

JavaFX Codeeditors



BestSolution

Tom Schindl <tom.schindl@bestsolution.at>

Twitter: @tomsontom

Blog: <http://tomsondev.bestsolution.at>

Website: <http://www.bestsolution.at>

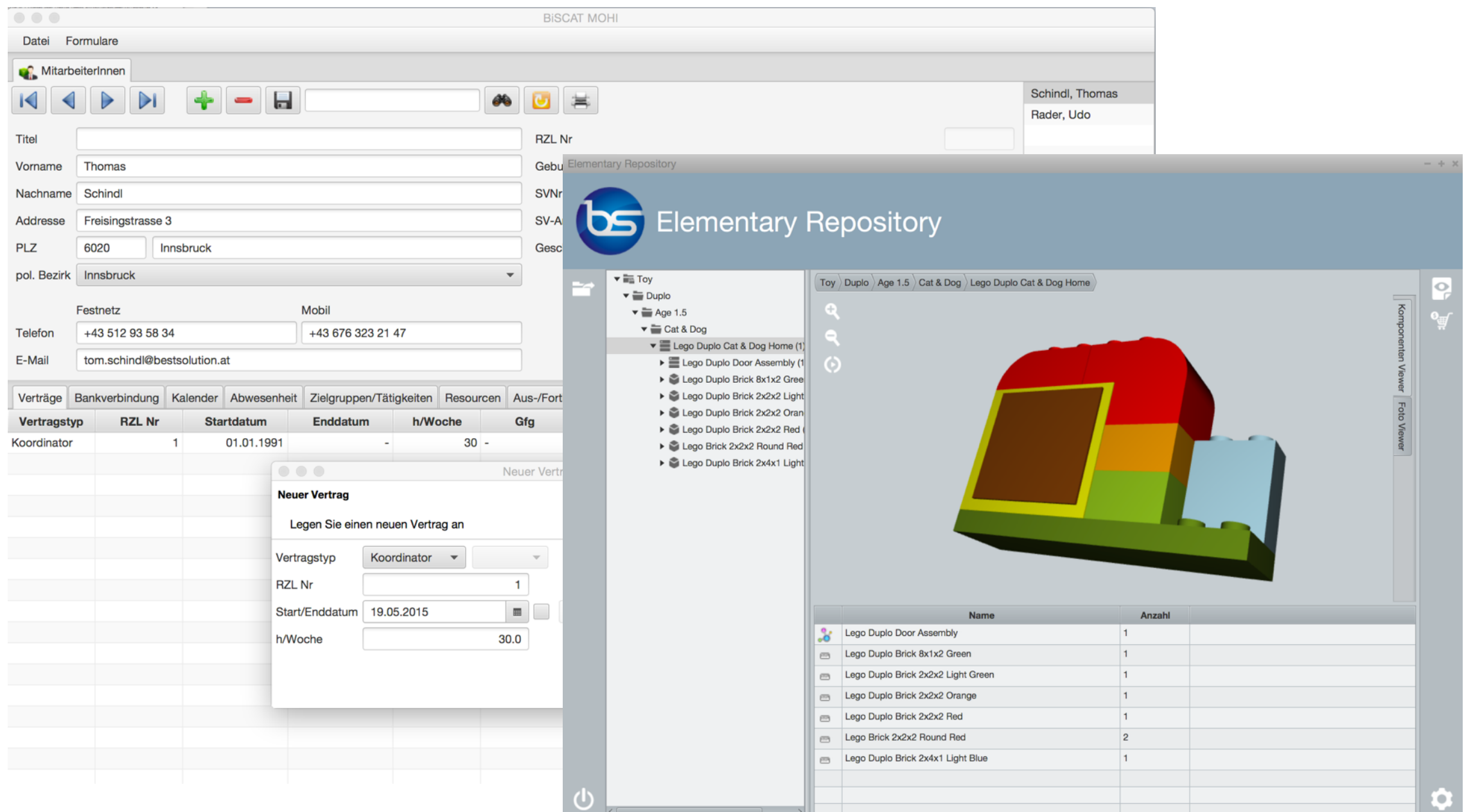
About Me

- ▶ CT0 BestSolution.at Systemhaus GmbH
- ▶ Eclipse Committer
 - ▶ e4
 - ▶ Platform
 - ▶ EMF
- ▶ Project lead
 - ▶ e(fx)clipse
- ▶ Twitter: @tomson tom
- ▶ Blog: tomsondev.bestsolution.at
- ▶ Cooperate: <http://bestsolution.at>



JavaFX

- Used for business UIs and simulation software



Demo

JavaFX in Toolspace

▸ Problems

- Misses basic controls like a StyledText-Widget to implement code editors
- A pure UI-Toolkit so no support for file-explorers, ...
- Docking framework, ...

