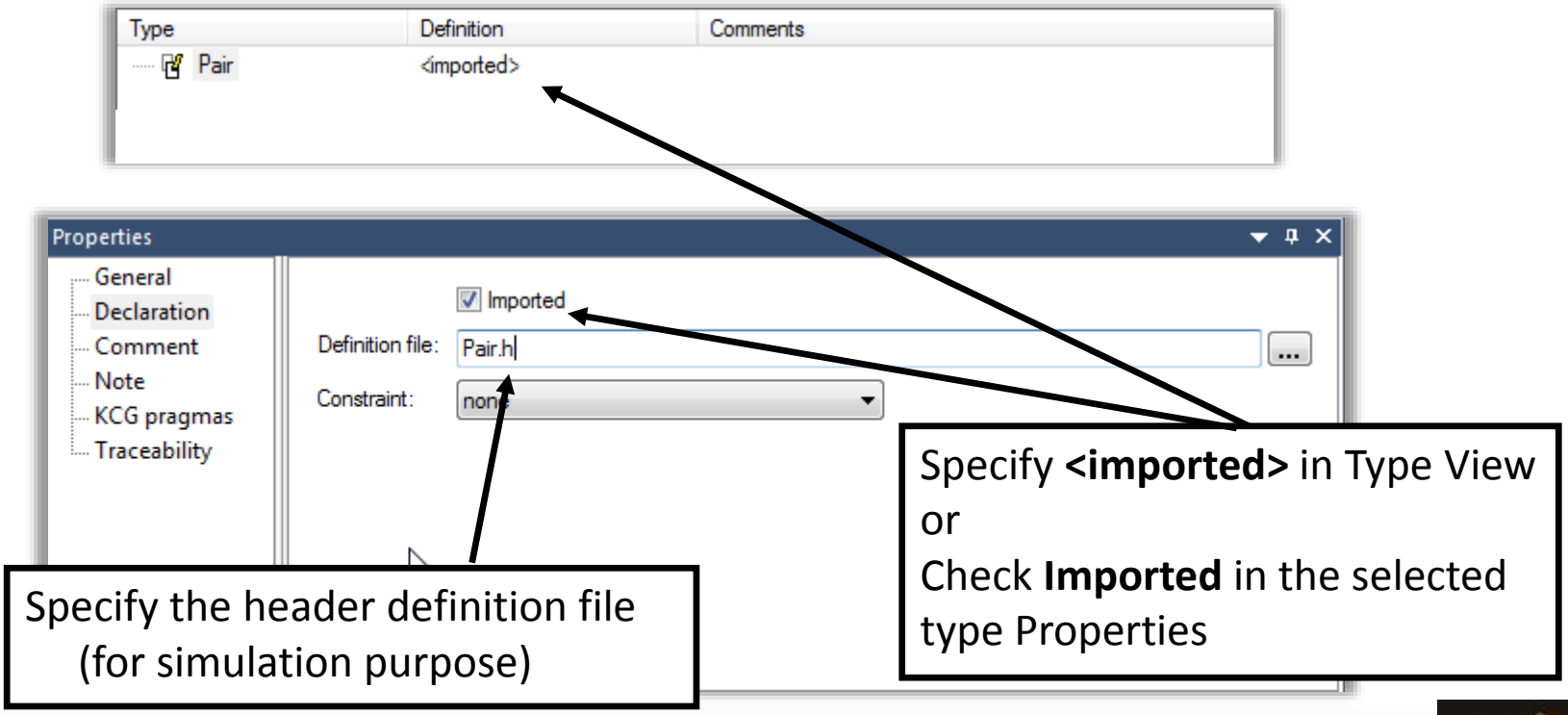
The imported type definition

The initialization definition

Imported Types

To set an imported type:

- Create a new type in a package or at the model level from Scade View
- Write the required definition(s) in the header file



Imported Types

Users must define conversion and comparison functions
used in the simulation mode
for imported types to simulate the model
(see user manual for details)

Imported Types

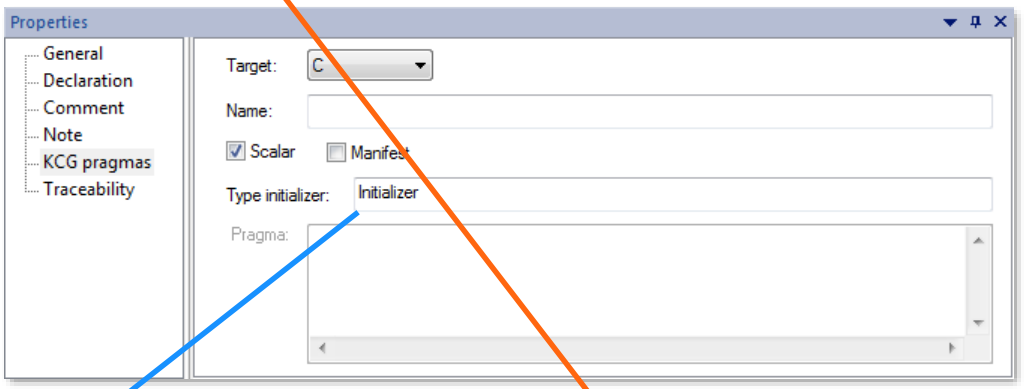
When **initializer** pragma is attached to an imported type, the initialization function of the associated type is named according to the pragma definition

