JHOVE2: Next-Generation Architecture for Format-Aware Characterization

Architectural Overview

Version 0.11
Issued 2009-08-05
Status Draft

1 Introduction

JHOVE2 is a Java framework and application for next-generation format-aware characterization of digital objects. Characterization is a form of *representation information* about a formatted digital object that is indicative of its significant nature and is useful for purposes of classification, analysis, and use. JHOVE2 supports four specific aspects of characterization:

- *Identification*. The process of determining the *presumptive* format of a digital object on the basis of suggestive extrinsic hints and intrinsic signatures, both internal (e.g. magic number) and external (e.g. file extension).
- *Validation*. The process of determining the level of *conformance* to the normative syntactic and semantic rules defined by the authoritative specification of the object's format.
- *Feature extraction*. The process of reporting the *intrinsic properties* of a digital object significant for purposes of classification, analysis, and use.
- *Assessment*. The process of determining the level of *acceptability* of a digital object for a specific purpose on the basis of locally-defined policy rules.

Additional information about JHOVE2 can be found on the public project wiki at confluence.ucop.edu/display/JHOVE2Info/Home.

2 Architecture

The overall JHOVE2 architecture is defined in terms of three layers:

- *Application*. The JHOVE2 application is responsible for managing the interface presented to the JHOVE2 user.

  JHOVE2 is distributed with a single command-line application that accepts a list of *names* (files, directories, URLs) and command options and displays formatted results. The command line syntax is:

  ```
  % jhove2 [-ik] [-b size] [-B Direct|NonDirect|Mapped]
          [-d JSON|Text|XML] [-f limit] [-o file] name ...
  ```

  where

`–i`	specifies that the unique identifiers for each reportable property are shown;
`-k`	specifies that message digests are to be calculated;
`–b size`	specifies the I/O buffer size (defaults to 131072 bytes);
`-B`	specifies the buffer type: `Direct` (default), `NonDirect`, or `Mapped`;
`-d`	specifies the display form: `JSON`, `Text` (default), or `XML`;
`-f limit`	specifies the fail fast limit (defaults to `0`);
`-o file`	specifies the name of an output (defaults to the standard output unit); and
`name`	is the file or directory name or URI to be characterized.

where square brackets `[` and `]` enclose optional elements; and a vertical bar `|` separates alternative choices. The fail fast limit specifies the number of errors that JHOVE2 will report before terminating the examination of a given object; the default value of `0` indicates that there is no limit and that error reporting will be exhaustive.

- *Framework*. The JHOVE2 framework coordinates all JHOVE2 processing. In short, the framework is presented with a list of *names* (files, directories, URLs) that are dispatched to various modules for appropriate processing and results display.

  Using the framework, the minimal JHOVE2 application is:

  ```
  JHOVE2 jhove2 = Configure.getReportable(JHOVE2.class, “JHOVE2”);
  jhove2.characterize(“file.ext”, ...);
  jhove2.display();
  ```

  which instantiates the JHOVE2 framework and uses it to characterize a list of named files, directories, or URLs, and display the results in structured text form (as name/value pairs) to the standard output unit.

- *Modules*. JHOVE2 modules are independent components that encapsulate specific processing behaviors. (Technically, the JHOVE2 application and framework are also modules, but due to their central role in JHOVE2 processing it is useful to consider them as conceptually distinct levels.)

  JHOVE2 uses the Spring Java enterprise platform (www.springsource.org) to manage module instantiation. JHOVE2 is implemented in terms of the Spring philosophy of *dependency injection*: all relationships between JHOVE2 components are defined in external configuration files, not the Java source code, so they can be modified easily by a JHOVE2 user at the time of installation or invocation. However, all Spring function is encapsulated in a single class (`org.jhove2.core.config.Configure`); all other classes are POJOs (“plain old Java objects”) that can be easily invoked in non-Spring contexts, if so desired.

  All JHOVE2 components are defined as Spring beans in the configuration file `jhove2-config.xml`.

  The JHOVE2 framework is configured with three modules, which may in turn invoke subsidiary modules:

  - *Characterizer* module. The characterizer module embodies a specific characterization *strategy*.

    The unit of JHOVE2 characterization examination and reporting is known as a *source unit*. A source unit can be an unitary entity such as a file or a bytestream within a file, or an aggregate entity such as a directory; a *file set*, a set of unrelated files (for example, the set of file names specified on the command line); or a *clump*, a set of related files that form a coherent logical object (for example, a Shapefile, which is defined by at least three independent files).

    The strategy used by the Characterizer module in the standard JHOVE2 distribution is (see Figure 1):

    - Signature-based unitary format identification performed by a configurable

Identifier module. The Identifier module in the standard JHOVE2 distribution is a wrapped version of DROID (droid.sourceforge.net) using signature data from the PRONOM format registry (www.nationalarchives.gov.uk/pronom).

- Dispatch to a format module associated with the source unit's format for parsing, feature extraction, and validation performed by a configurable *Dispatcher* module.
- Rules-based assessment performed by a configurable *Assessor* module.
- For unitary source units (i.e. file or bytestream), optional message digesting performed by a configurable *Digester* module. The set of digests to be calculated is configurable, and may include any combination of:
  - Adler-32
  - CRC-32
  - MD2, MD5
  - SHA-1, SHA-256, SHA-384, SHA512
- If the source unit is an *aggregate* (i.e. directory or file set), aggregate-level identification performed by a configurable *Aggrefier* module ("aggregate-identifier"). If an aggregate format is identified, the source unit is dispatched to the appropriate format module and the Assessor module.

o *Dispatcher* module. The Dispatcher module transfers control to a format module capable of parsing, extracting features, and validating a formatted source unit. The Dispatcher module in the standard JHOVE2 distribution is based on a user configurable map from unique format identifiers (in the JHOVE2 namespace) to format module classes, which must be found on the Java classpath. If a nested source unit is discovered by the format module during parsing, the Characterizer module is recursively invoked. By definition, the aggregate directory, file set, and clump source units all have nested child source units. Non-aggregate file and bytestream source units may have nested children, for example, a TIFF file with an embedded ICC profile and XMP metadata. Nested source units are automatically recursively characterized.

o *Displayer* module. The displayer module displays characterization results in one of three forms:

- JSON
- Text
- XML

The Text formatted results are in the form of property name/value pairs, where indentation indicates the reportable nesting level.

```
<name>: <value>
...
```

If the show identifiers command line option is specified (`-i`) the form is modified to:

```
<name> [<identifier>]: <value>
...
```