▸ Solutions

- e(fx)clipse & Eclipse Core to provide those

e(fx)clipse Compensator



- ▶ Mission 0: Must look slick!
- ▶ Mission 1: Create a simple source editor like Notepad++ who:
 - ▶ Is process light-weight
 - ▶ Makes it easy to add new language highlightings
- ▶ Mission 2: Allow the simple source editor to expand to a (simple) IDE:
 - ▶ where Source-Editor, VCS (git), Ticketsystem (eg. github), CI (eg. travis) are core components fully integrated with each other
 - ▶ Easy to integrate: Does not depend on core.resources

Demo

Built your own

- Compensator is NOT extensible but all components can be reused to build your own editor/IDE
- Syntax Highlighting with Eclipse Text

Eclipse Text Part 1

► Step 1 Partitioning

```
/*  
 * This is a multiline comment  
 */
```

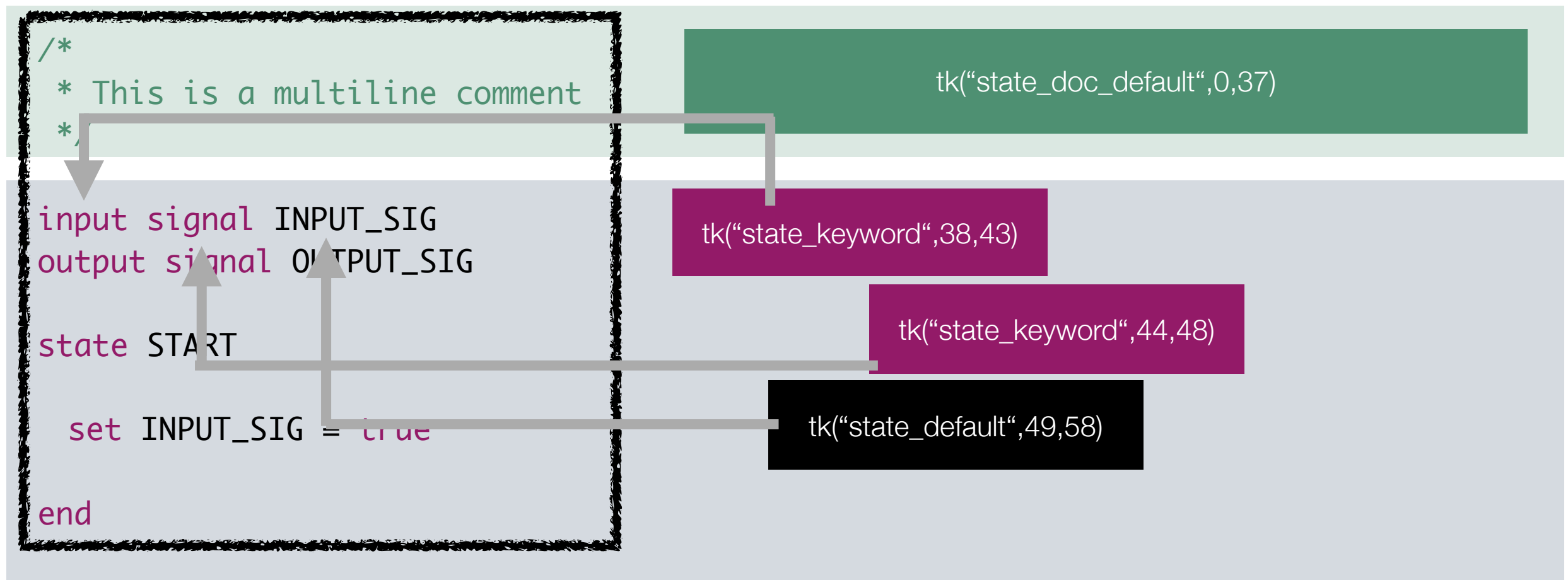
__state_comment

```
input signal INPUT_SIG  
output signal OUTPUT_SIG  
  
state START  
    set INPUT_SIG = true  
  
end
```

__dftl_partitioning

Eclipse Text Part 2

► Step 2 Tokenizing



efxclipse highlighting

```
.state.styled-text-area .state_default {  
    -styled-text-color: rgb(0,0,0);  
}  
  
.state.styled-text-area .state_keyword {  
    -styled-text-color: rgb(127, 0, 85);  
    -fx-font-weight: bold;  
}  
  
.state.styled-text-area .state_doc_default {  
    -styled-text-color: rgb(63, 127, 95);  
}
```