If an imported type has no pragma, the default name **<type_name>_init** is used

C Imported Types

Example

When an initializer pragma is attached to an imported type named **Pair**, exactly the **field name** is used as identifier for the initialization function of the associated C type. Without pragma, «**Pair_init**» is generated in the model initialization function.




```
void Operator1_init(outC_Operator1 *outC)
{
  outC->O_Bool = kcg_true;
 Initializer(&outC->O_GBPair);
  ImpNode_init(&outC->Context_1);
}
```

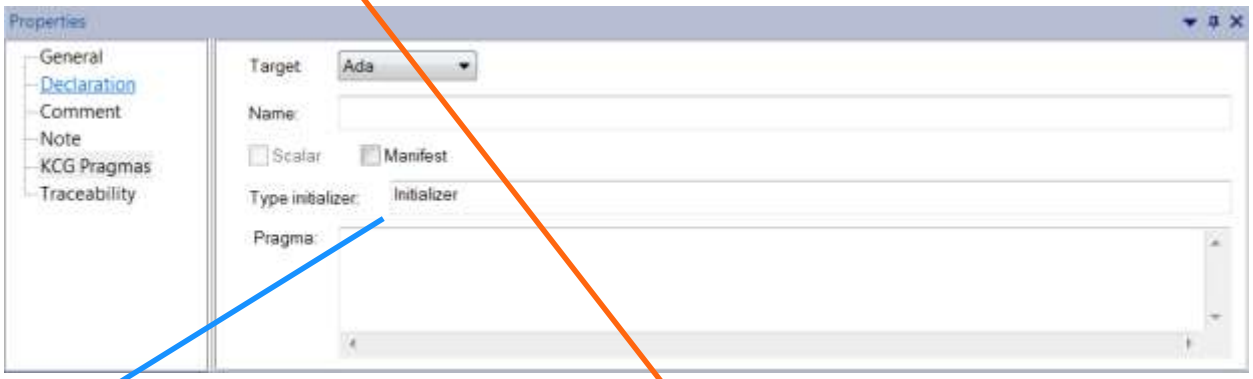
```
void Operator1_init(outC_Operator1 *outC)
{
  outC->O_Bool = kcg_true;
Pair_init(&outC->O_GBPair);
  ImpNode_init(&outC->Context_1);
}
```

Ada Imported Types

Ada

Example

When an initializer pragma is attached to an imported type named **Pair**, exactly the **field name** is used as identifier for the initialization function of the associated C type. Without pragma, « **Pair_init** » is generated in the model initialization function




```
package Pimp
 ....
 --- initial value to be provided
Initializer : constant Pair := Pair' (C1 => 28, C2 => 28);

end Pimp;
```

```
package Pimp
 ....
 --- initial value to be provided
 Pair_init : constant Pair := Pair' (C1 => 28, C2 => 28);

end Pimp;
```

AGENDA

Imported Types

Imported Constants

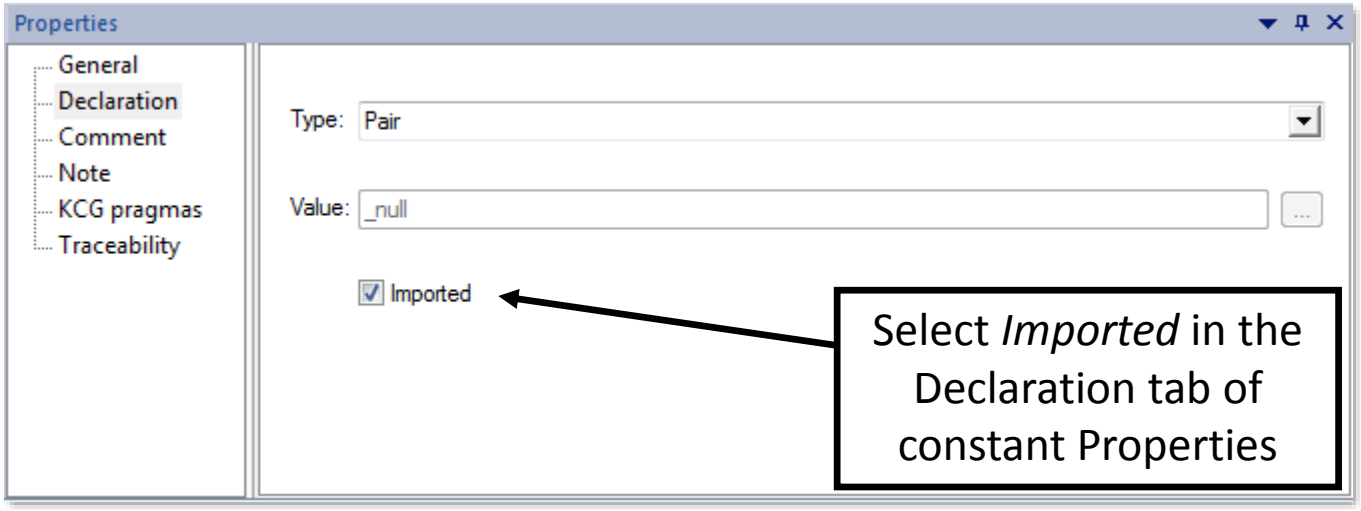
Imported Functions

Imported Nodes

Imported Constants

Externally defined user constants can be introduced in SCADE Suite models as imported constants

