External Format Reader Guide

Author	Nick Battle
Date	27/12/23
Issue	0.1

0 Document Control

0.1 Table of Contents

0.2 References

[1] Wikipedia entry for The Vienna Development Method, http://en.wikipedia.org/wiki/Vienna_Development_Method

[2] Wikipedia entry for Specification Languages, http://en.wikipedia.org/wiki/Specification_language

[3] VDMJ, https://github.com/nickbattle/vdmj

0.3 Document History

Issue 0.1	27/12/23	First draft.

0.4 Copyright

1 Overview

This document describes how to write *External Format Readers* for VDMJ.

Section 1 gives an overview of the architecture into which external readers fit. Section 2 gives detailed information about how to implement readers. Section 3 gives some examples of what would be possible with external readers and how to achieve it.

1.1 VDMJ

VDMJ provides basic tool support for the VDM-SL, VDM++ and VDM-RT specification languages, written in Java [1][2][3]. It includes a parser, a type checker, an interpreter and debugger with coverage recording, a proof obligation generator, user definable annotations and a combinatorial test generator, as well as *JUnit* support for automatic testing.

1.2 External Format Readers

VDM source files are normally plain text files that use the local character encoding for your locale or operating system (e.g. UTF8 or Windows-1252).

But VDM sources can be embedded in another document format, like MS Word, LaTeX or Markdown. For example, a VDM model may be in the appendix of a paper. In this case, the VDM source can be delimited by markers in the document, and VDMJ can extract the VDM content from between the markers. This means that there is only one copy of the VDM specification to maintain, and the tools can work with it directly from the document that contains/describes it.

In some cases, the VDM extracted from a document is *generated* from a completely different format. For example, an XML document can be read and converted to VDM record values.

The adapters in VDMJ that read and extract VDM sources are called *External Format Readers*, and these are specialized to read from a particular document format.

1.3 Source Reader Architecture

Source reading happens at the lowest level of the VDMJ parser architecture and the Java classes are all located in the *com.fujitsu.vdmj.lex* package.

Lexical language tokens are read from a file using *LexTokenReader.* Instances of this class wrap a particular *File* and are passed a *Dialect* and a *Charset* from the environment. Successive calls to the *nextToken* method return lexical tokens from the file.

The VDM parser requires backtracking, so *LexTokenReader* extends an abstract *BacktrackInputReader* class. This provides methods to push and pop locations within the file, remembering a stack of positions (pushed) and returning to the latest position in the event that the parse needs to backtrack (pops). The *BacktrackInputReader* class also applies an *IfdefProcessor* to the source, so that lines within *#ifdef, #elseif, #else* and *#endif* are either included or suppressed, depending on the values of the Java properties that are tested.

Finally, *BacktrackInputReader* uses an *ExternalFormatReader* to read the raw data from the file. The default external format reader is *LatexStreamReader*, which is capable of reading plain text files (*.vdmsl etc.) or files that contain LaTeX markup (*.tex or *.latex). But other external format readers are available that process MS Word documents (*.doc and *.docx), ODF[1] documents (*.odt), AsciiDoc files (*.adoc) and Markdown files (*.md or *.markdown).

BacktrackInputReader has a *readerFactory* method which returns an appropriate external format reader for a given *File*. If there is no specialised format reader available, the factory returns an instance of the default *LatexStreamReader*, which can read plain text.

[1] The Open Document Format, as used by *LibreOffice* and other tools.

Note that in order to do backtracking and to apply *#ifdef* processing, the external format readers must return the entire content of the file in one go, rather than reading it in an incremental way.

2 Writing External Format Readers

Users can write new external format readers and use these to process documents that either contain embedded VDM or from which VDM can be generated. This involves two steps:

- The creation of a new class, that implements the *ExternalFormatReader* interface, and extracts VDM source.
- The association of the new external reader with particular filename extension(s), so that a new instance of the reader will be used when trying to parse those files.

2.1 Implementing the Interface

The *ExternalFormatReader* interface is very simple and contains a single method:

```
public interface ExternalFormatReader
{
	/**
	 * Return a character array with the extracted VDM,
	 * or null if none available.
	 */
	public char[] getText(File file, Charset charset) throws IOException;
}
```

The *getText* method is responsible for extracting the entire VDM source from the *file* passed, and returning that as a character array. The *charset* passed is the default character encoding for the loaded specification (i.e. the *-c* option to VDMJ), and may be used to influence the reading, though often document formats define their own character set and this argument can sometimes be ignored.

The method for extracting the VDM source depends on the document format, but the following points may help:

- Look at the external readers provided with VDMJ – search for which classes implement *ExternalFormatReader*. There is an abstract *TextStreamReader*, which ignores the document format and treats it as a sequence of text, looking for a marker string. Many complex formats can be processed this way. For example, the older MS *.doc format stores its text content separate from formatting information, so as long as you are reasonably sure that a marker string cannot occur within the formatting, you can process *.doc files this way.
- There is a more complex abstract *XMLStreamReader* which is used to read *.docx and *.odt files. The class can extract a named member from a ZIP file, and convert layout elements to newlines and tabs. Concrete subclasses then do similar processing to the *TextStreamReader*, extracting text between markers.
- Typically you will want to collect character data into something like a *StringBuilder* and finally return that builder's *toString().toCharArray()*.
- The VDMJ project includes a simple example called *PDFStreamReader*, which uses the *pdftotext* command (Linux) to convert a PDF file to plain text and then look for markers.

