



ENGINEERING NOTE

Title:	KCG XML storage format.
Doc ID:	KCG-EN-001
Doc Date:	July 6, 2011
Abstract:	This document is the specification of the XML storage format for KCG
Author(s):	PROJECT MANAGER Bruno Martin
Reviewer(s):	CORE team members
Approval:	PROJECT MANAGER Bruno Martin

1 Description of the SCADE 6 graphics storage format

One of the possible inputs of KCG is the SCADE 6 graphics storage format (the SCADE 6 graphics storage format) which is an XML format.

1.1 Introduction to XML

The reader will refer to Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 4th February 2004 from François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler (reference available at <http://www.w3.org/TR/REC-xml/>) to find a detailed description of the XML language. XML syntax is not recalled in this document but some of its general principles are listed below to improve comprehension of the current document.

What is XML?

- XML is a mark-up language much like HTML.
- XML was designed to describe data.
- XML tags are not predefined. You have to define your own tags and your own document structure.
- XML uses a Document Type Definition (DTD) or an XML Schema to describe the data.
- XML with a DTD or XML Schema is designed to be self-descriptive.

Some general principles of the XML syntax are recalled:

XML declaration: The XML declaration corresponds to the first line of the document and defines the XML version and the character encoding used in the document. `<?xml version="1.0" encoding="UTF-8"? >`. In this case the document conforms to the 1.0 specification of XML and uses the UTF-8 character set. The line following it describes the root element of the document.

XML root element: All XML documents must contain a single tag pair to define a root element. All other elements must be within this root element. All elements can have sub-elements (also called child elements). Sub-elements must be correctly nested within their parent element:

```
<root>
  <child>
    <subchild> ... </subchild>
  </child>
</root>
```

XML closing tags: All XML elements must be properly closed either like `<tag>Hello world</tag>` or like `<tag attribute="..." />`

XML elements nesting: All XML elements must be properly nested. Improper nesting of tags makes no sense in XML.

XML tags are case sensitive: XML tags are case sensitive. Opening and closing tags must therefore be written with the same case.

Well-formed attribute values: XML elements can have attributes in name/value pairs. The attribute value must always be quoted. Quotation marks inside an attribute value must always be escaped (see Escape characters below).

Comments: The syntax for writing comments in XML is the following: `<!--This is a comment -->`

Escape characters: Illegal XML characters have to be replaced by entity references. If you place a character `<` inside an XML element, the parser will treat it as a markup and will interpret it as the start of a new element. If this character has another meaning (example: `<tag> 2 < 3 < /tag>`) it will generate an error because the parser interprets it as the start of a new element. To avoid this, you have to replace the `<` character with its entity reference `<` (example: `<tag> 2<3 < /tag>`). There are 5 predefined entity references in XML:

<code>&lt;</code>	<code><</code>	less than
<code>&gt;</code>	<code>></code>	greater than
<code>&amp;</code>	<code>&</code>	ampersand
<code>&apos;</code>	<code>'</code>	apostrophe
<code>&quot;</code>	<code>"</code>	quotation mark

PCDATA PCDATA means parsed character data. PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded. It means that `&` and `<` have special meaning and must be escaped (by replacing them with their predefined entity references) if they are not the start of markup.

1.2 The SCADE 6 graphics storage format

Any XML element will be referred to as a SCADE 6 graphics storage format element. As PCDATA is not used anywhere else than in pragmas, each SCADE 6 graphics storage format element of the format is therefore an XML element containing a tag, a list of attributes and a list of sub-element (possibly ordered). Several files may be used to build the SCADE 6 output file as packages or operators may be defined in their own files but only one file shall be given to KCG as an input. This input file shall be a root file (see section 1.2.4).

1.2.1 Namespaces

Each element and attribute used in the SCADE 6 graphics storage format is declared in a namespace defined using the URI¹ described in this section where the namespace is identified by "xxx" and its version number "Version".

In an SCADE 6 graphics storage format file, any Namespace shall be defined using the following syntax:

```
DefaultNamespace ::= xmlns="URI"
Namespace        ::= xmlns:prefix="URI"
```

where *prefix* is the prefix associated to the namespace.

¹A Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource. The main purpose of this identification is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. URIs are defined in schemes defining a specific syntax and associated protocols.