Constants can be typed with Scade types or imported types



Constant	Type	Value	Comments
Constant1	Pair	<Imported>	

C Imported Constants

Provide the value of the constant in a C file (according to the option with or without KCG Pragma Const):

- Constants typed with Scade types
- Const int8 constant1 = 7;

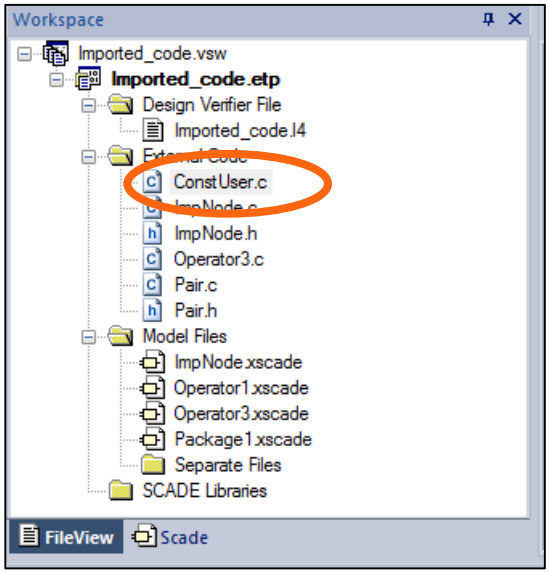
- Constants typed with imported types
- Const Pair_Package1 Constant1_Package1 = {1, 2};

C Imported Constants and Simulation

For the simulation mode, insert the definition file in the project:

- In FileView, select the project, right-click, and select Insert >Files

Open the Properties window and specify its type:



For operators and types: this action is not mandatory for files already imported through the Properties

Ada Imported Constants

Ada

Provide the value of the constant in .ads file (according to the option with or without KCG Pragma Const):

Constants typed with Scade types:

- Constant1 : constant Kcg_Config.Kcg_Int32 := 7;

Constants typed with imported types:

- Constant1 : constant Pimp.Pair := (0, 1);