Livecode

Partitioning & Tokenizing



- ▶ No Java code required but a special DSL
 - ▶ Advantage 1: You don't need to learn Eclipse Text API
 - ▶ Advantage 2: You can add new languages without the need to install a new OSGi bundle

Partitioning & Tokenizing

```
dart {
  partition __dftl_partition_content_type
  // ...
  partition __dart_string
  rule-damager __dftl_partition_content_type {
    default token dart_default
    token dart_operator
    token dart_keyword
    // ...
    keywords dart_keyword [ "break", "case", "catch", "class", "const", "continue", "default"
      , "do", "else", "enum", "extends", "false", "final", "finally", "for"
      , "if", "in", "is", "new", "null", "rethrow", "return", "super"
      , "switch", "this", "throw", "true", "try", "var", "void", "while"
      , "with" ]

    character-rule dart_operator [
      ';' , '.' , '=' , '/' , '\\' , '+' , '-' , '*' , '<' , '>' , ':' , '?' , '!' , ',' , '|' , '&' , '^' , '%' , '~' ]
    // ...
  }
  // ...
  rule-damager __dart_string {
    default token dart_string
    token dart_string_inter
    single_line dart_string_inter '${' => '}'
  }
  rule_partitioner {
    single_line __dart_string ''' => '''
    single_line __dart_string """ => """
    // ...
  }
} for "text/dart"
```

Coloring

```
.dart.styled-text-area .dart_default {
    -styled-text-color: rgb(0, 0, 0);
}

.dart.styled-text-area .dart_operator {
    -styled-text-color: rgb(0, 0, 0);
}

.dart.styled-text-area .dart_keyword {
    -styled-text-color: rgb(127, 0, 85);
    -fx-font-weight: bold;
}

/* ... */

.dart.styled-text-area .dart_string {
    -styled-text-color: rgb(42, 0, 255);
}

.dart.styled-text-area .dart_string_inter {
    -styled-text-color: rgb(42, 0, 255);
    -fx-font-weight: bold;
}

.dart-element-name {
}

.dart-type-info {
    -fx-fill: lightgray;
}
```

Xtext

FX & efx & Xtext

- ▶ Make use of the NEW „Generic IDE support“ who generates 2 projects who don't require OSGi nor Eclipse Framework
 - ▶ ...\$language: core parsing infrastructure
 - ▶ ...\$language.ide: ide specific infrastructure including a special parser & lexer
- ▶ ...\$language.fx (Handcrafted): FX-Text support like SourceViewerConfiguration, Partitioner, ...

FX & efx & Xtext

► Step 1: Setup Guice module for editing

```
class StatemachineFXModule extends AbstractGenericModule {  
  
    val ExecutorService executorService  
  
    def configureExecutorService(Binder binder) {  
        binder.bind(ExecutorService).toInstance(executorService)  
    }  
  
    def configureContentAssistLexer(Binder binder) {  
        binder  
            .bind(Lexer).annotatedWith(Names.named(LexerIdeBindings.CONTENT_ASSIST))  
            .to(InternalStateMachineLexer)  
    }  
  
    def Class<? extends IContentAssistParser> bindIContentAssistParser() {  
        StatemachineParser  
    }  
}
```

FX & efx & Xtext

► Step 2: Create Guice Injector

```
injector = new StatemachineStandaloneSetup() {  
  
    public Injector createInjector() {  
        StatemachineRuntimeModule runtimeModule = new StatemachineRuntimeModule();  
  
        StatemachineFXModule fxModule = new StatemachineFXModule(  
            Executors.newFixedThreadPool(3));  
  
        return Guice.createInjector((Module)runtimeModule, webModule);  
    }  
}.createInjectorAndDoEMFRegistration();
```