URI ::= http://www.esterel-technologies.com/ns/SCADE/xxx/Version
Namespaces ::= [*DefaultNamespace*] { *Namespace* }

1.2.2 Comments

In an SCADE 6 graphics storage format file, any comment shall be defined using the XML syntax: `<!-- This is a comment -->`

1.2.3 Prolog

In an SCADE 6 graphics storage format file, any prolog shall be defined using the XML syntax: `<?This is a prolog?>`

1.2.4 Root element

Two kinds of SCADE 6 graphics storage format files are defined: root files and non-root files.

RootElement ::= *File* | *Package* | *Operator*

In an SCADE 6 graphics storage format file, any root file shall have a single *File* element as root element which shall be defined using the following syntax:

File ::= `<File Namespaces>`
`[<declarations>`
`{ Open | Package | Constant | Sensor | Type | Operator }+`
`</declarations>]`
`[Pragmas]`
`</File>`

In an SCADE 6 graphics storage format file, any non-root file shall have either a *Package* element (see section 1.2.5) or an *Operator* one (see section 1.2.13) as root element.

Note: A non-root file is linked to its "including" father file by the href attribute associated to the *Package* or *Operator* (see KCG-046 and KCG-070) element declared in the father file.

1.2.5 Packages

In an SCADE 6 graphics storage format file, any *package* shall be defined using the following syntax:

Package ::= `<Package [Namespaces] [visibility="..."] name="...">`
`[<declarations>`
`{ Open | Package | Constant | Sensor | Type | Operator }+`
`</declarations>]`
`[Pragmas]`
`</Package>`
`| <Package name="..." href="..."> [Pragmas] </Package>`

In an SCADE 6 graphics storage format file, the visibility attribute shall have **private** or **public** as value.

In an SCADE 6 graphics storage format file, the definition of *Package* using the href attribute shall be used when the package is defined in another file. The path to the file shall be either absolute or relative to the SCADE 6 graphics storage format file containing the reference.

When defined in another SCADE 6 graphics storage format file, the translation process shall check that the name attribute given before the href one and the name attribute used when the package is defined are the same.

1.2.6 Open path

In an SCADE 6 graphics storage format file, any *Open* shall be defined using the following syntax:

```
Open ::= <Open >
        <package > PackageRef </package>
        [ Pragmas ]
    </Open>
```

In an SCADE 6 graphics storage format file, any *PackageRef* shall be defined using the following syntax:

```
PackageRef ::= <PackageRef name="..." > [ Pragmas ] </PackageRef>
```

1.2.7 Type declarations

In an SCADE 6 graphics storage format file, any *Type* shall be defined using the following syntax:

```
Type ::= <Type [ visibility="..." ] [ external="..." ] name="..." >
        [ <definition > TypeExpression | Enum </definition> ]
        [ Pragmas ]
    </Type>
```

Note: The visibility parameter is the same as the one described section 1.2.5.

In a SCADE 6 graphics storage format file, the external attribute shall have **none** or **imported** as value.

In a SCADE 6 graphics storage format file, any *Enum* shall be defined using the following syntax:

```
Enum ::= <Enum >
        <values >
            { <Value name="..." >
                [ Pragmas ]
            </Value> }+ordered
        </values>
        [ Pragmas ]
    </Enum>
```

1.2.8 Type expressions

In a SCADE 6 graphics storage format file, any *TypeExpression* shall either be a **NamedType**, a **Struct** or a **Table**.

```

TypeExpression ::= NamedType
                  | Struct
                  | Table

```

In a SCADE 6 graphics storage format file, any *NamedType* shall be defined using the following syntax:

```

NamedType ::= <NamedType >
              <type > TypeRef </type>
              [ Pragmas ]
              </NamedType>

```

In a SCADE 6 graphics storage format file, any *TypeRef* shall be defined using the following syntax:

```

TypeRef ::= <TypeRef name="..."> [ Pragmas ] </TypeRef>

```

In a SCADE 6 graphics storage format file, any *Struct* shall be defined using the following syntax:

```

Struct ::= <Struct >
            <fields >
              { <Field name="...">
                <type > TypeExpression </type>
                [ Pragmas ]
              }+ordered
            </fields>
            [ Pragmas ]
            </Struct>

```

In a SCADE 6 graphics storage format file, any *Table* shall be defined using the following syntax:

```

Table ::= <Table >
            <type > TypeExpression </type>
            <size > Expression </size>
            [ Pragmas ]
            </Table>

```

1.2.9 Constants

In a SCADE 6 graphics storage format file, any *Constant* shall be defined using the following syntax:

```

Constant ::= <Constant [ visibility="..." ] [ external="..." ] name="...">
              <type > TypeExpression </type>
              [ <value > Expression </value> ]
              [ Pragmas ]
              </Constant>

```

Note: The visibility and the external parameters are the same as previously defined.