The XML formatted results confirm to a JHOVE2-specific schema that is intended to be an *intermediate* form that can be converted by a stylesheet transformation into any desired *final* form. The general form for schema is:

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”yes”?>
<j2:jhove2 xmlns:j2=”http://jhove2.org/xsd/1.0.0”>
<j2:reportable>
  <j2:name>name</j2:name>
  <j2:identifier namespace=”namespace”>identifier</j2:identifier>
  <j2:properties>
    <j2:property>
      <j2:name>name</j2:name>
      <j2:identifier namespace=”namespace”>identifier</j2:identifier>
      <j2:value>value</j2:value>
    </j2:property>
    ...
  </j2:reportable>
  ...
</j2:property>
```

Since many formats define *families* of related sub-formats or *profiles*, an arbitrary number of format profiles can be registered with a format module. (Note that for JHOVE2 purposes certain aggregate entities such as directories and files sets are considered formats in their own right. An aggregate entity is processed by independently characterizing each of its subsidiary components and then trying to identify any recognizable aggregate formats.) The standard JHOVE2 distribution includes format modules and profiles for:

▪ Clump	
▪ dBASE	
▪ Directory	
▪ File set	
▪ ICC	
▪ JPEG 2000	JP2 (ISO/IEC 15444-1) and JPX (ISO/IEC 15444-2)
▪ PDF	1.0 – 1.7 (ISO 32000-1), PDF-A (ISO 19005-1), and PDF-X (ISO 15930)
▪ SGML	
▪ Shapefile	
▪ TIFF	4.0 – 6.0, Class B, F, G, P, R, and Y, TIFF-FX, RFC 1314, TIFF/EP (ISO 12234-2), TIFT/IT (ISO 12639), Exif (EITA CP-3451), GeoTIFF, and DNG
▪ UTF-8	ASCII
▪ WAVE	
▪ XML	BWF (EBU N22-1997)
▪ Zip	

3 Object Modeling

JHOVE2 is implemented in Java J2SE 6. The major JHOVE2 conceptual entities are described and their embodiment as Java interfaces and classes are summarized in the following sections. The general modeling paradigm is to define entity behavior in terms of *interfaces*, to implement common utility function applicable in general contexts in *abstract classes*, and to extend these to *concrete classes* for specific contexts.

JHOVE2 characterization results are defined in terms of *reportable properties*, which are named, typed values. These values can be of *primitive* or *complex* type.

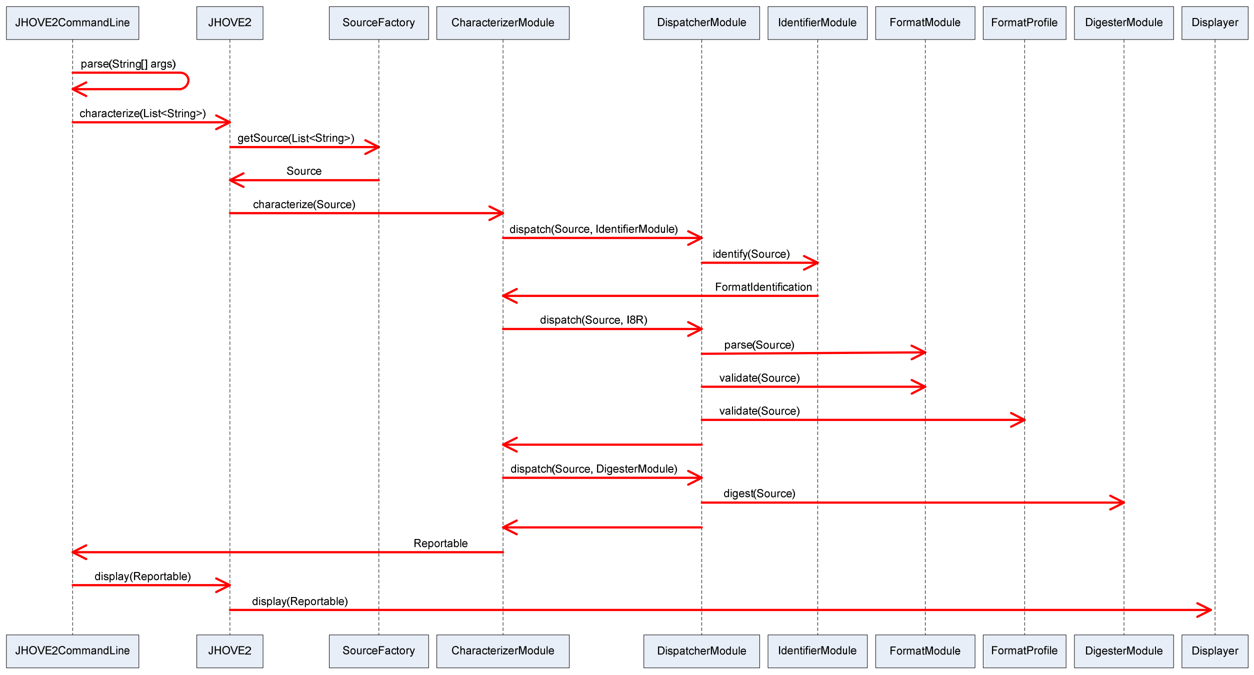

Figure 1 – JHOVE2 sequence diagram

3.1 Primitive types

JHOVE2 defines four primitive property types, each modeled by a concrete class:

- *Digest*. An algorithm-specific message digest.
- *Duration*. A unit of elapsed time.
- *I8R*. A namespace-typed identifier.
- *Message*. An error, warning, or informative message.

The class name "`I8R`" is used for the identifiers of reportable properties and other JHOVE2 entities; the name "`Identifier`" is reserved for the interface implemented by identification modules.

3.1.1 Digest

A JHOVE2 *digest* is an algorithm-specific message digest value.

```
package org.jhove2.core
public class Digest
   implements Comparable<Digest>
{
   public enum Algorithm {
      Adler32, CRC32, MD2, MD5, SHA1, SHA256, SHA384, SHA512
   }
   public Digest(String value, Algorithm algorithm);

   public Algorithm getAlgorithm();
   public String    getValue();
   public String    toString();
}
```

The `toString()` method returns a representation of the digest value in the form:

```
[<algorithm>] <hex-encoded-value>
```

3.1.2 Duration

A JHOVE2 *duration* is a unit of elapsed time with millisecond accuracy.

