Te Puna Mātauranga o Aotearoa
NATIONAL LIBRARY
OF NEW ZEALAND

Metadata Extraction Tool Developers Guide

Version***: 3.5.***

Metadata Extraction Tool Developers Guide

Version: 3.5.

Table of Contents

✓ **Setting the Environment.. 3**

✓ **Development Overview.. 3**

Introduction.. 3

ParserListener.. 3

Data Adapters.. 3

Harvester Development 5

Built in Harvesters... 5

Harvester Methods... 6

✓ **Building the Source 8**

✓ **Sample Code 8**

Harvester Usage 8

Adapter Usage.. 8

✓ *Setting the Environment*

Before running the examples, make sure that your JAVA_HOME environment variable points to a valid Java installation.

The CLASSPATH environment variable must include:

1. All JAR files in the lib directory.
2. The root directory of the application (to load the configuration files).

✓ *Development Overview*

Introduction

This section reviews some of the key components of the Metadata Extraction Tool that the open source community may be interested in extending.

ParserListener

ParserListeners listen for events coming from ***DataAdapter***s. The standard harvester comes with a ***DTDXmlParserListener***, which listens for the events and outputs them as XML into a byte stream. This byte stream can be written directly to a file in the "native" format, or can be transformed via XSLT as in the NLNZHarvester implementation.

New ParserListener implementations can be built that handle the metadata events in different ways, such as writing it in a CSV styled format.

Data Adapters

DataAdapter implementations are responsible for extracting metadata from data files. Developers may choose to create new adapters to allow the Metadata Extractor to extract data out of new file types. Alternatively, a developer may extend an existing adapter to extract additional information, or handle multiple versions of a data file type.

The development of new adapters can follow the ***BitmapAdapter***, which has been fully documented. The BitmapAdapter uses the ***Element*** architecture to define the file structure and extract metadata. This is not a requirement of the adapters; it is simply the way that the bitmap adapter has been built. Many of the built-in adapters use the Element architecture, but some, such as the Microsoft Office/Works document adapters, rely on third party libraries to extract metadata.

When an adapter reads a piece of metadata or the start of a new block of metadata, it fires an event to the ***ParserContext*** registered for the harvest. The ParserContext relays that event to all associated ***ParserListeners***.

The ParserListener is responsible for capturing these events and turning it into some type of output.

The table below provides the details of some example classes.

Class File	Description
`nz.govt.natlib.adapter.bmp.BitmapAdapter`	Documents an Adapter using the Element architecture for reading the data files and firing output events.
`nz.govt.natlib.adapter.excel.ExcelAdapter`	Illustrates an alternative mechanism for reading metadata. This class relies on both the Jakarta POI library and the Element architecture to read metadata out of an Excel document.

Harvester Development

A Harvester is responsible for running a Harvest. It ties together the creation of a ParserListener, the location of an appropriate adapter, and any recursion required for harvesting complex Harvest Sources (files or directories of data files to harvest).

Developers may choose to build new harvesters if they wish to:

- Change the behaviour of a Harvest;
- Change the output format of a Harvest – such as generating a different XML output, or sending the metadata to a non-XML format.

Built in Harvesters

The following harvesters are built into the Metadata Extraction Tool:

Harvester	**Purpose**
DefaultHarvester	Abstract base class for the Harvesters. This class holds the logic for recursing through complex structures within a HarvestSource.
NLNZHarvester	Delegates to the SimpleObjectHarvester or ComplexObjectHarvester depending on the type of object being harvested. This Harvester and the SimpleObjectHarvester and ComplexObjectHarvester were developed to meet the specific needs of the *National Library of New Zealand Te Puna Mātauranga o Aotearoa (National Library).*
SimpleObjectHarvester	Harvests Simple objects, creating a separate output file for each file in the object. The name of the output file is the same as the name of the file being harvested, with an XML extension. If you harvest multiple files, you will get one output file per source file. This harvester programmatically writes a prologue to the output file, which contains information about the harvest, such as date/time of harvest, software and hardware environment, etc. The metadata is harvested from each file in the native format. Each file's metadata is then transformed with an XSLT transformation that creates a FILE element that can be put into the output file.

