SCL Extensions

Deprecated since 0.12

This article is deprecated. The described features are no longer available in current releases.

There are several extensions implemented to ease the work with a scl model. (de.cau.cs.kieler.scl.extensions)

SCL Factory Extensions

You need several Factories to handle all aspects of Yakindu (SGraph + SText) and the extended SCChart models. The SCL Factory Extension provides you with shortcuts for all factories.

Shortcut	Factory	Description
SGraph()	SGraphFactory::eINSTANCE	Factory for yakindu statechart models
SText()	StextFactory::eINSTANCE	Factory for yakindu expressions
SyncGraph()	SyncgraphFactory::eINSTANCE	Factory for sgraph extensions
SyncText()	SynctextFactory::eINSTANCE	Factory for stext extensions
SCCExp()	SCChartsExpFactory::eINSTANCE	Factory for SCCharts specific expression extensions
SCL()	SclFactory::eINSTANCE	Factory for SCL

SCL Create Extensions

coming soon...

SCL Naming Extensions

SCL Naming Extensions provide helper functions for ID & naming services.

Method	Description
def void distributeStateIDs (Statechart)	Since yakindu does not make use of the ID field, one can use this method to make every ID in a statechart unique.
def String getHierarchicalName (SyncState, String)	Generates a (most likely unique) name for a state. The name is generated from all parent states and regions, which are separated by an underscore. If a region or state has no name, the element's hash code is used instead.

SCL Ordering Extensions

The Ordering Extensions provide functions, which can be used in the *xtend sort* context.

Method	Description
def **int** compareSCLRegionStateOrder(SyncState, SyncState)	Sorts states according to their type. Initial states come first, final states last.

SCL Statement Extensions

Method	Description
def boolean isEmpty(Statement)	Returns true, if the statement is an EmptyStatement.
def boolean hasInstruction(Statement)	Returns true, if the statement is an InstructionStatement containing an instruction.
def boolean isGoto(Statement)	Returns true, if the statement is an InstructionStatement containing a goto instruction.
def EmptyStatement asEmptyStatement(Statement)	Convenently type-cast the statement to an EmptyStatement
def InstructionStatement asInstructionStatement (Statement)	Convenently type-cast the statement to an InstructionStatement
def getInstruction(Statement)	Type-cast the statement to an InstructionStatement an return its instruction.
def EmptyStatement removeInstruction(Statement)	Creates a new EmptyStatement and copies the label and comment information from the old statement.
def getStatement(Instruction)	Returns the parent statement of a given instruction.

SCL Thread Extensions

The Thread Extensions provide functions to ease the handling of SCL threads and statements in the context of SCL threads.

Method	Description
def AbstractThread getThread(Instruction) def AbstractThread getThread(Statement)	Returns the SCL thread of a SCL statement or SCL instruction.
def AbstractThread getMainThread(Instruction) def AbstractThread getMainThread(Statement)	Returns the main thread of a SCL program.
def Statement[] getControlFlow(Instruction) def Statement[] getControlFlow(Statement)	Returns the control flow of an instruction/statement up to the parent of this control flow. In contrast to a thread this also includes control flows of conditional instructions.
def **boolean** isInSameThreadAs(Instruction, Instruction) def **boolean** isInSameThreadAs(Statement, Statement)	Returns true, if both instructions/statements are in the same thread.
def **boolean** isInMainThread(Instruction) def **boolean** isInMainThread(Statement)	Returns true, if the instruction/statement runs in the main thread.
def **boolean** isInThread(Instruction, AbstractThread) def **boolean** isInThread(Statement, AbstractThread)	Returns true, if the instruction/statement runs in the given thread.
def **boolean** contains(AbstractThread, Instruction) def **boolean** contains(AbstractThread, Statements)	Returns true, if the thread contains the given instruction/statement.
def dropPrevious(AbstractThread, Statement) def dropPrevious(List<Statement>, Statement)	Drops all preceding statements in a thread or a list of statements before the given statement.
def Statement getPreviousStatement(Statement)	Returns the preceding statement.
def Statement getPreviousStatementHierarchical (Statement)	Returns the preceding statement or the parent statement of that control flow, if no preceding statement is present.

def InstructionStatement getPreviousInstructionStatement(Statement)	Returns the preceding instruction statement. Empty statements are ignored.
def InstructionStatement getPreviousInstructionStatementHierarchical (Statement)	Returns the preceding instruction statement or the parent statement of that control flow, if no preceding statement is present.