```
package org.jhove2.core;
public class Duration {
   public Duration(long duration);

   public long   getDuration();
   public String toString();
}
```

The `toString()` method returns a representation of the duration in the form:

```
<hh>:<mm>:<ss>.<msec>
```

The shortest reportable duration is 1 millisecond (`00:00:00:001`), regardless of whether the actual duration is shorter.

3.1.3 Identifier

A JHOVE2 *identifier* is a namespace-typed value.

```
package org.jhove2.core;
public class I8R
   implements Comparable<I8R>
{
   public enum Namespace {
      AFNOR, AIIM, ANSI,..., JHOVE2,..., URI, URL, URN, UTI, OTHER
   }
   public Identifier(String value) {
      this(value, Namespace.JHOVE2);
   }
   public Identifier(String value, Namespace namespace);

   public Namespace getNamespace();
   public String    getValue();
   public String    toString();
}
```

The class name `I8R` is used since `Identifier` is in alternative use as the interface defining the behavior of the Identifier module.

The `toString()` method returns a representation of the identifier in the form:

```
[<namespace>] <value>
```

Every JHOVE2 entity, property, and format is given a unique formal identifier in the JHOVE2 namespace. This namespace uses the "info" URI identifier scheme, leading to identifiers of the general form:

`info:jhove2/<function>/<specific>`

where <*function*> describes a high-level functional category and <*specific*> describes a specific item within that category. The four defined identifier functions are:

- `reportable`. An identifier for an entity encapsulating one or more reportable properties. The specific portion of a reportable identifier is based on the package-qualified name of the class embodying the reportable with periods (.) replaced by slashes (/), e.g. `info:jhove2/reportable/org/jhove2/core/JHOVE2`.
- `property`. An identifier for a reportable property. The specific portion of a property identifier is the concatenation of the package-qualified name of the enclosing reportable and the property name, e.g. `info:jhove2/property/org/jhove2/core/JHOVE2/DateTime`.
- `message`. An identifier for a reportable message. The specific portion of the property identifier is the concatenation of the package-qualified name of the enclosing reportable and the message name, e.g. `info:jhove2/message/org/jhove2/module/format/utf8/UTF8Module/ByteOrderMark`.
- `format`. An identifier for a format. The specific portion of a format identifier is based on the format's common name, e.g. `info:jhove2/format/utf-8`.

3.1.4 Message

A JHOVE2 *message* is a property drawing notice to a specific unanticipated or unexpected condition or event. A message can be generated in one of two contexts:

- *Process*. A message documenting an unanticipated condition arising from the process of characterization, for example a file not found exception.
- *Object*. A message documenting an unexpected condition in the examined source unit, for example, a validation error.

A message can have one of three severities:

- *Error*. A message about a terminal condition generally requiring some remedial action or reaction by the user.
- *Warning*. A message about a condition that may requiring some further action by the user.
- *Informative*. A informative message requiring no further action.

```
package org.jhove2.core;
public class Message {
   public enum Context {
      Process,
      Object
   }
   public enum Severity {
      Error,
      Warning,
      Info
```

```
        }
        public Message(Context context, Severity severity, String message);

        public Context  getContext();
        public String   getMessage();
        public Severity getSeverity();
        public String   toString();
    }
```

The `toString()` method returning a String of the general form:

```
    [<context>/<severity>] <message>
```

3.2 Complex types

JHOVE2 defines a number of complex property types, which are aggregate containers of reportable properties.

3.2.1 Reportable

The fundamental JHOVE2 complex type is the *reportable*. A reportable is a named container of properties. A reportable has two types of names:

- A short descriptive name based on the simple, that is, non-package-qualified, name of the embodying class; and
- A formal identifier in the JHOVE2 namespace based on the package-qualified class name.

  ```
  info:jhove2/reportable/<package-path>/<name>
  ```

  Thus, the Agent reportable embodied by the class `org.jhove2.core.Agent` has the descriptive name "`Agent`" and the identifier "`info:jhove2/reportable/org/jhove2/core/Agent`".

Reportables are declared by implementing the Reportable marker interface.

```
    package org.jhove2.core;
    public interface Reportable {}
```

A reportable defines its properties by marking their accessor methods with the "`@ReportableProperty`" annotation. This annotation has three arguments:

- `order`. The ordinal position of the property, relative to all reportable properties, used when the reportable is displayed;
- `value`. A brief description of the property; and
- `ref`. An optional reference to external documentation related to the property.

A reportable property may be annotated in an interface implemented by the reportable or an abstract or concrete class extended by or embodying the reportable.

Like reportables, each property has two names:

- A short descriptive name based on its accessor method; and

* A unique identifier in the JHOVE2 namespace.

Thus Name property of the Agent reportable has the descriptive name "`Name`" and the identifier "`info:jhove2/property/org/jhove2/core/Agent/Name`", as defined by:

```
package org.jhove2.core;
public class Agent
   implements Reportable
{
   @ReportableProperty(order=1, value="Agent name.")
   public String getName();
}
```

Most major JHOVE2 components implement or extend the Reportable interface (see Figure 2).

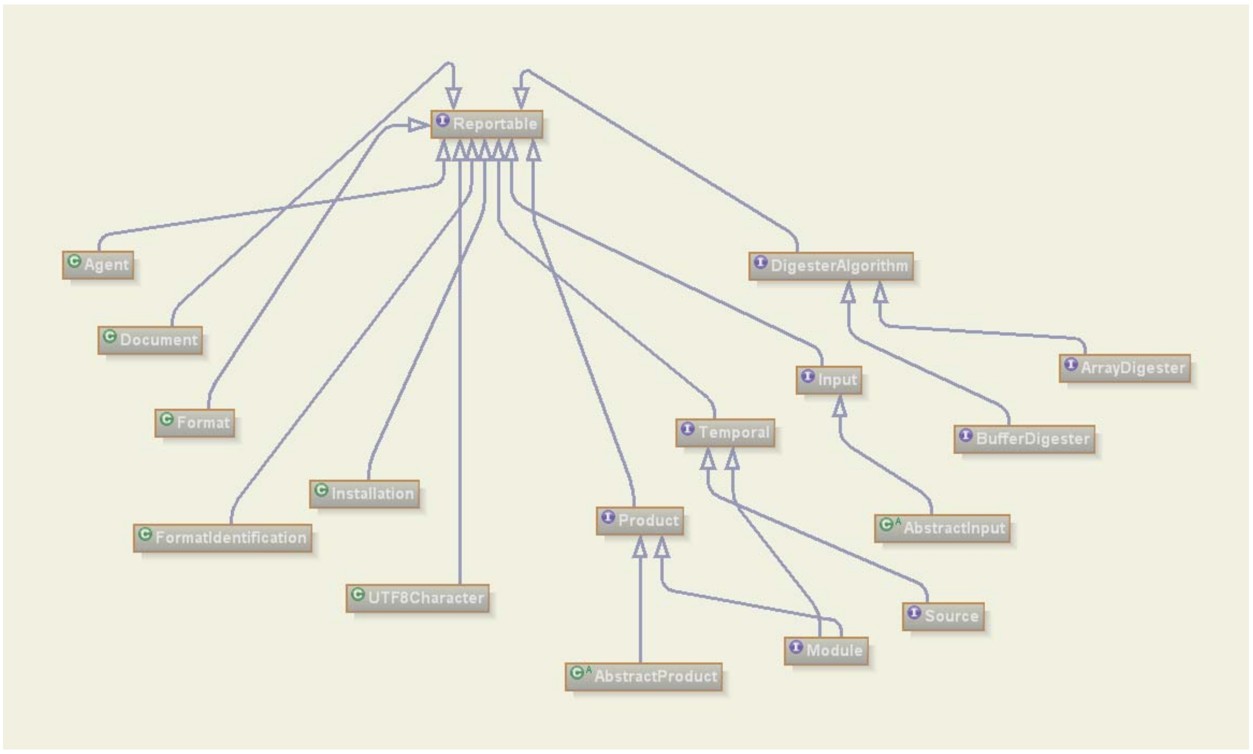

Figure 2 – Reportable object modeling

3.2.2 Temporal

A JHOVE2 *temporal* is a reportable that performs some process in a given amount of elapsed time.