ANSYS

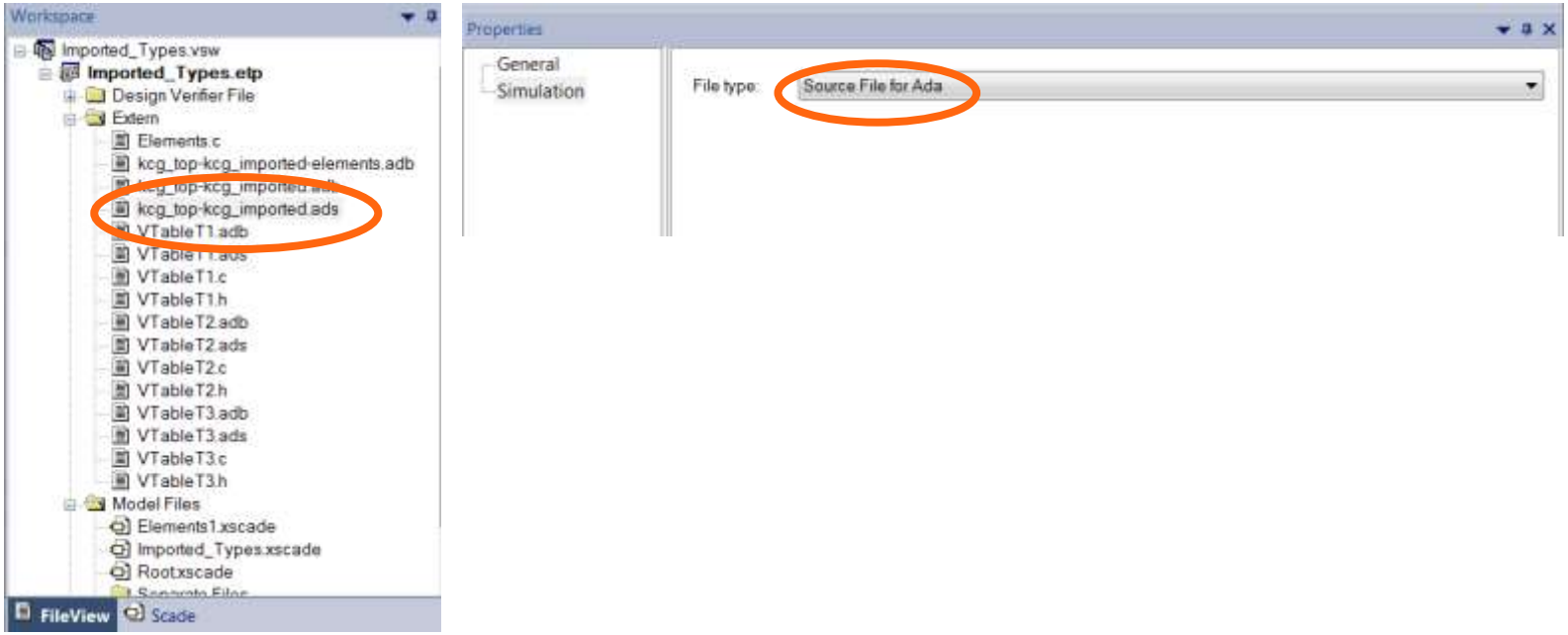
Ada Imported Constants and Simulation

Ada

For the simulation mode, insert the specification file in the project:

- In FileView, select the project, right-click, and select Insert > Files

Open the Properties window and specify its type:



For operators and types: this action is not mandatory for files already imported through the Properties

AGENDA

Imported Types

Imported Constants

Imported Functions

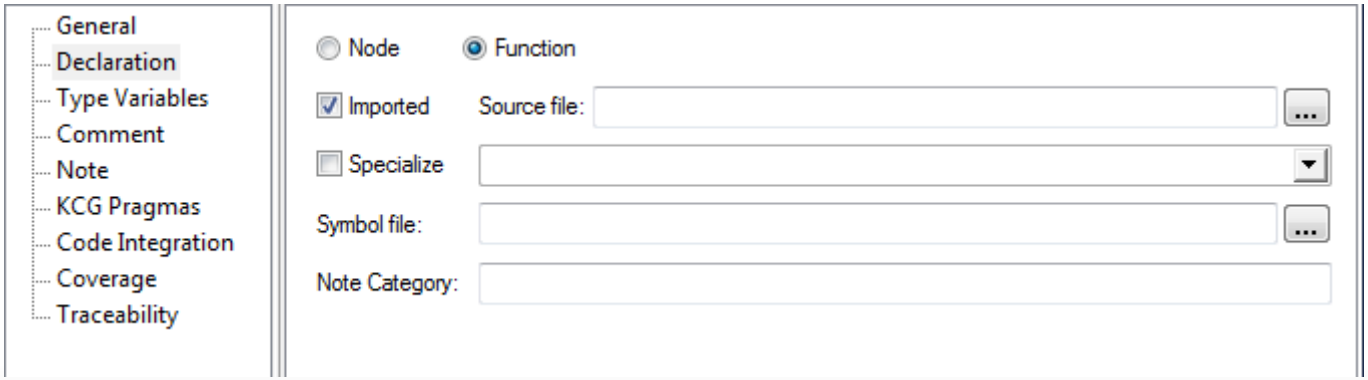
Imported Nodes

Imported Operators

To implement in the host language operators not suited for SCADE Suite modelling or already existing such as legacy code.

Right-click on Operators folder in Scade View and select:
Insert > Imported Operator

Operators can be a function or a node (with memory)

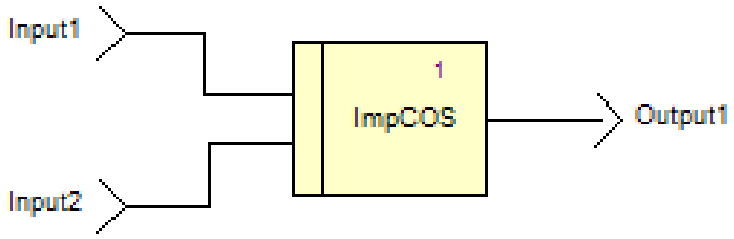


Imported Operators

The operator's name and interface are defined in SCADE Suite.

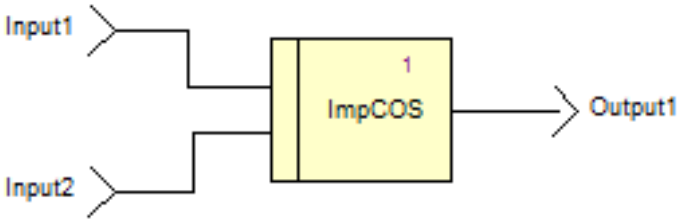
Its body is defined in the host language.

It graphically appears as a box with a vertical bar on the left hand side.



Imported Operators: Functions

Outputs only depend on inputs:



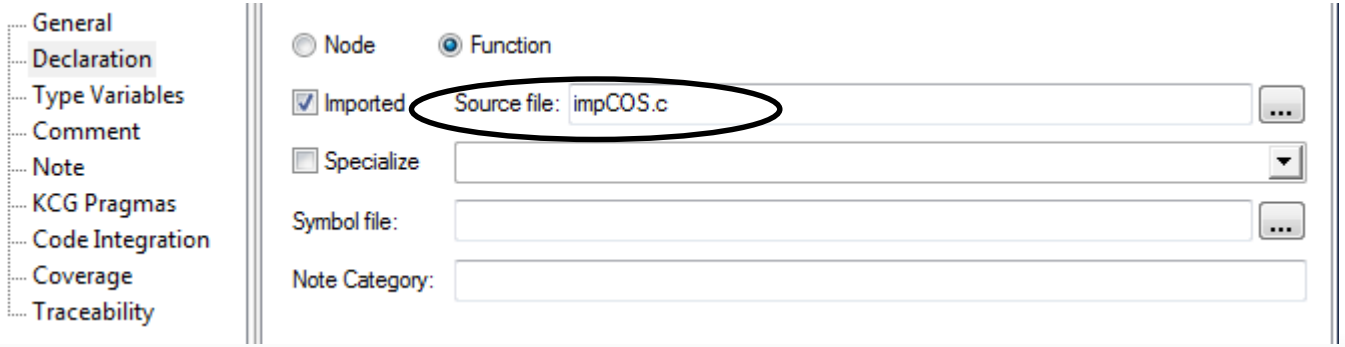

```
kcg_float32 ImpCOS(kcg_float32 Input1,kcg_float32 Input2)
{
  kcg_float32 Output1;
  Output1 = cos(Input1 + Input2);
  return Output1;
}
```

C Imported Function Implementation

KCG generates a kcg_imported_functions.h file containing the function prototypes for the imported function operators:

```
#ifndef ImpCOS
/* ImpCOS */
extern kcg_float32 ImpCOS(
  /* ImpCOS::Input1 */ kcg_float32 Input1,
  /* ImpCOS::Input2 */ kcg_float32 Input2);
#endif /* ImpCOS */
#endif
```

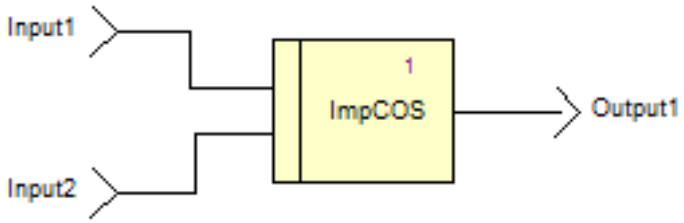
In the Properties window, specify the name of the source file for the simulation purpose (or add it in the project)



Imported Operators: Functions (Ada)

Ada

The output only depends on inputs:




```
function ImpCOS(
  Input1 : in Kcg_Config.Kcg_Float32;
  Input2 : in Kcg_Config.Kcg_Float32) return Kcg_Config.Kcg_Float32
is
  Output1: Kcg_Config.Kcg_Float32;
begin
  Output1:= cos(Input1 + Input2);
return Output1;
end ImpCOS;
```

Ada Imported Function Implementation

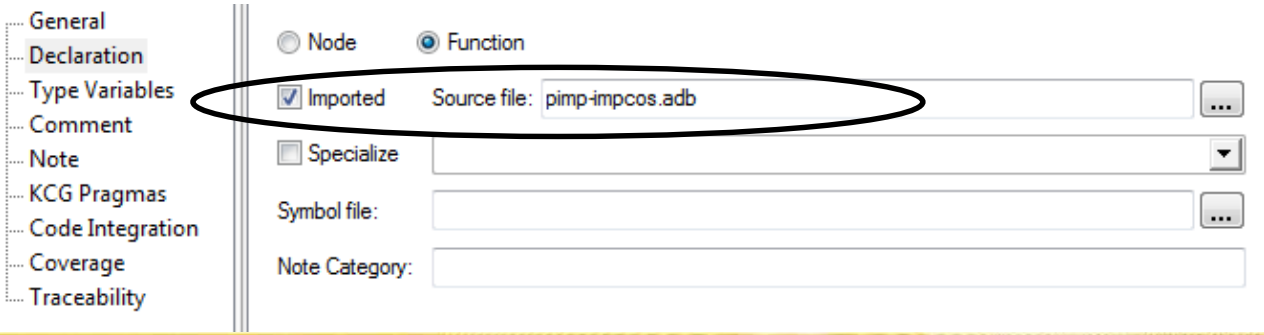
Ada

KCG generates a template of specification (.dads) and bodies (.dadb) files for the imported function operators:

```
package Pimp
-- Pimp::
is
  -- Pimp::ImpCOS/
  function ImpCOS(
    -- Input1/
    Input1 : in Kcg_Config.Kcg_Float32;
    -- Input2/
    Input2 : in Kcg_Config.Kcg_Float32) return
Kcg_Config.Kcg_Float32;
end Pimp;
```

```
function ImpCOS(
-- Input1/
Input1 : in Kcg_Config.Kcg_Float32;
  -- Input2/
  Input2 : in Kcg_Config.Kcg_Float32) return Kcg_Config.Kcg_Float32
is
Output1 : Kcg_Config.Kcg_Float32;
begin
  -- The body of this function must be provided
  null;
  return Output1;
end ImpCOS;
```

- In the Properties window, specify the name of the source file for the simulation purpose (or add it in the project)



