SCXML FrequentlyAskedQuestions

Commons SCXML Frequently Asked Questions (Commons-SCXML FAQ)

What is SCXML?

State Chart XML (SCXML) is a general-purpose event-based state machine language that can be used in many ways.

Do you have a simple example where Commons SCXML is used?

Sure, take a look at the stopwatch usecase.

What is a Context? And what is an Evaluator?

The SCXML specification allows an implementation to support multiple expressions languages. These expressions become part of attribute values for executable content, such as:

```
<var name="foo" expr="1 + 2 + bar" />
```

or are used to evaluate the boolean guard conditions that decide whether or not a particular transition is followed once its associated trigger event is received, such as:

```
<transition event="day.close" cond="day eq 'Friday'" target="weekend" />
```

In order to support multiple expression languages, Commons SCXML uses two interfaces which serve as adapters to the APIs for the particular expression language used for a particular SCXML document.

- The *Context* is a collection of variables that defines a variable "scope". Each <state> element within an SCXML document gets its own *Context* or variable scope.
- The *Evaluator* is a component with the capability of parsing and evaluating expressions. It is the "expression language engine".

Commons SCXML currently provide implementations for JEXL and JSP 2.0 EL.

Which expression languages does the Commons SCXML implementation support?

- JEXL
- JSP 2.0 EL

How do I try out the sample SCXML documents?

The SCXML distribution provides utility classes that offer mock command line environments allowing users to try out samples. The core dependencies for Commons SCXML are Commons Digester (which introduces a transitive dependency on Commons BeanUtils, at the least) and Commons Logging. View the dependencies page for the recommended version numbers. It may be possible to use lower version numbers for the Commons dependencies.

An environment specific expression language is used in SCXML documents. Commons SCXML currently supports the use of JEXL or JSP 2.0 EL in SCXML documents.

-- Using JEXL in SCXML documents --

The JEXL Standalone class anticipates expressions in JEXL and hence requires commons-jexl.jar.

So that amounts to (use the correct local paths and filenames to the `jar` files and the SCXML document, without the line breaks):

```
java -classpath

commons-digester-1.7.jar;commons-beanutils-1.7.0.jar;
commons-logging-1.0.4.jar;commons-scxml-1.0-SNAPSHOT.jar;
commons-jexl-1.0.jar

org.apache.commons.scxml.test.StandaloneJexlExpressions

microwave01.xml
```

-- Using JSP 2.0 EL in SCXML documents --

The JSP Standalone class anticipates expressions in the JSP 2.0 Expression Language, and hence requires commons-el.jar and jsp-api.jar

And that amounts to (use the correct local paths and filenames to the `jar` files and the SCXML document, without the line breaks):

```
java -classpath

commons-digester-1.7.jar;commons-beanutils-1.7.0.jar;
commons-logging-1.0.4.jar;commons-scxml-1.0-SNAPSHOT.jar;
commons-el-1.0.jar;jsp-api-2.0.jar

org.apache.commons.scxml.test.StandaloneElExpressions

microwave01.xml
```

You could set up something more elegant (a script, an ant task etc.), but that is what it boils down to. If the document is a well-formed SCXML document, you will be able to type ? or `help` at the console and you can follow the directions thereafter (to simulate events, set variable values, reset the state machine or quit).

A few examples are available as part of the Commons SCXML test suite (look in env packages as well). Enjoy, and feedback is always welcome.

Can I use more than one expression language in the same SCXML document?

No, the expressions throughout the document must be homogeneous. This also applies to any external documents that may be refered by this document, for example via `src` attributes, like so:

```
<state id="foo" src="foo.xml">
  <!-- Something, possibly very interesting, here -->
</state>
```

Here, `foo.xml` must use the same expression language as the document above that hosts the state `foo`.

Once I set up an SCXMLExecutor (call the constructor, set the state machine) is there anything I have to do to "activate" it?

Yes, you must call the marker method, `SCXMLExecutor#go()`. This serves as an indication that you have finished configuring the `SCXMLExecutor` instance and are now ready to begin executing the state machine described by your SCXML document. For example, you may attach zero, one or many `SCXMLListener}}s to interesting "nodes" within the SCXML document, such as the document root i.e. the {{SCXML` object, and/or particular `State` and `Transition` objects as well.

How do I enable / control the logging within the Commons SCXML package?

Commons SCXML uses Commons Logging. See the Commons Logging Website for more details.

Can multiple threads safely interact with an instance of SCXMLExecutor?

No. You have to worry about synchronizing access if you need to. From an implementation perspective, Commons SCXML does not assume that synchronization will **always** be necessary. Not all usecases for Commons SCXML require synchronization.

Can I have multiple instances of SCXMLExecutor all working off of a single instance of the SCXML class?

Yes. The Commons SCXML object model does not store any information related to a particular execution of the state machine. It is therefore possible to use a single `SCXML` instance as the state machine for multiple `SCXMLExecutor` instances.

This also means that a SCXML document needs to be parsed only once, irrespective of the number of "instances" of the state machine that may execute.

Can I use SCXML in a non-voice application?

Ofcourse, as mentioned in the overview of the specification itself.

Here are two example from our usecases that have nothing to do with voice applications:

- SCXML Stopwatch example
- SCXML in Shale dialogs

What are the core requirements of SCXML? Do I need to include the JSP and Servlet API or the Faces libraries? Do I need to include Commons JEXL or Commons EL?

The "core" requirements for Commons SCXML are Commons Digester and Commons Logging. Commons Digester, at the minimum (if you use Commons Digester 1.7+), has a runtime dependency on Commons BeanUtils.

You do **not** need to include JSP or Servlet or Faces libraries. These are meant to come in via the servlet container environment and the corresponding classes in the Commons SCXML codebase which have those dependencies are meant to be used only in JSP/Servlet/Faces environments.

In addition, you will need to choose an expression language for your SCXML documents. The recommended expression language for the Commons SCXML implementation is JEXL. Using JEXL for expressions introduces a dependency of Commons JEXL. For usecases in JSP-based environments, EL will be a prefered choice over JEXL, and if you choose to use JSP 2.0 EL, that introduces a dependency of Commons EL and a runtime dependency on the JSP API (again, these should come in via the servlet container).

See the dependencies page on the Commons SCXML website for details about the dependency versions.

Does Commons SCXML implement the entire Working Draft? Are there any parts of the SCXML draft that are not covered by Commons SCXML?

No. Speaking in terms of the July 2005 W3C SCXML Working Draft the items that are not covered are:

- Multiple (simultaneous) targets for a single transition (Section 3.3.1)
- JOIN (Section 4.3)
- SYNCH (Section 4.4)

Is there an XML Schema associated with the SCXML Working Draft?

Yes, a schema has been published, see appendix D of this W3C SCXML Working Draft.

Is the implementation currently Serializable, meaning that can an SCXMLExecutor be serialized, passed across a wire, be deserialized, and then continue processing events where it left off? If so, are there any limitations around listeners?

No, for the Commons SCXML 0.5 release, this cannot be done.

However, this has been implemented in trunk, so with the next release of Commons SCXML the SCXMLExecutor instances will be Serializable. Listeners and other associated user-supplied bits are expected to be Serializable.