FX & efx & Xtext

► Step 3: Create a content assist provider

```
@Singleton
public class ContentAssistProvider {
    @Inject
    Provider<ContentAssistContextFactory> contextFactoryProvider;
    @Inject
    Provider<XtextResourceSet> resourceSetProvider;
    @Inject
    ExecutorService pool;

    public List<ICompletionProposal> doContentAssist(String content, String uri, Integer offset) {
        XtextResource resource = getResource(uri);

        ContentAssistContextFactory contextFactory = contextFactoryProvider.get();
        contextFactory.setPool(pool);
        ContentAssistContext[] contexts = contextFactory.create(content, new TextRegion(0, 0), offset, resource);

        List<ICompletionProposal> proposal = new ArrayList<>();
        for (int i = 0; i < contexts.length; i++) {
            for (AbstractElement abstractElement : contexts[i].getFirstSetGrammarElements()) {
                createProposals(contexts[i], abstractElement, offset, proposal);
            }
        }

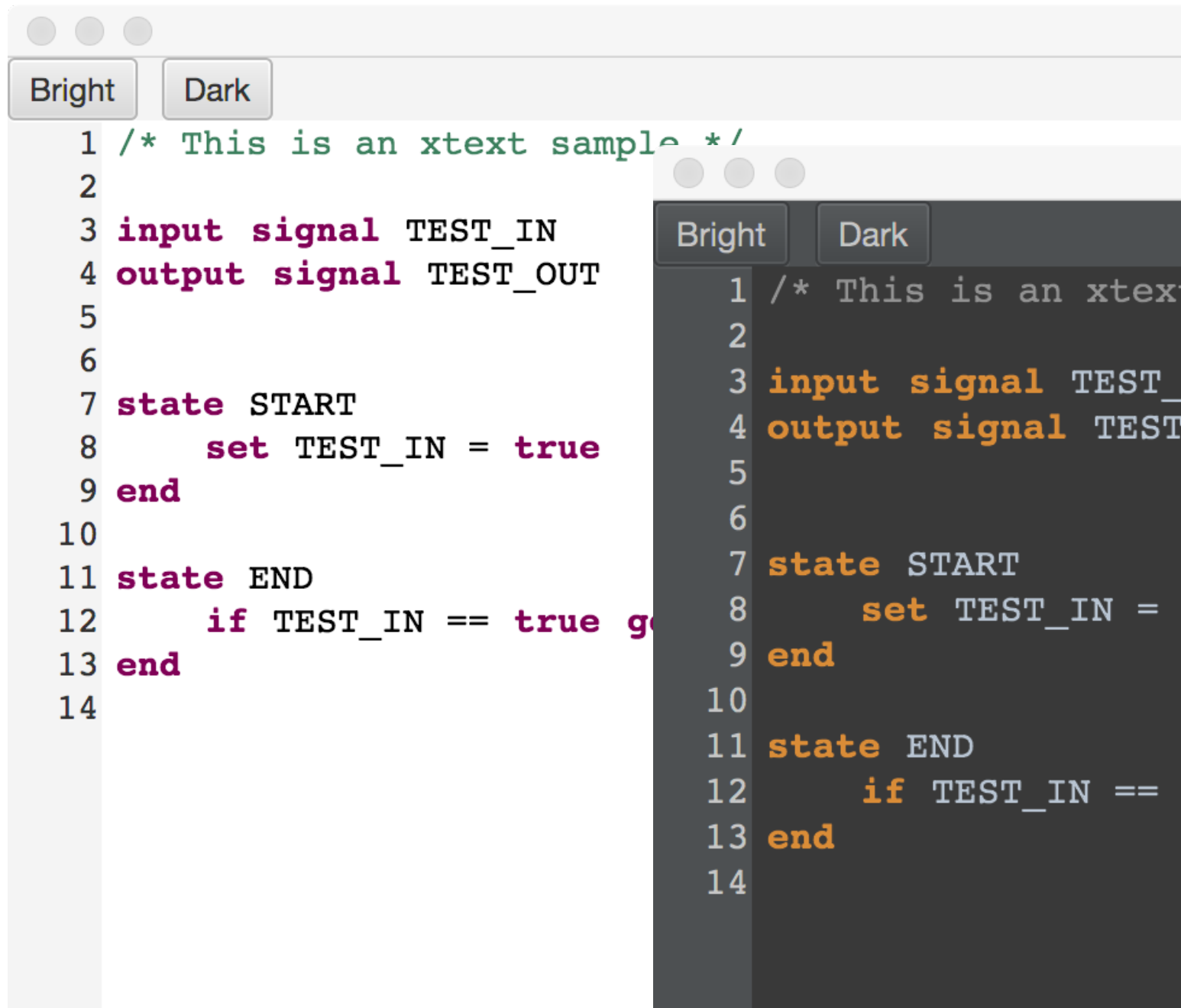
        return proposal;
    }
}
```