The original purpose of external format readers was to read VDM embedded in different document formats. But the same interface can be used to *generate* VDM source from formats that do not literally contain VDM, but which can be usefully represented in VDM for modelling purposes. Readers that do this are implemented using the same interface, but their *getText* method is really performing a data translation rather than an extraction.

Some examples of this usage are given in section 3 .

2.2 Linking with a Filename Suffix

Having written a new external format reader, VDMJ must be told to associate that reader class with a particular file suffix, so that when these files are parsed, the correct reader is used. This is done via a resource file on the classpath, though that can be overridden by a Java property. The resource file is “vdmj.readers” and the property name is “vdmj.parser.external_readers”. They contain lines like this:

```
.pdf = examples.parser.PDFStreamReader
```

The resource file contains a list of pairs, each matching a particular filename suffix with an external reader class. Each class must be on the classpath, and it must implement the *ExternalFormatReader* interface. The suffix is simply compared to the end of the filename (*filename.endsWith(suffix)*), so the dot is not mandatory, but this is the usual way to identify filename suffixes. The example above associates the PDF reader example with a .pdf suffix. Note that suffix matching is not case sensitive, so the example would also match files called *.PDF and *.Pdf etc.

If the overriding property is used instead of a resource file, it must consist of a CSV or semi-colon list of these pairs. But it is usually most convenient to package the external reader class and a resource file in the same jar.

Multiple filename suffixes can be associated with the same external reader class.

Associations are applied in the order given by the file or property, and can *override* the implicit associations for the external readers supplied with VDMJ (see Appendix A). So if you want to use (say) a new improved LaTeX reader, just associate your class with .tex and/or .latex.

3 Example Reader Functionality

Broadly, external format readers either extract VDM source from a particular document format, or generate VDM source from an external source that is associated with the file. But there are some subtle uses that may not be immediately obvious. This section mentions a few.

3.1 Document Processing Libraries

Usually, if a document format is not proprietary (and sometimes, even if it is) there will be libraries available for reading such files that allowing you to navigate their structure. In this case, you typically have full access to the style/formatting information in a document and it may not be necessary to use crude markers for delimiting text.

For example, it might be possible to extract all sections of a document that use a named “VDM Source” style.

3.2 Using Meta-Files

The external format reader interface requires a single Java *File* to identify the source document. But this document does not need to be the source from which the VDM source is extracted. For example, the file could contain meta-information about where to find the real file, and how that file is encoded.

VDMJ currently requires all of its source files to share the same character encoding, set by the *-c* command line option. But by using a meta-data external reader, it would be possible to include filename and encoding information in (say) a *.meta file, and use this to embed a Shift-JIS file in a project that is otherwise encoded in Windows-1252, for example.

3.3 Data Access and Conversion

As already mentioned, an external format reader can generate VDM rather than literally extracting it from a document. If we combine this capability with the meta-file point above, it would be possible to open (say) a *.jdbc file, which contains enough information to make a JDBC connection to a database and execute a SQL query defined in the same file, converting the resulting table of data into a VDM record values (the VDM types for which could be returned as well).

3.4 Encrypted Files

It is perhaps unlikely that people would want to work with encrypted files, but the same meta-data approach could easily be used, with the meta-data defining the encrypted file location and perhaps password values (or more likely, the means to obtain passwords from protected storage).

3.5 Unzipping Archives

A ZIP file is effectively an external format, and the embedded readers in VDMJ do already unzip *.docx and *.odt files to extract entries with a known name. The meta-file technique could be used to create an external reader to process an arbitrary member of the ZIP file.

This technique has been used to process *FMUs* (ZIP files of XML data). The external reader unzips the file, converts the XML files within to VDM-SL and returns that data. This allows VDM models to define formal rules about the XML, and directly open example FMUs to see whether they comply.

3.6 Chaining External Readers

Having opened a ZIP file or decrypted an archive or otherwise obtained access to a file, an external reader can instantiate a different external reader to process the data. The *readerFactory* method of *BacktrackInputReader* takes a *File* and returns an *ExternalFormatReader* whose *getText* method can be called to read that file.

A. VDMJ File Associations

The default file associations and external readers provided with VDMJ are as follows:

Suffix	Class	Markers
*.vdmsl, *.vdmpp, *.vdmrt	LatexStreamReader	Plain text (no markers)
*.tex, *.latex	LatexStreamReader	\begin{vdm_al} and \end{vdm_al}
*.doc	DocStreamReader	%%VDM%% before and after
*.docx	DocxStreamReader	%%VDM%% before and after
*.odt	ODFStreamReader	%%VDM%% before and after
*.md, *.markdown	MarkdownStreamReader	<!-- vdm --> before and after
*.adoc	AsciiDocStreamReader	// {vdm} before and after

New file associations (or overrides of these existing associations) are made using the *vdmj.readers* resource file (or *vdmj.parser.external_readers* property), which contains a list of pairs of file suffixes and class names (see 2.2). The classes must implement *ExternalFormatReader* and they must be on the classpath.