Source code

Source code

Some files are omitted due to space constraints:

- the build system
- element targeting annotations which have no parameters in addition to the standard 4 element targeting parameters and the `inverse` parameter.
- a few utility classes of minor importance.

and package imports are omitted too. The copyright is listed on the last page.

Each system test lists the servlet, the input html file and finally the html file with the expected output.

Appendix C

runtime.annotation

C.1 ACollection

```
package dk.vitality.util.jat.runtime.annotation.attribute.collection;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface ACollection {
    String[] value() default {};
    String[] id() default {};
    String[] className() default {};
    String[] tag() default {};
    String[] styles(); // name of the attributes which the callback generates
}
```

C.2 SCollection

```
package dk.vitality.util.jat.runtime.annotation.attribute.sub;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface SCollection {
    String[] value() default {};
    String[] id() default {};
    String[] className() default {};
    String[] tag() default {};
    String[] styles(); // css name of style(s)
}
```

C.3 SHasClass

```
package dk.vitality.util.jat.runtime.annotation.attribute.sub;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface SHasClass {
    String[] value() default {};
    String[] id() default {};
    String[] className() default {};
    String[] tag() default {};
    String[] classes(); // name collision with className, but this one is required
    boolean inverse() default false;
}
```

C.4 SValue

```
package dk.vitality.util.jat.runtime.annotation.attribute.sub;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface SValue {
    String[] value() default {};
    String[] id() default {};
    String[] className() default {};
    String[] tag() default {};
    String[] styles(); // css name of style(s)
}
```

C.5 EIterate

```
package dk.vitality.util.jat.runtime.annotation.flowControl;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface EIterate {
    String[] value() default {};

    String[] id() default {};

    String[] className() default {};

    String[] tag() default {};

    Class<?> scope() default NotSpecified.class;
}
```

C.6 EScopingWhile

```
package dk.vitality.util.jat.runtime.annotation.flowControl;

@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface EScopingWhile {
    String[] value() default {};

    String[] id() default {};

    String[] className() default {};

    String[] tag() default {};

    boolean inverse() default false;

    Class<?> scope() default NotSpecified.class;
}
```

C.7 NotSpecified

```
package dk.vitality.util.jat.runtime.annotation.global;

@SuppressWarnings({"ClassMayBeInterface"})
public final class NotSpecified {
}
```

C.8 TCompile

```
package dk.vitality.util.jat.runtime.annotation.global;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface TCompile {
    String filename() default "";

    String fragment() default ""; // empty == <html>

    String tpname() default "";
}
```

C.9 TIgnore

```
package dk.vitality.util.jat.runtime.annotation.global;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface TIgnore {
}
```

C.10 TOverrides

```
package dk.vitality.util.jat.runtime.annotation.global;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface TOverrides {
}
```

Appendix D

runtime.lib

D.1 Opcode

```
package dk.vitality.util.jat.runtime.lib;

public enum Opcode {
    PRINT_STRINGTABLE,
    PRINT_STRING_VAR,
    PRINT_HTML_ENCODED_STRING_VAR,
    PRINT_ATTR_ENCODED_STRING_VAR,
    PRINT_INT_VAR,
    CALLBACK,
    GOTO, CONDITIONAL_GOTO,
    NEGATE,
    RETURN;

    public final static int OPCODE_BITS = 4;
    public final static int OPCODE_MASK = (1 << OPCODE_BITS) - 1;

    public int encoded() {
        return ordinal();
    }

    public int encoded(int param) {
        return ordinal() | (param << OPCODE_BITS);
    }

    public static Opcode decodeOpcode(int encoded) {
        return opcodes[encoded & OPCODE_MASK];
    }

    private static Opcode[] opcodes = values();

    public static int decodeParam(int encoded) {
        return encoded >>> OPCODE_BITS;
    }

    public static String toString(int encoded) {
        Opcode opcode = Opcode.decodeOpcode(encoded);
        switch (opcode) {
            case PRINT_STRINGTABLE:
            case CALLBACK:
            case GOTO:
            case CONDITIONAL_GOTO:
                return opcode.name().toLowerCase() + "(" + decodeParam(encoded) + ")";
            default:
                return opcode.name().toLowerCase();
        }
    }
}
```

D.2 TemplateBase

```
package dk.vitality.util.jat.runtime.lib;

abstract public class TemplateBase {
    public String stringValue;
    public int intValue;
    public boolean boolValue;

    public final String javaSourceMD5;

    protected TemplateBase(String javaSourceMD5) {
        this.javaSourceMD5 = javaSourceMD5;
    }

    private static final boolean debugEmit = false;

    public void emit(Writer w) throws Exception {
        emit(new HtmlPrintWriter(w));
    }

    public void emitToStdout() throws Exception {
        HtmlPrintWriter pw = new HtmlPrintWriter(new OutputStreamWriter(System.out));
        emit(pw);
        pw.flush();
    }

    public void emitToStderr() throws Exception {
```

```
        HtmlPrintWriter pw = new HtmlPrintWriter(new OutputStreamWriter(System.err));
        emit(pw);
        pw.flush();
    }

    public void emit(HtmlPrintWriter pw) throws Exception {
        if (debugEmit)
            System.err.println("Emits template: " + getClass().getSimpleName());
        TemplateData data = getData();
        if (!data.sourceMD5.equals(javaSourceMD5)) {
            throw new TemplateJavaMD5Exception("Mismatch between javasource MD5 = " +
javaSourceMD5
                    + " and TemplateData MD5 = " + data.sourceMD5);
        }
        final int[] opcodes = data.opcodes;
        int pc = 0;
        while (true) {
            int encoded = opcodes[pc];
            Opcode opcode = Opcode.decodeOpcode(encoded);
            int param = Opcode.decodeParam(encoded);
            if (debugEmit)
                System.err.println("TmplOP @" + pc + ": " + Opcode.toString(encoded));
            pc++;
            switch (opcode) {
                case PRINT_STRINGTABLE:
                    pw.print(data.strings[param]);
                    continue;

                case PRINT_STRING_VAR:
                    pw.print(stringValue);
                    continue;

                case PRINT_HTML_ENCODED_STRING_VAR:
                    pw.htmlEncode(stringValue);
                    continue;

                case PRINT_ATTR_ENCODED_STRING_VAR:
                    pw.attrEncode(stringValue);
                    continue;

                case PRINT_INT_VAR:
                    pw.print(intValue);
                    continue;

                case CALLBACK:
                    executeCallback(param);
                    continue;

                case GOTO:
                    pc = param;
                    continue;

                case CONDITIONAL_GOTO:
                    if (boolValue)
                        pc = param;
                    continue;

                case NEGATE:
                    boolValue ^= true;
                    continue;

                case RETURN:
                    return;

                default:
                    throw new TemplateException("Invalid opcode = " + encoded + " at addr = " +
pc);
            }
        }
    }

    protected void executeCallback(int idx) throws Exception {
        throw new TemplateException(
"Template has been disabled by changing the generated source");
    }

    public TemplateData getData() throws Exception {
        return TemplateDataLoader.loader.load(getClass());
    }
}
```

D.3 TemplateData

```
package dk.vitality.util.jat.runtime.lib;

public class TemplateData {
    public final String[] strings;
    public final int[] opcodes;
    public final String sourceMD5;

    public final static String headerMagic = "TmplData";
    public final static String trailerMagic = "({end})";
    public final static int version = 2;

    public TemplateData(String[] strings, int[] opcodes, String sourceMD5) {
        this.strings = strings;
        this.opcodes = opcodes;
        this.sourceMD5 = sourceMD5;
    }

    public TemplateData(InputStream is) throws IOException {
        DataInputStream dis = new DataInputStream(is);
        String s = dis.readUTF();
        if (!headerMagic.equals(s))
            throw new IOException("Invalid magic - not a template data file: '" + s + "'");
        int v = dis.readShort();
        if (v != version)
            throw new IOException("Invalid version, expected " + version + " but is " + v);

        sourceMD5 = dis.readUTF();

        opcodes = new int[dis.readInt()];
        for (int i = 0; i < opcodes.length; i++)
            opcodes[i] = dis.readInt();

        strings = new String[dis.readInt()];
        for (int i = 0; i < strings.length; i++)
            strings[i] = dis.readUTF();

        s = dis.readUTF();
```

```
        if (!trailerMagic.equals(s))
            throw new IOException("TemplateData failed reading trailer marker.");
    }

    public byte[] toBytes(String fullClassName) throws IOException {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream os = new DataOutputStream(baos);
        try {
            os.writeUTF(headerMagic);
            os.writeShort(version);

            os.writeUTF(sourceMD5);

            os.writeInt(opcodes.length);
            for (int i : opcodes)
                os.writeInt(i);

            os.writeInt(strings.length);
            for (String s : strings)
                os.writeUTF(s);

            os.writeUTF(trailerMagic);
        }
        finally {
            os.close();
        }
        return baos.toByteArray();
    }
}
```

D.4 TemplateDataLoader

```
package dk.vitality.util.jat.runtime.lib;

abstract public class TemplateDataLoader {
    public static TemplateDataLoader loader = new UsingClassResource();

    abstract public TemplateData load(Class<? extends TemplateBase> cls) throws Exception;

    public static String getRelativePath(Class<? extends TemplateBase> cls) {
        return cls.getName().replace('.', '/') + ".tmpldata";
    }

    public static class UsingClassResource extends TemplateDataLoader {
        protected static Map<Class<?>, TemplateData> cache = new HashMap<Class<?>,
TemplateData>();

        public synchronized TemplateData load(Class<? extends TemplateBase> cls) throws
Exception {
            TemplateData td = cache.get(cls);
            if (td != null)
                return td;
            String path = "/" + getRelativePath(cls);
            InputStream is = getClass().getResourceAsStream(path);
            if (is == null)
                throw new TemplateException("TemplateData not found for " + path);
            if (!(is instanceof BufferedInputStream))
                is = new BufferedInputStream(is, 4096);
            try {
                td = new TemplateData(new DataInputStream(is));
            }
            finally {
                is.close();
            }
            cache.put(cls, td);
            return td;
        }
    }

    public static class UsingFilesystem extends TemplateDataLoader {
        public final File baseDirF;

        public UsingFilesystem(File baseDirF) {
            this.baseDirF = baseDirF;
        }

        protected static class MetaData {
            final TemplateData data;
            final long moddate;
            final File f;

            MetaData(TemplateData data, long moddate, File f) {
                this.data = data;
                this.moddate = moddate;
                this.f = f;
            }
        }

        protected static Map<Class<?>, MetaData> cache = new HashMap<Class<?>,
MetaData>();

        public synchronized TemplateData load(Class<? extends TemplateBase> cls) throws
Exception {
            MetaData meta = cache.get(cls);
            if (meta != null && meta.moddate == meta.f.lastModified())
                return meta.data;
            File f = new File(baseDirF, getRelativePath(cls));
            if (!f.exists())
                throw new TemplateException("TemplateData not found for " + cls.getName());
            long moddate = f.lastModified();
            FileInputStream fis = new FileInputStream(f);
            try {
                TemplateData td = new TemplateData(new BufferedInputStream(fis, 8196));
                meta = new MetaData(td, moddate, f);
                cache.put(cls, meta);
                return td;
            }
            finally {
                fis.close();
            }
        }
    }
}
```

Appendix E

translator.compiler

E.1 AnnotationCompiler

```
package dk.vitality.util.jat.translator.compiler;

public class AnnotationCompiler {
    public final CompilerOptions options;
    public final PersistentCompilationInfo pci;
    public final FileInfo javaFI;
    public final Class<?> loadedJavaClass;
    public final CompilerLog log;
    public final CompilationInfo compilationInfo;

    public TemplateLocation htmlLocation;
    public HtmlDocument doc;
    public Node rootE;
    public boolean includeDoctype;
    public ScopeImpl rootScope, annotationScope;
    public ProgramData programData;
    public String fullClassName;
    public TemplateData templateData;

    public AnnotationCompiler(CompilerOptions options, PersistentCompilationInfo pci,
    FileInfo javaFI, Class<?> loadedJavaClass,
    CompilerLog log, CompilationInfo compilationInfo) {
        this.options = options;
        this.pci = pci;
        this.javaFI = javaFI;
        this.loadedJavaClass = loadedJavaClass;
        this.log = log;
        this.compilationInfo = compilationInfo;
    }

    public void compile(ProgressInfo pi) throws CompilerException {
        findHtmlTemplate();
        if (htmlLocation == null) {
            pi.numErrors++;
            return;
        }
        loadHtmlTemplate();
        if (doc == null) {
            pi.numErrors++;
            return;
        }
        postProcessHtmlTemplate(doc);
        findRootElement(doc);
        if (rootE == null) {
            pi.numErrors++;
            return;
        }
        log.log(LogLevel.Verbose, "Root: " + rootE.getNodeName());
        pci.sourceFiles.add(javaFI);
        pci.sourceFiles.add(doc.fileInfo);

        makeScopes();
        emitTree();
        emitPostCompilationWarnings();
        if (log.hasError()) {
            pi.numErrors++;
            return;
        }
        emitProgram(pi);
        File templateDataF = new File(options.generatedTemplateDataBaseDirF,
fullClassName.replace('.', '/') + ".tmpldata");
        if (options.saveSourceToDisk)
            saveTemplateData(pi, templateDataF);
        pci.sourceFiles.add(new FileInfo(templateDataF));

    }

    protected void postProcessHtmlTemplate(HtmlDocument doc) throws CompilerException {
        zapSelectedMetaTags(doc);
    }

    protected void zapSelectedMetaTags(HtmlDocument doc) throws CompilerException {
        for (Node head = doc.doc.getDocumentElement().getFirstChild(); head != null; head =
head.getNextSibling()) {
```

```
            if (head.getNodeType() != Node.ELEMENT_NODE ||
!head.getNodeName().equalsIgnoreCase("head"))
                continue;
            Node next;
            for (Node meta = head.getFirstChild(); meta != null; meta = next) {
                next = meta.getNextSibling();
                if (meta.getNodeType() != Node.ELEMENT_NODE ||
!meta.getNodeName().equalsIgnoreCase("meta"))
                    continue;
                Element e = (Element) meta;
                if (shouldZapMetaElement(e))
                    head.removeChild(meta);
            }
        }
    }

    protected boolean shouldZapMetaElement(Element e) throws CompilerException {
        return e.getAttribute("name").equalsIgnoreCase("generator")
                || e.getAttribute("http-equiv").equalsIgnoreCase("Content-Type");

    }

    protected void findRootElement(HtmlDocument doc) throws CompilerException {
        if (compilationInfo.annot.fragment().length() == 0) {
            rootE = doc.rootE;
            includeDoctype = true;
            return;
        }
        includeDoctype = false;
        List<Element> rootList = doc.findMatchingElements(compilationInfo.annot.fragment());
        if (rootList.isEmpty()) {
            log.log(LogLevel.Error, "Error in @THtml.fragment '" +
compilationInfo.annot.fragment()
                    + "': not found in html template");
            return;
        }
        if (rootList.size() > 1) {
            log.log(LogLevel.Error, "Error in @THtml.fragment '" +
compilationInfo.annot.fragment()
                    + "': matches multiple elements in template");
            return;
        }
        rootE = rootList.get(0);
    }

    protected void findHtmlTemplate() throws CompilerException {
        String fn = FilenameUtils.basename(compilationInfo.annot.filename());
        if (fn.length() == 0)
            fn = FilenameUtils.basename(javaFI.f.getName());
        File dirF = javaFI.f.getParentFile();
        String packageName = loadedJavaClass.getPackage().getName();
        TemplateLocation loc = new TemplateLocation(dirF, packageName, fn);
        if (loc.htmlF.exists()) {
            htmlLocation = loc;
            return;
        }
        dirF = new File(dirF, "templates");
        packageName += ".templates";
        loc = new TemplateLocation(dirF, packageName, fn);
        if (loc.htmlF.exists()) {
            htmlLocation = loc;
            return;
        }
        log.log(LogLevel.Error, "Can't find html template: " + fn + ".html/.htm");
    }

    protected void loadHtmlTemplate() throws CompilerException {
        doc = new HtmlLoader(log).load(htmlLocation.htmlF);
    }

    protected void makeScopes() throws CompilerException {
        rootScope = new ScopeImpl(log, loadedJavaClass, compilationInfo);
        rootScope.process();
        makeSubScope(compilationInfo.scopeClass);
    }

    private ScopeImpl makeSubScope(Class<?> cls) throws CompilerException {
        if (cls == loadedJavaClass)
            return rootScope;
        ScopeImpl parentScope = makeSubScope(cls.getEnclosingClass());
        return parentScope.getSubScope(cls, null, null);
    }

    protected void emitTree() throws CompilerException {
        ProgramImpl program = new ProgramImpl(log, rootScope);
        annotationScope = enterScope(program, compilationInfo.scopeClass);
        TreeEmit te = new TreeEmit(program, log);
        te.emit(doc, includeDoctype, rootE, annotationScope, compilationInfo.annotDescription);
        program.addReturn();
        for (ScopeImpl scope = annotationScope; scope != rootScope; scope = scope.parent) {
            //log.log(LogLevel.Notice, "LEAVES AUTO-SCOPE: " +
scope.scopeClass.getSimpleName());
            program.addLeaveScopeInstance(scope);
        }
        programData = program.postProcess();
    }

    protected ScopeImpl enterScope(ProgramImpl program, Class<?> cls) throws
CompilerException {
        if (cls == loadedJavaClass)
            return rootScope;
        ScopeImpl parentScope = enterScope(program, cls.getEnclosingClass());
        //log.log(LogLevel.Notice, "ENTERS AUTO-SCOPE: " + cls.getSimpleName());
        ScopeImpl scope = parentScope.getSubScope(cls, null, null);
        program.addCreateScopeInstance(scope);
        return scope;
    }

    protected void emitProgram(ProgressInfo pi) throws CompilerException {
        String packageName = htmlLocation.packageName;
        File tdF = new File(htmlLocation.dirF, "templatedata");
        if (tdF.exists())
            packageName += ".templatedata";
        fullClassName = packageName + "." + compilationInfo.tpname;
        ProgramEmit pe = new ProgramEmit(options, log, programData, packageName,
fullClassName);
        pe.main(pi);
```

```
            templateData = pe.templateData;
            pci.sourceFiles.add(new FileInfo(pe.getGeneratedJavaF()));
        }

        protected void emitPostCompilationWarnings() throws CompilerException {
            annotationScope.emitPostCompilationWarnings();
        }

        protected void saveTemplateData(ProgressInfo pi, File templateDataF) throws
CompilerException {
            try {
                templateDataF.getParentFile().mkdirs();
                byte[] newData = templateData.toBytes(fullClassName);
                if (templateDataF.exists()) {
                    byte[] oldData = FileContentsUtils.readFile(templateDataF);
                    if (Arrays.equals(newData, oldData)) {
                        if (options.verbose)
                            log.log(LogLevel.Notice, "tmpldata is unchanged");
                        return;
                    }
                }
                if (options.verbose)
                    log.log(LogLevel.Notice, "Saves TemplateData in " + templateDataF.getPath());
                FileContentsUtils.saveFile(templateDataF, newData);
                pi.numTemplateDataUpdated++;
            }
            catch (IOException ex) {
                throw new CompilerException("Can't save template data: " +
ExceptionUtils.getSomeMessage(ex), ex);
            }
        }

        static class TemplateLocation {
            final File dirF, htmlF;
            final String packageName;

            TemplateLocation(File dirF, String packageName, String fn) throws CompilerException {
                this.dirF = dirF;
                File f1 = new File(dirF, fn + ".html");
                File f2 = new File(dirF, fn + ".htm");
                if (f1.exists() && f2.exists())
                    throw new CompilerException("Has both " + f1.getName() + " and " +
f2.getName());
                this.htmlF = f2.exists() ? f2 : f1;
                this.packageName = packageName;
            }
        }
}
```

Appendix F

translator.compiler.application

F.1 AttributeEmitter

```
package dk.vitality.util.jat.translator.compiler.application.attribute;

abstract public class AttributeEmitter<AP extends Application> extends
AbstractEmitter<AP> {
    public final String[] lcAttributeNames;
    public boolean processed; // used by TreeEmit for keeping track of attribute collections.

    protected AttributeEmitter(ElementEmitters emitters, AP handler,
    String... lcAttributeNames) {
        super(emitters.program, handler);
        this.lcAttributeNames = lcAttributeNames;
        for (String attrName : lcAttributeNames) {
            if (emitters.alreadyDefined(emitters.eAttributes.get(attrName), this, "attribute"))
                return;
            emitters.eAttributes.put(attrName, this);
        }
    }

    /**
    * Subclass must either:<br>
    * 1) emit nothing<br>
    * 2) the complete attribute include space before its name<br>
    */
    abstract public void generateAttribute() throws CompilerException;
}
```

F.2 PostAttributeEmitter

```
package dk.vitality.util.jat.translator.compiler.application.attribute;

abstract public class PostAttributeEmitter<AP extends Application> extends
AbstractEmitter<AP> {
    public final String[] lcAttributeNames;
    public boolean processed; // used by TreeEmit for keeping track of attribute collections.

    protected PostAttributeEmitter(ElementEmitters emitters, AP handler,
    String... lcAttributeNames) {
        super(emitters.program, handler);
        this.lcAttributeNames = lcAttributeNames;
        for (String attrName : lcAttributeNames) {
            if (emitters.alreadyDefined(emitters.ePostAttributes.get(attrName), this,
"post-content_attribute"))
                return;
            emitters.ePostAttributes.put(attrName, this);
        }
    }

    /**
    * Is not allowed to emit quotes but only the content inside the quotes.
    *
    * @param staticAttrValue the attribute content specified in the html template
    *                        which the method must prepend to the output. This is to enable proper use
    *                        of ";" and " " separators in style and class attributes.
    */
    abstract public void generatePostContent(String staticAttrValue) throws CompilerException;

}
```

F.3 ACollectionApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.collection;

public class ACollectionApplication extends AbstractStringApplication {
    public ACollectionApplication(TargetingInfo target) throws CompilerException {
        super(target, EncodingType.Raw);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        ACollection a = (ACollection) annotation;
```

```
        new AttributeEmitter<AbstractStringApplication>(emitters,
                ACollectionApplication.this, a.styles()) {
            public void generateAttribute() throws CompilerException {
                program.addEmitChar('␣');
                handler.emitStringCallback(program);
            }
        };
    }
}
```

F.4 ASrcWidthHeightApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.collection;

public class ASrcWidthHeightApplication extends AbstractStringApplication {
    public ASrcWidthHeightApplication(TargetingInfo target) throws CompilerException {
        super(target, EncodingType.Raw);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new AttributeEmitter<AbstractStringApplication>(emitters,
                ASrcWidthHeightApplication.this, "src", "width", "height") {
            public void generateAttribute() throws CompilerException {
                program.addEmitChar('␣');
                handler.emitStringCallback(program);
            }
        };
    }
}
```

F.5 BooleanAttributeApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.content;

public class BooleanAttributeApplication extends AbstractBooleanApplication {
    public final String attrName;
    public final boolean inverse;

    public BooleanAttributeApplication(TargetingInfo target,
    String attrName, boolean inverse) throws CompilerException {
        super(target);
        this.attrName = attrName;
        this.inverse = inverse;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new AttributeEmitter<AbstractBooleanApplication>(emitters,
BooleanAttributeApplication.this, attrName) {
            public void generateAttribute() throws CompilerException {
                handler.emitBooleanCallback(program);
                if (!inverse)
                    program.addNegateBoolean();
                PatchedGoto pg = program.addPatchedConditionalGoto();
                program.addEmitString("␣");
                program.addEmitString(lcAttributeNames[0]);
                program.getNextAddress();
                pg.patchToCurrentAddress();
            }
        };
    }
}
```

F.6 IntegerAttributeApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.content;

public class IntegerAttributeApplication extends AbstractIntegerApplication {
    public final String attrName;

    public IntegerAttributeApplication(TargetingInfo target, String attrName) throws
CompilerException {
        super(target);
        this.attrName = attrName;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new AttributeEmitter<AbstractIntegerApplication>(emitters,
IntegerAttributeApplication.this, attrName) {
            public void generateAttribute() throws CompilerException {
                program.addEmitString("␣");
                program.addEmitString(lcAttributeNames[0]);
                program.addEmitString("=");
                handler.emitIntegerCallback(program);
                program.addPrintIntegerResult();
            }
        };
    }
}
```

F.7 StringAttributeApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.content;

public class StringAttributeApplication extends AbstractStringApplication {
    public final String attrName;

    public StringAttributeApplication(TargetingInfo target, String attrName) throws
CompilerException {
        super(target, EncodingType.Attr);
        this.attrName = attrName;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new AttributeEmitter<AbstractStringApplication>(emitters,
StringAttributeApplication.this, attrName) {
            public void generateAttribute() throws CompilerException {
                program.addEmitString("␣");
                program.addEmitString(lcAttributeNames[0]);
```