1.2.10 Sensors

In a SCADE 6 graphics storage format file, any *Sensor* shall be defined using the following syntax:

```
Sensor ::= <Sensor name="...">
           <type> TypeExpression </type>
           [[ Pragmas ]]
           </Sensor>
```

1.2.11 Clock expressions

In a SCADE 6 graphics storage format file, any *ClockExpression* shall be a *SimpleClockExpression*, a *ClockNotExpression* or a *ClockMatchExpression*.

```
ClockExpression ::= SimpleClockExpression
                    | ClockNotExpression
                    | ClockMatchExpression
```

In a SCADE 6 graphics storage format file, any *SimpleClockExpression* shall be defined using the following syntax:

```
SimpleClockExpression ::= <ClockExpression >
                           <variable> VariableRef </variable>
                           [[ Pragmas ]]
                           </ClockExpression>
```

In a SCADE 6 graphics storage format file, any *ClockNotExpression* shall be defined using the following syntax:

```
ClockNotExpression ::= <ClockNotExpression >
                        <variable> VariableRef </variable>
                        [[ Pragmas ]]
                        </ClockNotExpression>
```

In a SCADE 6 graphics storage format file, any *ClockMatchExpression* shall be defined using the following syntax:

```
ClockMatchExpression ::= <ClockMatchExpression >
                           <variable> VariableRef </variable>
                           <value> <Pattern pattern="..." /> </value>
                           [[ Pragmas ]]
                           </ClockMatchExpression>
```

In a SCADE 6 graphics storage format file, any *VariableRef* shall be defined using the following syntax:

```
VariableRef ::= <VariableRef name="..."> [[ Pragmas ]] </VariableRef>
```

1.2.12 Variable declarations

In a SCADA 6 graphics storage format file, any *Variable* shall be defined using the following syntax:

```

Variable ::= <Variable [ clock="..." ] [ probe="..." ] name="..." >
            <type> TypeExpression </type>
            [ <when> ClockExpression </when> ]
            [ <default> Expression </default> ]
            [ <last> Expression </last> ]
            [ Pragmas ]
            </Variable>

```

In a SCADA 6 graphics storage format file, the clock attribute shall have **true** or **false** as value, **false** being the default one.

In a SCADA 6 graphics storage format file, the probe attribute shall have **true** or **false** as value, **false** being the default one.

1.2.13 User defined operators

In a SCADA 6 graphics storage format file, any *Operator* shall be defined using the following syntax:

```

Operator ::= <Operator [ Namespaces ]
            kind="..." [ visibility="..." ] [ external="..." ] name="..." >
            [ <sizeParameters> { SizeParameter }ordered </sizeParameters> ]
            [ <inputs> { Variable }ordered </inputs> ]
            [ <outputs> { Variable }ordered </outputs> ]
            [ <numericTypes> { NumericType }ordered </numericTypes> ]
            [ <specializedOperator> OperatorRef </specializedOperator> ]
            DataDefinition
            [ Pragmas ]
            </Operator>
            | <Operator name="..." href="..."> [ Pragmas ] </Operator>

```

In a SCADA 6 graphics storage format file, the kind attribute of *Operator* shall have **function** or **node** as value.

The definition of *Operator* using the href attribute shall be used when the operator is defined in another SCADA 6 graphics storage format file. The path to the file shall be either absolute or relative to the SCADA 6 graphics storage format file containing the reference.

When defined in another SCADA 6 graphics storage format file, the translation process shall check that the name attribute given before the href one and the name attribute used when the operator is defined are the same.

