Adding Tools to the DAP and DTS, Overview and Examples

As the name "Brown Dog" suggests the project aims at bringing together a number of external tools as part of the two services being constructed. For the DAP, which handles conversions, these tools are incorporated as scripts to the Software Servers in Polyglot or as DFDL schemas for Daffodil. For the DTS, which handles the automatic extraction of metadata and content signatures, tools are incorporated as either extractors for Medici or extractors for Versus. Below we show examples for incorporating each of these components. This overview assumes a basic level of knowledge about the four main components of the Brown Dog software platform, i.e. Polyglot, Medici, Versus, and Daffodil. For a more in depth overview of each of these components and their function it is recommended that you first read through their online documentation and/or go through one of the online tutorial videos:

- Polyglot Documentation
- Medici Documentation
- Versus Documentation
- Daffodil and DFDL Documentation
- Tutorial Videos

The purpose of this document is to provide quick examples of each means of incorporating tools so as to bootstrap ones ability to include their code within one of the two services.

Start Here

To begin does your code, software, or tool carry out a data conversion or a data extraction? If a conversion the tool should be included in the Data Access Proxy. If an extraction the tool should be included in the Data Tilling Service.

The Data Access Proxy (DAP)

The Data Access Proxy handles data conversions. If a piece of software or tool exists to carry out the conversion its incorporation into the DAP will be through Polyglot. If the specification of the file format is known then in can be incorporated as a DFDL schema within Daffodil.

Polyglot Software Server Scripts

Software Server scripts are used by Polyglot to automate the interaction with software that is capable of converting from one file format to another. These scripts can directly wrap command line utilities that carry out conversions for use in Polyglot or split the steps of opening a file in one format and saving a file in a different format, typical of GUI driven applications. These wrapper scripts can be written in pretty much any text based scripting language. Below we show a few simple examples. Full details on the creation of these wrapper scripts, the required naming conventions, and required header conventions please refer to the the Scripting Manual.

Command Line Applications

Bash Script

The following is an example of a bash wrapper script for ImageMagick. Note that it is fairly straight forward. The comments at the top contain the information Polyglot needs to use the application: the name and version of the application, they type of data it supports, the input formats it supports, and the output formats it supports.

ImgMgk_convert.sh

```
#!/bin/sh
#ImageMagick (v6.5.2)
#image
#bmp, dib, eps, fig, gif, ico, jpg, jpeg, pdf, pgm, pict, pix, png, pnm, ppm, ps, rgb, rgba, sgi, sun, svg,
tga, tif, tiff, ttf, x, xbm, xcf, xpm, xwd, yuv
#bmp, dib, eps, gif, jpg, jpeg, pdf, pgm, pict, png, pnm, ppm, ps, rgb, rgba, sgi, sun, svg, tga, tif, tiff,
ttf, x, xbm, xpm, xwd, yuv

convert $1 $2
```

Batch File

Some GUI based applications are capable of being called in a headless mode. The following is an example wrapper script for OpenOffice called in its headless mode.

OpenOffice_convert.bat

```
REM OpenOffice (v3.1.0)
REM document
REM doc, odt, rtf, txt
REM doc, odt, pdf, rtf, txt

"C:\Program Files\OpenOffice.org 3\program\soffice.exe" -headless -norestore "-accept=socket`,host=localhost`,
port=8100;urp;StarOffice.ServiceManager"
"C:\Program Files\OpenOffice.org 3\program\python.exe" "C:\Converters\DocumentConverter.py" "%1%" "%2%"
```

GUI Applications

AutoHotKey

The following is an example of an AutoHotKey script to convert files with Adobe Acrobat, a GUI driven application. Note it contains a similar header in the comments at the beginning of the script. Also note that the open and save operation can be broken into two separate scripts.

Acrobat_open.ahk

```
;Adobe Acrobat (v9.3.0 Pro Extended)
;document
;pdf

;Parse input filename
arg1 = %1%
StringGetPos, index, arg1, \, R
ifLess, index, 0, ExitApp
index += 2
input_filename := SubStr(arg1, index)

;Run program if not already running
IfWinNotExist, Adobe 3D Reviewer
{
  Run, C:\Program Files\Adobe\Acrobat 9.0\Acrobat\Acrobat.exe
  WinWait, Adobe Acrobat Pro Extended
}

;Activate the window
WinActivate, Adobe Acrobat Pro Extended
WinWaitActive, Adobe Acrobat Pro Extended

;Open document
Send, ^o
WinWait, Open
ControlSetText, Edit1, %1%
ControlSend, Edit1, {Enter}

;Make sure model is loaded before exiting
Loop
{
  IfWinExist, %input_filename% - Adobe Acrobat Pro Extended
  {
    break
  }

  Sleep, 500
}
```