```
                program.addEmitString("='");
                handler.emitStringCallback(program);
                program.addEmitString("'");
            }
        };
    }
}
```

F.8 StringPostAttributeApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.content;

public class StringPostAttributeApplication extends AbstractStringApplication {
    public final String attrName;
    protected final String postContentSeparator;

    public StringPostAttributeApplication(TargetingInfo target, String postContentSeparator,
String attrName) throws CompilerException {
        super(target, EncodingType.Attr);
        this.attrName = attrName;
        this.postContentSeparator = postContentSeparator;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new PostAttributeEmitter<AbstractStringApplication>(emitters,
StringPostAttributeApplication.this, attrName) {
            public void generatePostContent(String staticAttrValue) throws CompilerException {
                if (staticAttrValue.length() > 0) {
                    program.addEmitString(HtmlAttr.escape(staticAttrValue));
                    program.addEmitString(postContentSeparator);
                }
                handler.emitStringCallback(program);
            }
        };
    }
}
```

F.9 SCollectionApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SCollectionApplication extends AbstractStringApplication {
    public SCollectionApplication(TargetingInfo target) throws CompilerException {
        super(target, EncodingType.Attr);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final SCollection a = (SCollection) annotation;
        new SubAttributeEmitter<AbstractStringApplication>(emitters, this, "style", a.styles()) {
            public void generateSubAttribute(String itemPrefix) throws CompilerException {
                handler.emitStringCallback(program);
            }
        };
    }
}
```

F.10 SDisplayApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SDisplayApplication extends AbstractBooleanApplication {
    public SDisplayApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final SDisplay a = (SDisplay) annotation;
        new SubAttributeEmitter<AbstractBooleanApplication>(emitters, this, "style",
"display") {
            public void generateSubAttribute(String itemPrefix) throws CompilerException {
                emit(itemPrefix + "display:␣none");
            }

            public boolean canGenerateFullAttribute() {
                return true;
            }

            public void generateFullAttribute() throws CompilerException {
                emit("␣style='display:␣none'");
            }

            private void emit(String content) throws CompilerException {
                handler.emitBooleanCallback(program);
                if (a.inverse())
                    program.addNegateBoolean();
                PatchedGoto pg = program.addPatchedConditionalGoto();
                program.addEmitString(content);
                pg.patchToCurrentAddress();
            }
        };
    }
}
```

F.11 SDisplayBlockApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SDisplayBlockApplication extends AbstractBooleanApplication {
    public SDisplayBlockApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final SDisplayBlock a = (SDisplayBlock) annotation;
        new StyleDisplayEmitter(emitters, this, a.inverse(), "block");
    }
}
```

F.12 SDisplayInlineApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SDisplayInlineApplication extends AbstractBooleanApplication {
    public SDisplayInlineApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final SDisplayInline a = (SDisplayInline) annotation;
        new StyleDisplayEmitter(emitters, this, a.inverse(), "inline");
    }
}
```

F.13 SHasClassApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SHasClassApplication extends AbstractBooleanApplication {
    public SHasClassApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public static class SHasClassEmitter extends
SubAttributeEmitter<AbstractBooleanApplication> {
        final SHasClass annot;

        protected SHasClassEmitter(ElementEmitters emitters, AbstractBooleanApplication
handler,
        SHasClass annot) {
            super(emitters, handler, "class", annot.classes());
            this.annot = annot;
        }

        public void generateSubAttribute(String itemPrefix) throws CompilerException {
            emit(itemPrefix, null);
        }

        public boolean canGenerateFullAttribute() {
            return true;
        }

        public void generateFullAttribute() throws CompilerException {
            emit("␣class='", "'");
        }

        private void emit(String itemPrefix, String afterItems) throws CompilerException {
            if (lcSubAttributeNames.length == 0)
                return;
            handler.emitBooleanCallback(program);
            if (!annot.inverse())
                program.addNegateBoolean();
            PatchedGoto pg = program.addPatchedConditionalGoto();
            for (String cls : lcSubAttributeNames) {
                program.addEmitString(itemPrefix);
                itemPrefix = "␣";
                program.addEmitString(cls);
            }
            if (afterItems != null)
                program.addEmitString(afterItems);
            pg.patchToCurrentAddress();
        }
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new SHasClassEmitter(emitters, this, (SHasClass) annotation);
    }
}
```

F.14 SValueApplication

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

public class SValueApplication extends AbstractStringApplication {
    public SValueApplication(TargetingInfo target) throws CompilerException {
        super(target, AbstractStringApplication.EncodingType.Attr);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final SValue a = (SValue) annotation;
        final String[] styles = a.styles();
        new SubAttributeEmitter<AbstractStringApplication>(emitters, this, "style", styles) {
            public void generateSubAttribute(String itemPrefix) throws CompilerException {
                if (styles.length == 0)
                    return;
                program.addEmitString(itemPrefix);
                if (styles.length > 1 && callbackType !=
AbstractValueApplication.CallbackType.VoidMethod)
                    handler.emitStringCallback(program);
                for (String style : styles) {
                    program.addEmitString(style);
                    program.addEmitString(":␣");
                    if (styles.length > 1 && callbackType !=
AbstractValueApplication.CallbackType.VoidMethod)
                        program.addPrintHtmlEncodedStringResult();
                    else
                        handler.emitStringCallback(program);
                }
            }
        };
    }
}
```

F.15 StyleDisplayEmitter

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

class StyleDisplayEmitter extends SubAttributeEmitter<AbstractBooleanApplication> {
    final boolean inverse;
    final String visibleValue;
```

```
    protected StyleDisplayEmitter(ElementEmitters emitters,
    AbstractBooleanApplication handler,
    boolean inverse, String visibleValue) {
        super(emitters, handler, "style", "display");
        this.inverse = inverse;
        this.visibleValue = visibleValue;
    }

    public void generateSubAttribute(String itemPrefix) throws CompilerException {
        program.addEmitString(itemPrefix);
        program.addEmitString("display:␣");
        handler.emitBooleanCallback(program);
        if (inverse)
            program.addNegateBoolean();
        PatchedGoto handleTrue = program.addPatchedConditionalGoto();
        program.addEmitString("none");
        PatchedGoto done = program.addPatchedGoto();
        handleTrue.patchToCurrentAddress();
        program.addEmitString(visibleValue);
        done.patchToCurrentAddress();
    }
}
```

F.16 SubAttributeEmitter

```
package dk.vitality.util.jat.translator.compiler.application.attribute.sub;

abstract public class SubAttributeEmitter<AP extends Application> extends
AbstractEmitter<AP> {
    public final String lcAttributeName;
    public final String[] lcSubAttributeNames;

    protected SubAttributeEmitter(ElementEmitters emitters, AP handler,
    String lcAttributeName, String... lcSubAttributeNames) {
        super(emitters.program, handler);
        this.lcAttributeName = lcAttributeName;
        this.lcSubAttributeNames = lcSubAttributeNames;
        List<SubAttributeEmitter<?>> list = emitters.eSubAttributes.get(lcAttributeName);
        if (list == null) {
            list = new ArrayList<SubAttributeEmitter<?>>();
            emitters.eSubAttributes.put(lcAttributeName, list);
        }
        list.add(this);
    }

    /**
    * @param itemPrefix processor must emit string before first item
    * @return true if anything could be emitted
    */
    abstract public void generateSubAttribute(String itemPrefix) throws CompilerException;

    public boolean canGenerateFullAttribute() {
        return false;
    }

    public void generateFullAttribute() throws CompilerException {
        throw new InternalCompilerException("Not␣implemented");
    }
}
```

F.17 AbstractBooleanApplication

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

abstract public class AbstractBooleanApplication extends AbstractValueApplication {
    protected AbstractBooleanApplication(TargetingInfo target) throws CompilerException {
        super(target, false, Boolean.class);
    }

    public void emitBooleanCallback(Program program) throws CompilerException {
        switch (callbackType) {
            case Field:
                program.addGetBooleanField(scope, field);
                break;
            case Method:
                program.addCallback(scope, method);
                break;
            default:
                throw new CompilerException("Unsupported␣callbackType␣=␣" + callbackType + ":␣"
+ annotationToString());
        }
    }
}
```

F.18 AbstractIntegerApplication

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

abstract public class AbstractIntegerApplication extends AbstractValueApplication {
    protected AbstractIntegerApplication(TargetingInfo target) throws CompilerException {
        super(target, false, Integer.class);
    }

    public void emitIntegerCallback(Program program) throws CompilerException {
        switch (callbackType) {
            case Field:
                program.addGetIntegerField(scope, field);
                break;
            case Method:
                program.addCallback(scope, method);
                break;
            default:
                throw new CompilerException("Unsupported␣callbackType␣=␣" + callbackType + ":␣"
+ annotationToString());
        }
    }
}
```

F.19 AbstractIteratorApplication

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

abstract public class AbstractIteratorApplication extends AbstractValueApplication {
    public final Type itemType;

    protected AbstractIteratorApplication(TargetingInfo target) throws CompilerException {
        super(target, false, Iterable.class, Iterator.class);
        Type t = (method != null) ? method.getGenericReturnType() : field.getGenericType();
        if (!(t instanceof ParameterizedType))
            throw makeInvalidAnnotationApplianceException(
"Iterator/able_must_be_generic_parameterized_type.");
        ParameterizedType pt = (ParameterizedType) t;
        Type[] ata = pt.getActualTypeArguments();
        if (ata == null || ata.length != 1)
            throw makeInvalidAnnotationApplianceException(
"Iterator/able_must_have_exactly_1_generic_parameterized_type_argument.");
        itemType = ata[0];
    }

    public IteratorInfo makeIteratorInfo(Program program) throws CompilerException {
        String prefix = program.getNextVariableNamePrefix();
        boolean isIterable = Iterable.class.isAssignableFrom(resultType);
        return new IteratorInfo(scope, target.annotationToString(), isIterable, prefix, itemType);
    }

    public void emitIteratorCallback(Program program, IteratorInfo iterInfo) throws
CompilerException {
        switch (callbackType) {
            case Field:
                program.addGetIteratorField(scope, field, iterInfo);
                break;
            case Method:
                program.addCallback(iterInfo, method);
                break;
            default:
                throw new CompilerException("Unsupported_callbackType_=_" + callbackType + ":_"
+ annotationToString());
        }
    }
}
```

F.20 AbstractStringApplication

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

abstract public class AbstractStringApplication extends AbstractValueApplication {
    public enum EncodingType {
        Raw, Html, Attr
    }

    protected EncodingType encoding;

    protected AbstractStringApplication(TargetingInfo target, EncodingType encoding) throws
CompilerException {
        super(target, true, String.class);
        this.encoding = encoding;
        if (encoding != EncodingType.Raw && callbackType == CallbackType.VoidMethod) {
            log.log(LogLevel.Error, "Can't_use_@" +
target.annotation.annotationType().getSimpleName()
                    + "_on_void_method:_" + annotationToString());
        }
    }

    public void emitStringCallback(Program program) throws CompilerException {
        switch (callbackType) {
            case Field:
                program.addGetStringField(scope, field);
                printStringResult(program);
                break;
            case Method:
                program.addCallback(scope, method);
                printStringResult(program);
                break;
            case VoidMethod:
                program.addCallback(scope, method);
                break;
            default:
                throw new CompilerException("Unsupported_callbackType_=_" + callbackType + ":_"
+ annotationToString());
        }
    }

    protected void printStringResult(Program program) throws CompilerException {
        switch (encoding) {
            case Raw:
                program.addPrintRawStringResult();
                break;
            case Html:
                program.addPrintHtmlEncodedStringResult();
                break;
            case Attr:
                program.addPrintAttrEncodedStringResult();
                break;
            default:
                throw new InternalCompilerException("invalid_encoded_=_" + encoding);
        }
    }
}
```

F.21 AbstractValueApplication

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

abstract public class AbstractValueApplication extends Application {
    public final CallbackType callbackType;
    public final Class<?> resultType;

    public enum CallbackType {
        Field, Method, VoidMethod
    }
```

```
    protected AbstractValueApplication(TargetingInfo target,
    boolean allowVoidMethod, Class<?>... allowedTypes) throws CompilerException {
        super(target);
        if (method == null && field == null)
            throw makeInvalidAnnotationApplianceException(
"Annotation must be applied to field or method: " + annotationToString());
        Class<?> actualType = (method != null) ? method.getReturnType() : field.getType();
        if (method != null) {
            if (method.getParameterTypes().length > 0)
                throw makeInvalidAnnotationApplianceException(
"Method may not take any arguments: " + annotationToString());
            if (allowVoidMethod && actualType.getName().equals("void")) {
                callbackType = CallbackType.VoidMethod;
                resultType = null;
                return;
            }
        }
        callbackType = (method != null) ? CallbackType.Method : CallbackType.Field;
        Class<?> match = findMatchingType(actualType, allowedTypes);
        if (match != null) {
            resultType = match;
            return;
        }
        StringBuilder sb = new StringBuilder();
        sb.append(method != null ? "Method must return " : "Field must be ");
        if (allowedTypes.length == 1)
            sb.append("a ");
        for (int i = 0; i < allowedTypes.length; i++) {
            Class<?> t = allowedTypes[i];
            if (i > 0)
                sb.append(i + 1 == allowedTypes.length ? " or " : ", ");
            String s = t.getSimpleName();
            sb.append(s);
            if (t.isPrimitive())
                sb.append("/").append(s.toLowerCase());
        }
        sb.append(method != null ? " but returns " : " but is ");
        sb.append(actualType.getSimpleName());
        throw makeInvalidAnnotationApplianceException(sb.toString());
    }

    private static Class<?> findMatchingType(Class<?> actualType, Class<?>[]
allowedTypes) {
        String s = actualType.getSimpleName();
        for (Class<?> t : allowedTypes) {
            if (t == Integer.class && s.equals("int"))
                return t;
            if (t == Long.class && s.equals("long"))
                return t;
            if (t == Boolean.class && s.equals("boolean"))
                return t;
            if (t.isAssignableFrom(actualType))
                return t;
        }
        return null;
    }
}
```

F.22 Application

```
package dk.vitality.util.jat.translator.compiler.application.baseclass;

public abstract class Application {
    public final CompilerLog log;

    public final Scope scope;
    public final Annotation annotation;
    public final TargetingInfo target; // null if not targeting handler

    // item which the annotation is specified on. Exactly one is non-null.
    public final Class<?> cls;
    public final Field field;
    public final Method method;

    protected Application(TargetingInfo target) {
        this.log = target.log;
        this.scope = target.scope;
        this.annotation = target.annotation;
        this.cls = target.cls;
        this.field = target.field;
        this.method = target.method;
        this.target = target;
    }

    public Scope makeSubScope(Scope outerScope) throws CompilerException {
        return null;
    }

    abstract public void createEmitter(ElementEmitters emitters, Element e) throws
CompilerException;

    public String annotationToString() {
        return CompilerUtils.annotationToString(annotation, cls, field, method);
    }

    public InvalidApplicationException makeInvalidAnnotationApplianceException(String msg) {
        return new InvalidApplicationException(msg + ": " + annotationToString());
    }

    public void notifyMatched(SelectorType type) {
        if (target != null)
            target.notifyMatched(type);
    }

    public void emitPostCompilationWarnings() {
        if (target != null)
            target.emitPostCompilationWarnings();
    }
}
```

F.23 ElementLoopEmitter

```
package dk.vitality.util.jat.translator.compiler.application.element;

abstract public class ElementLoopEmitter<AP extends Application> extends
AbstractEmitter<AP> {
    /**
```

```
    * false for loop constructs to avoid duplicate id attributes in generated output.
    */
    public final boolean allowIdAttributeInSubTree;
    public final Scope subScope;

    protected ElementLoopEmitter(ElementEmitters emitters, AP handler,
    boolean allowIdAttributeInSubTree,
    Scope subScope) {
        super(emitters.program, handler);
        this.allowIdAttributeInSubTree = allowIdAttributeInSubTree;
        this.subScope = subScope;
        if (emitters.alreadyDefined(emitters.eLoop, this, "@EIterate/@EWhile/@EScopingWhile"))
            return;
        emitters.eLoop = this;
        if (!emitters.enterSubScope(this, subScope))
            return;
        emitters.allowIdAttribute &= allowIdAttributeInSubTree;
    }

    /**
    * Called before tag has been emitted.
    */
    abstract public void preElement() throws CompilerException;

    /**
    * Called after the compiler has processed the element.
    * Normally used for patching goto's for if/while constructs.
    */
    abstract public void postElement() throws CompilerException;
}
```

F.24 EIterateApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.scoped;

public class EIterateApplication extends AbstractIteratorApplication {
    public EIterateApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public Scope makeSubScope(Scope outerScope) throws CompilerException {
        final EIterate a = (EIterate) annotation;
        Scope subScope = outerScope.getSubScope(a.scope(), target, itemType);
        if (subScope == null)
            return null; // we have already logged the error
        if (subScope.scopeClass.isAnnotationPresent(EScope.class)) {
            log.log(LogLevel.Error,
"@EScope may not be present on inner-class used for @EIterate");
            return null;
        }
        return subScope;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final Scope subScope = makeSubScope(scope);
        if (subScope == null)
            return;
        final IteratorInfo iterInfo = makeIteratorInfo(emitters.program);
        new ElementLoopEmitter<AbstractIteratorApplication>(emitters, this, false, subScope) {
            PatchedGoto gotoTest;
            ProgramAddress bodyStart;

            public void preElement() throws CompilerException {
                handler.emitIteratorCallback(program, iterInfo);
                gotoTest = program.addPatchedGoto();
                bodyStart = program.getNextAddress();
                program.addCreateScopeInstance(iterInfo, subScope);
            }

            public void postElement() throws CompilerException {
                program.addLeaveScopeInstance(subScope);
                gotoTest.patchToCurrentAddress();
                program.addIteratorStep(iterInfo);
                program.addConditionalGoto(bodyStart);
            }
        };
    }
}
```

F.25 EScopedApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.scoped;

public class EScopedApplication extends Application {
    public EScopedApplication(TargetingInfo target) {
        super(target);
    }

    public Scope makeSubScope(Scope outerScope) throws CompilerException {
        return outerScope.getSubScope(cls, target, null);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        // annot er altid på inner-class.
        final Scope subScope = makeSubScope(scope);
        if (subScope == null)
            return; // we have already logged the error
        new Emitter(emitters, EScopedApplication.this, subScope);
    }

    public class Emitter extends AbstractEmitter<Application> {
        final Scope subScope;

        protected Emitter(ElementEmitters emitters, Application handler, Scope subScope) {
            super(emitters.program, handler);
            this.subScope = subScope;
            if (emitters.alreadyDefined(emitters.eScope, this, "@EScope"))
                return;
            emitters.eScope = this;
            emitters.enterSubScope(this, subScope);
        }


        public void preElement() throws CompilerException {
```

```
            program.addCreateScopeInstance(subScope);
        }
        /**
        * Called after the compiler has processed the element.
        */
        public void postElement() throws CompilerException {
            program.addLeaveScopeInstance(subScope);
        }
    }
}
```

F.26 EScopingWhileApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.scoped;

public class EScopingWhileApplication extends AbstractBooleanApplication {
    public EScopingWhileApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public Scope makeSubScope(Scope outerScope) throws CompilerException {
        final EScopingWhile a = (EScopingWhile) annotation;
        Scope subScope = outerScope.getSubScope(a.scope(), target, null);
        if (subScope == null)
            return null; // we have already logged the error
        if (subScope.scopeClass.isAnnotationPresent(EScope.class)) {
            log.log(LogLevel.Error,
"@EScope_may_not_be_present_on_inner-class_used_for_@EScopingWhile");
            return null;
        }
        return subScope;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        final Scope subScope = makeSubScope(scope);
        if (subScope == null)
            return;
        new ElementLoopEmitter<AbstractBooleanApplication>(emitters, this, false, subScope) {
            PatchedGoto gotoTest;
            ProgramAddress bodyStart;

            public void preElement() throws CompilerException {
                gotoTest = program.addPatchedGoto();
                bodyStart = program.getNextAddress();
                program.addCreateScopeInstance(subScope);
            }

            public void postElement() throws CompilerException {
                program.addLeaveScopeInstance(subScope);
                gotoTest.patchToCurrentAddress();
                handler.emitBooleanCallback(program);
                final EScopingWhile a = (EScopingWhile) annotation;
                if (a.inverse())
                    program.addNegateBoolean();
                program.addConditionalGoto(bodyStart);
            }
        };
    }
}
```

F.27 EContentApplication

```
packagedk.vitality.util.jat.translator.compiler.application.element.simple;

public class EContentApplication extends AbstractStringApplication {
    public EContentApplication(TargetingInfo target, EncodingType encoding) throws
CompilerException {
        super(target, encoding);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new Emitter(emitters, this);
    }

    public class Emitter extends AbstractEmitter<AbstractStringApplication> {
        protected Emitter(ElementEmitters emitters, AbstractStringApplication handler) {
            super(emitters.program, handler);
            if (!emitters.alreadyDefined(emitters.eContent, this, "@EText/@EHtml"))
                emitters.eContent = this;
        }

        public void generateContent() throws CompilerException {
            emitStringCallback(program);
        }
    }
}
```

F.28 EContentIfApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EContentIfApplication extends AbstractBooleanApplication {
    public EContentIfApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        EContentIf a = (EContentIf) annotation;
        new Emitter(emitters, this, a.inverse());
    }

    public class Emitter extends AbstractEmitter<AbstractBooleanApplication> {
        final boolean inverse;
        PatchedGoto pg;

        Emitter(ElementEmitters emitters, AbstractBooleanApplication handler, boolean inverse) {
            super(emitters.program, handler);
            this.inverse = inverse;
            emitters.eContentIf.add(this);
        }

        public void preElementContent() throws CompilerException {
```

```
            handler.emitBooleanCallback(program);
            if (!inverse)
                program.addNegateBoolean();
            pg = program.addPatchedConditionalGoto();
        }

        public void postElementContent() throws CompilerException {
            pg.patchToCurrentAddress();
        }
    }
}
```

F.29 EContentIfNotApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EContentIfNotApplication extends EContentIfApplication {
    public EContentIfNotApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new EContentIfNotApplication.Emitter(emitters, this, true);
    }
}
```

F.30 EContentPrePostApplication

```
packagedk.vitality.util.jat.translator.compiler.application.element.simple;

public class EContentPrePostApplication extends AbstractStringApplication {
    final boolean isPre;

    public EContentPrePostApplication(TargetingInfo target, EncodingType encoding, boolean
isPre) throws CompilerException {
        super(target, encoding);
        this.isPre = isPre;
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new Emitter(emitters, this);
    }

    public class Emitter extends AbstractEmitter<AbstractStringApplication> {
        protected Emitter(ElementEmitters emitters, AbstractStringApplication handler) {
            super(emitters.program, handler);
            emitters.ePrePostContent.add(this);
        }

        public void preContent() throws CompilerException {
            if (isPre)
                emitStringCallback(program);
        }

        public void postContent() throws CompilerException {
            if (!isPre)
                emitStringCallback(program);
        }
    }
}
```

F.31 EIfApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EIfApplication extends AbstractBooleanApplication {
    public EIfApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        EIf a = (EIf) annotation;
        new Emitter(emitters, this, a.inverse());
    }

    public class Emitter extends AbstractEmitter<AbstractBooleanApplication> {
        final boolean inverse;
        PatchedGoto pg;

        Emitter(ElementEmitters emitters, AbstractBooleanApplication handler, boolean inverse) {
            super(emitters.program, handler);
            this.inverse = inverse;
            emitters.eConditionals.add(this);
        }


        public void preElement() throws CompilerException {
            handler.emitBooleanCallback(program);
            if (!inverse)
                program.addNegateBoolean();
            pg = program.addPatchedConditionalGoto();
        }
        /**
        * Called after the compiler has processed the element.
        * Normally used for patching goto's for if/while constructs.
        */
        public void postElement() throws CompilerException {
            pg.patchToCurrentAddress();
        }
    }
}
```

F.32 EIfNotApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EIfNotApplication extends EIfApplication {
    public EIfNotApplication(TargetingInfo target) throws CompilerException {
        super(target);
```

```
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new Emitter(emitters, this, true);
    }
}
```

F.33 EReplaceApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EReplaceApplication extends AbstractStringApplication {
    public EReplaceApplication(TargetingInfo target) throws CompilerException {
        super(target, EncodingType.Raw);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new Emitter(emitters, this);
    }

    public class Emitter extends AbstractEmitter<Application> {
        protected Emitter(ElementEmitters emitters, Application handler) {
            super(emitters.program, handler);
            if (!emitters.alreadyDefined(emitters.eReplacer, this, "@EReplace"))
                emitters.eReplacer = this;
        }

        public void generateElement() throws CompilerException {
            emitStringCallback(program);
        }

    }
}
```