In a SCADA 6 graphics storage format file, any *SizeParameter* shall be defined using the following syntax:

```

SizeParameter ::= <SizeParameter name="..."> [ Pragmas ] </SizeParameter>

```

In a SCADA 6 graphics storage format file, any *NumericType* shall be defined using the following syntax:

```

NumericType ::= <NumericType >
                <typeVar> TypeRef </typeVar>
                [ Pragmas ]
            </NumericType>

```

1.2.14 Scope declarations

In a SCADE 6 graphics storage format file, any *DataDefinition* shall be defined using the following syntax:

$$\begin{aligned}
 \textit{DataDefinition} ::= & \left[\langle \mathbf{signals} \rangle \{ \{ \textit{Signal} \}^+ \langle / \mathbf{signals} \rangle \right] \\
 & \left[\langle \mathbf{locals} \rangle \{ \{ \textit{Variable} \}^+ \langle / \mathbf{locals} \rangle \right] \\
 & \left[\langle \mathbf{data} \rangle \right. \\
 & \quad \left. \{ \{ \textit{Equation} \mid \textit{Assertion} \mid \textit{Emission} \mid \textit{StateMachine} \mid \textit{IfBlock} \mid \textit{WhenBlock} \}^+ \right. \\
 & \quad \left. \langle / \mathbf{data} \rangle \right]
 \end{aligned}$$

Note: Unlike the SCADE 6 textual format, the SCADE 6 graphics storage format does not allow having a single equation without a dedicated scope. The equivalent writing is a *DataDefinition* composed only of one **data** sub-element containing only one equation as one would not be able to discriminate the two rules in the following case: a single equation versus a single equation in a dedicated scope.

In a SCADE 6 graphics storage format file, any *Signal* shall be defined using the following syntax:

$$\textit{Signal} ::= \langle \mathbf{Signal\ name} = \text{""} \dots \text{""} \rangle \left[\textit{Pragmas} \right] \langle / \mathbf{Signal} \rangle$$

1.2.15 Equations

In a SCADE 6 graphics storage format file, any *Equation* shall be defined using the following syntax:

$$\begin{aligned}
 \textit{Equation} ::= & \langle \mathbf{Equation} \rangle \\
 & \left[\langle \mathbf{lefts} \rangle \{ \{ \textit{VariableRef} \}^+_{\textit{ordered}} \langle / \mathbf{lefts} \rangle \right] \\
 & \langle \mathbf{right} \rangle \textit{Expression} \langle / \mathbf{right} \rangle \\
 & \left[\textit{Pragmas} \right] \\
 & \langle / \mathbf{Equation} \rangle
 \end{aligned}$$

The case of an expression returning no output shall be represented, in an SCADE 6 graphics storage format file, with no **lefts** element.

In a SCADE 6 graphics storage format file, any *Assertion* shall be defined using the following syntax:

$$\begin{aligned}
 \textit{Assertion} ::= & \langle \mathbf{Assertion\ kind} = \text{""} \dots \text{""} \ \mathbf{name} = \text{""} \dots \text{""} \rangle \\
 & \langle \mathbf{definition} \rangle \textit{Expression} \langle / \mathbf{definition} \rangle \\
 & \left[\textit{Pragmas} \right] \\
 & \langle / \mathbf{Assertion} \rangle
 \end{aligned}$$

In a SCADE 6 graphics storage format file, the kind attribute of *Assertion* shall have **assume** or **guarantee** as value.

In a SCADE 6 graphics storage format file, any *Emission* shall be defined using the following syntax:

$$\begin{aligned}
 \textit{Emission} ::= & \langle \mathbf{Emission} \rangle \\
 & \langle \mathbf{signals} \rangle \{ \{ \textit{SignalRef} \}^+_{\textit{ordered}} \langle / \mathbf{signals} \rangle \\
 & \left[\langle \mathbf{condition} \rangle \textit{Expression} \langle / \mathbf{condition} \rangle \right] \\
 & \left[\textit{Pragmas} \right] \\
 & \langle / \mathbf{Emission} \rangle
 \end{aligned}$$

In a SCADE 6 graphics storage format file, any *SignalRef* shall be defined using the following syntax:

$$\textit{SignalRef} ::= \langle \mathbf{SignalRef} \text{ name}=\text{""} \rangle \llbracket \textit{Pragmas} \rrbracket \langle / \mathbf{SignalRef} \rangle$$