```
package org.jhove2.core;
public interface Temporal
   extends Reportable
{
   @ReportableProperty("Elapsed time, in milliseconds")
   public Duration getElapsedTime();

   public void setStartTime();
```

```
      public void setRestartTime();
      public void setEndTime();
   }
```

A temporal reportable can accumulate its time in an arbitrary number of independent durations. The first duration is bracketed by invocations of the `setStartTime()/setEndTime()` methods; subsequent durations are bracketed by the `setResetTime()/setEndTime()` methods.

3.2.3 Product

A JHOVE2 *product* is a reportable that is independently distributable and configurable.

```
package org.jhove2.core;
public interface Product
   extends Reportable
{
   public Product(String version, String release, String rights);
   @ReportableProperty(order=4, value="Product developers.")
   public List<Agent> getDevelopers();

   @ReportableProperty(order=1, value="Product name.")
   public String getName();

   @ReportableProperty(order=6, value="Product informative note.")
   public String getNote();

   @ReportableProperty(order=3, value="Product release date.")
   public String getReleaseDate();

   @ReportableProperty(order=5, value="Product rights statement.")
   public String getRightsStatement();

   @ReportableProperty(order=2, value="Product version identifier.")
   public String getVersion();
}
public abstract class AbstractProduct
   implements Product;
```

3.2.4 Agent

A JHOVE2 *agent* is a reportable that represents an individual or corporate agent. Agents can be one of two types:

- Corporate. A institutional or organizational agent.
- Individual. An individual human agent.

```
package org.jhove2.core;
public class Agent
   implements Reportable
{
   public Type {
      Corporate,
      Individual
   }
```

```
      public Agent(String name, Type type);

      @ReportableProperty(order=4, value=”Agent postal address.”)
      public String getAddress();

      @ReportableProperty(order=3, value=”Agent affiliation.”)
      public Agent getAffiliation();

      @ReportableProperty(order=7, value=”Agent email address.”)
      public String getEmail();

      @ReportableProperty(order=6, value=”Agent fax number.”)
      public String getFax();

      @ReportableProperty(order=1, value=”Agent name.”)
      public String getName();

      @ReportableProperty(order=9, value=”Agent informative note.”)
      public String getNote();

      @ReportableProperty(order=5, value=”Agent telephone number.”)
      public String getTelephone();

      @ReportableProperty(order=2, value=”Agent type.”)
      public Type getType();

      @ReportableProperty(order=8, value=”Agent URI.”)
      public String getURI();
   }
```

3.2.5 Document

A JHOVE2 *document* is a reportable that represents a specification document. A document is classified with regard to its *intention*, that is, the degree of authority it carries; and its *type*.

```
   package org.jhove2.core;
   public class Document
      implements Reportable
   {
      public enum Intention {
         Authoritative,
         Informative,
         Speculative,
         Other,
         Unknown
      }
      public enum Type {
         Article, Codebook,..., Other, Unknown
      }
      public Document(String author, String title, Type type,
                      Intention intention);

      @ReportableProperty(order=1, value=”Document author or authors.”)
      public String getAuthor();

      @ReportableProperty(order=5, value=”Document publication date.”)
      public String getDate();

      @ReportableProperty(order=3, value=”Document edition.”)
      public String getEdition();

      @ReportableProperty(order=6, value=”Document formal identifiers.”)
      public List<Identifier> getIdentifiers();

      @ReportableProperty(order=4, value=”Document intention.”)
      public Intention getIntention();

      @ReportableProperty(order=8, value=”Document informative note.”)
      public String getNote();

      @ReportableProperty(order=4, value=”Document publisher or publishers.”)
      public String getPublisher();

      @ReportableProperty(order=2, value=”Document title.”)
      public String getTitle();

      @ReportableProperty(order=7, value=”Document type.”)
      public Type getType();
   }
```

3.2.6 Format

A JHOVE2 *format* is a reportable that represents a format. A format is classified with regard to its *ambiguity*, that is, the degree to which there is clear community consensus on the interpretation of the formats normative requirements; and its *type*.