F.34 ETagIfApplication

```
packagedk.vitality.util.jat.translator.compiler.application.element.simple;

public class ETagIfApplication extends AbstractBooleanApplication {
    public ETagIfApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        ETagIf a = (ETagIf) annotation;
        new Emitter(emitters, a.inverse());
    }

    public class Emitter extends AbstractEmitter<AbstractBooleanApplication> {
        PatchedGoto pgOpen, pgClose;
        final boolean inverse;

        protected Emitter(ElementEmitters emitters, boolean inverse) {
            super(emitters.program, ETagIfApplication.this);
            this.inverse = inverse;
            emitters.eTag.add(this);
        }

        public void preElementOpenTag() throws CompilerException {
            handler.emitBooleanCallback(program);
            if (!inverse)
                program.addNegateBoolean();
            pgOpen = program.addPatchedConditionalGoto();
        }

        public void postElementOpenTag() throws CompilerException {
            pgOpen.patchToCurrentAddress();
        }

        public void preElementCloseTag() throws CompilerException {
            handler.emitBooleanCallback(program);
            if (!inverse)
                program.addNegateBoolean();
            pgClose = program.addPatchedConditionalGoto();
        }

        public void postElementCloseTag() throws CompilerException {
            pgClose.patchToCurrentAddress();
        }
    }
}
```

F.35 ETagIfNotApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class ETagIfNotApplication extends ETagIfApplication {
    public ETagIfNotApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new Emitter(emitters, true);
    }
}
```

F.36 EWhileApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EWhileApplication extends AbstractBooleanApplication {
    public EWhileApplication(TargetingInfo target) throws CompilerException {
        super(target);
    }

    public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
        new ElementLoopEmitter<AbstractBooleanApplication>(emitters, this, false, null) {
            PatchedGoto gotoTest;
            ProgramAddress bodyStart;

            public void preElement() throws CompilerException {
```

```
            gotoTest = program.addPatchedGoto();
            bodyStart = program.getNextAddress();
        }

        public void postElement() throws CompilerException {
            gotoTest.patchToCurrentAddress();
            handler.emitBooleanCallback(program);
            EWhile a = (EWhile) annotation;
            if (a.inverse())
                program.addNegateBoolean();
            program.addConditionalGoto(bodyStart);
        }
    };
  }
}
```

F.37 EZapApplication

```
package dk.vitality.util.jat.translator.compiler.application.element.simple;

public class EZapApplication extends Application {
  public EZapApplication(TargetingInfo target) {
    super(target);
  }

  public void createEmitter(ElementEmitters emitters, Element e) throws CompilerException {
    throw new CompilerException(
"Should␣not␣be␣called␣as␣element␣isn't␣processed␣but␣completely␣ignored.");
  }
}
```

F.38 AnnotationApplicationMap

```
package dk.vitality.util.jat.translator.compiler.application.scope;

class AnnotationApplicationMap {
  public static ApplicationFactory get(Annotation annotation) {
    return map.get(annotation.annotationType().getName());
  }

  private static final Map<String, ApplicationFactory> map = new HashMap<String,
ApplicationFactory>();

  static {
    addStringAttr("action", AAction.class);
    addStringAttr("align", AAlign.class);
    addStringAttr("class", AClass.class);
    addStringPostAttr("class", AClassPost.class, "␣");
    addStringAttr("clear", AClear.class);
    addStringAttr("coords", ACoords.class);
    addStringAttr("enctype", AEncType.class);
    addStringAttr("for", AFor.class);
    addStringAttr("height", AHeight.class);
    addStringAttr("href", AHref.class);
    addStringAttr("id", AId.class);
    addStringAttr("name", AName.class);
    addStringAttr("name", AName.class);
    addStringAttr("onblur", AOnBlur.class);
    addStringAttr("onclick", AOnClick.class);
    addStringAttr("ondblclick", AOnDblClick.class);
    addStringAttr("onfocus", AOnFocus.class);
    addStringAttr("onkeydown", AOnKeyDown.class);
    addStringAttr("onkeypress", AOnKeyPress.class);
    addStringAttr("onkeyup", AOnKeyUp.class);
    addStringAttr("onload", AOnLoad.class);
    addStringAttr("onmousedown", AOnMouseDown.class);
    addStringAttr("onmousemove", AOnMouseMove.class);
    addStringAttr("onmouseout", AOnMouseOut.class);
    addStringAttr("onmouseover", AOnMouseOver.class);
    addStringAttr("onmouseup", AOnMouseUp.class);
    addStringAttr("onreset", AOnReset.class);
    addStringAttr("onselect", AOnSelect.class);
    addStringAttr("onsubmit", AOnSubmit.class);
    addStringAttr("onunload", AOnUnload.class);
    addStringAttr("src", ASrc.class);
    addStringAttr("style", AStyle.class);
    addStringPostAttr("style", AStylePost.class, ";␣");
    addStringAttr("type", AType.class);
    addStringAttr("valign", AValign.class);
    addStringAttr("value", AValue.class);
    addStringAttr("width", AWidth.class);
    //———
    addBooleanAttr("checked", AChecked.class);
    addBooleanAttr("disabled", AEnabled.class);
    addBooleanAttr("multiple", AMultiple.class);
    addBooleanAttr("ismap", AIsMap.class);
    addBooleanAttr("readonly", AReadOnly.class);
    addBooleanAttr("selected", ASelected.class);
    //———
    addIntegerAttr("colspan", AColSpan.class);
    addIntegerAttr("rowspan", ARowSpan.class);
    addIntegerAttr("tabindex", ATabIndex.class);

    //———————————————————
    add(ASrcWidthHeight.class, new ApplicationFactory() {
      public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new ASrcWidthHeightApplication(target);
      }
    });
    add(ACollection.class, new ApplicationFactory() {
      public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new ACollectionApplication(target);
      }
    });
    //———————————————————
    add(SHasClass.class, new ApplicationFactory() {
      public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SHasClassApplication(target);
      }
    });
    //———————————————————
```

```
add(SDisplay.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SDisplayApplication(target);
    }
});
add(SDisplayBlock.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SDisplayBlockApplication(target);
    }
});
add(SDisplayInline.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SDisplayInlineApplication(target);
    }
});
add(SValue.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SValueApplication(target);
    }
});
add(SCollection.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new SCollectionApplication(target);
    }
});
//————————————————————————

add(EText.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentApplication(target, EncodingType.Html);
    }
});
add(ETextPre.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentPrePostApplication(target, EncodingType.Html, true);
    }
});
add(ETextPost.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentPrePostApplication(target, EncodingType.Html, false);
    }
});
add(EHtml.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentApplication(target, EncodingType.Raw);
    }
});
add(EHtmlPre.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentPrePostApplication(target, EncodingType.Raw, true);
    }
});
add(EHtmlPost.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentPrePostApplication(target, EncodingType.Raw, false);
    }
});
add(EReplace.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EReplaceApplication(target);
    }
});
add(EZap.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) {
        return new EZapApplication(target);
    }
});
add(EIf.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EIfApplication(target);
    }
});
add(EIfNot.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EIfNotApplication(target);
    }
});
add(EContentIf.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentIfApplication(target);
    }
});
add(EContentIfNot.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EContentIfNotApplication(target);
    }
});
add(ETagIf.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new ETagIfApplication(target);
    }
});
add(ETagIfNot.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new ETagIfNotApplication(target);
    }
});
add(EWhile.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EWhileApplication(target);
    }
});
add(EScope.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) {
        return new EScopedApplication(target);
    }
});
add(EScopingWhile.class, new ApplicationFactory() {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException {
        return new EScopingWhileApplication(target);
    }
});
add(EIterate.class, new ApplicationFactory() {
```

```
            public Application create(Scope scope, TargetingInfo target) throws CompilerException {
                return new EIterateApplication(target);
            }
        });
        if (1 == 2) {
            System.err.println("Num: " + map.size());
            List<String> list = new ArrayList<String>(map.keySet());
            Collections.sort(list);
            for (String s : list)
                System.err.println(s.substring(1 + s.lastIndexOf('.')));
        }
    }

    private static void add(Class<?> cls, ApplicationFactory factory) {
        map.put(cls.getName(), factory);
    }

    private static void addStringAttr(final String attrName, Class<?> cls) {
        add(cls, new ApplicationFactory() {
            public Application create(Scope scope, TargetingInfo target) throws CompilerException {
                return new StringAttributeApplication(target, attrName);
            }
        });
    }

    private static void addStringPostAttr(final String attrName, Class<?> cls, final String
postContentSeparator) {
        add(cls, new ApplicationFactory() {
            public Application create(Scope scope, TargetingInfo target) throws CompilerException {
                return new StringPostAttributeApplication(target, postContentSeparator, attrName);
            }
        });
    }

    private static void addBooleanAttr(final String attrName, Class<?> cls) {
        add(cls, new ApplicationFactory() {
            public Application create(Scope scope, TargetingInfo target) throws CompilerException {
                boolean inverse = attrName.equals("disabled");
                return new BooleanAttributeApplication(target, attrName, inverse);
            }
        });
    }

    private static void addIntegerAttr(final String attrName, Class<?> cls) {
        add(cls, new ApplicationFactory() {
            public Application create(Scope scope, TargetingInfo target) throws CompilerException {
                return new IntegerAttributeApplication(target, attrName);
            }
        });
    }
}
```

F.39 ApplicationFactory

```
package dk.vitality.util.jat.translator.compiler.application.scope;

/**
* @see Application for explanation of name
*/
interface ApplicationFactory {
    public Application create(Scope scope, TargetingInfo target) throws CompilerException;
}
```

F.40 Scope

```
package dk.vitality.util.jat.translator.compiler.application.scope;

abstract public class Scope {
    public final CompilerLog log;
    public final Class<?> scopeClass;
    public final String scopeName, scopeFieldName;

    protected Scope(CompilerLog log, Class<?> scopeClass, String scopeName, String
scopeFieldName) {
        this.log = log;
        this.scopeClass = scopeClass;
        this.scopeName = scopeName;
        this.scopeFieldName = scopeFieldName;
    }

    abstract public List<Application> getApplicationsForElement(Element e, boolean addParents)
throws CompilerException;

    abstract public ScopeImpl getSubScope(Class<?> inner, TargetingInfo target, Type
optConstructorArgument) throws CompilerException;
}
```

F.41 ScopeImpl

```
package dk.vitality.util.jat.translator.compiler.application.scope;

public class ScopeImpl extends Scope {
    public final ScopeImpl parent;
    public final Method scopeInstanceCreatorMethod;
    public final String scopeClassname;
    public final CompilationInfo compilationInfo;

    private static final SelectorType[] selectorTypes = SelectorType.values();
    protected final Map<SelectorType, Map<String, List<Application>>> dict;
    protected final List<Application> applicationList;

    protected final Map<String, ScopeImpl> subScopeMap;
    protected final Set<String> scopeNames;
    protected final Set<String> overrideWarningsGiven = new HashSet<String>();

    public ScopeImpl(CompilerLog log, Class<?> scopeClass, CompilationInfo compilationInfo)
throws InternalCompilerException {
        this(log, scopeClass, "root", "scopeRoot", null, compilationInfo, null);
    }

    protected ScopeImpl(CompilerLog log, Class<?> scopeClass,
```

```
    String scopeName, String scopeFieldName,
    ScopeImpl parent, CompilationInfo compilationInfo, Type optConstructorArgument) throws
InternalCompilerException {
        super(log, scopeClass, scopeName, scopeFieldName);
        this.parent = parent;
        this.compilationInfo = compilationInfo;
        if (parent != null && parent.parent != null)
            throw new InternalCompilerException("Double-nested scope"); // might allow it...
        dict = new EnumMap<SelectorType, Map<String,
List<Application>>>(SelectorType.class);
        for (SelectorType type : selectorTypes)
            dict.put(type, new HashMap<String, List<Application>>());
        applicationList = new ArrayList<Application>();
        subScopeMap = new HashMap<String, ScopeImpl>();
        if (parent == null) {
            scopeInstanceCreatorMethod = null;
            scopeClassname = scopeClass.getSimpleName();
            scopeNames = new HashSet<String>();
        } else {
            scopeNames = parent.scopeNames;
            scopeClassname = parent.scopeClassname + "." +
scopeClass.getSimpleName().replace('$', '.');
            scopeInstanceCreatorMethod =
getScopeInstanceCreatorMethod(optConstructorArgument);
        }
        scopeNames.add(scopeFieldName);
    }

    protected Method getScopeInstanceCreatorMethod(Type optConstructorArgument) {
        String methodName = "new" + CompilerUtils.getInnerclassName(scopeClass);
        String wantedArgType = (optConstructorArgument != null) ?
boxedType(optConstructorArgument.toString()) : null;
        int wantedNumArguments = (optConstructorArgument != null) ? 1 : 0;
        for (Class<?> cls = parent.scopeClass; cls != null; cls = cls.getSuperclass()) {
            for (Method m : cls.getDeclaredMethods()) {
                if (!m.getName().equals(methodName))
                    continue;
                Type[] actualArgs = m.getGenericParameterTypes();
                if (actualArgs.length != wantedNumArguments) {
                    log.log(LogLevel.Error,
"Wrong number of arguments for scope instance creating method: " + m);
                    continue;
                }
                if (wantedNumArguments == 0)
                    return m;
                if (actualArgs[0].equals(optConstructorArgument))
                    return m;
                String actual = boxedType(actualArgs[0].toString());
                if (actual.equals(wantedArgType))
                    return m;
                log.log(LogLevel.Error,
"Wrong argument type for scope instance creating method: " + m);
            }
        }
        return null;
    }

    protected static String boxedType(String type) {
        String s = boxedType.get(type);
        return (s != null) ? s : type;
    }

    protected static final Map<String, String> boxedType = new HashMap<String, String>();

    static {
        boxedType.put("boolean", "class java.lang.Boolean");
        boxedType.put("byte", "class java.lang.Byte");
        boxedType.put("short", "class java.lang.Short");
        boxedType.put("int", "class java.lang.Integer");
        boxedType.put("long", "class java.lang.Long");
        boxedType.put("float", "class java.lang.Float");
        boxedType.put("double", "class java.lang.Double");
    }

    public Iterable<ScopeImpl> iterSubScopes() {
        return subScopeMap.values();
    }

    public boolean needsReflectionConstructor() {
        return parent != null && scopeInstanceCreatorMethod == null &&
!Modifier.isStatic(scopeClass.getModifiers());
    }

    //=========================================

    public List<Application> getApplicationsForElement(Element e, boolean addParents) throws
CompilerException {
        List<Application> list = new ArrayList<Application>();
        getApplicationsForElement(list, e, addParents);
        return list;
    }

    protected void getApplicationsForElement(List<Application> list, Element e, boolean
addParents) throws CompilerException {
        String id = e.getAttribute("id");
        if (id != null && id.length() > 0)
            getApplicationsForKey(list, SelectorType.Id, id, addParents);

        for (String s : CompilerUtils.splitCssClassNames(e.getAttribute("class")))
            getApplicationsForKey(list, SelectorType.Class, s, addParents);

        getApplicationsForKey(list, SelectorType.Tag, e.getTagName(), addParents);
    }

    protected void getApplicationsForKey(List<Application> list, SelectorType type, String
origKey, boolean addParents) throws CompilerException {
        String javaKey = makeJavaKey(origKey);
        if (doGetApplications(list, type, javaKey, addParents))
            return;
        if (origKey.charAt(0) == '_')
            log.log(LogLevel.Notice, type.description + "=" + origKey +
"_not_referred_by_annotation");//, java name = " + javaKey);
    }

    protected String makeJavaKey(String key) {
        if (key.indexOf('_') < 0 && key.indexOf('-') < 0 && key.indexOf('.') < 0)
            return key.toLowerCase();
        int len = key.length();
```

```
        StringBuilder sb = new StringBuilder(len);
        for (int i = 0; i < len; i++) {
            char ch = key.charAt(i);
            if (ch == '_' || ch == '-' || ch == '.')
                continue;
            sb.append(Character.toLowerCase(ch));
        }
        return sb.toString();
    }

    protected boolean doGetApplications(List<Application> list, SelectorType type, String
javaKey, boolean addParents) {
        List<Application> matched = dict.get(type).get(javaKey);
        if (matched != null) {
            list.addAll(matched);
            for (Application processor : matched)
                processor.notifyMatched(type);
            return true;
        }
        return addParents && parent != null && parent.doGetApplications(list, type, javaKey,
addParents);
    }

    public void emitPostCompilationWarnings() throws CompilerException {
        for (Map<String, List<Application>> map : dict.values()) {
            for (List<Application> list : map.values()) {
                for (Application appl : list)
                    appl.emitPostCompilationWarnings();
            }
        }
        for (ScopeImpl scope : subScopeMap.values())
            scope.emitPostCompilationWarnings();
    }
    //=========================================

    private ScopeImpl getSubScope(Class<?> inner, Type optConstructorArgument) throws
CompilerException {
        //log.log(LogLevel.Notice, "GetSubScope: " + inner.getSimpleName());
        for (ScopeImpl scope = this; scope != null; scope = scope.parent) {
            if (scope.hasProperEnclosingClass(inner))
                return scope.getOrCreateSubScope(inner, optConstructorArgument);
        }
        log.log(LogLevel.Error, "Scoping␣inner-class␣(" + inner.getSimpleName() + ")"
                + "␣is␣not␣enclosed␣by␣parent␣scope␣(" + scopeClass.getSimpleName() + ")"
                + "␣but␣by␣" + inner.getEnclosingClass().getSimpleName());
        return null;
    }

    private ScopeImpl getOrCreateSubScope(Class<?> inner, Type optConstructorArgument)
throws CompilerException {
        String key = CompilerUtils.getInnerclassBasename(inner);
        ScopeImpl scope = subScopeMap.get(key);
        if (scope != null)
            return scope;
        String name = key;
        for (int i = 2; scopeNames.contains(name); i++)
            name = key + i;
        scope = new ScopeImpl(log, inner, name, "scope" + name, this, compilationInfo,
optConstructorArgument);
        subScopeMap.put(key, scope);
        scope.process();
        return scope;
    }

    public ScopeImpl getSubScope(Class<?> inner, TargetingInfo target, Type
optConstructorArgument) throws CompilerException {
        if (inner != null && inner != NotSpecified.class)
            return getSubScope(inner, optConstructorArgument);
        //log.log(LogLevel.Notice, "GetSubScope: " + target.annotationToString());
        for (Class<?> cls = scopeClass; cls != null; cls = cls.getSuperclass()) {
            for (Class<?> i2 : cls.getDeclaredClasses()) {
                String bn = CompilerUtils.getInnerclassBasename(i2);
                //log.log(LogLevel.Notice, "SubScope candidate: " + bn + " for " +
target.basename);
                if (bn.equals(target.basename))
                    return getSubScope(i2, optConstructorArgument);
            }
        }
        log.log(LogLevel.Error, "Can't␣locate␣innerclass␣for␣scope␣'" + target.basename
                + "'␣inside␣" + scopeClassname + ":␣" + target.annotationToString());
        return null;
    }

    private boolean hasProperEnclosingClass(Class<?> inner) {
        Class<?> enclosing = inner.getEnclosingClass();
        for (Class<?> cls = scopeClass; cls != null; cls = cls.getSuperclass()) {
            if (cls == enclosing)
                return true;
        }
        return false;
    }
    //=========================================

    public void process() throws CompilerException {
        Set<String> seen = new HashSet<String>();
        processClass(scopeClass, seen, false);
        log.log(LogLevel.Verbose, "ScopeImpl." + scopeFieldName + "␣handler␣count:␣id␣=␣" +
dict.get(SelectorType.Id).size()
                + ",␣class␣=␣" + dict.get(SelectorType.Class).size()
                + ",␣tag␣=␣" + dict.get(SelectorType.Tag).size()
                + ",␣subs␣=␣" + subScopeMap.size());
        if (parent == null) {
            // processing innerclasses is delayed as to have the full declaration of the parent scope
            // before processing the subscope as we otherwise can't make proper "overrides"
warnings.
            processInnerClasses(scopeClass, false);
        }
        for (Application appl : applicationList)
            appl.makeSubScope(this);
    }

    protected void processClass(Class<?> cls, Set<String> seen, boolean isInherited) throws
CompilerException {
        if (cls == null || cls.getName().startsWith("java."))
            return; // ignore java.lang.Object
```

```
        if (!cls.isAnnotationPresent(TIgnore.class)) {
            if (parent == null) {
                boolean hasTOverrides = cls.isAnnotationPresent(TOverrides.class);
                for (Annotation a : cls.getDeclaredAnnotations())
                    processAnnotation(new AnnotationHelper(log, a, cls, null, null), isInherited,
hasTOverrides);
            }
            for (Field f : cls.getDeclaredFields()) {
                if (f.isAnnotationPresent(TIgnore.class))
                    continue;
                String key = f.getType().getName() + "␣" + f.getName();
                if (seen.contains(key))
                    continue;
                seen.add(key);
                boolean hasTOverrides = f.isAnnotationPresent(TOverrides.class);
                for (Annotation a : f.getDeclaredAnnotations())
                    processAnnotation(new AnnotationHelper(log, a, cls, f, null), isInherited,
hasTOverrides);
            }
            for (Method m : cls.getDeclaredMethods()) {
                if (m.isAnnotationPresent(TIgnore.class))
                    continue;
                String key = m.getReturnType().getName() + "␣" + m.getName() + "␣" +
Arrays.toString(m.getParameterTypes());
                if (seen.contains(key))
                    continue;
                seen.add(key);
                boolean hasTOverrides = m.isAnnotationPresent(TOverrides.class);
                for (Annotation a : m.getDeclaredAnnotations())
                    processAnnotation(new AnnotationHelper(log, a, cls, null, m), isInherited,
hasTOverrides);
            }
        }
        processClass(cls.getSuperclass(), seen, true);
    }

    protected void processInnerClasses(Class<?> cls, boolean isInherited) throws
CompilerException {
        if (cls == null || cls.getName().startsWith("java."))
            return; // ignore java.lang.Object
        processInnerClasses(cls.getSuperclass(), isInherited);
        if (cls.isAnnotationPresent(TIgnore.class))
            return;
        for (Class<?> inner : cls.getDeclaredClasses()) {
            //log.log(LogLevel.Verbose, "Inner class " + inner.getSimpleName() + " inside " +
cls.getSimpleName());
            EScope es = inner.getAnnotation(EScope.class);
            if (es == null)
                continue;
            log.log(LogLevel.Verbose, "Inner␣class␣" + inner.getSimpleName() + "␣has␣@EScope:␣"
+ Modifier.toString(inner.getModifiers()));
            getSubScope(inner, null);
            boolean hasTOverrides = inner.isAnnotationPresent(TOverrides.class);
            processAnnotation(new AnnotationHelper(log, es, inner, null, null), isInherited,
hasTOverrides);
        }
    }

    protected void processAnnotation(AnnotationHelper helper, boolean isInherited, boolean
hasTOverrides) throws CompilerException {
        ApplicationFactory factory = AnnotationApplicationMap.get(helper.annotation);
        if (factory == null) {
            String annotationClassname = helper.annotation.annotationType().getName();
            if (annotationClassname.startsWith("dk.vitality.util.jat.runtime.annotation.")
                    && !(helper.annotation instanceof TCompile)
                    && !(helper.annotation instanceof TOverrides)) {
                log.log(LogLevel.Notice, "Missing␣handler␣for␣annotation:␣" +
annotationClassname);
            }
            return;
        }
        TargetingInfo target;
        Application appl;
        try {
            target = new TargetingInfo(this, helper, isInherited);
            appl = factory.create(this, target);
        }
        catch (InvalidApplicationException ex) {
            log.log(LogLevel.Error, ExceptionUtils.getSomeMessage(ex));
            return;
        }
        applicationList.add(appl);
        switch (target.type) {
            case DefaultParam:
                addApplication(SelectorType.Id, target.defparam, appl, hasTOverrides);
                addApplication(SelectorType.Class, target.defparam, appl, hasTOverrides);
                break;
            case Triplets:
                addApplication(SelectorType.Id, target.id, appl, hasTOverrides);
                addApplication(SelectorType.Class, target.className, appl, hasTOverrides);
                addApplication(SelectorType.Tag, target.tag, appl, hasTOverrides);
                break;
            case Appliance:
                addApplication(SelectorType.Id, target.basename, appl, hasTOverrides);
                addApplication(SelectorType.Class, target.basename, appl, hasTOverrides);
                break;
            default:
                throw new InternalCompilerException("Invalid␣value␣of␣target.type␣=␣" +
target.type);
        }
    }

    protected void addApplication(SelectorType type, String[] keys, Application processor,
    boolean hasTOverrides) throws CompilerException {
        for (String key : keys)
            addApplication(type, key, processor, hasTOverrides);
    }

    protected void addApplication(SelectorType type, String key, Application processor,
    boolean hasTOverrides) throws CompilerException {
        if (key == null || key.length() == 0)
            throw new InternalCompilerException("key␣is␣null/empty-string");
        key = key.toLowerCase();
        Map<String, List<Application>> map = dict.get(type);
        List<Application> list = map.get(key);
```

```
        if (list == null) {
            list = new ArrayList<Application>();
            map.put(key, list);
        }
        list.add(processor);
        if (hasTOverrides)
            return;
        if (parent == null)
            return;
        List<Application> overridden = new ArrayList<Application>(1);
        parent.getApplicationsForKey(overridden, type, key, true);
        if (overridden.isEmpty())
            return;
        String msg = "Annotation␣overrides␣enclosing␣class:␣" + processor.annotationToString();
        if (overrideWarningsGiven.contains(msg))
            return;
        overrideWarningsGiven.add(msg);
        log.log(LogLevel.Notice, msg);
    }
}
```

