



ENGINEERING NOTE

Title:	KCG mapping file format.
Doc ID:	KCG-EN-002
Doc Date:	July 6, 2011
Abstract:	This document is the specification of the KCG mapping file
Author(s):	PROJECT MANAGER Bruno Martin
Reviewer(s):	CORE team members
Approval:	PROJECT MANAGER Bruno Martin

1 Format of the mapping file

The following section describes the XML format of the traceability file for a full traceability.

The idea is to have a mapping file containing a "forest" of nodes (or one forest per package if packages are used). Each non-expanded node is described once and linked to the relevant target code once. Each instance of expanded node is inlined in every non-expanded node using them.

Any traceability file shall start with a banner using the following syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"? >
```

The XML root element contains a "node forest". The `commandLine` attribute will contain the command line used to generate the mapping file.

Any traceability file shall have a single root element using the following syntax:

```
Root ::= <Model date="..." commandLine="...">
        { Package | RootNode | Node | Type | Constant | Sensor | Option }
        </Model>
```

1.1 Options

The *Options* tags list the status of all the options for the generation that led to the creation of the current mapping file. If an option does not take any argument, its possible values are either `#on#` or `#off#` depending on its presence or not in the command line. If an option takes an argument, its value is either `#off#` or the corresponding argument if the option is present in the command line.

```
Option ::= <Option name="..." value="..." />
```

Note: the following options are not reported: `-h/-help`, `-xml_filter` and `-xml_basename`.

1.2 Package

A SCAD 6 package will have no corresponding structure in the target code. Each XML package element will contain the "forest" of the sub packages, nodes, types, constants and sensors it contains in the input model

In the traceability file, any package shall be traced using the following syntax:

```
Package ::= <Package scadeName="...">
            { Package | RootNode | Node | Type | Constant | Sensor }
            </Package>
```

1.3 Type

Any SCADE 6 type will have a corresponding definition in the target code. Several kind of types will therefore be traced:

```

Type ::= PredefType
      | ArrayType
      | EnumType
      | StructType
      | NamedType
      | ImportedType

```

In the traceability file, any predefined type shall be traced using the following syntax:

```
PredefType ::= <PredefType scadeName="..." targetName="..." />
```

In the traceability file, any array type shall be traced using the following syntax:

```
ArrayType ::= <ArrayType targetName="..." cellType="..." size="..."
              targetCopyFct="..." targetCompFct="..." />
```

In the traceability file, any enumeration shall be traced using the following syntax:

```
EnumType ::= <EnumType scadeName="..." targetName="...">
             { <EnumVal scadeName="..." targetName="..." /> }+
             </EnumType>
```

In the traceability file, any structure shall be traced using the following syntax:

```
StructType ::= <StructType targetName="..." targetCopyFct="..." targetCompFct="...">
               { <Field [ scadeName="..." scadeType="..." ] targetName="..."
                 targetType="..." /> }+
               </StructType>
```

Each field sub element shall link one field of the structure SCADE 6 traced type definition to the relevant structure field of the target code. The scadeType attribute shall contain the SCADE 6 type of each field.

In the traceability file, any named type shall be traced using the following syntax:

```
NamedType ::= <NamedType scadeName="..." refType="..." targetName="...">
```

In the traceability file, any imported type shall be traced using the following syntax:

```
ImportedType ::= <ImportedType scadeName="..." targetName="...">
```

1.4 Constant

In the traceability file, any constant shall be traced using the following syntax:

```
Constant ::= <Constant scadeName="..." scadeType="..." targetName="..." targetType="..."
              [ imported="true" ][ macroDef="true" ] />
```

Note: `macroDef="true"` is present when the constant gives rise to a macro definition (`#define`) in the C code.

1.5 Sensor

In the traceability file, any sensor shall be traced using the following syntax:

```
Sensor ::= <Sensor scadeName="..." scadeType="..." targetName="..." targetType="..." />
```

1.6 Local variables

Operators declarations and definitions introduce identifiers for flows, signal and assertions. Some of these may not have a direct counterpart in the generated code, depending on options (mostly optimization level and activation of assertions and probes) and/or use of the “keep” pragma. Those that do have a counterpart in the generated code will be traced.

In the traceability file, any input with a counterpart shall be traced using the following syntax:

```
Input ::= <Input scadeName="..." scadeType="..." targetName="..." targetType="..."
  [[ mem="true" ] [ inCtx="true" ] [[ clockVar="..." clockVal="..." ] />
```

In the traceability file, any output with a counterpart shall be traced using the following syntax:

```
Output ::= <Output scadeName="..." scadeType="..." [ targetName="..." ] targetType="..."
  [[ inCtx="true" ] [[ clockVar="..." clockVal="..." ] [[ mem="true" ] />
```

Note: `targetName="..."` is optional because there is no named variable in the generated C code for imported functions returning a scalar.

In the traceability file, any local variable having a counterpart in the generated code shall be traced using the following syntax:

```
Local ::= <Local scadeName="..." scadeType="..." targetName="..." targetType="..."
  [ inCtx="true" ] [[ clockVar="..." clockVal="..." ] />
```

In the traceability file, any local memory shall be traced using the following syntax:

```
Memory ::= <Memory [ scadeName="..." scadeType="..." ] targetName="..." targetType="..."
  inCtx="true" [[ clockVar="..." clockVal="..." ] />
```

In the traceability file, any initialization variable shall be traced using the following syntax:

```
Init ::= <Init targetName="..." targetType="..."
  inCtx="true" [[ clockVar="..." clockVal="..." ] />
```

In the traceability file, any local memory coming from a **fby** whose delay is greater than or equal to 2 shall be traced using the following syntax:

```
Fby ::= <Fby [ scadeName="..." scadeType="..." ] targetName="..." targetType="..."
  inCtx="true" [[ clockVar="..." clockVal="..." ] />
```

Requirement KCG-485 applies to local memories coming from a **fby** whose delay is equal to 1.

In the traceability file, any local variable marked as **probe** that has a counterpart in the generated code shall be traced using the following syntax:

Probe ::= <Probe scadeName="..." scadeType="..." targetName="..." targetType="..."
 [[inCtx="true"]][clockVar="..." clockVal="..."]][mem="true"]/ >

Generated clock variables which are present in the traceability file but are not otherwise described shall be traced using the following syntax:

Clock ::= <Clock targetName="..." targetType="..."
 [[clockVar="..." clockVal="..."]][inCtx="true"]][mem="true"]/ >

In the traceability file, any signal having a counterpart in the generated code shall be traced using the following syntax:

Signal ::= <Signal scadeName="..." targetName="..."
 [[inCtx="true"]][clockVar="..." clockVal="..."]/ >

In the traceability file, any identifier of an **assume** assertion having a counterpart in the generated code shall be traced using the following syntax:

Assume ::= <Assume scadeName="..." targetName="..."
 [[inCtx="true"]][clockVar="..." clockVal="..."]/ >

In the traceability file, any identifier of a **guarantee** assertion having a counterpart in the generated code shall be traced using the following syntax:

Guarantee ::= <Guarantee scadeName="..." targetName="..."
 [[inCtx="true"]][clockVar="..." clockVal="..."]/ >

1.7 Local scopes

Inside a node, the control block constructions introduce different scope levels where local variables and inner scopes may be defined.

All local scope constructs having a counterpart in the generated code shall be traced.

Scope ::=
 { { Local | Init | Memory | Fby | Probe | Clock | Signal | Assume | Guarantee }
 { NodeInstance | NodeInlining | Automaton | ClockedBlock | Iterator } }

1.8 Node

The "forest" of nodes will contain, for each node, at least one XML element making possible to link it to the relevant target code structure. If the node is not expanded then its whole structure is detailed. If the node is expanded, its whole body will be inlined therefore is not described there. If a node is imported, the SCAD 6 input entity is linked to the relevant target code entity, including this target code entity signature.

In the traceability file, the root node shall be traced using the following syntax:

```

RootNode ::= <RootNode scadeName="..." headerFile="..."
            targetCycleFct="..." [[ targetInitFct="..." ]]>
            [[ <InCtxType targetName="..." />
              <OutCtxType targetName="..." /> ]
            [[ <MemCtxVar targetName="..." /> ]
            [[ <StateVectorType targetName="..." loadFct="..." saveFct="..." /> ]
            { Input | Output | Memory | Init | Fby }
            Scope
        < /RootNode>

```

There shall be one and only one RootNode element in the traceability file.

MemCtxVar gives the name of the global variable holding the context of the root node when option `-global_root_context` is set.

StateVectorType gives the name of the state vector type (if any) when option `-state_vector` is set, and the names of the associated functions for saving (resp. loading) a context to (resp. from) a state vector.

In the traceability file, any node shall be traced using the following syntax:

```

Node ::= <NoExpNode scadeName="..." headerFile="..." targetCycleFct="..."
        [[ targetInitFct="..." ] [[ imported="..." ]]>
        [[ <InCtxType targetName="..." />
          <OutCtxType targetName="..." />
          <StateVectorType targetName="..." loadFct="..." saveFct="..." /> ]
        { Input | Output | Memory | Init | Fby }
        Scope
    < /NoExpNode>
    | <ExpNode scadeName="..." />

```

A NoExpNode tag shall correspond to each different type instantiation of a non-expanded polymorph node.

1.9 Node Instance

There are two different ways of tracing a node instance according to the fact that the node is expanded or not. If a node is expanded, it shall be inlined. If a node is not expanded, the call shall be traced and the user shall refer to the node trace done within the "forest".

1.9.1 Unexpanded node

The XML element will give the context and the other inputs given as parameters to the node.

In the traceability file, any unexpanded node instance shall be traced using the following syntax:

```

NodeInstance ::= <NodeInstance scadeName="..." [[ refName="..." ] [[ instName="..." ]
                [[ clockVar="..." clockVal="..." ]>
                [[ <InCtxVar targetName="..." />
                  <OutCtxVar targetName="..." [[ clockVar="..." clockVal="..." ] /> ]
            < /NodeInstance>

```

The `instName` attribute shall contain the pragma instance name used in the input model (if any). The `refName` attribute shall refer to the actual cycle function of the node being instantiated. It allows to discriminate between different monomorph versions of a polymorph node.

Predefined operators are traced only when they have an occurrence pragma.

1.9.2 Expanded node

Any expanded node is inlined. It will therefore contain every single structure making the trace complete. If an expanded node is instanced several time, the XML element describing it will be repeated wherever it is needed.

In the traceability file, any expanded node shall be traced using the following syntax:

```
NodeInlining ::= <NodeInlining scadeName="..." [ instName="..." ]>
                { Input | Output }
                Scope
            < /NodeInlining>
```

1.10 Automaton

Any transition and state of an automaton in the SCADE 6 input model may have a corresponding structure in the target code.

In the traceability file, any automaton having a counterpart in the generated code shall be traced using the following syntax:

```
Automaton ::= <Automaton
                scadeName="..." targetStatesType="..." targetTransitionsType="..." >
                { State }
                [ <WeakTransition targetName="..." /> [ inCtx="true" ] ]
                [ <StrongTransition targetName="..." /> [ inCtx="true" ] ]
                [ <ActiveState targetName="..." /> [ inCtx="true" ] ]
                [ <SelectedState targetName="..." /> [ inCtx="true" ] ]
                [ <NextState targetName="..." /> [ inCtx="true" ] ]
                [ <ResetActiveState targetName="..." /> [ inCtx="true" ] ]
                [ <ResetSelectedState targetName="..." /> [ inCtx="true" ] ]
                [ <ResetNextState targetName="..." /> [ inCtx="true" ] ]
                { Clock }
            < /Automaton>

State ::= <State scadeName="..." targetName="...">
                Scope
                { Fork }
            < /State>

Fork ::= <Fork priority="...">
                [ <Condition > { Iterator | NodeInstance | NodeInlining } < /Condition > ]
                [ <Action > Scope < /Action > ]
                Fork | Transition
            < /Fork>
```

Transition ::= <**Transition** *scadeDest*="..." *targetName*="..." *kind*="...">

The *targetStatesType* and the *targetTransitionsType* attributes shall contain the name of the corresponding target code structure. The *ResetActiveState*, *ResetNextState* and *ResetSelectedState* variables are always boolean. The *kind* attribute shall either be “weak” or “strong”.

1.11 ClockedBlock

A clocked block in the SCADE 6 input model may have a corresponding structure in the target code.

Any clocked block having a counterpart in the generated code shall be traced.

ClockedBlock ::= *ActivateIf*
| *ActivateWhen*

1.11.1 ActivateIf

In the traceability file, any activate with an if block shall be traced using the following syntax:

ActivateIf ::= <**ActivateIf** [*scadeName*="..."]>
 IfBlock
 </**ActivateIf**>

IfBlock ::= [<**Condition** [*targetName*="..."] [*inCtx*="true"]>
 { *Iterator* | *NodeInstance* | *NodeInlining* }
 </**Condition**>]
 <**Then**> *Scope* | *IfBlock* </**Then**>
 <**Else**> *Scope* | *IfBlock* </**Else**>

The *targetName*="..." attribute in the condition tag shall be the generated variable – if any – for the implicit clock introduced by the activate ... if clocked block.

1.11.2 ActivateWhen

In the traceability file, any activate with an match block shall be traced using the following syntax:

ActivateWhen ::= <**ActivateWhen** [*scadeName*="..."] *targetWhenType*="...">
 [<**Condition** [*targetName*="..."] [*inCtx*="true"]>
 { *Iterator* | *NodeInstance* | *NodeInlining* }
 </**Condition**>]
 { <**Match** *scadeName*="..." *targetName*="...">
 Scope
 </**Match**> }
 </**ActivateWhen**>

The *targetName*="..." attribute in the condition tag shall be the generated variable – if any – for the implicit clock introduced by the activate ... when clocked block.

1.12 Iterator

In the traceability file, iterations on traced operators shall be traced using the following syntax:

```
Iterator ::= <Iterator iteratorKind="..." scadeSize="..." targetSize="...">  
            Iterator | NodeInstance | NodeInlining  
            </Iterator>
```

The value of attribute `iteratorKind` shall be one of the following: **map**, **fold**, **mapfold**, **mapi**, **foldi**, **foldw**, **foldwi**