SCL Goto Extensions

To help with the handling of the goto statement and its target instruction, use the SCL Goto Extensions.

Method	Description
def Statement getTargetStatement (Goto) def Statement getTargetStatement (Goto, AbstractThread)	Returns the target statement of a goto instruction (in the context of the (given) thread).
def boolean targetExists(Goto) def boolean targetExists(Goto, AbstractThread)	Returns true, if the target of a goto instruction exists (in the context of the (given) thread).
def InstructionStatement getInstructionStatement(Statement)	Returns the first valid InstructionStatement in a thread after the given statement. **May return null, if no further InstructionStatement exists.** To get a valid instruction form a goto jump, one can write "goto.getTargetStatement?. getInstructionStatement?.instruction". The result value will be the instruction or null.
def getIncomingGotos(Statement)	Returns a list of all gotos that target the given statement. If you want to retrieve the count of incoming goto jumps, use "getIncomingGotos.size".

SCL Expression Extensions

The SCL Expression Extension holds methods to help with the manipulation and constructions of the SText (and extended) expressions.

Method	Description
def Expression toExpression (RegularEventSpec)	Transforms a SGraph RegularEventSpec to a SText Expression. The resulting expression will be an ElementReferenceExpression.
def Expression negate(Expression)	Negates the given expression. If the Expression is an ElementReferenceExpression the result will be a LogicalNotExpression containing the expression. Otherwise the result will be a LogicalNotExpression containing a ParenthesizedExpression, which then holds the original expression.
def String correctSerialization (String)	Since the actual implementation of the SText parser may parse artefacts (like linebreaks) until matching a preceding delimeter, correctSerialization removes these artifacts, when serializing an expression.

SCL Dependency Extensions

coming soon...

SCL Basic Block Extensions

The Basic Block Extensions retrieve information about basic blocks in the SCL model. A basic block can be identified by any statement in the block. Usually the first statement in the block is used.

Method	Description
def ArrayList<Statement> getBasicBlock (Statement) def ArrayList<Statement> getBasicBlock (Statement, List<Statement>)	Retrieves all statements of the basic block in which the given statement is located.

def Statement getBasicBlockFirst(Statement) def Statement getBasicBlockFirst (List<Statement>)	Returns the first statement of a basic block.
def boolean isInBasicBlock(Statement, Statement) def boolean isInBasicBlock(Statement, List<Statement>)	Returns true, if the caller statement is contained in the given basic block.
def String getBasicBlockID(Statement) def String getBasicBlockID(List<Statement>)	Returns an unique ID for the given basic block. To create this ID, the hash code of the root statement is used.
def ArrayList<Statement> getBasicBlockRoots (Statement)	Returns a list of all basic block root statements in the contorl flow of the calling statement.
def ArrayList<Statement> getAllBasicBlockRoots (Statement)	Returns a list of all basic block root statements in the program that contains the calling statement.
def int getBasicBlockIndex(Statement)	Returns the index of the given basic block.
def ArrayList<Statement> getBasicBlockPredecessors(Statement)	Returns a list of basic block root statements, that identify the predecessor basic blocks of the basic block identified by the given statement.

SCL Basic Block Extensions Code Examples

```
for (predecessor : basicBlockData.BasicBlockRootStatement.getBasicBlockPredecessors) {
        goLabelText = goLabelText + 'P' + predecessor.getBasicBlockIndex + "\n"
}
```