F.42 SelectorType

```
package dk.vitality.util.jat.translator.compiler.application.scope;

public enum SelectorType {
    Id("id"),
    Class("class"),
    Tag("tag");

    public final String description;

    SelectorType(String description) {
        this.description = description;
    }
}
```

F.43 TargetingInfo

```
package dk.vitality.util.jat.translator.compiler.application.scope;

public class TargetingInfo extends AnnotationHelper {
    public final Scope scope;
    public final boolean isInherited;
    public final String[] defparam, id, className, tag;

    public final String basename;
    public final Type type;

    public enum Type {
        DefaultParam, Triplets, Appliance
    }

    public TargetingInfo(Scope scope, AnnotationHelper helper, boolean isInherited) throws
CompilerException {
        super(helper.log, helper.annotation, helper.cls, helper.field, helper.method);
        this.scope = scope;
        this.isInherited = isInherited;
        this.defparam = getAnnotationStringArrayParam("value");
        this.id = getAnnotationStringArrayParam("id");
        this.className = getAnnotationStringArrayParam("className");
        this.tag = getAnnotationStringArrayParam("tag");
        String bn;
        if (defparam.length > 0) {
            type = Type.DefaultParam;
            if (id.length + className.length + tag.length > 0) {
                throw new InvalidApplicationException("id/className/tag␣may␣not␣be␣specified" +
                        "␣when␣the␣default␣parameter␣is␣specified:␣" + annotationToString());
            }
            bn = defparam[0];
        } else if (id.length + className.length + tag.length > 0) {
            bn = (id.length > 0) ? id[0] : ((className.length > 0) ? className[0] : tag[0]);
            type = Type.Triplets;
        } else if (method != null) {
            bn = CompilerUtils.stripLowercasePrefix(method.getName());
            if (bn.length() == 0)
                bn = method.getName();
            type = Type.Appliance;
        } else if (field != null) {
            bn = field.getName();
            type = Type.Appliance;
        } else if (cls.getEnclosingClass() != null) {
            // innerclass
            bn = cls.getSimpleName();
            type = Type.Appliance;
        } else {
            throw new InvalidApplicationException(
"id/className/tag␣or␣default-parm␣must␣be␣specified" +
                    "␣when␣annotation␣is␣applied␣directly␣on␣the␣class:␣" +
annotationToString());
        }
        if (bn.length() == 0)
            throw new InternalCompilerException("bn␣is␣null␣for␣" + annotationToString());
        // make initial letter uppercase
        if (!Character.isJavaIdentifierStart(bn.charAt(0)))
            bn = "X" + bn;
        basename = CompilerUtils.makeFirstCharacterUppercase(bn);
    }

    public String targetToString() {
        return type.name()
                + (defparam.length > 0 ? "{defparam␣=␣" + Arrays.toString(defparam) + "}" : "")
                + (id.length > 0 ? "{id␣=␣" + Arrays.toString(id) + "}" : "")
                + (className.length > 0 ? "{class␣=␣" + Arrays.toString(className) + "}" : "")
                + (tag.length > 0 ? "{tag␣=␣" + Arrays.toString(tag) + "}" : "")
                ;
    }

    protected EnumSet<SelectorType> matchedBy = EnumSet.noneOf(SelectorType.class);

    public void notifyMatched(SelectorType st) {
        // →|Id, Class, Tag
```

```
        if (matchedBy.contains(st))
            return;
        if (type != Type.DefaultParam && !matchedBy.isEmpty()) {
            log.log(LogLevel.Notice, "Matched␣by␣"
                    + matchedBy.iterator().next().description
                    + "␣and␣"
                    + st.description
                    + ":␣" + annotationToString());
        }
        matchedBy.add(st);
    }

    protected boolean hasMadeNonMatchedWarning;

    /**
    * can be called several times
    */
    public void emitPostCompilationWarnings() {
        if (hasMadeNonMatchedWarning || !matchedBy.isEmpty() || isInherited)
            return;
        hasMadeNonMatchedWarning = true;
        log.log(LogLevel.Notice, "Not␣matched␣by␣html:␣" + annotationToString());
    }
}
```

Appendix G

translator.compiler.codeGeneration

G.1 IteratorInfo

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.api;

public class IteratorInfo {
    public final Scope scope;
    public boolean isIterable;
    public final String description;
    public final String iterVar, itemVar, itemType, constructorVar;

    public IteratorInfo(Scope scope, String description, boolean isIterable, String namePrefix, Type
itemType) {
        this.scope = scope;
        this.description = description;
        this.isIterable = isIterable;
        this.iterVar = namePrefix + "iter";
        this.itemVar = namePrefix + "item";
        this.constructorVar = namePrefix + "cnstr";
        this.itemType = StringUtils.zapPrefix(itemType.toString(), "class ");
    }
}
```

G.2 PatchedGoto

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.api;

public interface PatchedGoto {
    public void patchToCurrentAddress() throws CompilerException;
}
```

G.3 Program

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.api;

public interface Program {
    public ProgramAddress getNextAddress() throws CompilerException; // flushes current string,
so next addEmitString is at new address.

    public void addGoto(ProgramAddress destination) throws CompilerException;

    public void addConditionalGoto(ProgramAddress destination) throws CompilerException;

    public PatchedGoto addPatchedGoto() throws CompilerException;

    public PatchedGoto addPatchedConditionalGoto() throws CompilerException;

    public void addNegateBoolean() throws CompilerException;

    public void addEmitChar(char ch) throws CompilerException;

    public void addEmitString(String s) throws CompilerException;

    public void addSoftNewline() throws CompilerException; // adds newline if text doesn't already
end with newline, i.e. inverse of trim()

    public void addCallback(Scope scope, Method method) throws CompilerException;

    public void addCallback(IteratorInfo info, Method method) throws CompilerException;

    public void addGetStringField(Scope scope, Field field) throws CompilerException;

    public void addGetBooleanField(Scope scope, Field field) throws CompilerException;

    public void addGetIntegerField(Scope scope, Field field) throws CompilerException;

    public void addGetIteratorField(Scope scope, Field field, IteratorInfo info) throws
CompilerException;

    public void addIteratorStep(IteratorInfo info) throws CompilerException;

    public void addPrintHtmlEncodedStringResult() throws CompilerException;

    public void addPrintAttrEncodedStringResult() throws CompilerException;

    public void addPrintRawStringResult() throws CompilerException;

    public void addPrintIntegerResult() throws CompilerException;

    public void addCreateScopeInstance(Scope scope) throws CompilerException;
```

```
    public void addCreateScopeInstance(IteratorInfo info, Scope subScope) throws
CompilerException;

    public void addLeaveScopeInstance(Scope scope) throws CompilerException;

    public String getNextVariableNamePrefix();
}
```

G.4 ProgramAddress

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.api;

public interface ProgramAddress {
}
```

G.5 Callback

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

class Callback {
    final ScopeImpl scope;
    final Field field;
    final Method method;
    final CallbackType callbackType;
    final IteratorInfo iterInfo;
    final String key;

    enum CallbackType {
        CallMethod,
        GetStringField, GetBooleanField, GetIntegerField, GetIteratorField,
        EnterScope, LeaveScope,
        IteratorStep
    }

    Callback(Scope scope, Field field, Method method, IteratorInfo iterInfo, CallbackType
callbackType, String key) {
        this.scope = (ScopeImpl) scope;
        this.field = field;
        this.method = method;
        this.iterInfo = iterInfo;
        this.callbackType = callbackType;
        this.key = key;
    }
}
```

G.6 PatchedGotoImpl

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

class PatchedGotoImpl implements PatchedGoto {
    protected boolean isPatched;
    protected final ProgramAddressImpl opcodeAddr;
    protected final ProgramImpl program;
    protected final Opcode opcode;

    PatchedGotoImpl(ProgramImpl program, ProgramAddressImpl opcodeAddr, Opcode opcode) {
        this.program = program;
        this.opcodeAddr = opcodeAddr;
        this.opcode = opcode;
    }

    public void patchToCurrentAddress() throws CompilerException {
        if (isPatched)
            throw new CompilerException("Goto_is_already_patched");
        isPatched = true;
        program.patchToCurrentAddress(opcodeAddr, opcode);
    }
}
```

G.7 ProgramAddressImpl

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

class ProgramAddressImpl implements ProgramAddress, Comparable<ProgramAddressImpl> {
    final int idx;

    ProgramAddressImpl(int addr) {
        this.idx = addr;
    }

    public boolean equals(Object o) {
        return o instanceof ProgramAddressImpl && idx == ((ProgramAddressImpl) o).idx;
    }

    public int hashCode() {
        return idx;
    }

    public String toString() {
        return "Addr{" + idx + '}';
    }

    public int compareTo(ProgramAddressImpl b) {
        return idx - b.idx;
    }
}
```

G.8 ProgramData

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

/**
* This class contains the final program data that 'ProgramImpl' delivers to
* the output class ProgramEmit.
*/
public class ProgramData {
    public final CompilerLog log;
```

```
    public final int[] opcodes;
    public final String[] texts;
    public final ScopeImpl rootScope;
    public final ScopeImpl[] scopes; // sorted by name
    public final Callback[] callbacks; // sorted by key

    ProgramData(CompilerLog log, int[] opcodes, String[] texts,
    ScopeImpl rootScope, ScopeImpl[] scopes, Callback[] callbacks) {
        this.log = log;
        this.opcodes = opcodes;
        this.texts = texts;
        this.rootScope = rootScope;
        this.scopes = scopes;
        this.callbacks = callbacks;
    }
}
```

G.9 ProgramEmit

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

public class ProgramEmit {
    protected final CompilerOptions options;
    protected final CompilerLog log;
    protected final ProgramData data;
    protected final String packageNameDot, fullClassName, simpleClassName;
    protected final List<Callback> unqiueIterCallbacks;

    protected JavaClass jclass;
    protected JavaPrinter jp;

    public TemplateData templateData;

    protected static final String md5Marker = "MD5.MD5.MD5.MD5.MD5.MD5.MD5.MD5.";

    public ProgramEmit(CompilerOptions options, CompilerLog log, ProgramData data, String
packageName, String fullClassName) throws CompilerException {
        this.options = options;
        this.log = log;
        this.data = data;
        this.fullClassName = fullClassName;
        this.packageNameDot = packageName + ".";
        this.simpleClassName = fullClassName.substring(packageNameDot.length());
        if (Opcode.values().length > Opcode.OPCODE_MASK)
            throw new CompilerException("Invalid␣Opcode.OPCODE_MASK:␣too␣few␣bits.");
        Set<String> seen = new HashSet<String>();
        unqiueIterCallbacks = new ArrayList<Callback>(); // must keep sorting
        for (Callback cb : data.callbacks) {
            if (cb.iterInfo == null)
                continue;
            if (seen.contains(cb.iterInfo.itemVar))
                continue;
            seen.add(cb.iterInfo.itemVar);
            unqiueIterCallbacks.add(cb);
        }
    }

public void main(ProgressInfo pi) throws CompilerException {
    try {
        jclass = new EmitJavaClass(options, fullClassName);
        jclass.encoding = options.sourceEncoding;
        jclass.addImport(TemplateBase.class);
        jclass.classExtends = "TemplateBase";
        jp = new JavaPrinter(jclass.pw, 1);
        makeConstructor();
        makeRootScopeField();
        makeMagicCommentStart();
        makeEmitSetup();
        makeSubScopeFields();
        makeCallbackFields();
        makeCallbacks();
        //if (options.verbose)
        //→|System.err.println(jclass.getCode());
        jclass.javaF.getParentFile().mkdirs();
        if (options.saveSourceToDisk) {
            if (jclass.save()) {
                pi.numJavaSourcesUpdated++;
                log.log(LogLevel.Notice, "Updated␣java␣source:␣" + jclass.shortClassName);
            }
        } else {
            jclass.getCode(); // to calc MD5s
        }
        if (options.verbose) {
            dumpOpcodes();
            //dumpStringTable();
        }
    }
    catch (CompilerException ex) {
        throw ex;
    }
    catch (Exception ex) {
        throw new CompilerException(ex);
    }
}

public File getGeneratedJavaF() {
    return jclass.javaF;
}

public class EmitJavaClass extends JavaClass {
    public EmitJavaClass(CompilerOptions options, String fullClassName) {
        super(options.generatedSourceBaseDirF, fullClassName, ClassType.Class);
    }

    public String getCode() throws Exception {
        String s = super.getCode();
        String md5 = MessageDigestUtils.md5LcHex(s);
        templateData = new TemplateData(data.texts, data.opcodes, md5);
        int idx = s.indexOf(md5Marker);
        return s.substring(0, idx) + md5 + s.substring(idx + md5Marker.length());
    }
}

protected void dumpStringTable() throws CompilerException {
    System.err.println("---------␣string␣table");
```

```
        StringBuilder sb = new StringBuilder(200);
        for (int i = 0; i < data.texts.length; i++) {
            String s = data.texts[i];
            System.err.println("[" + i + "]: {{" + s + "}}");
            sb.append(s);
        }
        int chars = sb.length();
        int bytes;
        try {
            bytes = sb.toString().getBytes("utf-8").length;
        }
        catch (UnsupportedEncodingException ex) {
            throw new CompilerException(ex.getMessage(), ex);
        }
        System.err.println("--------- end, " + chars + " chars, " + bytes + " utf-8 bytes.");
    }

    protected void dumpOpcodes() throws CompilerException {
        System.err.println("--------- opcodes");
        for (int i = 0; i < data.opcodes.length; i++) {
            int encoded = data.opcodes[i];
            String txt = String.format("%8s", "[" + i + "]: ");
            try {
                int param = Opcode.decodeParam(encoded);
                String s;
                switch (Opcode.decodeOpcode(encoded)) {
                    case PRINT_STRINGTABLE:
                        s = data.texts[param];
                        if (s.length() > 40)
                            s = s.substring(0, 20) + "[...]" + s.substring(s.length() - 20);
                        s = StringUtils.replaceSubstring(s, "\n", "\\n");
                        txt += String.format("%-25s %s", Opcode.toString(encoded), s);
                        break;
                    case CALLBACK:
                        Callback cb = data.callbacks[param];
                        txt += String.format("%-25s %s", Opcode.toString(encoded), cb.key);
                        //txt += ": " + cb.key;//cb.callbackType.name();
                        break;
                    default:
                        txt += Opcode.toString(encoded);
                }
            }
            catch (Exception ex) {
                txt += ": INVALID encoded = " + encoded + ": " +
ExceptionUtils.getSomeMessage(ex);
            }
            System.err.println(txt);
        }
        int stringsLen = 0;
        //noinspection UnusedCatchParameter
        try {
            for (String s : data.texts)
                stringsLen += s.getBytes("utf-8").length;
        }
        catch (UnsupportedEncodingException ex) {
            stringsLen = -1;
        }
        System.err.println("--------- end, " + data.opcodes.length * 4 + " bytes, strings = " +
stringsLen + " bytes.");
    }

    protected void makeConstructor() throws CompilerException {
        jp.println("public " + jclass.shortClassName + "("
                + relClassName(data.rootScope) + " " + data.rootScope.scopeFieldName + ")");
        jp.println("{");
        jp.println("super(\"" + md5Marker + "\");");
        jp.println("this." + data.rootScope.scopeFieldName + " = " +
data.rootScope.scopeFieldName + ";");
        jp.println("}");
    }

    protected void makeMagicCommentStart() throws CompilerException {
        jp.println();
        jp.println("/*");
        jp.println("// remove this line to comment out rest of class" +
                " (to temporary silence compile errors) */");
    }

    protected void makeRootScopeField() throws CompilerException {
        jp.println();
        jp.println("final " + relClassName(data.rootScope) + " " +
data.rootScope.scopeFieldName + ";");
    }

    protected void makeSubScopeFields() throws CompilerException {
        if (data.scopes.length <= 1)
            return;
        jp.println();
        jp.println("// subscopes:");
        Set<String> needsConstructor = new HashSet<String>();
        for (Callback cb : data.callbacks) {
            if (cb.callbackType == Callback.CallbackType.EnterScope && cb.iterInfo == null &&
cb.scope.needsReflectionConstructor())
                needsConstructor.add(cb.scope.scopeName);
        }
        for (ScopeImpl scope : data.scopes) {
            if (scope == data.rootScope)
                continue;
            jp.println();
            jp.println(relClassName(scope) + " " + scope.scopeFieldName + ";");
            if (needsConstructor.contains(scope.scopeName)) {
                jclass.addImport(Constructor.class);
                jp.println("static Constructor<" + relClassName(scope) + "> cnstr" +
scope.scopeName + ";");
            }
        }
    }

    protected void makeCallbackFields() throws CompilerException {
        for (Callback cb : unqiueIterCallbacks) {
            jclass.addImport(Iterator.class);
            jp.println();
            jp.println("// callback fields for " + cb.iterInfo.description);
            jp.println("Iterator<" + relClassName(cb.iterInfo.itemType) + "> " +
cb.iterInfo.iterVar + ";");
```

```
            jp.println(relClassName(cb.iterInfo.itemType) + " " + cb.iterInfo.itemVar + ";");
            if (!cb.scope.needsReflectionConstructor())
                continue;
            jclass.addImport(Constructor.class);
            jp.println("static_Constructor<" + relClassName(cb.scope) + ">_" +
cb.iterInfo.constructorVar + ";");
        }
    }

    protected void makeEmitSetup() throws CompilerException {
        jclass.addImport(HtmlPrintWriter.class);
        jp.println();
        jp.println("public_void_emit(HtmlPrintWriter_pw)_throws_Exception");
        jp.println("{");
        for (Callback cb : unqiueIterCallbacks) {
            if (!cb.scope.needsReflectionConstructor())
                continue;
            jp.print("if_(" + cb.iterInfo.constructorVar + "_==_null)");
            jp.println((cb.iterInfo != null) ? "_{" : "");
            jp.print(cb.iterInfo.constructorVar + "_=_" + relClassName(cb.scope)
                    + ".class.getDeclaredConstructor(\n"
                    + "\t\t" + relClassName(cb.scope.scopeClass.getEnclosingClass()) + ".class");
            if (cb.iterInfo != null) {
                jp.println(",\n\t\t" + relClassName(cb.iterInfo.itemType) + ".class);");
                jp.println("}");
            } else {
                jp.println(");");
            }
        }
        Set<String> seen = new HashSet<String>();
        for (Callback cb : data.callbacks) {
            if (cb.callbackType != Callback.CallbackType.EnterScope || cb.iterInfo != null)
                continue;
            if (!cb.scope.needsReflectionConstructor())
                continue;
            if (seen.contains(cb.scope.scopeName))
                continue;
            seen.add(cb.scope.scopeName);
            jp.println("if_(cnstr" + cb.scope.scopeName + "_==_null)");
            jp.println("cnstr" + cb.scope.scopeName + "_=_" + relClassName(cb.scope) + ".class"
+
                    ".getDeclaredConstructor(" + relClassName(cb.scope.parent) + ".class);");
        }
        jp.println("super.emit(pw);");
        jp.println("}");
    }

    protected void makeCallbacks() throws CompilerException {
        jp.println();
        jp.println("protected_void_executeCallback(int_idx)_throws_Exception");
        jp.println("{");
        jp.println("switch_(idx)_{");
        for (int cbIdx = 0; cbIdx < data.callbacks.length; cbIdx++) {
            Callback cb = data.callbacks[cbIdx];
            jp.println("case_" + cbIdx + ":");
            jp.indent++;
            switch (cb.callbackType) {
                case CallMethod:
                    if (cb.iterInfo != null) {
                        emitSetupIterator(cb, false);
                        break;
                    }
                    Class<?> t = cb.method.getReturnType();
                    if (t.equals(String.class))
                        jp.print("stringValue_=_");
                    else if (t.equals(Integer.class) || t.getName().equals("int"))
                        jp.print("intValue_=_");
                    else if (t.equals(Boolean.class) || t.getName().equals("boolean"))
                        jp.print("boolValue_=_");
                    jp.println(getMethodRef(cb.scope, cb.method) + "();");
                    break;

                case GetStringField:
                    jp.println("stringValue_=_" + getFieldRef(cb.scope, cb.field) + ";");
                    break;

                case GetIntegerField:
                    jp.println("intValue_=_" + getFieldRef(cb.scope, cb.field) + ";");
                    break;

                case GetBooleanField:
                    jp.println("boolValue_=_" + getFieldRef(cb.scope, cb.field) + ";");
                    break;

                case GetIteratorField:
                    emitSetupIterator(cb, true);
                    break;

                case EnterScope:
                    jp.print(cb.scope.scopeFieldName + "_=_");
                    String prefix = "";
                    String whoseFault = null;
                    if (cb.scope.scopeInstanceCreatorMethod != null) {
                        jp.print(getMethodRef(cb.scope.parent, cb.scope.scopeInstanceCreatorMethod)
+ "(");
                        whoseFault = relClassName(cb.scope.parent) + "." +
cb.scope.scopeInstanceCreatorMethod.getName() + "()";
                    } else if (Modifier.isStatic(cb.scope.scopeClass.getModifiers())) {
                        jp.print("new_" + relClassName(cb.scope) + "(");
                    } else if (cb.iterInfo != null) {
                        prefix = ",_";
                        jp.print(cb.iterInfo.constructorVar + ".newInstance(" +
cb.scope.parent.scopeFieldName + "");
                    } else {
                        prefix = ",_";
                        jp.print("cnstr" + cb.scope.scopeName + ".newInstance(" +
cb.scope.parent.scopeFieldName + "");
                    }
                    if (cb.iterInfo != null) {
                        jp.print(prefix);
                        jp.print(cb.iterInfo.itemVar);
                    }
                    jp.println(");");
                    if (whoseFault != null) {
                        jp.println("if_(" + cb.scope.scopeFieldName + "_==_null)");
```