1.2.16 Clocked blocks

In a SCADE 6 graphics storage format file, any *IfBlock* shall be defined using the following syntax:

$$\begin{aligned} \textit{IfBlock} ::= & \langle \mathbf{IfBlock} \llbracket \text{ name}=\text{""} \rrbracket \llbracket \text{ exhaustiveReturn}=\text{""} \rrbracket \rangle \\ & \langle \mathbf{block} \rangle \textit{IfNode} \langle / \mathbf{block} \rangle \\ & \llbracket \langle \mathbf{returns} \rangle \{ \{ \text{VariableRef} \}^+ \langle / \mathbf{returns} \rangle \rrbracket \\ & \llbracket \textit{Pragmas} \rrbracket \\ & \langle / \mathbf{IfBlock} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, any *IfNode* shall be defined using the following syntax:

$$\begin{aligned} \textit{IfNode} ::= & \langle \mathbf{IfNode} \rangle \\ & \langle \mathbf{if} \rangle \textit{Expression} \langle / \mathbf{if} \rangle \\ & \langle \mathbf{then} \rangle \textit{IfAction} \mid \textit{IfNode} \langle / \mathbf{then} \rangle \\ & \langle \mathbf{else} \rangle \textit{IfAction} \mid \textit{IfNode} \langle / \mathbf{else} \rangle \\ & \llbracket \textit{Pragmas} \rrbracket \\ & \langle / \mathbf{IfNode} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, any *IfAction* shall be defined using the following syntax:

$$\begin{aligned} \textit{IfAction} ::= & \langle \mathbf{Action} \rangle \\ & \textit{DataDefinition} \\ & \llbracket \textit{Pragmas} \rrbracket \\ & \langle / \mathbf{Action} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, any *WhenBlock* shall be defined using the following syntax:

$$\begin{aligned} \textit{WhenBlock} ::= & \langle \mathbf{WhenBlock} \llbracket \text{ name}=\text{""} \rrbracket \llbracket \text{ exhaustiveReturn}=\text{""} \rrbracket \rangle \\ & \langle \mathbf{when} \rangle \textit{Expression} \langle / \mathbf{when} \rangle \\ & \langle \mathbf{matches} \rangle \{ \{ \text{Match} \}_{\textit{ordered}}^+ \langle / \mathbf{matches} \rangle \\ & \llbracket \langle \mathbf{returns} \rangle \{ \{ \text{VariableRef} \}^+ \langle / \mathbf{returns} \rangle \rrbracket \\ & \llbracket \textit{Pragmas} \rrbracket \\ & \langle / \mathbf{WhenBlock} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, any *Match* shall be defined using the following syntax:

$$\begin{aligned} \textit{Match} ::= & \langle \mathbf{Match} \text{ pattern}=\text{""} \rangle \\ & \textit{DataDefinition} \\ & \llbracket \textit{Pragmas} \rrbracket \\ & \langle / \mathbf{Match} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, the *exhaustiveReturn* attribute shall have **true** or **false** as value, **false** being the default one.

The *returns* sub-element shall be compulsory if the *exhaustiveReturn* attribute has **true** as value, .

1.2.17 State machines

In a SCADE 6 graphics storage format file, any *StateMachine* shall be defined using the following syntax:

```

StateMachine ::= <StateMachine [ name="..." ] [ exhaustiveReturn="..." ]>
                [ <states> { State }+ </states> ]
                [ <returns> { VariableRef }+ </returns> ]
                [ Pragmas ]
                </StateMachine>

```

In a SCADE 6 graphics storage format file, any *State* shall be defined using the following syntax:

```

State ::= <State [ initial="..." ] [ final="..." ] name="...">
          [ <unless> { Transition }+ordered </unless> ]
          DataDefinition
          [ <until> { Transition }+ordered [ Synchro ] </until> ]
          [ Pragmas ]
          </State>

```

In a SCADE 6 graphics storage format file, the initial attribute shall have **true** or **false** as value, **false** being the default one.

In a SCADE 6 graphics storage format file, the final attribute shall have **true** or **false** as value, **false** being the default one.

1.2.18 Transitions

In a SCADE 6 graphics storage format file, any *Transition* shall be defined using the following syntax:

```

Transition ::= <Transition [ kind="..." ]>
                [ <target> StateRef </target> ]
                <condition> Expression </condition>
                [ <effect> Action </effect> ]
                [ <forks> { ForkBranch }+ordered [ DefaultForkBranch ] </forks> ]
                [ Pragmas ]
                </Transition>

```

In a SCADE 6 graphics storage format file, any *StateRef* shall be defined using the following syntax:

```

StateRef ::= <StateRef name="..."> [ Pragmas ] </StateRef>

```

In a SCADE 6 graphics storage format file, any *Synchro* shall be defined using the following syntax:

```

Synchro ::= <Synchro [ kind="..." ]>
            [ <target> StateRef </target> ]
            [ <effect> Action </effect> ]
            [ <forks> { ForkBranch }+ordered [ DefaultForkBranch ] </forks> ]
            [ Pragmas ]
            </Synchro>