```
package org.jhove2.core;
public class Format
   implements Reportable
{
   public enum Ambiguity {
      Ambiguous, Unambiguous
   }
   public enum Type {
      Family, Format
   }
   public Format(String name, I8R identifier, Type type,
                 Ambiguity ambiguity);

   @ReportableProperty(order=5, value=”Format alias identifiers.”)
   public Set<Identifier> getAliasIdentifiers();

   @ReportableProperty(order=6, value=”Format alias names.”)
   public Set<String> getAliasNames();

   @ReportableProperty(order=8, value=”Format ambiguity.”)
   public Ambiguity getAmbiguity();

   @ReportableProperty(order=9, value=”Format caveats.”)
   public String getCaveats();
```

```
      @ReportableProperty(order=2, value=”Format canonical identifier.”)
      public I8R getIdentifier();

      @ReportableProperty(order=1, value=”Format canonical name.”)
      public String getName();

      @ReportableProperty(order=10, value=”Format informative note.”)
      public String getNote();

      @ReportableProperty(order=7, value=”Format specification documents.”)
      public List<Document> getSpecifications();

      @ReportableProperty(order=4, value=”Format type.”)
      public Type getType();

      @ReportableProperty(order=3, value=”Format version.”)
      public String getVersion();
   }
```

3.2.7 Format identification

A JHOVE2 *format identification* is a reportable that represents a presumptive identification of a format based on some identification strategy embodied by the Identification module. All format identifications are associated with the *process* used to perform the identification and a level of *confidence* about the identification. Confidence levels are based on the union of those defined by DROID (droid.sourceforge.net) and the DSpace format identification framework (wiki.dspace.org/index.php/BitstreamFormat_Renovation)..

```
   package org.jhove2.core;
   public class FormatIdentification
      implements Reportable, Comparable<FormatIdentification>
   {
      public enum Confidence {
         Negative,
         Tentative,
         Heuristic,
         PositiveGeneric,
         PositiveSpecific,
         Validated
      }
      public FormatIdentification(Product process, Confidence confidence,
                                  Format format);

      @ReportableProperty(order=2, value=”Level of confidence.”)
      public Confidence getConfidence();

      @ReportableProperty(orde=3, value=”Presumptively-identified format.”)
      public Format getPresumptiveFormat();

      @ReportableProperty(order=1, value=”Identifying process.”)
      public Product getProcess();

      public List<Source> getSources();
   }
```

3.3 Source units

A JHOVE2 *source unit* is a temporal reportable that is the unit of characterization. Source units are classified as being either *unitary* or *aggregate*. The following source units are defined (see Figure 3):

• File set	[*aggregate*].
• Directory	[*aggregate*].
• Clump	[*aggregate*].
• File	[*unitary*].
• Bytestream	[*unitary*].
• URL	[*unitary*].
• Zip directory entry	[*aggregate*].
• Zip file entry	[*unitary*].

Note that source units are defined with a granularity both larger than a file (i.e. file set, directory, clump) and smaller than a file (i.e. bytestream). Source units can be arbitrarily nested. Aggregate source units can be the structural parent of files or other aggregates; unitary source units can be the structural parent of bytestreams. Nested source units encountered during the parsing of unitary source units are automatically recursively characterized.

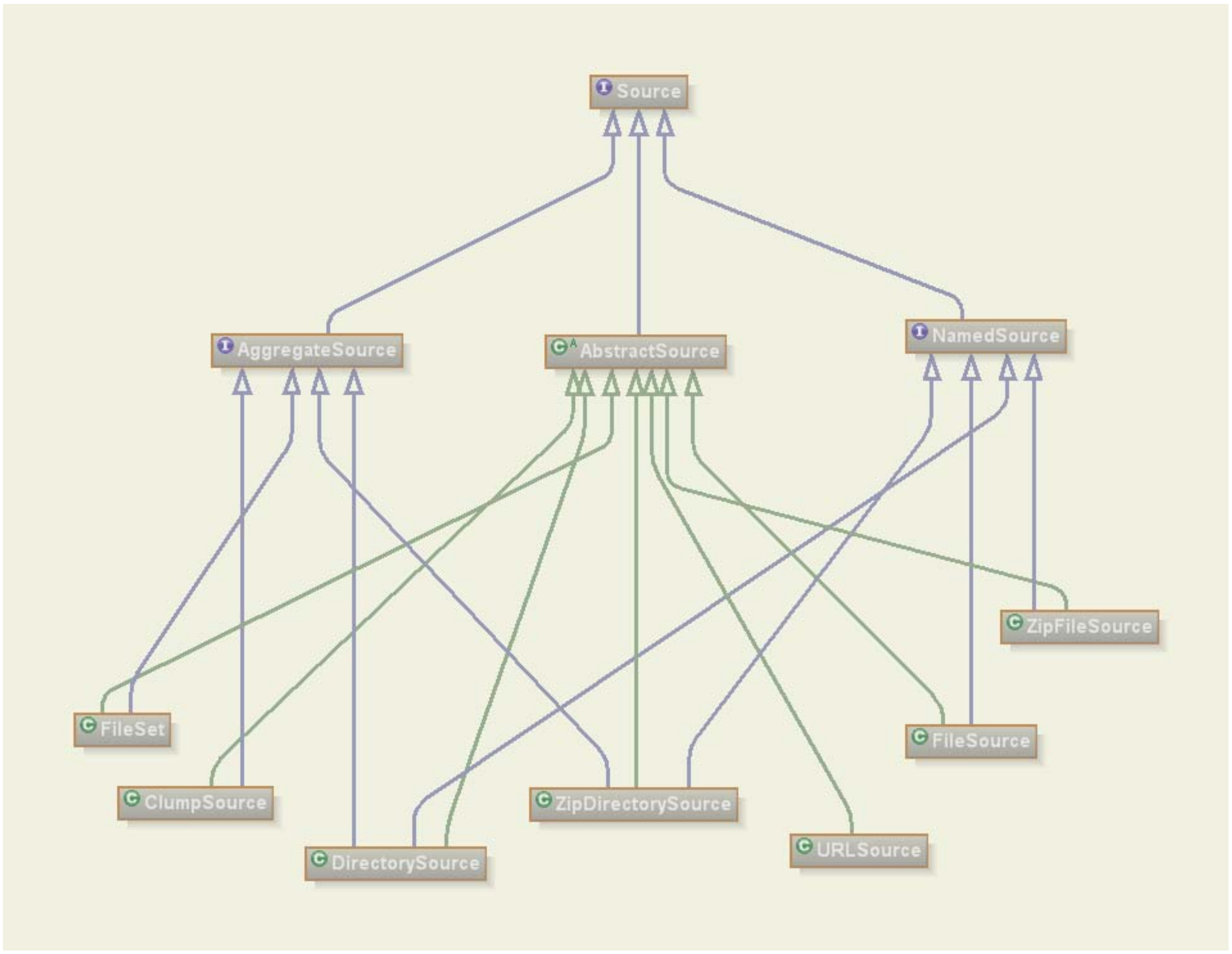

Figure 3 – Source unit object modeling

A unitary source unit can expose its content as:

- Java `File` [`java.io.File`].
- Java `InputStream` [`java.io.InputStream`].
- JHOVE2 `Input`.

If there is no pre-existing file backing the source unit, a temporary file is automatically generated.

A JHOVE2 *input* is a reportable that provides uniform access the contents of source units in terms of the Java buffered I/O package [`java.nio`]

A factory class is available to instantiate new source units.