```
                    jp.println("throw new NullPointerException(\"" + whoseFault +
"returned null\");");
                }
                break;

            case LeaveScope:
                jp.println(cb.scope.scopeFieldName + " = null;");
                break;

            case IteratorStep:
                jp.println("boolValue = " + cb.iterInfo.iterVar + ".hasNext();");
                jp.println("if (boolValue) {");
                jp.println(cb.iterInfo.itemVar + " = " + cb.iterInfo.iterVar + ".next();");
                jp.println("} else {");
                jp.println(cb.iterInfo.itemVar + " = null;");
                jp.println(cb.iterInfo.iterVar + " = null;");
                jp.println("}");
                break;

            default:
                throw new CompilerException("Invalid type = " + cb.callbackType);
            }
            jp.println("break;");
            jp.println();
            jp.indent--;
        }
        jp.println("default: throw new Exception(\"Unknown callback idx = \" + idx);");
        jp.println("}");
        jp.println("}");
    }

    protected void emitSetupIterator(Callback cb, boolean isField) throws CompilerException {
        String who = isField
                ? relClassName(cb.scope) + "." + cb.field.getName()
                : relClassName(cb.scope) + "." + cb.method.getName();
        String get = isField ? getFieldRef(cb.scope, cb.field) : getMethodRef(cb.scope, cb.method)
+ "()";
        String isNullTxt = isField ? " is null" : " returned null";
        if (cb.iterInfo.isIterable) {
            jp.println("{");
            jp.println("Iterable<" + relClassName(cb.iterInfo.itemType) + "> able = " + get +
";");
            jp.println("if (able == null)");
            jp.println("throw new NullPointerException(\"" + who + isNullTxt + "\");");
            jp.println(cb.iterInfo.iterVar + " = able.iterator();");
            jp.println("if (" + cb.iterInfo.iterVar + " == null)");
            jp.println("throw new NullPointerException(\"" + who +
".iterator() returned null\");");
            jp.println("}");
        } else {
            jp.println(cb.iterInfo.iterVar + " = " + get + ";");
            jp.println("if (" + cb.iterInfo.iterVar + "== null)");
            jp.println("throw new NullPointerException(\"" + who + isNullTxt + "\");");
        }
    }

    protected String getMethodRef(ScopeImpl scope, Method method) throws CompilerException {
        checkAccess(method.getDeclaringClass(), method.getModifiers(), method.getName());
        if (Modifier.isStatic(method.getModifiers()))
            return relClassName(scope) + "." + method.getName();
        else
            return scope.scopeFieldName + "." + method.getName();
    }

    protected String getFieldRef(ScopeImpl scope, Field field) throws CompilerException {
        checkAccess(field.getDeclaringClass(), field.getModifiers(), field.getName());
        if (Modifier.isStatic(field.getModifiers()))
            return relClassName(scope) + "." + field.getName();
        else
            return scope.scopeFieldName + "." + field.getName();
    }

    protected void checkAccess(Class<?> cls, int modifiers, String name) {
        if (isSamePackage(cls)) {
            if (Modifier.isPrivate(modifiers))
                log.log(LogLevel.Error, relClassName(cls) + "." + name +
" is not private: cannot be accessed");
        } else if (!Modifier.isPublic(modifiers)) {
            log.log(LogLevel.Error, relClassName(cls) + "." + name +
" is not public: cannot be accessed");
        }
    }

    static protected final Pattern javaLangPattern = Pattern.compile(
"(?<![\\.\\w$])(java\\.lang\\.)");

    protected String relClassName(String fcn) {
        if (isSamePackage(fcn))
            return StringUtils.zapPrefix(fcn, packageNameDot).replace('$', '.');
        fcn = javaLangPattern.matcher(fcn).replaceAll("");
        return fcn.replace('$', '.');
    }

    protected boolean isSamePackage(Class<?> cls) {
        return isSamePackage(cls.getName());
    }

    protected boolean isSamePackage(String s) {
        if (!s.startsWith(packageNameDot))
            return false;
        int idx = s.indexOf('$');
        if (idx > 0)
            s = s.substring(0, idx);
        return s.indexOf('.', packageNameDot.length()) < 0;
    }

    protected String relClassName(Class<?> cls) {
        if (cls == null)
            new Exception("cls is null").printStackTrace();
        return relClassName(cls.getName());
    }

    protected String relClassName(ScopeImpl scope) {
        if (scope == null)
            new Exception("scope is null").printStackTrace();
        return relClassName(scope.scopeClass);
    }
}
```

G.10 ProgramImpl

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

public class ProgramImpl implements Program {
    final CompilerLog log;
    final ScopeImpl rootScope;

    final StringTable stringTable;
    final List<Integer> opcodes;
    protected boolean isPostprocessed;
    int variableCount;
    private final Set<ProgramAddressImpl> nonpatchedAddresses = new
HashSet<ProgramAddressImpl>();

    protected StringBuilder textSB = new StringBuilder();
    protected boolean textIsCommitted = true;

    final Map<String, Callback> callbackMap = new HashMap<String, Callback>();
    final Map<String, List<ProgramAddressImpl>> callbackOpcodeAddrsMap = new
HashMap<String, List<ProgramAddressImpl>>();
    final LinkedList<Callback> subScopeStack = new LinkedList<Callback>();

    public ProgramImpl(CompilerLog log, ScopeImpl rootScope) {
        this.log = log;
        this.rootScope = rootScope;
        this.stringTable = new StringTable();
        this.opcodes = new ArrayList<Integer>();
    }

    public ProgramAddressImpl getNextAddress() throws CompilerException {
        checkModifyable(true);
        return new ProgramAddressImpl(opcodes.size());
    }

    public void addGoto(ProgramAddress destination) throws CompilerException {
        checkModifyable(true);
        ProgramAddressImpl dst = (ProgramAddressImpl) destination;
        opcodes.add(Opcode.GOTO.encoded(dst.idx));
    }

    public void addConditionalGoto(ProgramAddress destination) throws CompilerException {
        checkModifyable(true);
        ProgramAddressImpl dst = (ProgramAddressImpl) destination;
        opcodes.add(Opcode.CONDITIONAL_GOTO.encoded(dst.idx));
    }

    public PatchedGotoImpl addPatchedGoto() throws CompilerException {
        checkModifyable(true);
        return new PatchedGotoImpl(this, makePatchedOpcodeAddr(), Opcode.GOTO);
    }

    public PatchedGoto addPatchedConditionalGoto() throws CompilerException {
        checkModifyable(true);
        return new PatchedGotoImpl(this, makePatchedOpcodeAddr(),
Opcode.CONDITIONAL_GOTO);
    }

    public void addNegateBoolean() throws CompilerException {
        checkModifyable(true);
        opcodes.add(Opcode.NEGATE.encoded());
    }

    public void addEmitChar(char ch) throws CompilerException {
        checkModifyable(false);
        textIsCommitted = false;
        textSB.append(ch);
    }

    public void addEmitString(String s) throws CompilerException {
        checkModifyable(false);
        textIsCommitted = false;
        textSB.append(s);
    }

    public void addSoftNewline() throws CompilerException {
        checkModifyable(false);
        if (!endsWithSoftNewline())
            addEmitChar('\n');
    }

    private boolean endsWithSoftNewline() {
        if (textIsCommitted)
            return stringTable.isEmpty();
        else
            return textSB.length() > 0 && textSB.charAt(textSB.length() - 1) == '\n';
    }

    public void addCallback(Scope scope, Method method) throws CompilerException {
        makeCallback(scope, null, method, null, Callback.CallbackType.CallMethod);
    }

    public void addCallback(IteratorInfo iterInfo, Method method) throws CompilerException {
        makeCallback(iterInfo.scope, null, method, iterInfo, Callback.CallbackType.CallMethod);
    }

    public void addIteratorStep(IteratorInfo iterInfo) throws CompilerException {
        makeCallback(iterInfo.scope, null, null, iterInfo, Callback.CallbackType.IteratorStep);
    }

    public void addGetStringField(Scope scope, Field field) throws CompilerException {
        makeCallback(scope, field, null, null, Callback.CallbackType.GetStringField);
    }

    public void addGetBooleanField(Scope scope, Field field) throws CompilerException {
        makeCallback(scope, field, null, null, Callback.CallbackType.GetBooleanField);
    }

    public void addGetIntegerField(Scope scope, Field field) throws CompilerException {
        makeCallback(scope, field, null, null, Callback.CallbackType.GetIntegerField);
    }

    public void addGetIteratorField(Scope scope, Field field, IteratorInfo info) throws
CompilerException {
        makeCallback(scope, field, null, info, Callback.CallbackType.GetIteratorField);
    }

    public void addPrintHtmlEncodedStringResult() throws CompilerException {
        checkModifyable(true);
        opcodes.add(Opcode.PRINT_HTML_ENCODED_STRING_VAR.encoded());
    }

    public void addPrintAttrEncodedStringResult() throws CompilerException {
```

```
        checkModifyable(true);
        opcodes.add(Opcode.PRINT_ATTR_ENCODED_STRING_VAR.encoded());
    }

    public void addPrintRawStringResult() throws CompilerException {
        checkModifyable(true);
        opcodes.add(Opcode.PRINT_STRING_VAR.encoded());
    }

    public void addPrintIntegerResult() throws CompilerException {
        checkModifyable(true);
        opcodes.add(Opcode.PRINT_INT_VAR.encoded());
    }

    public String getNextVariableNamePrefix() {
        return "v" + (++variableCount);
    }

    public void addCreateScopeInstance(Scope scope) throws CompilerException {
        if (scope == null)
            throw new InternalCompilerException("scope == null");
        if (scope == rootScope)
            throw new InternalCompilerException("scope == rootScope");
        subScopeStack.add(makeCallback(scope, null, null, null,
Callback.CallbackType.EnterScope));
    }

    public void addCreateScopeInstance(IteratorInfo iterInfo, Scope subScope) throws
CompilerException {
        if (subScope == null)
            throw new InternalCompilerException("subScope == null");
        if (subScope == rootScope)
            throw new InternalCompilerException("subScope == rootScope");
        subScopeStack.add(makeCallback(subScope, null, null, iterInfo,
Callback.CallbackType.EnterScope));
    }

    public void addLeaveScopeInstance(Scope scope) throws CompilerException {
        if (subScopeStack.isEmpty())
            throw new InternalCompilerException("subScopeStack is empty");
        Callback inner = subScopeStack.getLast();
        if (inner.scope != scope)
            throw new InternalCompilerException("leaving non-current scope = " +
scope.scopeName + ", current = " + inner.scope.scopeName);
        makeCallback(inner.scope, null, null, inner.iterInfo, Callback.CallbackType.LeaveScope);
    }

    public void addReturn() throws CompilerException {
        checkModifyable(true);
        opcodes.add(Opcode.RETURN.encoded());
    }

    protected ProgramAddressImpl makePatchedOpcodeAddr() throws CompilerException {
        checkModifyable(true);
        ProgramAddressImpl addr = getNextAddress();
        opcodes.add(-1);
        nonpatchedAddresses.add(addr);
        //log.log(LogLevel.Notice, "makePatchedOpcodeAddr: " + addr);
        return addr;
    }

    void patchToOpcode(ProgramAddressImpl addr, int encoded) throws CompilerException {
        checkModifyable(true);
        //log.log(LogLevel.Notice, "patches " + addr.idx + " to " + encoded);
        if (!nonpatchedAddresses.contains(addr))
            throw new CompilerException("Patching non-open address");
        if (opcodes.get(addr.idx) != -1)
            throw new CompilerException("Patching open address, but has encoded = " +
opcodes.get(addr.idx));
        if (!nonpatchedAddresses.remove(addr))
            throw new CompilerException("nonpatchedAddresses did not remove " + addr);
        opcodes.set(addr.idx, encoded);
    }

    void patchToCurrentAddress(ProgramAddressImpl opcodeAddr, Opcode opcode) throws
CompilerException {
        checkModifyable(true);
        patchToOpcode(opcodeAddr, opcode.encoded(getNextAddress().idx));
    }

    protected Callback makeCallback(Scope scope, Field field, Method method, IteratorInfo
iteratorInfo,
    Callback.CallbackType callbackType) throws CompilerException {
        checkModifyable(true);
        String key = scope.scopeFieldName + ",  " + callbackType.name();
        if (iteratorInfo != null)
            key += ",  " + iteratorInfo.itemVar;
        if (method != null)
            key += ",  " + method.getName();
        if (field != null)
            key += ",  " + field.getName();
        Callback cb = callbackMap.get(key);
        List<ProgramAddressImpl> opcodeAddrsList;
        if (cb == null) {
            cb = new Callback(scope, field, method, iteratorInfo, callbackType, key);
            callbackMap.put(key, cb);
            opcodeAddrsList = new ArrayList<ProgramAddressImpl>();
            callbackOpcodeAddrsMap.put(key, opcodeAddrsList);
        } else {
            opcodeAddrsList = callbackOpcodeAddrsMap.get(key);
        }
        opcodeAddrsList.add(makePatchedOpcodeAddr());
        return cb;
    }

    protected void checkModifyable(boolean commitText) throws CompilerException {
        if (isPostprocessed)
            throw new InternalCompilerException(
"Can't modify program at this point: it is already post-processed");
        if (!commitText || textIsCommitted)
            return;
        int idx = stringTable.addText(textSB.toString());
        textSB.setLength(0);
        opcodes.add(Opcode.PRINT_STRINGTABLE.encoded(idx));
        textIsCommitted = true;
    }

    public ProgramData postProcess() throws CompilerException {
        checkModifyable(true);
```

```
        Callback[] callbacks = callbackMap.values().toArray(new Callback[callbackMap.size()]);
        Arrays.sort(callbacks, new Comparator<Callback>() {
            public int compare(Callback a, Callback b) {
                return a.key.compareTo(b.key);
            }
        });
        for (int i = 0; i < callbacks.length; i++) {
            for (ProgramAddressImpl addr : callbackOpcodeAddrsMap.get(callbacks[i].key))
                patchToOpcode(addr, Opcode.CALLBACK.encoded(i));
        }
        if (!nonpatchedAddresses.isEmpty()) {
            List<ProgramAddressImpl> list = new
ArrayList<ProgramAddressImpl>(nonpatchedAddresses);
            Collections.sort(list);
            throw new CompilerException(nonpatchedAddresses.size() +
"␣patched-goto's␣have␣not␣been␣patched:␣" + list);
        }
        int[] ops = new int[opcodes.size()];
        for (int idx = 0; idx < ops.length; idx++)
            ops[idx] = opcodes.get(idx);
        isPostprocessed = true;
        List<ScopeImpl> scopes = new ArrayList<ScopeImpl>();
        getFlatScopeList(scopes, rootScope);
        Collections.sort(scopes, new Comparator<ScopeImpl>() {
            public int compare(ScopeImpl a, ScopeImpl b) {
                return a.scopeFieldName.compareTo(b.scopeFieldName);
            }
        });
        return new ProgramData(log, ops, stringTable.toArray(),
                rootScope, scopes.toArray(new ScopeImpl[scopes.size()]),
                callbacks);
    }

    protected void getFlatScopeList(List<ScopeImpl> list, ScopeImpl scope) {
        list.add(scope);
        for (ScopeImpl sc : scope.iterSubScopes())
            getFlatScopeList(list, sc);
    }
}
```

G.11 StringTable

```
package dk.vitality.util.jat.translator.compiler.codeGeneration.impl;

public class StringTable {
    protected List<String> list = new ArrayList<String>();
    protected Map<String, Integer> map = new HashMap<String, Integer>();

    /**
    * @return index of string
    */
    public int addText(String s) {
        Integer idx = map.get(s);
        if (idx != null)
            return idx;
        idx = list.size();
        list.add(s);
        map.put(s, idx);
        return idx;
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public int size() {
        return list.size();
    }

    public String[] toArray() {
        return list.toArray(new String[list.size()]);
    }
}
```

Appendix H

translator.compiler.htmlLoader

H.1 HtmlDocument

```
package dk.vitality.util.jat.translator.compiler.htmlLoader;

public class HtmlDocument {
    public final FileInfo fileInfo;
    public final Document doc;
    public final Element rootE;
    public String docType;

    public HtmlDocument(File f, long moddate, Document doc, String docType) {
        fileInfo = new FileInfo(f, moddate);
        this.doc = doc;
        this.rootE = doc.getDocumentElement();
        this.docType = docType;
    }

    public List<Element> findMatchingElements(String specification) {
        List<Element> list = new ArrayList<Element>();
        findMatchingElements(list, rootE, specification);
        return list;
    }

    private void findMatchingElements(List<Element> list, Element e, String specification) {
        if (specification.equals(e.getAttribute("id"))) {
            list.add(e);
        } else {
            for (String s :
CompilerUtils.splitCssClassNamesWithoutPrefixedUnderscore(e.getAttribute("class"))) {
                if (specification.equals(s)) {
                    list.add(e);
                    break;
                }
            }
        }
        for (Node child = e.getFirstChild(); child != null; child = child.getNextSibling()) {
            if (child.getNodeType() == Node.ELEMENT_NODE)
                findMatchingElements(list, (Element) child, specification);
        }
    }
}
```

H.2 HtmlLoader

```
package dk.vitality.util.jat.translator.compiler.htmlLoader;

public class HtmlLoader {
    public final CompilerLog log;

    public HtmlLoader(CompilerLog log) {
        this.log = log;
    }

    public HtmlDocument load(File htmlF) throws CompilerException {
        try {
            long moddate = htmlF.lastModified();
            StringWriter errorSW = new StringWriter();
            PrintWriter errorPW = new PrintWriter(errorSW);
            Document tidyDoc = isEmptyTemplate(htmlF) ? null : runJTidy(htmlF, errorPW);
            Document xercesDoc = importIntoXerces(tidyDoc);
            errorPW.flush();
            String warnings = processTidyWarnings(errorSW.toString());
            if (warnings != null)
                log.log(LogLevel.Notice, warnings);
            String docType = (tidyDoc != null && tidyDoc.getDoctype() != null) ?
tidyDoc.getDoctype().getName() : null;
            return new HtmlDocument(htmlF, moddate, xercesDoc, docType);
        }
        catch (CompilerException ex) {
            throw ex;
        }
        catch (Exception ex) {
```

```
            log.log(LogLevel.Error, "Can't load html template: " +
ExceptionUtils.getSomeMessage(ex) + "\n" +
                    "   " + htmlF.getPath());
            System.err.println("Load_html_exception: " + ex.getMessage());
            ex.printStackTrace();
            return null;
        }
    }

    protected Document runJTidy(File htmlF, PrintWriter errorPW) throws Exception {
        Tidy t = new Tidy();
        t.setDropEmptyParas(false);
        t.setCharEncoding(Configuration.UTF8);
        BufferedInputStream is = new BufferedInputStream(new FileInputStream(htmlF), 4096);
        t.setErrout(errorPW);
        t.setShowWarnings(false);
        t.setDropEmptyParas(false);
        Document tidyDoc;
        try {
            tidyDoc = t.parseDOM(is, null);
        }
        finally {
            is.close();
        }
        errorPW.close();
        return tidyDoc;
    }

    /**
    * JTidy crashes in Node.getAttributes() with NullPointerException
    * if the template only contains whitespace....
    */
    protected boolean isEmptyTemplate(File htmlF) throws Exception {
        if (htmlF.length() > 1000)
            return false;
        String s = new String(FileContentsUtils.readFile(htmlF), "ISO-8859-1");
        return s.trim().length() == 0;
    }

    protected Document importIntoXerces(Document tidyDoc) throws
ParserConfigurationException {
        // Tidy includes a very stupid DOM implementation, so move it
        // to Xerces, so we have stuff like importNode....
        // but Xerces crashes in importNode due to the stupid JTidy DOM implementation,
        // so this is done manually...
        Document xercesDoc = CompilerUtils.createDocument();
        Element xercesRoot = xercesDoc.createElement("html");
        xercesDoc.appendChild(xercesRoot);
        if (tidyDoc != null)
            importElementData(xercesDoc, xercesRoot, tidyDoc.getDocumentElement(),
xercesRoot);
        return xercesDoc;
    }

    protected void importTidyElement(Document dstDoc, Node dstParent, Node srcNode) throws
DOMException {
        switch (srcNode.getNodeType()) {
            case Node.CDATA_SECTION_NODE:
                dstParent.appendChild(dstDoc.createCDATASection(srcNode.getNodeValue()));
                return;
            case Node.COMMENT_NODE:
                dstParent.appendChild(dstDoc.createComment(srcNode.getNodeValue()));
                return;
            case Node.TEXT_NODE:
                dstParent.appendChild(dstDoc.createTextNode(srcNode.getNodeValue()));
                return;
            case Node.ELEMENT_NODE:
                break;
            default:
                throw new DOMException(DOMException.NOT_SUPPORTED_ERR,
"Invalid_HTML_element_type_=_" + srcNode.getNodeType());
        }
        Element e = dstDoc.createElement(srcNode.getNodeName().toLowerCase());
        dstParent.appendChild(e);
        importElementData(dstDoc, dstParent, srcNode, e);
    }

    protected void importElementData(Document dstDoc, Node dstParent, Node srcNode, Element
e) throws DOMException {
        NamedNodeMap nnm = srcNode.getAttributes();
        int idx = 0;
        while (true) {
            Node n = nnm.item(idx++);
            if (n == null)
                break;
            e.setAttribute(n.getNodeName().toLowerCase(), n.getNodeValue());
        }
        Node child = srcNode.getFirstChild();
        while (child != null) {
            importTidyElement(dstDoc, e, child);
            child = child.getNextSibling();
        }
    }

    /**
    * JTidy makes very verbose warnings. Zap the less interesting ones.
    */
    protected String processTidyWarnings(String msgs) {
        for (Pattern re : documentZapsRE)
            msgs = re.matcher(msgs).replaceAll("");
        msgs = tidyEmptyLinesRE.matcher(msgs).replaceAll("\n").trim();
        return (msgs.length() == 0) ? null : msgs;
    }

    static final protected Pattern[] documentZapsRE;
    static final protected Pattern tidyEmptyLinesRE;
    static protected final String warn = "line_\\d+_column_\\d+(,\\d+)*_-_Warning:_";

    static {
        documentZapsRE = new Pattern[]{
                r("Tidy_!(vers_4th_August_2000!)_Parsing.*"),
                r(warn + "<table>_lacks_.summary._attribute!s*"),
                r(warn + "<img>_lacks_.alt._attribute!s*"),
                r(warn + "unescaped_&_or_unknown_entity_.&!w+.!s*"),
                r("InputStream:_Doctype_given_is.*"),
                r("InputStream:_Document_content_looks_like_HTML.*"),
```

```
            rdn("The␣table␣summary␣attribute␣should␣be␣used" +
                  ".*?a␣meaningful␣context␣for␣each␣cell!."),
            rdn("The␣alt␣attribute␣should␣be␣used" +
                  ".*?longdesc␣attribute␣which␣takes␣a␣URL␣linked␣to␣the␣description!."),
            r("These␣measures␣are␣needed␣for␣people␣using␣non-graphical␣browsers!."),
            rdn("For␣further␣advice␣on␣how␣to␣make␣your␣pages␣accessible" +
                  ".*?service␣for␣checking␣URLs␣for␣accessibility!."),
            r("!d+␣warnings/errors␣were␣found."), // ender med !
            r("no␣warnings␣or␣errors␣were␣found"),
            rdn("You␣are␣recommended␣to␣use␣CSS␣to␣specify.*?using␣<FONT>␣elements!.")
      };
      tidyEmptyLinesRE = Pattern.compile("\\n\\s+");
   }

   private static Pattern r(String s) {
      return Pattern.compile(s.replace('!', '\\'));
   }

   private static Pattern rdn(String s) {
      return Pattern.compile(s.replace('!', '\\'), Pattern.DOTALL);
   }
}
```

Appendix I

translator.compiler.tree

I.1 AbstractEmitter

```
package dk.vitality.util.jat.translator.compiler.tree;

abstract public class AbstractEmitter<AP extends Application> {
    public final Program program;
    public final AP handler;

    protected AbstractEmitter(Program program, AP handler) {
        this.program = program;
        this.handler = handler;
    }

    public String annotationToString() {
        return handler.annotationToString();
    }
}
```

I.2 AbstractSubAttributesHandler

```
package dk.vitality.util.jat.translator.compiler.tree;

abstract class AbstractSubAttributesHandler {
    final Program program;
    final CompilerLog log;
    final List<SubAttributeEmitter<?>> subEmitters;

    protected AbstractSubAttributesHandler(Program program, CompilerLog log,
    List<SubAttributeEmitter<?>> subEmitters) throws InternalCompilerException {
        this.program = program;
        this.log = log;
        this.subEmitters = subEmitters;
        if (subEmitters.isEmpty())
            throw new InternalCompilerException("empty_list");
    }

    abstract void main(String attrValue) throws CompilerException;
}
```

I.3 ClassSubAttributesHandler

```
package dk.vitality.util.jat.translator.compiler.tree;

class ClassSubAttributesHandler extends AbstractSubAttributesHandler {
    protected ClassSubAttributesHandler(Program program, CompilerLog log,
    List<SubAttributeEmitter<?>> subEmitters) throws InternalCompilerException {
        super(program, log, subEmitters);
    }

    void main(String attrValue) throws CompilerException {
        String itemPrefix = "␣" + subEmitters.get(0).lcAttributeName + "='";
        if (attrValue.length() > 0) {
            Set<String> handledItems = new HashSet<String>();
            for (SubAttributeEmitter<?> p : subEmitters) {
                for (String s : p.lcSubAttributeNames)
                    handledItems.add(s);
            }
            for (String s : CompilerUtils.splitCssClassNames(attrValue)) {
                if (s.startsWith("-"))
                    continue;
                if (handledItems.contains(s))
                    continue;
                program.addEmitString(itemPrefix);
                itemPrefix = "␣";
                program.addEmitString(s);
            }
        }
        if (subEmitters.size() == 1 && itemPrefix.length() > 1 &&
subEmitters.get(0).canGenerateFullAttribute()) {
            // if only one conditional and we have no static classnames, then make the complete
element
```

```
            // attribute inside the conditional so we avoid "class="" when the conditional is false
            subEmitters.get(0).generateFullAttribute();
        } else {
            if (itemPrefix.length() > 1) {
                program.addEmitString(itemPrefix);
                itemPrefix = "";
            }
            for (SubAttributeEmitter<?> p : subEmitters) {
                p.generateSubAttribute(itemPrefix);
                itemPrefix = "␣";
            }
            program.addEmitString("'");
        }
    }
}
```