AGENDA

Imported Types

Imported Constants

Imported Function

Imported Nodes

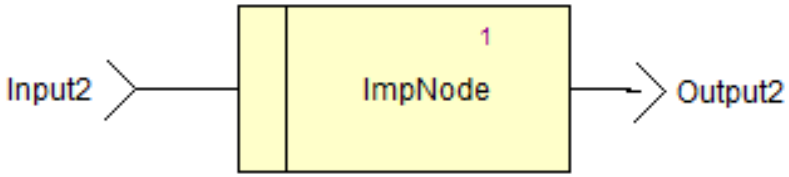
Imported Operators: Nodes

Outputs depend on inputs and internal states

Enable multiple instances

Are reset-able

Called in following the data flow order




```
void ImpNode(kcg_real Input2, outC_ImpNode *outC)
{
  outC->Output2 = cos(Input1) + 20.0;
}
```

C Imported Node Implementation

For each imported node, name_<path>, two template files are generated:

- `name_<path>.dh`: contains the template declarations of:
  - The reset function: name_reset_<path>() and
  - The initialization function: name_init_<path>()
  - The cyclic function: name_<path>()
  - The context structure of the C functions
- `name_<path>.dc`: contains the template definitions of the C functions:

name_reset_<path>(), name_<path>() and name_init_<path>())

- Users provide the initialization, reset and cyclic functions for imported nodes

Note: <path> corresponds to the package hierarchy leading to the operator

C Imported Node Implementation: Header

Example

Copy the header template in name_<path>.h

Fill the context structure adding the initialization and memory fields ()

```
typedef struct {
  /* ---------------------------  outputs  --------------------------- */
  kcg_float32 /* ImpNode::Output1 */ Output1;
  /* -------------  insert eventual inits and memories  -------------- */
  kcg_bool init;
} outC_ImpNode;
```

C Imported Node Implementation: Body

Example

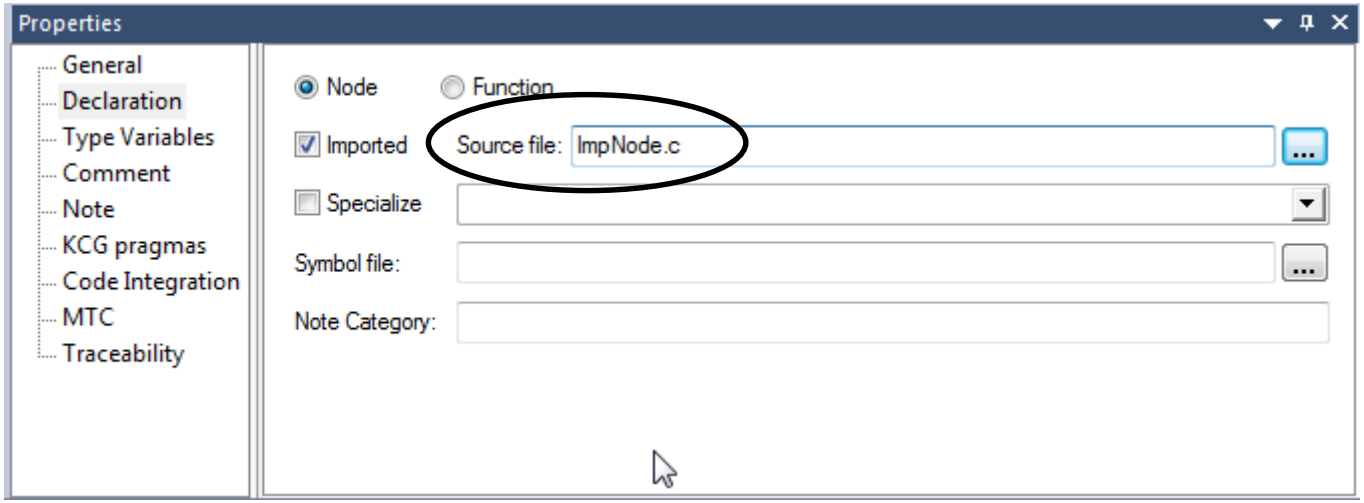
Copy the body template in name_<path>.c

Fill the function bodies:

```
void ImpNode_init(outC_ImpNode *outC)
{
  /* The body of this function must be provided */
  outC->init= kcg_true;
  outC->Output1 = 0.0;
}

void ImpNode(/* ImpNode::Input1 */ kcg_float32 Input1, outC_ImpNode *outC)
{
  /* The body of this function must be provided */
 if (outC->init)
 {
  outC->init= kcg_false;
  outC->Output1 = 0.0;
 }
 else
 {
  outC->Output1 = cos(Input1) + 20.0;
 }
}
```