```

In a SCADE 6 graphics storage format file, any *ForkBranch* shall be defined using the following syntax:

```

ForkBranch ::= <ForkBranch [ kind="..." ]>
                [ <target > StateRef </target> ]
                <condition > Expression </condition>
                [ <effect > Action </effect> ]
                [ <forks > { { ForkBranch }+ordered [ DefaultForkBranch ] </forks> ]
                [ Pragmas ]
                </ForkBranch>

```

In a SCADE 6 graphics storage format file, any *DefaultForkBranch* shall be defined using the following syntax:

```

DefaultForkBranch ::= <DefaultForkBranch [ kind="..." ]>
                        [ <target > StateRef </target> ]
                        [ <effect > Action </effect> ]
                        [ <forks > { { ForkBranch }+ordered [ DefaultForkBranch ] </forks> ]
                        [ Pragmas ]
                        </DefaultForkBranch>

```

In a SCADE 6 graphics storage format file, the kind attribute of *Synchro* shall have **restart** or **resume** as value.

In a SCADE 6 graphics storage format file, the kind attribute of *Transition*, *Synchro*, *ForkBranch* and *DefaultForkBranch* shall be exclusive with forks sub-element.

In a SCADE 6 graphics storage format file, the target sub-element of *Transition*, *Synchro*, *ForkBranch* and *DefaultForkBranch* shall be exclusive with forks sub-element.

1.2.19 Actions

In a SCADE 6 graphics storage format file, any *Action* shall be defined using the following syntax:

```

Action ::= <Action >
            DataDefinition
            [ Pragmas ]
            </Action>

```

Note: Unlike the SCADE 6 format, the SCADE 6 graphics storage format does not allow having signal emissions without a dedicated scope. The equivalent writing is a *DataDefinition* composed only of one **data** sub-element containing only signal emissions as one would not be able to discriminate the two rules in the following case: signal emissions versus signal emissions in a dedicated scope.

1.2.20 Expressions

Note: The name or identifier of the called operator is stored either in a sub-element or in an operator attribute (not in the name attribute).

In a SCADE 6 graphics storage format file, any *Expression* shall be defined using the following syntax:

```

Expression ::= <IdExpression >
    <path > ConstVarRef </path>
    [ Pragmas ]
</IdExpression>
| <Last >
    <variable > ConstVarRef </variable>
    [ Pragmas ]
</Last>
| <Present >
    <signal > SignalRef </signal>
    [ Pragmas ]
</Present>
| <ConstValue value="..." >
    [ Pragmas ]
</ConstValue>
| ListExpression
| <UnaryOp [ name="..." ] operator="..." >
    <operand > Expression </operand>
    [ Pragmas ]
</UnaryOp>
| <BinaryOp [ name="..." ] operator="..." >
    <operands > (( Expression Expression ))ordered </operands>
    [ Pragmas ]
</BinaryOp>
| <NAryOp [ name="..." ] operator="..." >
    <operands > (( Expression { Expression }+ ))ordered </operands>
    [ <pragmas > { OtherPragma } </pragmas> ]
</NAryOp>
| <SharpOp [ name="..." ] >
    [ <flows > Expressions </flows> ]
    [ Pragmas ]
</SharpOp>
| <PreOp [ name="..." ] >
    <flow > Expression </flow>
    [ Pragmas ]
</PreOp>
| <InitOp [ name="..." ] >
    <value > Expression </value>
    <flow > Expression </flow>
    [ Pragmas ]
</InitOp>
| <WhenOp [ name="..." ] >

```

```

    <flow > Expression </flow>
    <when > ClockExpression </when>
    [ Pragma ]
</WhenOp>
| <FbyOp [ name="..." ]>
    <flows > Expressions </flows>
    <delay > Expression </delay>
    <values > Expressions </values>
    [ Pragma ]
</FbyOp>
| <MergeOp [ name="..." ]>
    <clock > Expression </clock>
    <flows > (( ListExpression { ListExpression }+ ))ordered </flows>
    [ Pragma ]
</MergeOp>
| <PrjOp [ name="..." ]>
    <flow > Expression </flow>
    <with > { LabelOrIndex }+ordered </with>
    [ <pragmas > { OtherPragma } </pragmas > ]
</PrjOp>
| <SliceOp [ name="..." ]>
    <array > Expression </array>
    <fromIndex > Expression </fromIndex>
    <toIndex > Expression </toIndex>
    [ Pragma ]
</SliceOp>
| <PrjDynOp [ name="..." ]>
    <array > Expression </array>
    <indexes > { Expression }+ordered </indexes>
    <default > Expression </default>
    [ Pragma ]
</PrjDynOp>
| <TransposeOp [ name="..." ]>
    <array > Expression </array>
    <dimensions > (( Dimension Dimension ))ordered </dimensions>
    [ Pragma ]
</TransposeOp>
| <DataStructOp [ name="..." ]>
    <data >
        { <LabelledExpression label="...">
            <flow > Expression </flow>
            [ Pragma ]
        }
    </data >