I.4 ElementEmitters

```
package dk.vitality.util.jat.translator.compiler.tree;

public class ElementEmitters {
    public final CompilerLog log;
    public final Program program;
    public boolean allowIdAttribute;
    public boolean shouldProcessElement;

    public final List<EIfApplication.Emitter> eConditionals = newList();
    public final List<ETagIfApplication.Emitter> eTag = newList();
    public final List<EContentIfApplication.Emitter> eContentIf = newList();
    public final List<EContentPrePostApplication.Emitter> ePrePostContent = newList();

    public ElementLoopEmitter<?> eLoop;
    public EReplaceApplication.Emitter eReplacer;
    public EScopedApplication.Emitter eScope;
    public EContentApplication.Emitter eContent;

    public final Map<String, AttributeEmitter<?>> eAttributes = newMap();
    public final Map<String, PostAttributeEmitter<?>> ePostAttributes = newMap();
    public final Map<String, List<SubAttributeEmitter<?>>> eSubAttributes = newMap();

    final Element e;
    final String nodeName;
    final int level;
    final boolean debugFlag;

    ScopeLink scopeLink;

    ElementEmitters(CompilerLog log, boolean debugFlag, Program program,
    ScopeLink scopeLink,
    Element e, int level,
    boolean allowIdAttribute) throws CompilerException {
        this.log = log;
        this.debugFlag = debugFlag;
        this.program = program;
        this.e = e;
        this.level = level;
        this.nodeName = e.getNodeName();
        if (debugFlag)
            log.log(LogLevel.Verbose, indent() + "Element:␣" + nodeName);
        this.shouldProcessElement = true;
        this.allowIdAttribute = allowIdAttribute;
        this.scopeLink = scopeLink;
        List<Application> applicationList = scopeLink.scope.getApplicationsForElement(e, true);
        for (Application processor : applicationList) {
            if (processor instanceof EZapApplication) {
                shouldProcessElement = false;
                return;
            }
        }
        for (Application appl : applicationList) {
            if (debugFlag)
                log.log(LogLevel.Verbose, indent() + "␣␣␣␣␣" + appl.getClass().getSimpleName());
            appl.createEmitter(this, e);
        }
    }

    public boolean alreadyDefined(AbstractEmitter<?> previous, AbstractEmitter<?> emitter,
String what) {
        if (previous == null)
            return false;
        error("" +
                "Multiple␣" + what + "␣annotations␣for␣same␣element␣are␣not␣allowed:\n" +
                "␣␣" + previous.annotationToString() + "\n" +
                "␣␣" + emitter.handler.annotationToString());
        return true;
    }

    public boolean enterSubScope(AbstractEmitter<?> emitter, Scope subScope) {
        if (subScope == null)
            return true;
        // Making non-used warnings become complex when getProcessors... is split over two
scopes...
        //List<Application> additional = emitter.scope.getProcessorsForElement(e, false);
        //System.err.println("subscope " + usages.size() + " + " + additional.size() + ": " +
usage.processor.annotationToString());
        //processors.addAll(additional);
        StringBuilder tooDeepSB = (scopeLink.level > 25) ? new StringBuilder() : null;
        ScopeLink sl = scopeLink;
        while (sl != null) {
            if (tooDeepSB != null) {
                tooDeepSB.append(".....Level␣").append(sl.level).append(":␣");
                tooDeepSB.append(sl.annotDescription).append("\n");
            }
            if (sl.scope == subScope) {
                error("Scopes␣are␣not␣reentrant:\n" +
                        "......Entered␣" + sl.scope.scopeName + "␣using␣" + sl.annotDescription +
"\n" +
                        "......Reenters␣" + subScope.scopeName + "␣using␣" +
emitter.annotationToString());
                return false;
            }
            sl = sl.prev;
        }
        if (tooDeepSB != null) {
```

```
            error("Too deep scope nesting (" + scopeLink.level + " levels!):\n" + tooDeepSB);
            return false;
        }
        scopeLink = new ScopeLink(scopeLink, subScope, emitter.annotationToString());
        return true;
    }

    protected String indent() {
        char[] ch = new char[level];
        Arrays.fill(ch, ' ');
        return new String(ch);
    }

    protected void error(String msg) {
        attrError(null, msg);
    }

    protected void attrError(String attrName, String msg) {
        StringBuilder sb = new StringBuilder();
        makeLocation(sb, e);
        if (attrName != null)
            sb.append("/@").append(attrName);
        sb.append(": ").append(msg);
        log.log(LogLevel.Error, sb.toString());
    }

    /**
    * Appends path to current node to the StringBuilder, used for error reporting.
    */
    protected void makeLocation(StringBuilder sb, Node e) {
        Node parent = e.getParentNode();
        if (parent != null && parent.getNodeType() != Node.DOCUMENT_NODE) {
            makeLocation(sb, parent);
            sb.append("/");
        }
        sb.append(e.getNodeName());
        if (e.getNodeType() != Node.ELEMENT_NODE)
            sb.append("[type=").append(e.getNodeType()).append("]");
    }

    static private <T> List<T> newList() {
        return new ArrayList<T>();
    }

    static private <K, V> Map<K, V> newMap() {
        return new HashMap<K, V>();
    }
}
```

I.5 ScopeLink

```
package dk.vitality.util.jat.translator.compiler.tree;

class ScopeLink {
    final ScopeLink prev;
    final Scope scope;
    final String annotDescription;
    final int level;

    ScopeLink(ScopeLink prev, Scope scope, String annotDescription) {
        this.prev = prev;
        this.scope = scope;
        this.annotDescription = annotDescription;
        this.level = (prev != null) ? prev.level + 1 : 0;
    }
}
```

I.6 StyleSubAttributesHandler

```
package dk.vitality.util.jat.translator.compiler.tree;

class StyleSubAttributesHandler extends AbstractSubAttributesHandler {
    StyleSubAttributesHandler(Program program, CompilerLog log,
    List<SubAttributeEmitter<?>> subEmitters) throws InternalCompilerException {
        super(program, log, subEmitters);
    }

    String itemPrefix;

    void main(String attrValue) throws CompilerException {
        itemPrefix = " " + subEmitters.get(0).lcAttributeName + "='";
        if (attrValue.length() > 0) {
            Set<String> handledItems = new HashSet<String>();
            for (SubAttributeEmitter<?> p : subEmitters) {
                for (String s : p.lcSubAttributeNames)
                    handledItems.add(s);
            }
            new StyleParser(attrValue, handledItems);
        }
        if (subEmitters.size() == 1 && !itemPrefix.contains(";") &&
subEmitters.get(0).canGenerateFullAttribute()) {
            // if only one conditional and we have no static value, then make the complete element
            // attribute inside the conditional so we avoid "style="" when the conditional is false
            subEmitters.get(0).generateFullAttribute();
        } else {
            program.addEmitString(itemPrefix);
            itemPrefix = "";
            for (SubAttributeEmitter<?> p : subEmitters) {
                p.generateSubAttribute(itemPrefix);
                itemPrefix = "; ";
            }
        }
        if (itemPrefix.contains(";"))
            program.addEmitString("'");
    }

    protected void parseStyles(String attrValue) throws CompilerException {
        try {
            int idx = 0;
            while (idx < attrValue.length()) {
                while (idx < attrValue.length() && Character.isWhitespace(attrValue.charAt(idx)))
                    idx++;
            }
```

```
        }
        catch (Exception ex) {
            log.log(LogLevel.Error, "Can't parse style='" + attrValue + "': " +
ExceptionUtils.getSomeMessage(ex));
        }
    }

    final class StyleParser {
        final String attrValue;
        int idx;

        StyleParser(String attrValue, Set<String> handledItems) {
            this.attrValue = attrValue;
            try {
                while (true) {
                    char ch = next(null);
                    if (ch == 0)
                        break;
                    if (Character.isWhitespace(ch))
                        continue;
                    String name = parseName(ch);
                    String value = parseValue(name);
                    if (handledItems.contains(name.toLowerCase()))
                        continue;
                    program.addEmitString(itemPrefix);
                    itemPrefix = "; ";
                    program.addEmitString(name);
                    program.addEmitString(": ");
                    program.addEmitString(value);
                }
            }
            catch (Exception ex) {
                log.log(LogLevel.Error, "Can't parse style='" + attrValue + "': " +
ExceptionUtils.getSomeMessage(ex));
            }
        }

        void skipWhiteSpace() {
            while (idx < attrValue.length() && Character.isWhitespace(attrValue.charAt(idx)))
                idx++;
        }

        char next(String msg) throws CompilerException {
            if (idx < attrValue.length())
                return attrValue.charAt(idx++);
            if (msg != null)
                error(msg);
            return 0;
        }

        String parseName(char ch) throws CompilerException {
            StringBuilder sb = new StringBuilder();
            do {
                sb.append(ch);
                ch = next("end of style attribute when expecting ':' after name");
                if (ch == ';')
                    error("';' when expecting ':' after name");
            } while (ch != ':');
            String s = sb.toString().trim();
            for (int i = 0; i < s.length(); i++) {
                ch = s.charAt(i);
                if (ch != '-' && !Character.isLetter(ch)) {
                    log.log(LogLevel.Notice, "Weird css style name: '" + s + "'");
                    break;
                }
            }
            return s;
        }

        String parseValue(String name) throws CompilerException {
            StringBuilder sb = new StringBuilder();
            while (true) {
                char ch = next(null);
                if (ch == 0)
                    break;
                if (ch <= 32 && sb.length() == 0)
                    continue;
                if (ch == ';')
                    break;
                sb.append(ch);
                if (ch == '\\') {
                    sb.append(next("backslash-escaped character"));
                    continue;
                }
                if (ch != '"' && ch != '\'')
                    continue;
                char quote = ch;
                do {
                    ch = next("inside string");
                    sb.append(ch);
                    if (ch == '\\')
                        sb.append(next("backslash-escaped character inside string"));
                } while (ch != quote);
            }
            String s = sb.toString().trim();
            if (s.length() == 0)
                error("empty style value for '" + name + "'");
            return s;
        }

        void error(String msg) throws CompilerException {
            throw new CompilerException(msg + " at character " + idx + " of " +
attrValue.length());
        }

    }
}
```

I.7 TreeEmit

```
packagedk.vitality.util.jat.translator.compiler.tree;

public class TreeEmit {
    public final Program program;
```

```
    public final CompilerLog log;

    boolean debugFlag = false;

    public TreeEmit(Program program, CompilerLog log) {
        this.program = program;
        this.log = log;
    }

    public void emit(HtmlDocument hfi, boolean includeDoctype, Node rootE, Scope scope, String
annotDescription) throws CompilerException {
        if (includeDoctype && hfi.docType != null) {
            program.addEmitString("<!DOCTYPE␣");
            program.addEmitString(hfi.docType);
            program.addEmitString(">\n");
        }
        emitNode(rootE, new ScopeLink(null, scope, annotDescription), 0, "", true);
    }

    protected void emitNode(Node n, ScopeLink scopeLink, int level, String indent, boolean
allowIdAttribute) throws CompilerException {
        switch (n.getNodeType()) {
            case Node.ELEMENT_NODE:
                emitElement((Element) n, scopeLink, level, indent, allowIdAttribute);
                break;
            case Node.TEXT_NODE:
                emitTextNode(n);
                break;
            case Node.COMMENT_NODE:
                program.addEmitString("<!--");
                program.addEmitString(n.getNodeValue());
                program.addEmitString("-->");
                break;
            default:
                error(n, "Unsupported␣node␣type␣for␣html");
        }
    }

    protected void emitElement(Element e, ScopeLink scopeLink, int level, String indent, boolean
allowIdAttribute) throws CompilerException {
        String nodeName = e.getNodeName();
        if (debugFlag)
            log.log(LogLevel.Verbose, indent + "Element:␣" + nodeName);
        ElementEmitters emitters = new ElementEmitters(log, debugFlag, program, scopeLink, e,
level, allowIdAttribute);
        if (!emitters.shouldProcessElement)
            return;
        for (EIfApplication.Emitter emit : emitters.eConditionals)
            emit.preElement();
        if (emitters.eLoop != null)
            emitters.eLoop.preElement();
        if (emitters.eScope != null)
            emitters.eScope.preElement();
        if (HtmlTags.elementsPrependNewline.contains(nodeName))
            program.addSoftNewline();
        if (emitters.eReplacer != null) {
            emitters.eReplacer.generateElement();
        } else {
            for (ETagIfApplication.Emitter emit : emitters.eTag)
                emit.preElementOpenTag();
            program.addEmitString("<");
            program.addEmitString(nodeName);
            emitAttributes(e, scopeLink.scope, level, emitters.allowIdAttribute, emitters);
            program.addEmitString(">");
            Collections.reverse(emitters.eTag);
            for (ETagIfApplication.Emitter emit : emitters.eTag)
                emit.postElementOpenTag();
            if (!HtmlTags.emptyElements.contains(nodeName)) {
                for (EContentIfApplication.Emitter emit : emitters.eContentIf)
                    emit.preElementContent();
                for (EContentPrePostApplication.Emitter emit : emitters.ePrePostContent)
                    emit.preContent();
                if (emitters.eContent != null) {
                    emitters.eContent.generateContent();
                } else {
                    String childIndent = indent + "␣␣";
                    for (Node child = e.getFirstChild(); child != null; child = child.getNextSibling())
                        emitNode(child, emitters.scopeLink, level + 1, childIndent,
emitters.allowIdAttribute);
                }
                Collections.reverse(emitters.ePrePostContent);
                for (EContentPrePostApplication.Emitter emit : emitters.ePrePostContent)
                    emit.postContent();
                Collections.reverse(emitters.eContentIf);
                for (EContentIfApplication.Emitter emit : emitters.eContentIf)
                    emit.postElementContent();
                for (ETagIfApplication.Emitter emit : emitters.eTag)
                    emit.preElementCloseTag();
                program.addEmitString("</");
                program.addEmitString(nodeName);
                program.addEmitString(">");
                Collections.reverse(emitters.eTag);
                for (ETagIfApplication.Emitter emit : emitters.eTag)
                    emit.postElementCloseTag();
            }
        }
        if (HtmlTags.elementsAppendNewline.contains(nodeName))
            program.addSoftNewline();
        if (emitters.eScope != null)
            emitters.eScope.postElement();
        if (emitters.eLoop != null)
            emitters.eLoop.postElement();
        for (EIfApplication.Emitter emit : emitters.eConditionals)
            emit.postElement();
    }

    protected void emitAttributes(Element e, Scope scope, int level, boolean allowIdAttribute,
    ElementEmitters emitters) throws CompilerException {
        // make union set of attributes specified on the element and those we have emitters for.
        Set<String> sortedAttributeNames = new TreeSet<String>();
        NamedNodeMap map = e.getAttributes();
        int num = map.getLength();
        for (int ai = 0; ai < num; ai++)
```

```
            sortedAttributeNames.add(map.item(ai).getNodeName());
        sortedAttributeNames.addAll(emitters.eAttributes.keySet());
        sortedAttributeNames.addAll(emitters.eSubAttributes.keySet());
        for (String attrName : sortedAttributeNames) {
            AttributeEmitter<?> valueEmitter = emitters.eAttributes.get(attrName);
            PostAttributeEmitter<?> postValueEmitter = emitters.ePostAttributes.get(attrName);
            List<SubAttributeEmitter<?>> subEmitters = emitters.eSubAttributes.get(attrName);
            emitOneAttribute(e, attrName, allowIdAttribute, valueEmitter, postValueEmitter,
subEmitters);
        }
    }

    protected void emitOneAttribute(Element e, String attrName, boolean allowIdAttribute,
    AttributeEmitter<?> valueEmitter,
    PostAttributeEmitter<?> postValueEmitter,
    List<SubAttributeEmitter<?>> subEmitters)
            throws CompilerException {
        String value = e.getAttribute(attrName);
        if (attrName.equals("id") && valueEmitter == null) {
            // if we have a processor, it is probably for generating unique id's, so keep it.
            if (!allowIdAttribute || value.startsWith("_"))
                return;
        }
        boolean empty = HtmlTags.emptyAttributes.contains(attrName);
        if (valueEmitter != null) {
            if (postValueEmitter != null) {
                attrError(e, attrName, "Having␣both␣plain␣attribute␣annotation␣" +
                        "and␣post-content␣attribute␣annotation␣is␣not␣allowed");
            }
            if (subEmitters != null) {
                attrError(e, attrName, "Having␣both␣plain␣attribute␣annotation␣" +
                        "and␣sub-attribute␣annotation␣is␣not␣allowed");
            }
            if (valueEmitter.processed) // attribute collection which is already generated.
                return;
            valueEmitter.processed = true;
            valueEmitter.generateAttribute();
            return;
        }
        if (postValueEmitter != null) {
            if (empty)
                attrError(e, attrName, "Can't␣make␣post-content␣for␣empty␣(boolean)␣attribute!");
            if (subEmitters != null) {
                attrError(e, attrName, "Having␣both␣post-content␣attribute␣annotation␣" +
                        "and␣sub-attribute␣annotation␣is␣not␣allowed");
            }
        }
        if (subEmitters != null) {
            if (empty) {
                attrError(e, attrName, "Can't␣make␣sub-attrs␣for␣empty␣(boolean)␣attribute");
                return;
            }
            if (attrName.equals("class"))
                new ClassSubAttributesHandler(program, log, subEmitters).main(value);
            else if (attrName.equals("style"))
                new StyleSubAttributesHandler(program, log, subEmitters).main(value);
            else
                attrError(e, attrName, "Internal␣error:␣Unsupported␣sub␣attr␣name:␣" +
attrName);
            return;
        }
        if (empty) {
            program.addEmitString("␣");
            program.addEmitString(attrName);
            return;
        }
        if (attrName.equals("class")) {
            value = CompilerUtils.zapUnderscorePrefixedClassNames(value);
            if (postValueEmitter == null && value.length() == 0)
                return;
        }
        program.addEmitString("␣");
        program.addEmitString(attrName);
        if (postValueEmitter != null) {
            // always quote
            program.addEmitString("='");
            postValueEmitter.generatePostContent(value);
            program.addEmitString("'");
            return;
        }
        if (value == null || value.length() == 0) {
            program.addEmitString("=''");
        } else if (!HtmlAttr.needsQuoting(value)) {
            program.addEmitString("=");
            program.addEmitString(value);
        } else {
            program.addEmitString("=");
            program.addEmitString(HtmlAttr.quoted(value));
        }
    }

    protected void error(Node e, String msg) {
        attrError(e, null, msg);
    }

    protected void attrError(Node e, String attrName, String msg) {
        StringBuilder sb = new StringBuilder();
        makeLocation(sb, e);
        if (attrName != null)
            sb.append("/@").append(attrName);
        sb.append(":␣").append(msg);
        log.log(LogLevel.Error, sb.toString());
    }

    /**
    * Appends path to current node to the StringBuilder, used for error reporting.
    */
    protected void makeLocation(StringBuilder sb, Node e) {
        Node parent = e.getParentNode();
        if (parent != null && parent.getNodeType() != Node.DOCUMENT_NODE) {
            makeLocation(sb, parent);
            sb.append("/");
        }
```

```
        sb.append(e.getNodeName());
        if (e.getNodeType() != Node.ELEMENT_NODE)
            sb.append("[type=").append(e.getNodeType()).append("]");
    }

    protected void emitTextNode(Node n) throws CompilerException {
        Node parent = n.getParentNode();
        if (parent != null && parent.getNodeType() == Node.ELEMENT_NODE) {
            String name = parent.getNodeName();
            if (name.equals("style")) {
                emitStyleTagContent(n);
                return;
            }
            if (name.equals("script")) {
                emitScriptTagContent(n);
                return;
            }
        }
        program.addEmitString(n.getNodeValue());
    }

    /**
    * Special handling of &lt;style&gt;...&lt;/style&gt; content: prepend newline and remove
    * leading spaces from each line. End with newline.
    */
    protected void emitStyleTagContent(Node n) throws CompilerException {
        String s = n.getNodeValue().trim();
        if (s.length() == 0)
            return;
        program.addEmitChar('\n');
        boolean afterLF = true;
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (ch == '\n') {
                afterLF = true;
            } else if (ch <= 32 && afterLF) {
                continue;
            } else {
                afterLF = false;
            }
            program.addEmitChar(ch);
        }
        program.addEmitChar('\n');
    }

    /**
    * Special handling of &lt;script&gt;...&lt;/script&gt; content if it is not empty:
    * prepend newline and append newline.
    */
    protected void emitScriptTagContent(Node n) throws CompilerException {
        String s = n.getNodeValue();
        if (s.length() == 0)
            return;
        if (s.indexOf('\n') < 0) {
            program.addEmitString(s);
            return;
        }
            program.addSoftNewline();
            program.addEmitString(s);
            program.addSoftNewline();
        }
    }
```

Appendix J

translator.compiler.utils

J.1 AnnotationHelper

```
package dk.vitality.util.jat.translator.compiler.utils;

public class AnnotationHelper {
    public final CompilerLog log;
    public final Annotation annotation;
    public final Class<?> cls;
    public final Field field;
    public final Method method;

    public AnnotationHelper(CompilerLog log, Annotation annotation, Class<?> cls, Field field,
Method method) {
        this.log = log;
        this.annotation = annotation;
        this.cls = cls;
        this.field = field;
        this.method = method;
    }

    public String getAnnotationStringParam(String name) throws InternalCompilerException {
        try {
            Class<?> atype = annotation.annotationType();
            Method m = atype.getMethod(name);
            return (String) m.invoke(annotation);
        }
        catch (Exception ex) {
            throw new InternalCompilerException("Can't get String annotation value (" + name +
"): "
                    + ExceptionUtils.getSomeMessage(ex), ex);
        }
    }

    public String[] getAnnotationStringArrayParam(String name) throws
InternalCompilerException {
        try {
            Object o = annotation.annotationType().getMethod(name).invoke(annotation);
            if (o == null)
                return ArrayUtils.emptyStringArray;
            if (!(o instanceof String[])) {
                log.log(LogLevel.Error, "Annotation parameter '" + name + "' is not String[]: "
                        + CompilerUtils.annotationToString(annotation, cls, field, method));
                return ArrayUtils.emptyStringArray;
            }
            String[] array = (String[]) o;
            for (String s : array) {
                if (s == null || s.length() == 0) {
                    log.log(LogLevel.Error, "Annotation parameter '" + name +
"' may not contain nulls or empty strings: "
                            + CompilerUtils.annotationToString(annotation, cls, field, method));
                }
            }
            return array;
        }
        catch (Exception ex) {
            throw new InternalCompilerException("Can't get String[] annotation value (" + name
+ "): "
                    + ExceptionUtils.getSomeMessage(ex), ex);
        }
    }

    public String annotationToString() {
        StringBuilder sb = new StringBuilder(20);
        sb.append(toString(annotation)).append(' ');
        if (field != null || method != null) {
            if (field != null)
                sb.append(field.getName());
            if (method != null) {
                sb.append(method.getReturnType().getSimpleName()).append(' ');
                sb.append(method.getName()).append("()");
            }
            sb.append(" in ").append(CompilerUtils.getInnerclassName(cls));
        } else if (cls != null) {
            sb.append("class ").append(cls.getSimpleName());
        }
```

```
        return sb.toString();
    }

    public static String toString(TCompile a) {
        StringBuilder sb = new StringBuilder();
        sb.append('@').append(a.annotationType().getSimpleName());
        String sep = "(";
        if (a.filename().length() > 0) {
            sb.append(sep).append("filename='").append(a.filename()).append("'");
            sep = ",␣";
        }
        if (a.fragment().length() > 0) {
            sb.append(sep).append("fragment='").append(a.fragment()).append("'");
            sep = ",␣";
        }
        if (a.tpname().length() > 0) {
            sb.append(sep).append("tpname='").append(a.tpname()).append("'");
            sep = ",␣";
        }
        if (!sep.equals("("))
            sb.append(")");
        return sb.toString();
    }

    public static String toString(Annotation a) {
        return '@' + a.annotationType().getSimpleName();
    }
}
```