C Imported Node Implementation

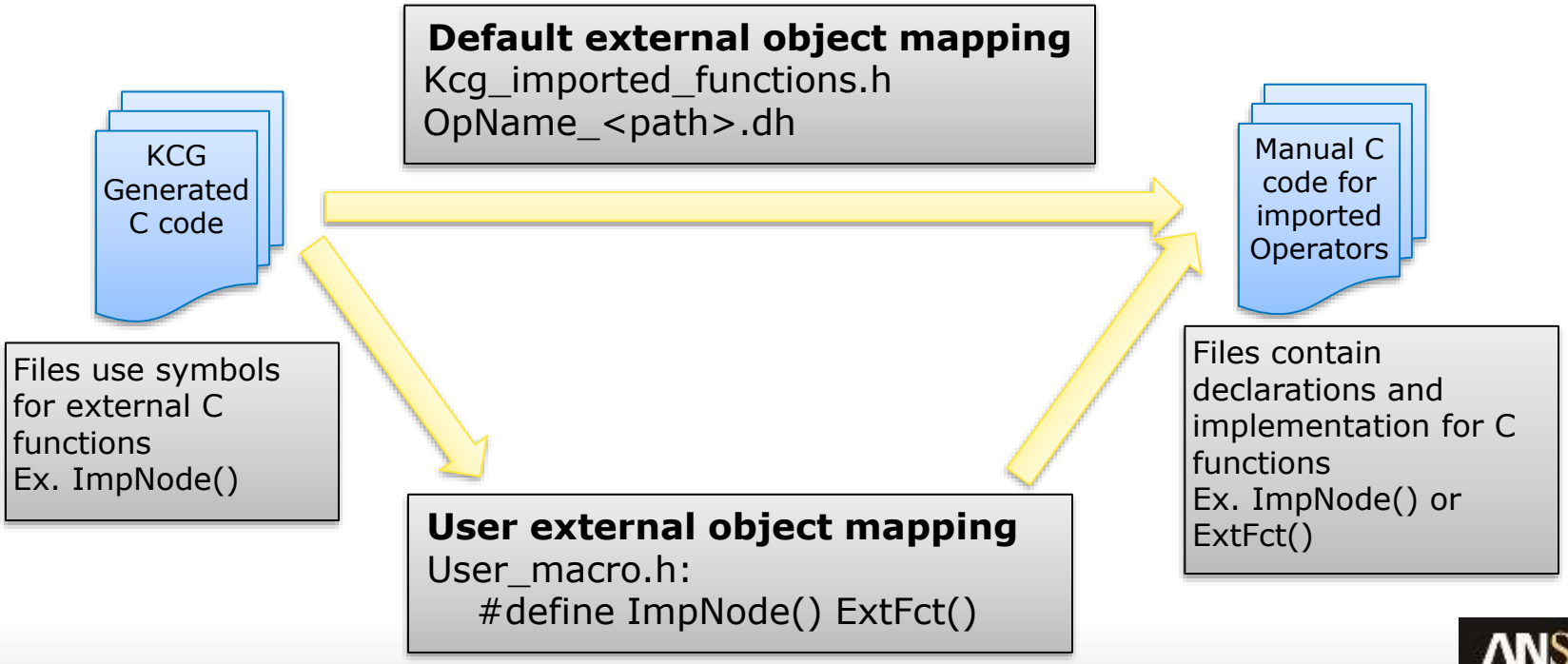
Specify the created C file in the properties of the SCADE Suite imported operator for simulation purposes:



Change Default Mapping to C External Code

The default mapping between the generated code and the external code can be changed in two ways:

- Add a KCG pragma name to change the name of the C symbol used in the generated code: ImpNode()
- Define a new mapping using a macro in a user macro file: ExtFct()



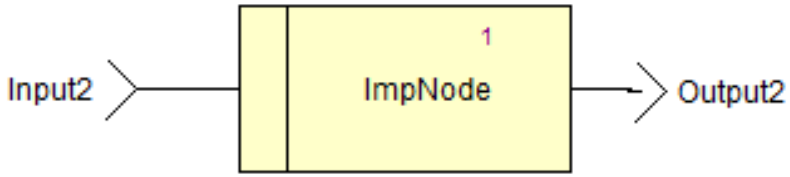
Imported Operators: Nodes (Ada)

Ada

Outputs depend on inputs and internal states

Enable multiple instances

Are reset-able




```
procedure ImpNode(Input2 : in Kcg_Config.Kcg_Float32;
                  Ctx : in out Context_ImpNode)
is
begin
Ctx.Output2 := cos(Input2) + 20.0;
end ImpNode;
```

ANSYS

Ada Imported Node Implementation

Ada

KCG generates a template of specification (.dads) and bodies (.dadb) files for the imported nodes (separate files for reset, init and cyclic procedures)

```
package Pimp
Is
....
procedure ImpNode_Reset(Ctx : in out Context_ImpNode);
procedure ImpNode_Init(Ctx : out Context_ImpNode);
procedure ImpNode(Input1 : in Kcg_Config.Kcg_Float32; Ctx :
in out Context_ImpNode);

end Pimp;

procedure ImpNode_Init(
  Ctx : out Context_ImpNode)
is
begin
  -- The body of this function must be provided
  null;
end ImpNode_Init;
```