```
package org.jhove2.core.source;
public interface Source
   extends Temporal
{
   public void close();
   public void deleteChildSource(Source child);

   @ReportableProperty(“Child source units.”)
   public List<Source> getChildSources();
   public File getFile();
   public Input getInput(int bufferSize, Type bufferType, ByteOrder order);
   public InputStream getInputStream();

   @ReportableProperty(value=”Modules that processed the source unit.”)
   public List<Module> getModules();
   public int getNumModules();
   public void setChildSource(Source child);
   public void setModule(Module module);
}
public abstract class AbstractSource
   implements Source;
public interface NamedSource
   extends Source
{
   @ReportableProperty(“Source unit name.”)
   pubic String getName();
}
public interface AggregateSource
   extends Source;
public class ClumpSource
   extends AbstractSource
   implements AggregateSource;
public class DirectorySource
   extends AbstractSource
   implements AggregateSource, NamedSource;
public class FileSetSource
   extends AbstractSource
   implements AggregateSource;
public class FileSource
   extends AbstractSource
   implements NamedSource;
public class URLSource
```

```
      extends AbstractSource;
   public class ZipDirectorySource
      extends AbstractSource
      implements AggregateSource, NamedSource;
   public class ZipEntrySource
      extends AbstractSource
      implements NamedSource;
   public abstract class SourceFactory {
      public static Source getSource(String pathname);
      public static Source getSource(File file);
      public static Source getSource(JHOVE2 jhove2, ZipFile zipFile,
                                     ZipEntry entry);
   }
```

Once all processing has been completed on a source unit, it most be closed [`close()`] in order to release all open system resources.

3.4 Input

A JHOVE2 *input* is a reportable that provides uniform access to the contents of source units in terms of the Java buffered I/O package [`java.nio`].

Inputs can be defined with three underlying buffer types:

- Direct.
- Non-direct.
- Memory mapped.

An input can expose its content as:

• Java `File`	[`java.io.File`].
• Java `InputStream`	[`java.io.InputStream`].

A factory class is available to instantiate new source units.

```
   package org.jhov2.core.io;
   public interface Input
      Reportable
   {
      public enum Type {
         Direct, NonDirect, MemoryMapped
      };
      public final static int EOF = -1;

      public void close();
      public ByteBuffer getBuffer();
      public int getBufferSize();
      public byte [] getByteArray();
      public ByteOrder getByteOrder();
      public File getFile();
      public InputStream getInputStream();
      public int getMaxBufferSize();
      public long getPosition();
      public long getSize();
      public void setByteOrder(ByteOrder order);
      public void setPosition(long position);

      public byte  readSignedByte();
      public int   readSignedInt();
      public short readSignedShort();
      public long  readSignedLong();
      public short readUnsignedByte();
      public int   readUnsignedShort();
      public long  readUnsignedInt();
   }
   public abstract class AbstractInput
      implements Input;
   public class DirectInput
      extends AbstractInput;
   public class NonDirectInput
      extends AbstractInput;
   public class MappedInput
      extends AbstractInput;
   public abstract class InputFactory {
      public static Input getInput(File file, int bufferSize, Type type);
   }
```

Once all processing has been completed on a source unit, it most be closed [`close()`] in order to release all open system resources.

3.5 Modules

A JHOVE2 *module* is a temporal product reportable.

```
package org.jhove2.module;
public interface Module
   extends Product, Temporal
{
   @ReportableProperty("External tool wrapped by the module.")
   public Product getWrappedProduct();
}
public abstract class AbstractModule
   extends AbstractProduct
   implements Module
{
   public AbstractModule(String version, String release, String rights);
}
```

Modules embody specific JHOVE2 processing capabilities as defined by appropriate interfaces (see Figure 4).

3.5.1 Assessment modules

JHOVE2 *assessment* modules perform assessments of source units based on prior characterization information and locally-configured assessment assertions. [*TBD*…]

```
package org.jhove2.module.assess;
...
```

3.5.2 Characterization modules

JHOVE2 *characterization* modules encapsulate specific characterization strategies.

```
package org.jhove2.module.characterize;
public interface Characterizer
   extends Module
{
   public void characterize(JHOVE2 jhove2, Source source);
}
public class CharacterizerModule
   extends AbstractModule
   implements Characterizer;
```

The default strategy of the characterization module supplied in the standard JHOVE2 distribution is:

- Unitary identification of the source unit.
- Dispatch of the source unit to an appropriate format module for feature extraction, and validation, with appropriate recursive characterization of any nested source units that are encountered.
- Optional message digesting of unitary source units.
- Aggregate identification of aggregate source units with appropriate recursive characterization.

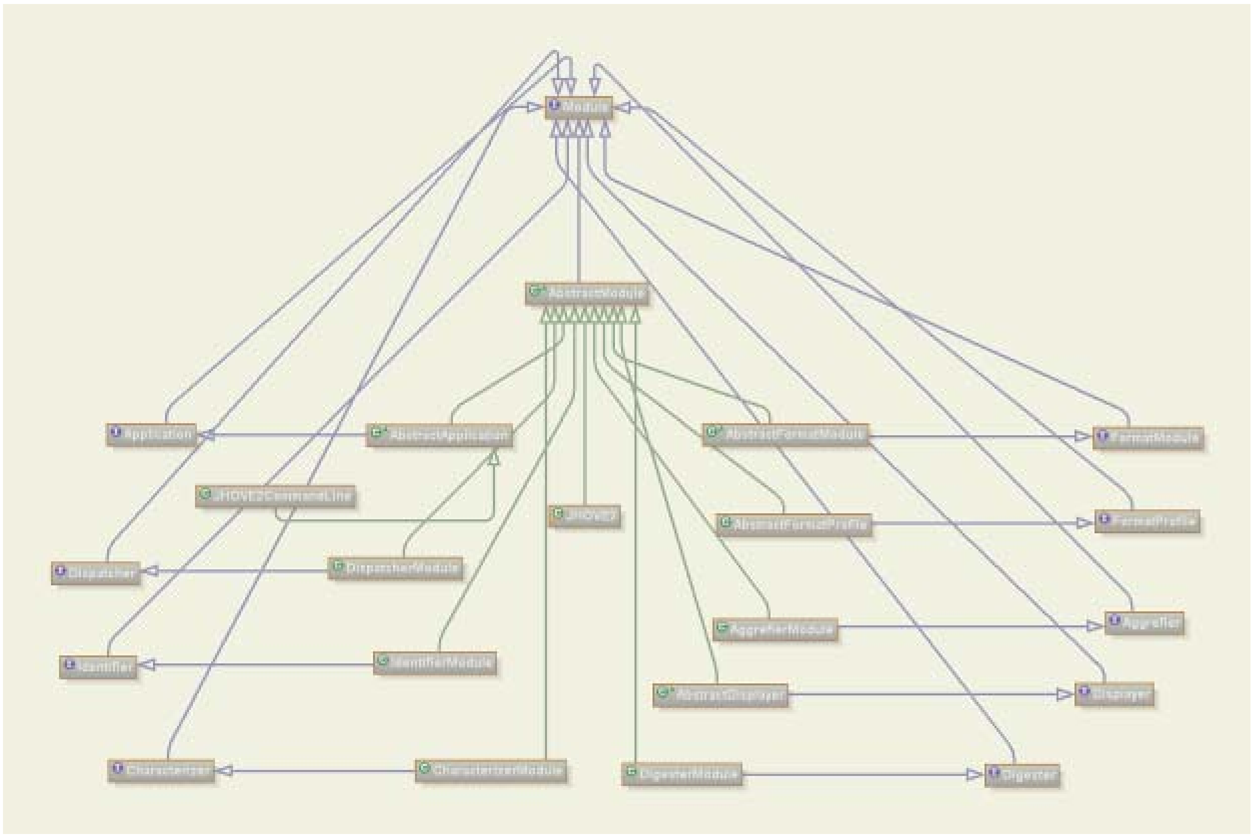

Figure 4 – Module object modeling

3.5.3 Digesting modules

JHOVE2 *digesting* modules calculate message digests.