J.2 CompilerUtils

```
package dk.vitality.util.jat.translator.compiler.utils;

public class CompilerUtils {
    public static String[] splitCssClassNames(String attr) {
        if (attr == null || attr.length() == 0)
            return ArrayUtils.emptyStringArray;
        StringTokenizer st = new StringTokenizer(attr, "␣\t\n\r");
        String[] array = new String[st.countTokens()];
        for (int i = 0; st.hasMoreTokens(); i++)
            array[i] = st.nextToken();
        return array;
    }

    public static String[] splitCssClassNamesWithoutPrefixedUnderscore(String attr) {
        String[] a = splitCssClassNames(attr);
        for (int i = 0; i < a.length; i++) {
            String s = a[i];
            if (s.charAt(0) == '_')
                a[i] = s.substring(1);
        }
        return a;
    }

    public static String zapUnderscorePrefixedClassNames(String attr) {
        if (attr == null || attr.indexOf('_') < 0)
            return attr;
        String[] a = splitCssClassNames(attr);
        List<String> list = new ArrayList<String>(a.length);
        for (String s : a) {
            if (s.charAt(0) != '_')
                list.add(s);
        }
        return StringUtils.join("␣", list);
    }

    public static <T> List<T> concatLists(List<T> accum, List<T> additional) {
        if (additional == null || additional.isEmpty())
            return accum;
        if (accum == null)
            accum = new ArrayList<T>();
        accum.addAll(additional);
        return accum;
    }

    public static String stripLowercasePrefix(String s) throws CompilerException {
        int len = s.length();
        for (int i = 0; i < len; i++) {
            if (!Character.isLowerCase(s.charAt(i)))
                return s.substring(i);
        }
        throw new InvalidElementTargetException(
"Can't␣create␣default␣name␣by␣stripping␣[a-z]+␣from␣'" + s + "'");
    }

    public static String makeFirstCharacterUppercase(String s) {
        if (s == null || s.length() == 0 || Character.isUpperCase(s.charAt(0)))
            return s;
        else
            return Character.toUpperCase(s.charAt(0)) + s.substring(1);
    }

    public static String annotationToString(Annotation a, Class<?> cls, Field f, Method m) {
        StringBuilder sb = new StringBuilder(20);
        sb.append('@').append(a.annotationType().getSimpleName()).append('␣');
        if (f != null || m != null) {
            if (f != null)
                sb.append(f.getName());
            if (m != null) {
                sb.append(m.getReturnType().getSimpleName());
                sb.append('␣').append(m.getName()).append("()");
            }
            sb.append("␣in␣").append(CompilerUtils.getInnerclassName(cls));
        } else {
            sb.append("class␣").append(cls.getSimpleName());
        }
        return sb.toString();//StringUtils.replaceSubstring(sb.toString(), " java.lang.", " ");
    }

    public static String getInnerclassBasename(Class<?> innerClass) {
        return StringUtils.zapPrefix(getInnerclassName(innerClass), "Emit");
    }

    public static String getInnerclassName(Class<?> innerClass) {
        String s = innerClass.getSimpleName();
```

```
        int idx = s.lastIndexOf('$');
        if (idx >= 0)
            s = s.substring(idx + 1);
        return s;
    }

    public static boolean isValidJavaIdentifier(String s) {
        if (s.length() == 0)
            return false;
        if (!Character.isJavaIdentifierStart(s.charAt(0)))
            return false;
        for (int i = 1; i < s.length(); i++) {
            if (!Character.isJavaIdentifierPart(s.charAt(i)))
                return false;
        }
        return true;
    }

    private static DocumentBuilder myDocumentBuilder = null;

    public synchronized static Document createDocument() throws ParserConfigurationException {
        if (myDocumentBuilder == null) {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            myDocumentBuilder = dbf.newDocumentBuilder();
        }
        return myDocumentBuilder.newDocument();
    }
}
```

J.3 HtmlTags

```
package dk.vitality.util.jat.translator.compiler.utils;

public class HtmlTags {
    public final static Set<String> emptyElements;
    public final static Set<String> emptyAttributes;
    public final static Set<String> elementsPrependNewline;
    public final static Set<String> elementsAppendNewline;

    static {
        emptyElements = makeSet(new String[]{
                "area", "base", "basefont", "br", "col", "frame", "hr", "img",
                "input", "isindex", "link", "meta", "param"});

        emptyAttributes = makeSet(new String[]{
                "checked", "compact", "declare", "defer", "disabled", "ismap",
                "nohref", "noresize", "noshade", "nowrap", "readonly", "selected"});

        Set<String> set = new HashSet<String>();
        for (String s : new String[]{
                "area", "base", "basefont", "body", "col", "div", "form", "frame",
                "h1", "h2", "h3", "h4", "h5", "h6",
                "head", "hr", "link", "meta", "param", "p", "script", "style",
                "table", "td", "title", "th", "tr"}) {
            set.add(s);
        }
        elementsPrependNewline = Collections.unmodifiableSet(new HashSet<String>(set));
        set.add("br");
        set.add("html");
        elementsAppendNewline = Collections.unmodifiableSet(set);
    }

    static private Set<String> makeSet(String[] a) {
        Set<String> set = new HashSet<String>();
        for (String s : a)
            set.add(s);
        return Collections.unmodifiableSet(set);
    }

    public static String elementToString(Element e) {
        StringBuilder sb = new StringBuilder(10);
        sb.append('<').append(e.getNodeName());
        NamedNodeMap map = e.getAttributes();
        int num = map.getLength();
        for (int i = 0; i < num; i++) {
            Node a = map.item(i);
            sb.append('␣').append(a.getNodeName());
            if (emptyElements.contains(a.getNodeName()))
                continue;
            sb.append('=');
            sb.append(HtmlAttr.quoted(a.getNodeValue()));
        }
        sb.append('>');
        return sb.toString();
    }
}
```

J.4 LogLevel

```
package dk.vitality.util.jat.translator.compiler.utils;

public enum LogLevel {
    Verbose, Notice, Error, InternalError
}
```

Appendix K

junit

K.1 TestBase

```
package dk.vitality.util.jat.junit.lib;

abstract public class TestBase {
    public final HtmlPrintWriter pw;
    private final StringWriter sw;

    protected final String defaultHtmlBasename = getClass().getSimpleName();
    protected String htmlBasename = defaultHtmlBasename;

    protected TestBase() {
        sw = new StringWriter();
        pw = new HtmlPrintWriter(sw);
    }

    @Test(timeout = 5000)
    public void test() throws Exception {
        try {
            emitTemplate();
            validate();
        }
        catch (TemplateJavaMD5Exception ex) {
            throw ex;
            //Assert.fail("MD5 mismatch: " + ex.getMessage());
        }
    }

    abstract protected void emitTemplate() throws Exception;

    protected void validate() throws Exception {
        pw.flush();
        String actual = sw.toString();

        File actualF = getHtmlF(true);
        softSave(actualF, actual);
        File expectedF = getHtmlF(false);
        String expected = new String(FileContentsUtils.readFile(expectedF), "utf-8");

        if (actual.equals(expected))
            return;
        System.err.println("#######_Failure_in_" + getClass().getName());
        if (!htmlBasename.equals(defaultHtmlBasename))
            System.err.println("___using_html_basename:_" + htmlBasename);
        Process p = Runtime.getRuntime().exec(new String[]{"diff", "-u", expectedF.getPath(),
actualF.getPath()});
        p.getOutputStream().close();
        InputStream is = new BufferedInputStream(p.getInputStream(), 2048);
        p.getErrorStream().close();
        int i;
        ByteBuffer bb = new ByteBuffer(100);
        while ((i = is.read()) != -1)
            bb.append(i);
        System.err.println(bb.toString("utf-8"));
        assert false;
    }

    protected File getHtmlF(boolean actual) throws Exception {
        String basename = (actual ? defaultHtmlBasename : htmlBasename + "-expected");
        String basedir = System.getenv("JAT_BASEDIR");
        if (basedir == null)
            throw new Exception("Missing_environment_variable_JAT_BASEDIR");
        File f = new File(getClass().getName().replace('.', '/')).getParentFile();
        if (actual)
            f = new File(f.getParentFile().getParentFile(), "actualOutput");
        return new File(basedir, f.getPath() + "/" + basename + ".html");
    }

    protected void softSave(File f, String contents) throws Exception {
        byte[] newData = contents.getBytes("utf-8");
        if (f.exists()) {
            byte[] oldData = FileContentsUtils.readFile(f);
            if (Arrays.equals(newData, oldData))
                return;
        }
        FileContentsUtils.saveFile(f, newData);
    }
```

```
}
```

K.2 SimpleContent

```
package dk.vitality.util.jat.junit.tests.aSimpleContent;

@TCompile
public class SimpleContent extends TestBase {
    protected void emitTemplate() throws Exception {
        new SimpleContentTp(this).emit(pw);
    }

    @EText
    public String getPersonFullName() {
        return "Elliott␣Smith";
    }
}
```

K.2.1 SimpleContent.html

```
<div>You're logged in as <span id=PersonFullName>John Doe</span>.</div>
```

K.2.2 SimpleContent-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div>You're logged in as <span id=PersonFullName>Elliott Smith</span>.</div>
</body>
</html>
```

K.3 MoreContent

```
package dk.vitality.util.jat.junit.tests.bMoreContent;

@TCompile
public class MoreContent extends TestBase {
    protected void emitTemplate() throws Exception {
        new MoreContentTp(this).emit(pw);
    }

    //————————————————————— TEXT, instance

    @EText
    String textInstanceContent1 = "<i>You␣&␣me</i>!";

    @ETextPre
    String textInstanceLeader1 = "<i>Keith␣Jarret</i>:␣";

    @ETextPost
    String textInstanceExtra1 = ":␣<b>OK␣Computer</b>.";

    @EText
    String getTextInstanceContent2() {
        return "Beth␣Gibbons␣&␣Rustin'␣Man";
    }

    @ETextPre
    String getTextInstanceLeader2() {
        return "<b>Kathusja␣Babikian</b>:␣";
    }

    @ETextPost
    String getTextInstanceExtra2() {
        return ":␣<i>OK␣Computer</i>.";
    }

    //————————————————————— TEXT, static

    @EText
    static String textStaticContent1 = "<i>You␣&␣me</i>!";

    @ETextPre
    static String textStaticLeader1 = "<i>Keith␣Jarret</i>:␣";

    @ETextPost
    static String textStaticExtra1 = ":␣<b>OK␣Computer</b>.";

    @EText
    static String getTextStaticContent2() {
        return "Beth␣Gibbons␣&␣Rustin'␣Man";
    }

    @ETextPre
    static String getTextStaticLeader2() {
        return "<b>Kathusja␣Babikian</b>:␣";
    }

    @ETextPost
    static String getTextStaticExtra2() {
        return ":␣<i>OK␣Computer</i>.";
    }
    //————————————————————— HTML, instance

    @EHtml
    String htmlInstanceContent1 = "<i>You␣&␣me</i>!";

    @EHtmlPre
    String htmlInstanceLeader1 = "<i>Keith␣Jarret</i>:␣";

    @EHtmlPost
    String htmlInstanceExtra1 = ":␣<b>OK␣Computer</b>.";

    @EHtml
    String getHtmlInstanceContent2() {
        return "Beth␣Gibbons␣&␣Rustin'␣Man";
    }

    @EHtmlPre
    String getHtmlInstanceLeader2() {
        return "<b>Kathusja␣Babikian</b>:␣";
    }
```

```
    @EHtmlPost
    String getHtmlInstanceExtra2() {
        return ":␣<i>OK␣Computer</i>.";
    }
    @EHtml
    void printHtmlInstanceContent3() {
        pw.print("(<b>Finally␣We␣Are␣No␣One!</b>)");
    }
    @EHtmlPre
    void printHtmlInstanceLeader3() {
        pw.print("<i>album</i>:␣");
    }
    @EHtmlPost
    void printHtmlInstanceExtra3() {
        pw.print(":␣<i>Don't␣Be␣Afraid,␣You␣Have␣Just␣Got␣Your␣Eyes␣Closed</i>.");
    }
    //———————————————— HTML, static
    @EHtml
    static String htmlStaticContent1 = "<i>You␣&␣me</i>!";
    @EHtmlPre
    static String htmlStaticLeader1 = "<i>Keith␣Jarret</i>:␣";
    @EHtmlPost
    static String htmlStaticExtra1 = ":␣<b>OK␣Computer</b>.";
    @EHtml
    static String getHtmlStaticContent2() {
        return "Beth␣Gibbons␣&␣Rustin'␣Man";
    }
    @EHtmlPre
    static String getHtmlStaticLeader2() {
        return "<b>Kathusja␣Babikian</b>:␣";
    }
    @EHtmlPost
    static String getHtmlStaticExtra2() {
        return ":␣<i>OK␣Computer</i>.";
    }
    //———————————————— MISC
    @EHtml(tag = "title")
    String title = "Just␣a␣few␣names";
}
```

K.3.1 MoreContent.html

```
<html>
<head>
    <title>This is my title</title>
</head>
<body>
<div class=TextInstance>
    <div class=TextInstanceContent1>Hello, world</div>
    <div class=TextInstanceLeader1>The Celestial Hawk</div>
    <div class=TextInstanceExtra1>Radiohead</div>
    <div class=TextInstanceContent2>Hello, world</div>
    <div class=TextInstanceLeader2>So Real</div>
    <div class=TextInstanceExtra2>Radiohead</div>
</div>
<div class=TextStatic>
    <div class=TextStaticContent1>Hello, world</div>
    <div class=TextStaticLeader1>The Celestial Hawk</div>
    <div class=TextStaticExtra1>Radiohead</div>
    <div class=TextStaticContent2>Hello, world</div>
    <div class=TextStaticLeader2>So Real</div>
    <div class=TextStaticExtra2>Radiohead</div>
</div>
<div class=HtmlInstance>
    <div class=HtmlInstanceContent1>Hello, world</div>
    <div class=HtmlInstanceLeader1>The Celestial Hawk</div>
    <div class=HtmlInstanceExtra1>Radiohead</div>
    <div class=HtmlInstanceContent2>Hello, world</div>
    <div class=HtmlInstanceLeader2>So Real</div>
    <div class=HtmlInstanceExtra2>Radiohead</div>
    <div class=HtmlInstanceContent3>Hello, world</div>
    <div class=HtmlInstanceLeader3>Yesterday Was Dramatic Today Is OK</div>
    <div class=HtmlInstanceExtra3>Múm</div>
</div>
<div class=HtmlStatic>
    <div class=HtmlStaticContent1>Hello, world</div>
    <div class=HtmlStaticLeader1>The Celestial Hawk</div>
    <div class=HtmlStaticExtra1>Radiohead</div>
    <div class=HtmlStaticContent2>Hello, world</div>
    <div class=HtmlStaticLeader2>So Real</div>
    <div class=HtmlStaticExtra2>Radiohead</div>
</div>
</body>

</html>
```

K.3.2 MoreContent-expected.html

```
<html>
<head>
<title>Just a few names</title>
</head>
<body>
<div class=TextInstance>
<div class=TextInstanceContent1>&lt;i&gt;You &amp; me&lt;/i&gt;!</div>
<div class=TextInstanceLeader1>&lt;i&gt;Keith Jarret&lt;/i&gt;: The Celestial Hawk</div>
<div class=TextInstanceExtra1>Radiohead: &lt;b&gt;OK Computer&lt;/b&gt;.</div>
<div class=TextInstanceContent2>Beth Gibbons &amp; Rustin&#39; Man</div>
<div class=TextInstanceLeader2>&lt;b&gt;Kathusja Babikian&lt;/b&gt;: So Real</div>
<div class=TextInstanceExtra2>Radiohead: &lt;i&gt;OK Computer&lt;/i&gt;.</div>
</div>
<div class=TextStatic>
<div class=TextStaticContent1>&lt;i&gt;You &amp; me&lt;/i&gt;!</div>
<div class=TextStaticLeader1>&lt;i&gt;Keith Jarret&lt;/i&gt;: The Celestial Hawk</div>
```

```
<div class=TextStaticExtra1>Radiohead: &lt;b&gt;OK Computer&lt;/b&gt;.</div>
<div class=TextStaticContent2>Beth Gibbons &amp; Rustin&#39; Man</div>
<div class=TextStaticLeader2>&lt;b&gt;Kathusja Babikian&lt;/b&gt;: So Real</div>
<div class=TextStaticExtra2>Radiohead: &lt;i&gt;OK Computer&lt;/i&gt;.</div>
</div>
<div class=HtmlInstance>
<div class=HtmlInstanceContent1><i>You & me</i>!</div>
<div class=HtmlInstanceLeader1><i>Keith Jarret</i>: The Celestial Hawk</div>
<div class=HtmlInstanceExtra1>Radiohead: <b>OK Computer</b>.</div>
<div class=HtmlInstanceContent2>Beth Gibbons & Rustin' Man</div>
<div class=HtmlInstanceLeader2><b>Kathusja Babikian</b>: So Real</div>
<div class=HtmlInstanceExtra2>Radiohead: <i>OK Computer</i>.</div>
<div class=HtmlInstanceContent3>(<b>Finally We Are No One!</b>)</div>
<div class=HtmlInstanceLeader3><i>album</i>: Yesterday Was Dramatic Today Is OK</div>
<div class=HtmlInstanceExtra3>Múm: <i>Don't Be Afraid, You Have Just Got Your Eyes
Closed</i>.</div>
</div>
<div class=HtmlStatic>
<div class=HtmlStaticContent1><i>You & me</i>!</div>
<div class=HtmlStaticLeader1><i>Keith Jarret</i>: The Celestial Hawk</div>
<div class=HtmlStaticExtra1>Radiohead: <b>OK Computer</b>.</div>
<div class=HtmlStaticContent2>Beth Gibbons & Rustin' Man</div>
<div class=HtmlStaticLeader2><b>Kathusja Babikian</b>: So Real</div>
<div class=HtmlStaticExtra2>Radiohead: <i>OK Computer</i>.</div>
</div>
</body>
</html>
```

K.4 SimpleAttributes

```
package dk.vitality.util.jat.junit.tests.dSimpleAttributes;

@TCompile
public class SimpleAttributes extends TestBase {
    protected void emitTemplate() throws Exception {
        new SimpleAttributesTp(this).emit(pw);
    }

    @AName
    String nameUser() {
        return "fullname";
    }

    @AValue
    String valueUser() {
        return "Crash_Test_Dummies_and_\"Beth_Gibbons_&_Rustin'_Man\"";
    }

    @AEnabled
    boolean enabledUser() {
        return true;
    }

    @AId
    String idUser() {
        return "U42";
    }

    @ATabIndex
    int tabUser() {
        return 7;
    }
}
```

K.4.1 SimpleAttributes.html

```
<div>Please type your name: <input disabled id=_User name=username maxlength="50"
style="width: 200px;"></div>
```

K.4.2 SimpleAttributes-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div>Please type your name: <input id='U42' maxlength=50 name='fullname' style='width:
200px;' tabindex=7 value='Crash Test Dummies and &quot;Beth Gibbons &amp; Rustin&#39;
Man&quot;'></div>
</body>
</html>
```

K.5 TestAttributeCollections

```
package dk.vitality.util.jat.junit.tests.eAttributeCollections;

@TCompile
public class TestAttributeCollections extends TestBase {
    @ASrcWidthHeight
    String image1 = "src='j1'_width=10_height=12";

    @ASrcWidthHeight
    String image2 = "src='j1'_width=20_height=22";

    @ASrcWidthHeight
    String image3 = "src='j3'_width=30_height=32";

    @ASrcWidthHeight
    String image4 = "src='j4'_width=40_height=42";

    @ACollection(styles = {"style", "class", "title"})
    String multiMulti =
"style='border:_1px_solid_green;'_class=RealClass_title='Comment_here'";

    protected void emitTemplate() throws Exception {
        new TestAttributeCollectionsTp(this).emit(pw);
    }

    @AClassPost
```

```
    String header = "DynamicClass ExtraClass";

    @AStylePost
    String addedStyles = "color: green; font: 12px Courier";
}
```

K.5.1 TestAttributeCollections.html

```
<html>
<head>
    <title>Testing attribute collections</title>
</head>
<body>
<div><img id=Image1 src="a" alt="number 1"></div>
<div><img id=Image2 src="b" width=15 alt="number 2"></div>
<div><img id=Image3 src="c" height=20 alt="number 3"></div>
<div><img id=Image4 src="d" width=50 height=25 alt="number 4"></div>
<div id=MultiMulti class="Dummy">Hello, world!</div>
<div id=MoreClasses class="Header Keeper">This is a header</div>
<div id=AddedStyles style="border: 1px dotted blue">This is important</div>
</body>

</html>
```

K.5.2 TestAttributeCollections-expected.html

```
<html>
<head>
<title>Testing attribute collections</title>
</head>
<body>
<div><img alt='number 1' src='j1' width=10 height=12 id=Image1></div>
<div><img alt='number 2' src='j1' width=20 height=22 id=Image2></div>
<div><img alt='number 3' src='j3' width=30 height=32 id=Image3></div>
<div><img alt='number 4' src='j4' width=40 height=42 id=Image4></div>
<div style='border: 1px solid green;' class=RealClass title='Comment here' id=MultiMulti>Hello,
world!</div>
<div class='Header Keeper DynamicClass ExtraClass' id=MoreClasses>This is a header</div>
<div id=AddedStyles style='border: 1px dotted blue; color: green; font: 12px Courier'>This is
important</div>
</body>

</html>
```

K.6 TestHasClass

```
package dk.vitality.util.jat.junit.tests.fHasClass;

@TCompile
public class TestHasClass extends TestBase {
    @EText
    String getTextOne() {
        return "Making a test";
    }

    @SHasClass(classes = "AlwaysPresent")
    boolean textOne = true;

    @SHasClass(classes = "BlockOutline")
    boolean grayBlockOne() {
        return true;
    }

    @SHasClass(classes = "BlockTwoToRemove")
    boolean zapBlockTwo() {
        return false;
    }

    @SHasClass(classes = "BlockTreeToRemove")
    boolean zapBlockThree() {
        return false;
    }

    boolean odd = false;

    @SHasClass(classes = "GreenText")
    boolean greenString1() {
        odd ^= true;
        return odd;
    }

    @SHasClass(classes = "BlueText")
    boolean blueString1() {
        return !odd;
    }

    @SHasClass(classes = "Number2")
    boolean string2;

    @SHasClass(classes = {"abc def ghi jkl mno pqr"})
    boolean blockFour = true;

    protected void emitTemplate() throws Exception {
        new TestHasClassTp(this).emit(pw);
    }
}
```

K.6.1 TestHasClass.html

```
<html>
<head>
    <title>Testing classes</title>
    <style type="text/css">
        .GreenText {
            color: green;
        }
        .BlueText {
            color: blue;
        }
    </style>
</head>
<body>
```

```
<div class=_TextOne>This is a text</div>
<div class=String1>global-s1</div>
<div class=String2>global-s2</div>
<div class=BlockOne>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div id=BlockTwo class='BlockTwoToRemove'>
    <div class=_String1>b3s1</div>
    <div class=_String2>b2s2</div>
</div>
<div class='BlockThree BlockTreeToRemove TheLast3'>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div class=BlockFour>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div><span class=GreenText>green</span> and <span
class=BlueText>blue</span></div>
</body>
</html>
```

K.6.2 TestHasClass-expected.html

```
<html>
<head>
<title>Testing classes</title>
<style type='text/css'>
.GreenText {
color: green;
}
.BlueText {
color: blue;
}
</style>
</head>
<body>
<div class='AlwaysPresent'>Making a test</div>
<div class='String1 GreenText'>global-s1</div>
<div class='String2'>global-s2</div>
<div class='BlockOne BlockOutline'>
<div class='String1 BlueText'>b2s1</div>
<div class='String2'>b2s2</div>
</div>
<div id=BlockTwo>
<div class='GreenText'>b3s1</div>
<div>b2s2</div>
</div>
<div class='BlockThree TheLast3'>
<div class='String1 BlueText'>b2s1</div>
<div class='String2'>b2s2</div>
</div>
<div class='BlockFour abc def ghi jkl mno pqr'>
<div class='String1 GreenText'>b2s1</div>
<div class='String2'>b2s2</div>
</div>
<div><span class=GreenText>green</span> and <span
class=BlueText>blue</span></div>
</body>
</html>
```

K.7 TestStyles

```
package dk.vitality.util.jat.junit.tests.gStyles;

@TCompile
public class TestStyles extends TestBase {
    @EText
    String getTextOne() {
        return "Making_a_styled_test";
    }

    @SDisplayBlock
    boolean textOne = true;

    @SDisplayBlock
    boolean displayBlockOne() {
        return false;
    }

    @SDisplay
    static boolean hideBlockTwo() {
        return false;
    }

    @SDisplay
    boolean showString1() {
        return true;
    }

    boolean odd = false;

    @SValue(styles = "color")
    String colorString2() {
        odd ^= true;
        return odd ? "yellow" : "cyan";
    }

    @SDisplayInline
    static boolean seeMe1 = true;

    @SDisplayInline
    boolean invisible1 = false;

    @SDisplayInline
    boolean seeMe2 = true;

    @SDisplayInline
    static final boolean invisible2 = false;

    @SCollection(styles = {"color", "margin-top", "margin-bottom"})
```

```
    String multiStyles = "color: red; margin-top: 10px; margin-bottom: 5px";

    protected void emitTemplate() throws Exception {
        new TestStylesTp(this).emit(pw);
    }
}
```

K.7.1 TestStyles.html

```
<html>
<head>
    <title>Testing classes</title>
    <style type="text/css">
        .GreenText {
            color: green;
        }
        .BlueText {
            color: blue;
        }
    </style>
</head>
<body>
<div class=_TextOne>This is a text</div>
<div class=String1>global-s1</div>
<div class=String2>global-s2</div>
<div class=BlockOne>
    <div class=String1 style="border: 1px solid brown">b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div id=BlockTwo class='BlockTwoToRemove'>
    <div class=_String1>b3s1</div>
    <div class=_String2>b2s2</div>
</div>
<div class='BlockThree'>
    <div class=SeeMe1>I'm emo</div>
    <div class=Invisible1>at night</div>
</div>
<div class=BlockFour>
    (((
    <span class=SeeMe2>I'm emo</span>
    |||
    <span class=Invisible2>at night</span>
    )))
</div>
<div id=MultiStyles style="color: blue; font-family: Times, Sans Serif; margin: 2px;">Hello,
world!</div>
<div><span class=GreenText>green</span> and <span
class=BlueText>blue</span></div>
</body>

</html>
```