	Once all files are harvested, the SimpleObjectHarvester closes the tags opened in the prologue. Note that while this uses XSLT to transform native metadata into the National Library's *Preservation Metadata Data Dictionary (presmet)* format, the prologue and epilogue are coded into the harvester itself. This means that while the XSLT can be changed to alter the <FILE> element in the presmet format, it cannot be used to create a completely different output format, as the prologue will remain the same. Further note that the XSLT transformation may cause a significant amount of metadata to be lost, as not all information harvested from the file has a place in the presmet format.
ComplexObjectHarvester	The ComplexObjectHarvester extends the SimpleObjectHarvester to provide some additional functionality for handling Complex objects. The primary differences between this and the SimpleObjectHarvester are: • A single output file is created and contains the metadata for all of the files in the complex object within a nested XML structure; • The name of the output file is determined by the name of the complex object (not the name of the files being harvested); • For recursive harvests, the ComplexObjectHarvester records the relative paths.
POCHarvester	The POCHarvester extracts metadata from files in the native metadata extractor format. This format is proprietary, but holds all metadata extracted from the file.

Harvester Methods

The following table describes the primary methods of a Harvester, and how they are implemented in the built-in harvesters.

Method	**Description**	**Simple**	**Complex**	**POCHarvester**
startHarvest	Called when the Harvest is started.	Does nothing.	Creates an output file based on the name of the complex object being harvested. Writes the prologue for the NLNZ file format.	Does nothing.

openFolder	For recursive harvests, this is called every time we recurse into a folder.	Does nothing.	Writes an open *<FOLDER>* and *<PATH>foldername</PATH>* tag to the output file.	Does nothing.
closeFolder	For recursive harvests, this is called every time we recurse out of a folder.	Does nothing.	Writes a close *</FOLDER>* tag.	Does nothing.
startHarvestFile	Called when the Harvester begins harvesting a particular file.	Creates an output file, based on the name of the source file. Writes the prologue for the presmet file format.	Does nothing.	Does nothing.
harvestFile	Performs the actual harvest.	Locates the correct Adapter for the file being harvested. Creates an XML listener to capture the metadata events. Runs the adapter. Transforms the native output of the adapter into the presmet format, based on an XSLT transformation.	As for the Simple Harvester.	Locates the correct Adapter for the file being harvested. Opens an output file based on the name of the source file. Creates an XML listener to capture the metadata events. Runs the adapter. Uses the DoNothingTransformer to write the XML file to disk.
endHarvestFile	Called when the harvest of a single file is finished.	Closes the output file.	Does nothing.	Does nothing.
endHarvest	Called when the harvest is completed.	Does nothing.	Closes the output file.	Does nothing.

✓ *Building the Source*

The Metadata Extraction Tool is built from source using ANT. ANT can be downloaded from http://ant.apache.org/. The build file has been tested against version 1.6.1.

With ANT in the classpath, change into the root directory of the Metadata Extraction Tool and run ***ant***. The default target will clean the directories, compile the code, and produce the binary and source distributables.

To regenerate the JavaDocs, run ***ant javadoc***.

When ANT has finished, a binary distributable will be found in

BASE\metadata-bin-2-0.zip

Once a binary distributable version is built, you can install from binary as described in the Installation Guide.

✓ *Sample Code*

The nz.govt.natlib.samples package in the source code contains two samples of how the Metadata Extraction Tool can be embedded into another tool.

Harvester Usage

The Harvester Usage sample (nz.govt.natlib.samples.HarvesterUsage) illustrates how to call the Harvester and use one of the existing harvester configurations. The key code is wrapped in a try/catch block.

```
// Create a HarvestSource of the object we want to harvest.
FileHarvestSource source = new FileHarvestSource(file);

// Get the native Configuration.
Configuration c = Config.getInstance().getConfiguration(
                                        "Extract in Native form");

// Harvest the file. Note that the output is sent to a file as
// specified in the configuration. This class produces very little
// in the way of direct output.
c.getHarvester().harvest(c, source, new PropsManager());
```

Note that the name of the configuration must match a configuration in the config.xml file. The extracted metadata is saved to the directory configured in config.xml.

Adapter Usage

Instead of using the Harvester, you can choose to use an Adapter directly. The Adapter Usage sample (nz.govt.natlib.samples.AdapterUsage) shows how this can be done.

Using an Adapter directly allows additional control over how the metadata is harvested and how it is output. In the AdapterUsage sample, we have created our own

ParserListener to listen for metadata events. In this case, we are simply delegating to an XmlParserListener.

```
// Attempt to harvest the metadata.
try {
      // Make sure the Harvester System is initialised.
      Config.getInstance();

      // Get the appropriate adapter.
      DataAdapter adapter = AdapterFactory.getInstance().getAdapter(f);

      // Extract the metadata.
      adapter.adapt(f, ctx);

      // Display the metadata.
      System.out.println(listener.getContents());

} catch (IOException e) {
      // We failed to harvest the metadata, display the
      // exception.
      e.printStackTrace();
}
```