```

```

        < /LabelledExpression> }+ordered
    < /data>
    [ Pragmas ]
< /DataStructOp>
| <ScalarToVectorOp [ name="..." ]>
    <flow > Expression < /flow>
    <size > Expression < /size>
    [ Pragmas ]
< /ScalarToVectorOp>
| <DataArrayOp [ name="..." ]>
    <data > Expressions < /data>
    [ Pragmas ]
< /DataArrayOp>
| <ChgIthOp [ name="..." ]>
    <flow > Expression < /flow>
    <with > { LabelOrIndex }+ordered < /with>
    <value > Expression < /value>
    [ Pragmas ]
< /ChgIthOp>
| <IfThenElseOp [ name="..." ]>
    <if > Expression < /if>
    <then > Expression < /then>
    <else > Expression < /else>
    [ Pragmas ]
< /IfThenElseOp>
| <CaseOp [ name="..." ]>
    <switch > Expression < /switch>
    <cases >
        { <Case [ pattern="..." ]>
            <flow > Expression < /flow>
            [ Pragmas ]
            < /Case> }+ordered
        < /cases>
    [ Pragmas ]
< /CaseOp>
| <CallExpression >
    <operator > ExprOp < /operator>
    [ <callParameters > Expressions < /callParameters> ]
    [ Pragmas ]
< /CallExpression>

```

In a SCADE 6 graphics storage format file, the operator attribute of UnaryOp shall have +, -, **not**, **reverse**, **int** or **real** as value.

In a SCADE 6 graphics storage format file, the operator attribute of BinaryOp shall have -, /, **mod**, **div**, =, **<>**; (use of escaped characters, stands for <>), **<**; (stands for <), **>**; (stands for >), **<=** (stand for <=), **>=** (stands for >=) or **times** as value.

In a SCADE 6 graphics storage format file, the operator attribute of NArYOp shall have **+**, *****, **and**, **or**, **xor** or **@** as value.

Note1: The NArYOp element was created due to the SCADE 6 graphical editor where **+**, *****, **and**, **or**, **xor** and **@** are considered (and are graphically represented) as n-ary operators. As it does not exist in the SCADE 6 language, it is considered, for example, that $a + b + c$ is equivalent to $(a + b) + c$.

Note2: The NArYOp and the PrjOp elements cannot hold any *KCGpragma* as there is no equivalent operator in the SCADE 6 language.

In a SCADE 6 graphics storage format file, any *Dimension* shall be defined using the following syntax:

$$Dimension ::= \langle \mathbf{ConstValue\ value="..."} \rangle \langle \mathbf{/ConstValue} \rangle$$

In a SCADE 6 graphics storage format file, any *ConstVarRef* shall be defined using the following syntax:

$$ConstVarRef ::= \langle \mathbf{ConstVarRef\ name="..."} \rangle \llbracket Pragma \rrbracket \langle \mathbf{/ConstVarRef} \rangle$$

In a SCADE 6 graphics storage format file, any *ListExpression* shall be defined using the following syntax:

$$ListExpression ::= \langle \mathbf{ListExpression} \rangle \\ \llbracket \langle \mathbf{items} \rangle Expressions \langle \mathbf{/items} \rangle \rrbracket \\ \llbracket Pragma \rrbracket \\ \langle \mathbf{/ListExpression} \rangle$$

In a SCADE 6 graphics storage format file, any *Expressions* shall be a list of *Expression*:

$$Expressions ::= \{ \{ Expression \}_{ordered}^+$$

In a SCADE 6 graphics storage format file, any *LabelOrIndex* shall be defined using the following syntax:

$$LabelOrIndex ::= \langle \mathbf{Label\ name="..."} \rangle \llbracket Pragma \rrbracket \langle \mathbf{/Label} \rangle \\ | Expression$$