FX & efx & Xtext

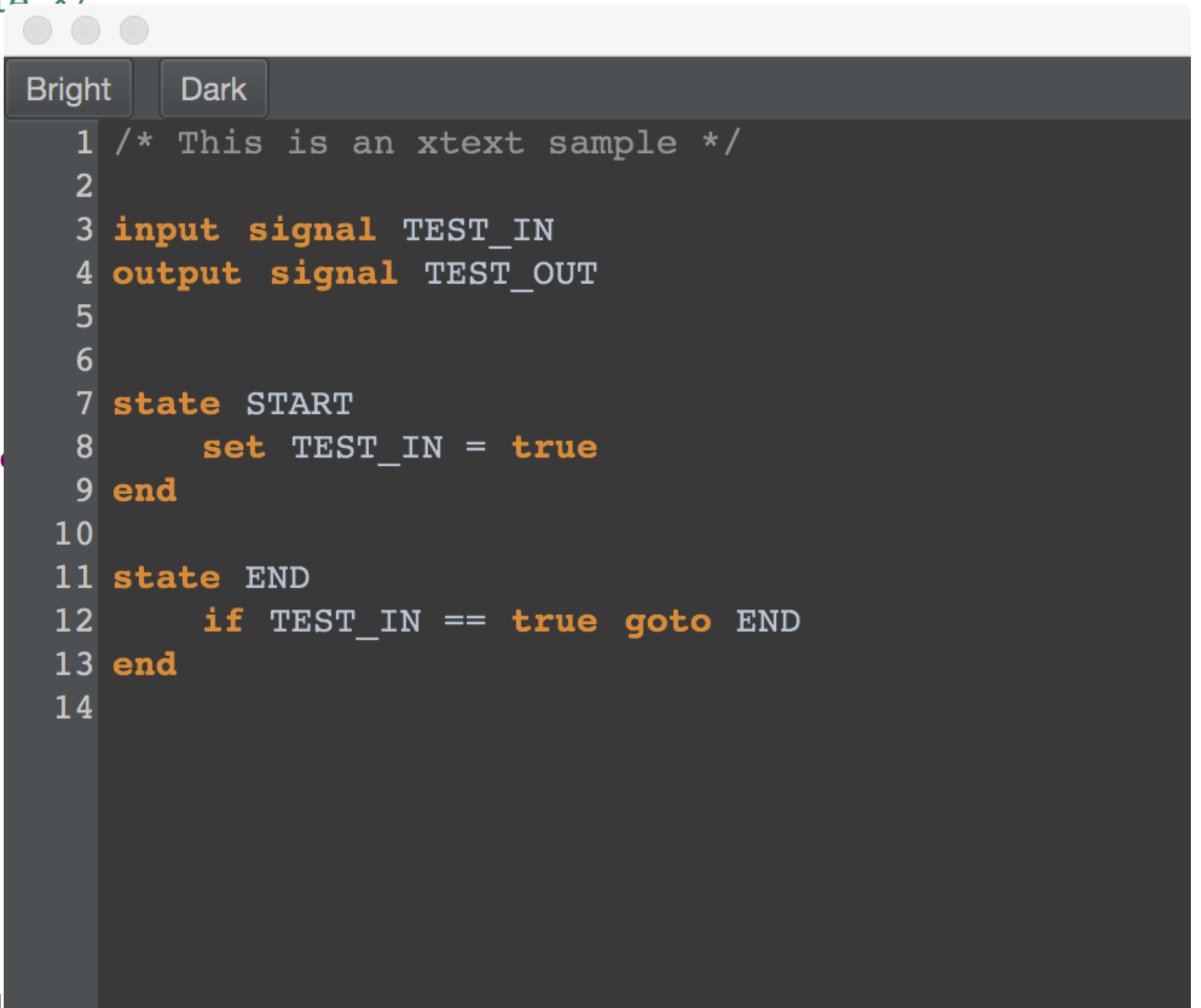
► Step 4: Setup Content assist in SourceViewerConfiguration

```
public class StatemachineSourceConfiguration extends SourceViewerConfiguration {  
    public StatemachineSourceConfiguration(Document doc, File f) {  
        contentAssistProvider = injector.getInstance(ContentAssistProvider.class);  
    }  
  
    @Override  
    public IContentAssistant getContentAssist() {  
        return new ContentAssistant(this::computeProposals);  
    }  
  
    private List<ICompletionProposal> computeProposals(Integer offset) {  
        return contentAssistProvider.doContentAssist(doc.get(),  
            f.toURI().toString(), offset);  
    }  
}
```

FX & efx & Xtext



```
1 /* This is an xtext sample */
2
3 input signal TEST_IN
4 output signal TEST_OUT
5
6
7 state START
8     set TEST_IN = true
9 end
10
11 state END
12     if TEST_IN == true goto END
13 end
14
```



```
1 /* This is an xtext sample */
2
3 input signal TEST_IN
4 output signal TEST_OUT
5
6
7 state START
8     set TEST_IN = true
9 end
10
11 state END
12     if TEST_IN == true goto END
13 end
14
```