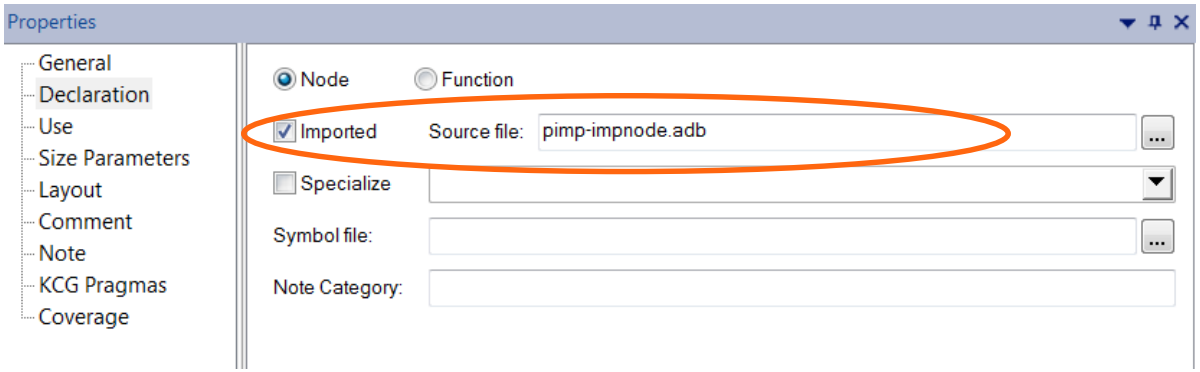
```
procedure ImpNode(
  -- Input1/
  Input1 : in Kcg_Config.Kcg_Float32;
  Ctx : in out Context_ImpNode)
is
begin
  -- The body of this function must be provided
  null;
end ImpNode;

procedure ImpNode_Reset(
  Ctx : in out Context_ImpNode)
is
begin
  -- The body of this function must be provided
  null;
end ImpNode_Reset;
```

ANSYS

Ada Imported Node Implementation

Ada

In the Properties window, specify the name of the source file for simulation purposes (or add it in the project)



Introduction to Best Practices

Engineering Best Practices With the Help of the SCADE Toolset

Day 4 presents a menu to related tools, processes and concepts.

It leads to other training materials and voluntarily open discussion to other topics.

Should you need more information on any of the presented items, please request the necessary training courses

Basic Trainings

SCADE Basic Trainings present the **methodology**, **language** and **tools usage** according to the steps of a software development process.

They are meant for beginners with **no prior, or little experience** of the SCADE products and environment.

The Basic Trainings are composed of different modules presenting the modeling activity using **SCADE Suite**, **SCADE Display**, **SCADE Architect, SCADE Test,** and **SCADE Solutions for ARINC 661**

Proposed trainings are:

- Model-Based Systems Engineering with SCADE Architect
- Model-Based Design with SCADE Display
- Testing SCADE Suite Applications with SCADE Test
- Testing SCADE Display applications with SCADE Test
- Model-Based Virtual Prototyping with SCADE Test Rapid Prototyper
- Model-Based Design of ARINC 661 Compliant Cockpit Display Systems with SCADE Solutions for ARINC 661

Advanced Trainings

SCADE Advanced Training focus on **SCADE solutions** and **best practices**. They **combine your experience to ours** to make **you become an expert**.

Advanced training courses are ideal for SCADE users who already have an understanding and some experience in using the SCADE tools:

- Optimize your SCADE Suite Models and Code Performance
- Optimize your SCADE Display Models, Code and Integration Performance
- Extend the SCADE Suite Capabilities using TCL Scripts
- SCADE Suite Modeling with Import of Simulink/Stateflow® Models
- Model-based Formal Verification with SCADE Suite Design Verifier
- SCADE Architect Configurator
- SCADE Avionics Package / SCADE Automotive Package

Process Trainings

They provide key insights on how to comply with the **specific requirements of the certification standards** and aim at **sharing our expertise** and experience in system and software engineering processes.

They support you in **implementing quickly and efficiently your projects** and **reaching your objectives**.

- Effectively Manage a DO-178C (or DO-178B) Certified Model-Based Project with SCADE
- Optimize Verification and Validation Strategies for DO-178C (or DO-178B) Compliant Applications using SCADE
- Transition to DO-178C
- SCADE Model-Based Systems Engineering of ARP-4754A Compliant Aeronautics Systems
- Realization of a Railway Application Compliant with the EN 50128:2011 Standard with SCADE

Online Introduction Trainings

On the **YouTube ANSYS How To channel**, you will find **introduction videos** on the complete SCADE toolset.

Entry level information for the safety critical industry-related standards is provided, as well as **SCADE best-practices** to follow them and take the tools as a support on implementing a compliant methodology.

For more advanced users, **tips and tricks videos** demonstrate the use of the tools on industry-based examples by our experts.

Come follow us on YouTube!

Contacts

Legal Contact
Esterel Technologies SAS
14/15, Place Georges Pompidou
78180 Montigny Le Bretonneux
FRANCE
Phone: +33 1 30 68 61 60
Fax: +33 1 30 68 61 61

Technical Support
Esterel Technologies SAS
Parc Avenue - 9 rue Michel Labrousse
31100 Toulouse FRANCE
Phone: +33 5 34 60 90 50
Fax: +33 5 34 60 90 41

Submit questions to Technical Support: scade-support@ansys.com
Contact one of our Sales representatives at: scade-sales@ansys.com
Direct general questions about Esterel Technologies to: scade-info@ansys.com
Discover the latest news on our products and technology at: http://www.ansys.com/products/embedded-software

Legal Information