```
package org.jhove2.module.digest;
public interface Digester
   extends Module
{
   public void digest(JHOVE2 jhove2, Source source);
   @ReportableProperty("Get message digests.")
   public Set<Digest> getDigests();
}
public class DigesterModule
   extends AbstractModule
   implements Digester;
```

The default digesting module supplied in the standard JHOVE2 distribution can calculate digests for any combination of the following algorithms:

- Adler-32
- CRC-32
- MD2, MD5
- SHA-1, SHA-256, SHA-384, SHA-512

3.5.4 Dispatching modules

JHOVE2 *dispatching* modules transfer control to an explicitly provided module or the module associated with a format identifier.

```
package org.jhove2.module.dispatch;
public interface Dispatcher
   extends Module
{
   public Module dispatch(JHOVE2 jhove2, Source source, I8R identifier);
   public Module dispatch(JHOVE2 jhove2, Source source, Module module);
}
public class DispatcherModule
   extends AbstractModule
   implements Dispatcher;
```

The default dispatching module supplied in the standard JHOVE2 is initialized with a set of mappings defined by the `dispatcher.properties` file. Each line of this file contains a mapping from a I8R format identifier to the Spring bean name of a format module, for example:

```
info\:jhove2/format/utf-8 UTF8Module
```

(The colon in the identifier string must be escaped to conform with standard Java property file semantics.)

3.5.5 Displayer modules

JHOVE2 *displayer* modules present formatted results to an output print stream.

```
package org.jhove2.module.display;
public interface Displayer
   extends Module
```

```
    {
       public void startDisplay(PrintStream out, int level);
       public void startReportable(PrintStream out, int level, String name,
                                   I8R identifier, int order);
       public void startCollection(PrintStream out, int level, String name,
                                   I8R identifier, int size, int order);
       public void displayProperty(PrintStream out, int level, String name,
                                   I8R identifier, Object value, int order);
       public void endCollection(PrintStream out, int level, String name,
                                 I8R identifier, int size);
       public void endReportable(PrintStream out, int level, String name,
                                  I8R identifier);
       public void endDisplay(PrintStream out, int level);
    }
```

For purposes of display, a *collection* is a non-scalar property such as a list or a set. The `level` argument indicates the nesting level of the various reportables, collections, and properties. The `order` argument indicates the ordinal position of the entity (reportable, collection, property) relative to its siblings.

The default displayer modules supplied in the standard JHOVE2 distribution can provide results in the following formats:

- JSON
- Text (i.e. name/value pairs; default)
- XML

By default, all characterization properties are displayed. However, the visibility of individual properties can be controlled by setting directives in the `displayer.properties` file.

```
    <property-identifier> <display-directive>
```

where <*`property-identifier`*> is the unique I8R identifier for a reportable, collection , or property; and <*`display-directive`*> is a directive that is evaluated with respect to the property value to determine whether or not the property is displayed:

- Never
- IfFalse
- IfTrue
- IfZero
- IfNonZero
- IfNegative [display if *value* < 0]
- IfPositive [display if *value* > 0]
- IfNonNegative [display if *value* ≥ 0]
- IfNonPositive [display if *value* ≥ 0]
- Always

3.5.6 Format modules

JHOVE2 *format* modules are able to parse, extract features from, and validate formatted source units. For JHOVE2 purposes, file sets, directories, and clumps are all considered to be formats.

Since formats are often defined in terms of related sets of formats, JHOVE2 formats are classified as:

- Format *families*.
- *Formats* or format *profiles*.

Format modules are generally defined at the level of a format family. An arbitrary number of format profiles can be registered with a format module. All of these profiles are automatically invoked following the processing of a source by a format module.

```
package org.jhove2.module.format;
public interface FormatModule
   extends Module
{
   @ReportableProperty(“Format module format.”)
   public Format getFormat();

   @ReportableProperty(“Format module format profiles.”)
   public List<FormatProfile> getProfiles();
   public void setProfile(FormatProfile profile);
}
public abstract class AbstractFormatModule
   extends AbstractModule
   implements FormatModule;
```

3.5.6.1 *Parsing format modules*

JHOVE2 *parsing* format modules are capable of parsing and extract features from the contents of formatted source units.

```
package org.jhove2.module.format;
public interface Parser {
   public long parse(JHOVE2 jhove2, Source source);
}
```

An aggregate source unit is parsed by iteratively characterizing each of its child source units. During the parsing of unitary source units, additional nested source units may be encountered; if so, they are automatically recursively characterized.

3.5.6.2 *Validating format modules*

JHOVE2 *validating* format modules are capable of validating formatted source units.