K.7.2 TestStyles-expected.html

```
<html>
<head>
<title>Testing classes</title>
<style type='text/css'>
.GreenText {
color: green;
}
.BlueText {
color: blue;
}
</style>
</head>
<body>
<div style='display: block'>Making a styled test</div>
<div class=String1>global-s1</div>
<div class=String2 style='color: yellow'>global-s2</div>
<div class=BlockOne style='display: none'>
<div class=String1 style='border: 1px solid brown; '>b2s1</div>
<div class=String2 style='color: cyan'>b2s2</div>
</div>
<div class=BlockTwoToRemove id=BlockTwo style='display: none'>
<div>b3s1</div>
<div style='color: yellow'>b2s2</div>
</div>
<div class=BlockThree>
<div class=SeeMe1 style='display: inline'>I'm emo</div>
<div class=Invisible1 style='display: none'>at night</div>
</div>
<div class=BlockFour>((( <span class=SeeMe2 style='display: inline'>I'm emo</span> |||
<span class=Invisible2 style='display: none'>at night</span> )))</div>
<div id=MultiStyles style='font-family: Times, Sans Serif; margin: 2px; color: red; margin-top:
10px; margin-bottom: 5px'>Hello, world!</div>
<div><span class=GreenText>green</span> and <span
class=BlueText>blue</span></div>
</body>

</html>
```

K.8 ConditionalsA

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsA extends TestBase {
    protected void emitTemplate() throws Exception {
        new ConditionalsATp(this).emit(pw);
    }

    @EIf
    @EIfNot("NoPrinters")
    public boolean testHasPrinters() {
        return true;
    }
}
```

K.9 ConditionalsB

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsB extends TestBase {
    protected void emitTemplate() throws Exception {
        new ConditionalsBTp(this).emit(pw);
    }

    @EIf
    @EIfNot("NoPrinters")
    public boolean testHasPrinters() {
        return false;
    }
}
```

K.10 ConditionalsC

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsC extends TestBase {
    protected void emitTemplate() throws Exception {
        htmlBasename = "ConditionalsA";
        new ConditionalsCTp(this).emit(pw);
    }

    @EIf
    public boolean testHasPrinters() {
        return true;
    }

    @EIf(value = "NoPrinters", inverse = true)
    public boolean noHasPrinters() {
        return true;
    }
}
```

K.11 ConditionalsF

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsF extends TestBase {
    protected void emitTemplate() throws Exception {
        htmlBasename = "ConditionalsA";
        new ConditionalsFTp(this).emit(pw);
    }

    @EIf
    @EIfNot("NoPrinters")
    boolean hasPrinters = true;
}
```

K.12 ConditionalsG

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsG extends TestBase {
    protected void emitTemplate() throws Exception {
        htmlBasename = "ConditionalsB";
        new ConditionalsGTp(this).emit(pw);
    }

    @EIf
    @EIfNot("NoPrinters")
    boolean hasPrinters = false;
}
```

K.13 ConditionalsH

```
package dk.vitality.util.jat.junit.tests.iConditional;

@TCompile(filename = "Conditionals")
public class ConditionalsH extends TestBase {
    protected void emitTemplate() throws Exception {
        htmlBasename = "ConditionalsA";
        new ConditionalsHTp(this).emit(pw);
    }

    @EIf
    boolean hasPrinters = true;

    @EIf(value = "NoPrinters", inverse = true)
    boolean has2Printers = true;
}
```

K.13.1 Conditionals.html

```
<div id=NoPrinters>You don't have access to any printer</div>
<form class='HasPrinters' action=.>
    <div>You have access to the following printers:</div>

</form>
```

K.13.2 ConditionalsA-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<form action=. class=HasPrinters>
<div>You have access to the following printers:</div>
</form>
</body>
```

```
</html>
```

K.13.3 ConditionalsB-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div id=NoPrinters>You don't have access to any printer</div>
</body>
</html>
```

K.14 ConditionalContentA

```
package dk.vitality.util.jat.junit.tests.iConditionalContent;

@TCompile(filename = "ConditionalContent")
public class ConditionalContentA extends TestBase {
    protected void emitTemplate() throws Exception {
        new ConditionalContentATp(this).emit(pw);
    }

    @EContentIf
    public boolean testHasPrinters() {
        return true;
    }

    @EContentIf("NoPrinters")
    public boolean missingPrinters() {
        return !testHasPrinters();
    }
}
```

K.15 ConditionalContentB

```
package dk.vitality.util.jat.junit.tests.iConditionalContent;

@TCompile(filename = "ConditionalContent")
public class ConditionalContentB extends TestBase {
    protected void emitTemplate() throws Exception {
        new ConditionalContentBTp(this).emit(pw);
    }

    @EContentIf
    public boolean testHasPrinters() {
        return false;
    }

    @EContentIf("NoPrinters")
    public boolean missingPrinters() {
        return !testHasPrinters();
    }
}
```

K.15.1 ConditionalContent.html

```
<div id=NoPrinters>You don't have access to any printer</div>
<form class='HasPrinters' action=.>
    <div>You have access to the following printers:</div>

</form>
```

K.15.2 ConditionalContentA-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div id=NoPrinters></div>
<form action=. class=HasPrinters>
<div>You have access to the following printers:</div>
</form>
</body>
</html>
```

K.15.3 ConditionalContentB-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div id=NoPrinters>You don't have access to any printer</div>
<form action=. class=HasPrinters></form>
</body>
</html>
```

K.16 TestConditionalTag

```
package dk.vitality.util.jat.junit.tests.iConditionalTag;

@TCompile
public class TestConditionalTag extends TestBase {
    protected void emitTemplate() throws Exception {
        new TestConditionalTagTp(this).emit(pw);
    }

    @ETagIf
    public boolean testKeeper() {
        return true;
```

```
    }
    @ETagIf
    public boolean wantsKeeper() {
        return true;
    }
    @ETagIf
    public boolean testDropMe() {
        return false;
    }
    @ETagIf
    public boolean failedDropMe() {
        return true;
    }
}
```

K.16.1 TestConditionalTag.html

```
<html>
<body>
<div><a id=Keeper href="http://www.ruc.dk">Roskilde University</a></div>
<div>Goto: <a id=DropMe href="http://example.com">No place in particular</a>.</div>
</body>
</html>
```

K.16.2 TestConditionalTag-expected.html

```
<html>
<head>
<title></title>
</head>
<body>
<div><a href='http://www.ruc.dk' id=Keeper>Roskilde University</a></div>
<div>Goto: No place in particular.</div>
</body>
</html>
```

K.17 SimpleLoop

```
package dk.vitality.util.jat.junit.tests.kSimpleLoop;

@TCompile
public class SimpleLoop extends TestBase {
    int rownum;
    @EText
    String itemName;
    @EWhile
    boolean testItemRow() {
        rownum++;
        itemName = "name_#" + rownum;
        return rownum <= 5;
    }
    @EText
    String getItemDescription() {
        return "Making_a_" + rownum + "-long_story_longer";
    }
    protected void emitTemplate() throws Exception {
        new SimpleLoopTp(this).emit(pw);
    }
}
```

K.17.1 SimpleLoop.html

```
<html>
<head>
    <title>Just a test</title>
</head>
<body>
<table>
    <tr>
        <th>Name</th>
        <th>Description</th>
    </tr>
    <tr id=ItemRow>
        <td class=ItemName>Sulfur</td>
        <td id=ItemDescription>Something</td>
    </tr>
    <tr>
        <td>My</td>
        <td>footer</td>
    </tr>
</table>
</body>
</html>
```

K.17.2 SimpleLoop-expected.html

```
<html>
<head>
<title>Just a test</title>
</head>
<body>
<table>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
<tr>
<td class=ItemName>name #1</td>
<td>Making a 1-long story longer</td>
</tr>
```

```
<tr>
<td class=ItemName>name #2</td>
<td>Making a 2-long story longer</td>
</tr>
<tr>
<td class=ItemName>name #3</td>
<td>Making a 3-long story longer</td>
</tr>
<tr>
<td class=ItemName>name #4</td>
<td>Making a 4-long story longer</td>
</tr>
<tr>
<td class=ItemName>name #5</td>
<td>Making a 5-long story longer</td>
</tr>
<tr>
<td>My</td>
<td>footer</td>
</tr>
</table>
</body>

</html>
```

K.18 TestScope

```
package dk.vitality.util.jat.junit.tests.lScope;

@TCompile
public class TestScope extends TestBase {
    @EText
    String getTextOne() {
        return "Making_a_test";
    }

    @EScope
    class BlockOne {
        @EText
        String getString1() {
            return "scope-1-get";
        }

        @EHtml
        void printString2() {
            pw.htmlEncode("scope-1-print");
        }
    }

    @EScope
    static class BlockTwo {
        @EText
        String getString1() {
            return "scope-2-get";
        }
    }

    /*
    Support for double-processing of element for scopes disabled due to complex warning handling
    @EScope(tag = "title")
    class Title {
        @EContent(tag = "title")
        void printTitle() {
            pw.print("Testing gone amuck");
        }
    }
    */

    @EText(tag = "strong")
    String strong = "Testing_gone_amuck";

    @ETextPre("StrongContainer")
    String empty = "";

    protected void emitTemplate() throws Exception {
        new TestScopeTp(this).emit(pw);
    }
}
```

K.19 TestScopeInstance

```
package dk.vitality.util.jat.junit.tests.lScope;

@TCompile(filename = "TestScope")
public class TestScopeInstance extends TestBase {
    @EText
    String getTextOne() {
        return "Making_a_test";
    }

    @EScope
    class BlockOne {
        final String msg1;

        BlockOne() throws Exception {
            throw new Exception("Should_not_be_called");
        }

        BlockOne(String msg1) {
            this.msg1 = msg1;
        }

        @EText
        String getString1() {
            return msg1;
        }

        @EHtml
        void printString2() {
            pw.htmlEncode("scope-1-print");
        }
    }

    BlockOne newBlockOne() {
        return new BlockOne("scope-1-get");
```

```
    }
    @EScope
    static class BlockTwo {
        final String msg1;
        BlockTwo() throws Exception {
            throw new Exception("Should not be called");
        }
        BlockTwo(String msg1) {
            this.msg1 = msg1;
        }
        @EText
        String getString1() {
            return msg1;
        }
    }
    BlockTwo newBlockTwo() {
        return new BlockTwo("scope-2-get");
    }
    @EScope("StrongContainer")
    static class Severance {
        final String msg1;
        Severance() throws Exception {
            throw new Exception("Should not be called");
        }
        Severance(String msg1) {
            this.msg1 = msg1;
        }
        @EText(tag = "strong")
        String mymy() {
            return msg1;
        }
    }
    static Severance newSeverance() {
        return new Severance("Testing gone amuck");
    }
    protected void emitTemplate() throws Exception {
        htmlBasename = "TestScope";
        new TestScopeInstanceTp(this).emit(pw);
    }
}
```

K.19.1 TestScope.html

```
<html>
<head>
    <title>Testing scopes</title>
</head>
<body>
<div class=_TextOne>This is a text</div>
<div class=BlockOne>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div class='BlockOne second-time'>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div class=BlockTwo>
    <div class=String1>b3s1</div>
</div>
<div class=_StrongContainer>Severance: <strong>message here</strong></div>
</body>
</html>
```

K.19.2 TestScope-expected.html

```
<html>
<head>
<title>Testing scopes</title>
</head>
<body>
<div>Making a test</div>
<div class=BlockOne>
<div class=String1>scope-1-get</div>
<div class=String2>scope-1-print</div>
</div>
<div class='BlockOne second-time'>
<div class=String1>scope-1-get</div>
<div class=String2>scope-1-print</div>
</div>
<div class=BlockTwo>
<div class=String1>scope-2-get</div>
</div>
<div>Severance: <strong>Testing gone amuck</strong></div>
</body>
</html>
```

K.20 TestScopingWhile

```
package dk.vitality.util.jat.junit.tests.nScopingWhile;

@TCompile
public class TestScopingWhile extends TestBase {
    @EText
    String getTextOne() {
        return "Making_a_test";
    }
    @EText
    String getString1() {
        return "Outer-string-1";
```

```
}
@EHtml
void printString2() {
    pw.htmlEncode("Outer-string-2");
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int blockOneCount;
String blockOneName;
@EWhile
boolean testBlockOne() {
    blockOneCount++;
    blockOneName = "scope-1-get_#" + blockOneCount;
    return blockOneCount <= 5;
}
@EScope
class BlockOne {
    @EText
    @TOverrides
    String getString1() {
        return blockOneName;
    }
    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("scope-1-print-" + blockOneCount);
    }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int blockTwoCount;
@EScopingWhile
boolean testBlockTwo() {
    blockTwoCount++;
    return blockTwoCount <= 3;
}
BlockTwo newBlockTwo() {
    return new BlockTwo("scope-2-get_#" + blockTwoCount);
}
class BlockTwo {
    final String msg;
    BlockTwo() throws Exception {
        throw new Exception("Should_not_be_called");
    }
    BlockTwo(String msg1) {
        this.msg = msg1;
    }
    @EText
    @TOverrides
    String getString1() {
        return msg;
    }
    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("scope-2-print-" + blockTwoCount);
    }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int blockThreeCount;
@EWhile
boolean testBlockThree() {
    blockThreeCount++;
    return blockThreeCount <= 2;
}
BlockThree newBlockThree() {
    return new BlockThree("scope-3-get_#" + blockThreeCount);
}
@EScope
class BlockThree {
    final String msg;
    BlockThree() throws Exception {
        throw new Exception("Should_not_be_called");
    }
    BlockThree(String msg1) {
        this.msg = msg1;
    }
    @EText
    @TOverrides
    String getString1() {
        return msg;
    }
    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("scope-3-print-" + blockThreeCount);
    }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
@EHtml(tag = "title")
void printTitle() {
    pw.htmlEncode("Testing_gone_amuck_in_iterating");
}
protected void emitTemplate() throws Exception {
    new TestScopingWhileTp(this).emit(pw);
}
}
```

K.20.1 TestScopingWhile.html

```
<html>
<head>
   <title>Testing scopes</title>
</head>
<body>
<div class=_TextOne>This is a text</div>
<div class=String1>global-s1</div>
<div class=String2>global-s1</div>
<div class=BlockOne>
   <div class=String1>b1s1</div>
   <div class=String2>b1s2</div>
</div>
<div class=_BlockTwo>
   <div class=_String1>b2s1</div>
   <div class=_String2>b2s2</div>
</div>
<div class=BlockThree>
   <div class=String1>b3s1</div>
   <div class=String2>b3s2</div>
</div>
</body>

</html>
```

K.20.2 TestScopingWhile-expected.html

```
<html>
<head>
<title>Testing gone amuck in iterating</title>
</head>
<body>
<div>Making a test</div>
<div class=String1>Outer-string-1</div>
<div class=String2>Outer-string-2</div>
<div class=BlockOne>
<div class=String1>scope-1-get #1</div>
<div class=String2>scope-1-print-1</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #2</div>
<div class=String2>scope-1-print-2</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #3</div>
<div class=String2>scope-1-print-3</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #4</div>
<div class=String2>scope-1-print-4</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #5</div>
<div class=String2>scope-1-print-5</div>
</div>
<div>
<div>scope-2-get #1</div>
<div>scope-2-print-1</div>
</div>
<div>
<div>scope-2-get #2</div>
<div>scope-2-print-2</div>
</div>
<div>
<div>scope-2-get #3</div>
<div>scope-2-print-3</div>
</div>
<div class=BlockThree>
<div class=String1>scope-3-get #1</div>
<div class=String2>scope-3-print-1</div>
</div>
<div class=BlockThree>
<div class=String1>scope-3-get #2</div>
<div class=String2>scope-3-print-2</div>
</div>
</body>

</html>
```

K.21 TestIterate

```
package dk.vitality.util.jat.junit.tests.pIterate;

@TCompile
public class TestIterate extends TestBase {
   public TestIterate() {
      blockOneItems = new ArrayList<String>();
      for (int i = 1; i <= 5; i++)
         blockOneItems.add("scope-1-get_#" + i);

      blockTwoItems = new ArrayList<ComplexDataStructure>();
      for (int i = 1; i <= 3; i++)
         blockTwoItems.add(new ComplexDataStructure(i));

      blockThreeItems = new TreeSet<Integer>();
      for (int i = 1; i <= 2; i++)
         blockThreeItems.add(i);

      blockFourItems = new ArrayList<Map<String, Integer>>();
      for (int i = 1; i <= 3; i++)
         blockFourItems.add(new HashMap<String, Integer>());
   }

   @EText
   static String getTextOne() {
      return "Making_a_test";
   }

   @EText
   String getString1() {
      return "Outer-string-1";
   }
```

```
@EHtml
void printString2() {
    pw.htmlEncode("Outer-string-2");
}

//-----------------------------

@EIterate("BlockOne")
List<String> blockOneItems;

class BlockOne {
    final String msg;

    BlockOne(String msg) {
        this.msg = msg;
    }

    @EText
    @TOverrides
    String getString1() {
        return msg;
    }

    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("str2-" + msg);
    }
}

//-----------------------------
class ComplexDataStructure {
    final int number;
    final String msg;

    ComplexDataStructure(int number) {
        this.number = number * (pw != null ? 1 : 42);
        this.msg = "scope-2-get_#" + number;
    }
}

@EIterate("BlockTwo")
Collection<ComplexDataStructure> blockTwoItems;

BlockTwo newBlockTwo(ComplexDataStructure data) throws Exception {
    return new BlockTwo(data, true);
}

class BlockTwo {
    final ComplexDataStructure data;

    BlockTwo() throws Exception {
        throw new Exception("Should_not_be_called");
    }

    @SuppressWarnings({"UnusedDeclaration"})
    BlockTwo(ComplexDataStructure data) throws Exception {
        throw new Exception("Should_not_be_called");
    }

    BlockTwo(ComplexDataStructure data, boolean dummy) throws Exception {
        if (!dummy)
            throw new Exception("Bad_mojo");
        this.data = data;
    }

    @EText
    @TOverrides
    String getString1() {
        return data.msg;
    }

    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("str2-" + data.msg);
    }
}

//-----------------------------

Set<Integer> blockThreeItems;

@EIterate
Collection<Integer> iterBlockThree() {
    return blockThreeItems;
}

BlockThree newBlockThree(int x) throws Exception {
    return new BlockThree(x, 42);
}

class BlockThree {
    final int number;

    BlockThree(int number, int tt) throws Exception {
        this.number = number;
        if (tt != 42)
            throw new Exception("bad_mojo");
    }

    @EText
    @TOverrides
    String getString1() {
        return "scope-3-get_#" + number;
    }

    @EHtml
    @TOverrides
    void printString2() {
        pw.htmlEncode("scope-3-print-" + number);
    }
}

//-----------------------------

List<Map<String, Integer>> blockFourItems;

@EIterate
Collection<Map<String, Integer>> iterBlockFour() {
    return blockFourItems;
}

BlockFour newBlockFour(Map<String, Integer> x) throws Exception {
```

```
        x.put("num-" + (x.size() + 1), x.size() + 1);
        return new BlockFour(x, 42);
    }

    class BlockFour {
        final int number;

        BlockFour(Map<String, Integer> map, int tt) throws Exception {
            this.number = map.size();
            if (tt != 42)
                throw new Exception("bad_mojo");
        }

        @EText
        @TOverrides
        String getString1() {
            return "scope-4-get_#" + number;
        }

        @EHtml
        @TOverrides
        void printString2() {
            pw.htmlEncode("scope-4-print-" + number);
        }
    }

    //-----------------------

    @EHtml(tag = "title")
    void printTitle() {
        pw.htmlEncode("Testing_gone_amuck_in_iterating");
    }

    protected void emitTemplate() throws Exception {
        new TestIterateTp(this).emit(pw);
    }
}
```

K.21.1 TestIterate.html

```
<html>
<head>
    <title>Testing scopes</title>
</head>
<body>
<div class=_TextOne>This is a text</div>
<div class=String1>global-s1</div>
<div class=String2>global-s1</div>
<div class=BlockOne>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div class=_BlockTwo>
    <div class=_String1>b3s1</div>
    <div class=_String2>b2s2</div>
</div>
<div class=BlockThree>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
<div class=BlockFour>
    <div class=String1>b2s1</div>
    <div class=String2>b2s2</div>
</div>
</body>

</html>
```

K.21.2 TestIterate-expected.html

```
<html>
<head>
<title>Testing gone amuck in iterating</title>
</head>
<body>
<div>Making a test</div>
<div class=String1>Outer-string-1</div>
<div class=String2>Outer-string-2</div>
<div class=BlockOne>
<div class=String1>scope-1-get #1</div>
<div class=String2>str2-scope-1-get #1</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #2</div>
<div class=String2>str2-scope-1-get #2</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #3</div>
<div class=String2>str2-scope-1-get #3</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #4</div>
<div class=String2>str2-scope-1-get #4</div>
</div>
<div class=BlockOne>
<div class=String1>scope-1-get #5</div>
<div class=String2>str2-scope-1-get #5</div>
</div>
<div>
<div>scope-2-get #1</div>
<div>str2-scope-2-get #1</div>
</div>
<div>
<div>scope-2-get #2</div>
<div>str2-scope-2-get #2</div>
</div>
<div>
<div>scope-2-get #3</div>
<div>str2-scope-2-get #3</div>
</div>
<div class=BlockThree>
<div class=String1>scope-3-get #1</div>
<div class=String2>scope-3-print-1</div>
```

```
</div>
<div class=BlockThree>
<div class=String1>scope-3-get #2</div>
<div class=String2>scope-3-print-2</div>
</div>
<div class=BlockFour>
<div class=String1>scope-4-get #1</div>
<div class=String2>scope-4-print-1</div>
</div>
<div class=BlockFour>
<div class=String1>scope-4-get #1</div>
<div class=String2>scope-4-print-1</div>
</div>
<div class=BlockFour>
<div class=String1>scope-4-get #1</div>
<div class=String2>scope-4-print-1</div>
</div>
</body>
</html>
```

K.22 SplitTemplate

```
package dk.vitality.util.jat.junit.tests.sSplitTemplate;

public class SplitTemplate extends TestBase {

    @TCompile(tpname = "SplitTemplatePageTp")
    class Page {
        @EText(tag = "title")
        String pageTitle = "The_split_test_title";

        @EText(tag = "h2")
        String header = "This_is_the_heading";

        @EReplace
        void makeSharedPart() throws Exception {
            pw.println("<!--_pre_SplitTemplateSharedTp_-->");
            new SplitTemplateSharedTp(SplitTemplate.this).emit(pw);
            pw.println("<!--_post_SplitTemplateSharedTp_-->");
        }

        @EReplace
        void makeReplaceMe() throws Exception {
            pw.println("<!--_pre_SplitTemplateReplacedTp_-->");
            new SplitTemplateReplacedTp(SplitTemplate.this).emit(pw);
            pw.println("<!--_post_SplitTemplateReplacedTp_-->");
        }
    }

    protected void emitTemplate() throws Exception {
        new SplitTemplatePageTp(this).emit(pw);
    }

    @TCompile(tpname = "SplitTemplateSharedTp", fragment = "SharedPart")
    static class SplitTemplateShared {
        @EText
        String name1 = "Hilmar_Örn_Hilmarsson";

        @EText
        String name2 = "Herbie_Hancock";

        @EText
        String name3 = "Frankie_Sparo";

        @EText
        String name4 = "Faun_Fables";
    }

    @TCompile(tpname = "SplitTemplateReplacedTp", fragment = "ReplaceMe")
    static class SplitTemplateReplaced {
        @AStyle
        String adjective = "font-weight:_bold;";

        @AStyle
        String replaceme = "color:_green;";
    }
}
```

K.22.1 SplitTemplate.html

```
<html>
<head>
    <title>This is the title</title>
</head>
<body>
<h2>Page header</h2>
<div id=SharedPart>
    <div id=Name1>John A</div>
    <div id=Name2>John A+B</div>
    <div id=Name3>John B</div>
    <div id=Name4>John #2 A+B</div>
</div>
<div id=ReplaceMe>This is <span id=_Adjective>some</span> text</div>
<div>Page footer</div>
</body>
</html>
```

K.22.2 SplitTemplate-expected.html

```
<html>
<head>
<title>The split test title</title>
</head>
<body>
<h2>This is the heading</h2>
<!– pre SplitTemplateSharedTp –>
<div id=SharedPart>
<div id=Name1>Hilmar Örn Hilmarsson</div>
<div id=Name2>Herbie Hancock</div>
<div id=Name3>Frankie Sparo</div>
<div id=Name4>Faun Fables</div>
```

```
</div>
<!– post SplitTemplateSharedTp –>
<!– pre SplitTemplateReplacedTp –>
<div id=ReplaceMe style='color: green;'>This is <span style='font-weight:
bold;'>some</span> text</div>
<!– post SplitTemplateReplacedTp –>
<div>Page footer</div>
</body>
</html>
```

Copyright