In a SCADE 6 graphics storage format file, any *ExprOp* shall be defined using the following syntax:

$$ExprOp ::= \langle \mathbf{MakeOp} \llbracket \mathbf{name="..."} \rrbracket \rangle \\ \langle \mathbf{type} \rangle TypeRef \langle \mathbf{/type} \rangle \\ \llbracket Pragma \rrbracket \\ \langle \mathbf{/MakeOp} \rangle \\ | \langle \mathbf{FlattenOp} \llbracket \mathbf{name="..."} \rrbracket \rangle \\ \langle \mathbf{type} \rangle TypeRef \langle \mathbf{/type} \rangle \\ \llbracket Pragma \rrbracket \\ \langle \mathbf{/FlattenOp} \rangle \\ | \langle \mathbf{IteratorOp\ iterator="..."} \rangle \\ \langle \mathbf{operator} \rangle ExprOp \langle \mathbf{/operator} \rangle \\ \langle \mathbf{size} \rangle Expression \langle \mathbf{/size} \rangle$$

```

    [[ Pragmas ]]
  </IteratorOp>
| <PartialIteratorOp iterator="...">
  <operator > ExprOp </operator>
  <size > Expression </size>
  <if > Expression </if>
  [[ <default > Expression </default> ]]
  [[ Pragmas ]]
</PartialIteratorOp>
| <ActivateOp >
  <operator > ExprOp </operator>
  <every > Expression </every>
  <default > Expression </default>
  [[ Pragmas ]]
</ActivateOp>
| <ActivateNoInitOp >
  <operator > ExprOp </operator>
  <every > Expression </every>
  <default > Expression </default>
  [[ Pragmas ]]
</ActivateNoInitOp>
| <ActivateClockOp >
  <operator > ExprOp </operator>
  <every > ClockExpression </every>
  [[ Pragmas ]]
</ActivateClockOp>
| <RestartOp >
  <operator > ExprOp </operator>
  <every > Expression </every>
  [[ Pragmas ]]
</RestartOp>
| <OpCall [[ name="..." ]]>
  <operator > OperatorRef </operator>
  [[ <instanceParameters > Expressions </instanceParameters> ]]
  [[ Pragmas ]]
</OpCall>

```

Note: In **PartialIteratorOp**, the **default** sub-element shall be present if and only if the value of the **iterator** attribute is either 'mapw' or 'mapwi'.

In a SCAD 6 graphics storage format file, any *OperatorRef* shall be defined using the following syntax:

$$OperatorRef ::= <OperatorRef name="..."> [[*Pragmas*]] </OperatorRef>$$

In a SCAD 6 graphics storage format file, the iterator attribute of *IteratorOp* shall have **map**, **fold**, **mapfold**, **mapi** or **foldi** as value.

In a SCAD 6 graphics storage format file, the iterator attribute of *PartialIteratorOp* shall have **mapw**, **mapwi**, **foldw** or **foldwi** as value.

The default attribute shall be used for **mapw** and **mapwi** partial iterators only.

1.2.21 Pragmas

In a SCADE 6 graphics storage format file, any *Pragmas* shall be defined using the following syntax:

$$\begin{aligned} \textit{Pragmas} ::= & \langle \mathbf{pragmas} \rangle \\ & \{ \{ \textit{KCGpragma} \mid \textit{OtherPragma} \} \\ & \langle / \mathbf{pragmas} \rangle \end{aligned}$$

In a SCADE 6 graphics storage format file, any *KCGpragma* shall be defined using the following syntax:

$$\begin{aligned} \textit{KCGpragma} ::= & \langle \mathbf{kcg:Pragma} \rangle \\ & \dots \\ & \langle / \mathbf{kcg:Pragma} \rangle \end{aligned}$$

The ... shall be replaced by the content of the pragma.

In a SCADE 6 graphics storage format file, any *OtherPragma* shall be defined using the following syntax:

$$\begin{aligned} \textit{OtherPragma} ::= & \langle \mathbf{xxx:yyy} \dots \rangle \\ & \dots \\ & \langle / \mathbf{xxx:yyy} \rangle \end{aligned}$$

where **xxx** represents the prefix of the tag associated to the tool that recognises and defines the pragma. A tool can define one (or more) tags and can recognise other tags. Pragmas syntax (contents: attributes, CDATA, PCDATA or XML sub-tree) will be defined by the owner tool.