```
package org.jhove2.module.format;
public interface Validator {
   public enum Validity {
      True, False, Undetermined
   }
   public Validity validate(JHOVE2 jhove2, Source source);
   @ReportableProperty(“Validation status.”)
   public Validity isValid();
}
```

3.5.6.3 *Format profiles*

JHOVE2 format *profiles* are modules that can validate profiles of a format family. In general format profiles are expected to implement the `Validator` interface, but *not* the `Parser` interface. Validation is performed on the basis of the pre-existing characterization information.

All of the format profiles registered with a given format module are automatically invoked following the conclusion of that module's processing of a source unit.

```
package org.jhove2.module.format;
public interface FormatProfile
   extends Module
{
   @ReportableProperty("Format profile format.")
   public Format getFormat();
   public void setFormatModule(FormatModule module);
}
```

3.5.7 Identification modules

JHOVE2 *identification* modules make presumptive assertions about the formats of examined source units on the basis of suggestive hints and intrinsic internal and external signatures. Identification can occur at two levels:

- *Unitary*. Unitary identification is performed relative to file and bytestream source units.
- *Aggregate*. Aggregate identification is performed relative to file set and directory source units.

Unitary identification is defined by the following interface and module:

```
package org.jhove2.module.identify;
public interface Identifier
   extends Module
{
   public Set<FormatIdentification> identify(JHOVE2 jhove2, Source source);

   @ReportableProperty("Presumptive format identifications.")
   public Set<FormatIdentification> getPresumptiveFormats();
}
public class IdentifierModule
   extends AbstractModule
   implements Identifier;
```

Aggregate identification is defined by the following interface and module:

```
package org.jhove2.module.identify;
public interface Identifier
   extends Module
{
   public Set<FormatIdentification> identify(JHOVE2 jhove2,
                                             AggregateSource source);
   @ReportableProperty("Presumptive format identifications.")
   public Set<FormatIdentification> getPresumptiveFormats();
}
public class AggrefierModule
```

```
      extends AbstractModule
      implements Aggrefier;
```

3.6 Spring instantiation

All Spring function is consolidated into a single class, which exposes methods for dynamically instantiating and initializing JHOVE2 reportables, including all dependencies, by class type and name.

```
package org.jhove2.core.config;
public class Configure {
   public static <R> R getReportable(Class<? extends Reportable> cl,
                                     String name);
}
```

The class argument specifies the class of, a superclass of, or an interface implemented by the named reportable. The use of a generic method obviates the need for a cast in the invoking context, for example:

```
FormatModule utf8 = Configure.getReportable(FormatModule.class,
                                      “UTF8Module”);
```

compiles and executes cleanly without an unchecked cast exception.

4 Distribution Package

The JHOVE2 project uses the Maven for its build management (maven.apache.org). The project structure is:

```
jhove2/
       doc/
           index.html
           ...
       jhove2.bat
       jhove2.sh
       lib/
           *.jar
       src/
           main/
                java/
                          org/
                                     jhove2/
                                             ...
                resources/
                          config/
                                     jhove2-config.xml
                          properties/
                                     dispatcher.properties
                                     displayer.properties
                                     unicode/
                                             c0control.properties
                                             c1control.properties
                                             codeblock.properties
       test/
            ...
```

The “`doc/`” directory contains the Javadoc documentation; the “`lib/`” directory contains all JHOVE2 jar files; the “`src/`” directory contains source code and configuration files; and the “`test/`” directory holds sample data.

The files “`jhove2.bat`” and “`jhove2.sh`” are DOS and Bourne shell invocation scripts, respectively. Both of these files *must* be modified during installation to define the “JHOVE2_HOME” directory and the command path for the java command, for example:

```
SET JHOVE2_HOME=”C:\Program Files\jhove2”
SET JAVA=”C:\Program Files\java\jre1.6.0_15\bin\java

JHOVE2_HOME=/usr/jhove2
JAVA=/usr/bin/java
```

4.1 Package namespaces

The JHOVE2 interfaces and classes exist within a structured package hierarchy.

```
package org.jhove2;
package org.jhove2.annotation;            // JHOVE2 annotations
package org.jhove2.app;                   // JHOVE2 application classes
package org.jhove2.core;                  // JHOVE2 core classes
package org.jhove2.core.config;           // JHOVE2 instantiation classes
package org.jhove2.core.io;               // JHOVE2 core I/O classes
package org.jhove2.core.source;           // JHOVE2 core source units
package org.jhove2.core.util;             // JHOVE2 core utility classes
package org.jhove2.module;                // JHOVE2 modules
package org.jhove2.module.assess;         // Assessment module
package org.jhove2.module.characterize;   // Characterization module
package org.jhove2.module.digest;         // Message digesting module
package org.jhove2.module.dispatch;       // Dispatching module
package org.jhove2.module.display;        // Display modules
package org.jhove2.module.format;         // Format modules
package org.jhove2.module.identify;       // Identification module
```

4.2 Configuration

JHOVE2 offers extensive opportunities for local configuration. Configuration information is defined in two forms:

- Spring XML configuration files
- Java properties files

```
jhove2/
       src/
           main/
                resources/
                          config/
                                     jhove2-config.xml
                          properties/
                                     dispatcher.properties
                                     displayer.properties
```

```
                                unicode/
                                        c0control.properties
                                        c1control.properties
                                        codeblock.properties
```

The "`jhove2-config.xml`" file contains the definitions of all JHOVE2 dynamically instantiated components in terms of Spring beans. The "`dispatcher.properties`" file contains the format identifier-to-format module mappings used by the dispatcher module. The "`displayer.properties`" file contains property display directives. The "`c0control.properties`" and "`c1control.properties`" define the Unicode C0 and C1 control characters. The "`codeblock.properties`" file defines the Unicode code blocks.

5 Adding a New Format Module

The general steps to add a new format module to the JHOVE2 framework include:

- Assign a unique format identifier in the JHOVE2 namespace.

  ```
  info:jhove2/format/format
  ```

- Define a singleton format object to be referenced by the module.
- Create the module
  - Determine the name and type of all properties reported by the module.
  - The module will implement the Module or FormatModule interfaces and any specific behavioral interfaces appropriate for the module's processing capability.
- Make the module available on the Java classpath
- Add the module to the Spring configuration file "`jhove2-config.xml`", defining a module bean, a format bean, and a format identifier bean.

  ```
  <bean name="FormatModule" class="package.class" scope="prototype">
  </bean>
  ```

- Add the module to the dispatcher map properties file "`dispatcher.properties`"

  ```
  info\:jhove2/format/format package.class
  ```