Acrobat_save.ahk

```
;Adobe Acrobat (v9.3.0 Pro Extended)
;document
;doc, html, jpg, pdf, ps, rtf, txt
```

```
;Parse output format
arg1 = %1%
StringGetPos, index, arg1, ., R
ifLess, index, 0, ExitApp
index += 2
out := SubStr(arg1, index)

;Parse filename root
StringGetPos, index, arg1, \, R
ifLess, index, 0, ExitApp
index += 2
name := SubStr(arg1, index)
StringGetPos, index, name, ., R
ifLess, index, 0, ExitApp
name := SubStr(name, 1, index)

;Activate the window
WinActivate, %name%.pdf - Adobe Acrobat Pro Extended
WinWaitActive, %name%.pdf - Adobe Acrobat Pro Extended

;Save document
Send, ^S
WinWait, Save As

if(out = "doc"){
  ControlSend, ComboBox3, m
}else if(out = "html"){
  controlSend, ComboBox3, h
}else if(out = "jpg"){
  controlSend, ComboBox3, j
}else if(out = "pdf"){
  controlSend, ComboBox3, a
}else if(out = "ps"){
  controlSend, ComboBox3, p
  controlSend, ComboBox3, p
  controlSend, ComboBox3, p
  controlSend, ComboBox3, p
  controlSend, ComboBox3, p
}else if(out = "rtf"){
  controlSend, ComboBox3, r
}else if(out = "txt"){
  controlSend, ComboBox3, t
  controlSend, ComboBox3, t
}

ControlSetText, Edit1, %1%
ControlSend, Edit1, {Enter}

;Return to main window before exiting
Loop
{
  ;Continue on if main window is active
  IfWinActive, %name%.pdf - Adobe Acrobat Pro Extended
  {
    break
  }

  ;Click "Yes" if asked to overwrite files
  IfWinExist, Save As
  {
    ControlGetText, tmp, Button1, Save As

    if(tmp = "&Yes")
    {
      ControlClick, Button1, Save As
    }
  }

  Sleep, 500
}
```

```
;Wait a lit bit more just in case
Sleep, 1000

;Close whatever document is currently open
Send, ^w

;Make sure it actually closed before exiting
Loop
{
  ;Continue on if main window is active
  IfWinActive, Adobe Acrobat Pro Extended
  {
    break
  }

  Sleep, 500
}
```

DFDL Schemas

The Data Format Description Language (DFDL) allows one to write an XML schema definition which defines how to automatically parse a file in that format into an XML representation of the data. DFDL provides an ideal means of preserving the many ad hoc formats created and used in labs. The DFDL schema below is a simple example that parses the data from a PGM image file.

pgm.dfdl.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Load image data from a PGM file and represent the data as a sequence of pixels in row major order.
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/" xmlns:ex="
http://example.com" targetNamespace="http://example.com">
  <xs:include schemaLocation="xsd/built-in-formats.xsd"/>

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="ex:daffodilTest1" separator="" initiator="" terminator="" leadingSkip='0' textTrimKind="
none" initiatedContent="no"
        alignment="implicit" alignmentUnits="bits" trailingSkip="0" ignoreCase="no" separatorPolicy="
suppressed"
        separatorPosition="infix" occursCountKind="parsed" emptyValueDelimiterPolicy="both" representation="
text"
        textNumberRep="standard" lengthKind="delimited" encoding="ASCII"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="file">
    <xs:complexType>
      <xs:sequence>

        <xs:element name="header" dfdl:lengthKind="implicit" maxOccurs="1">
          <xs:complexType>
            <xs:sequence dfdl:sequenceKind="ordered" dfdl:separator="%NL;" dfdl:separatorPosition="postfix">
              <xs:element name="type" type="xs:string"/>
              <xs:element name="dimensions" maxOccurs="1" dfdl:occursCountKind="implicit">
                <xs:complexType>
                  <xs:sequence dfdl:sequenceKind="ordered" dfdl:separator="%SP;">
                    <xs:element name="width" type="xs:integer"/>
                    <xs:element name="height" type="xs:integer"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="depth" type="xs:integer"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="pixels" dfdl:lengthKind="implicit" maxOccurs="1">
          <xs:complexType>
            <xs:sequence dfdl:separator="%SP; %NL; %SP;%NL;" dfdl:separatorPosition="postfix" dfdl:
separatorSuppressionPolicy="anyEmpty">
              <xs:element name="pixel" type="xs:integer" maxOccurs="unbounded" dfdl:occursCountKind="expression"
                dfdl:occursCount="{../../ex:header/ex:dimensions/ex:width * ../../ex:header/ex:dimensions/ex:
height }"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

An example of the XML produced the above DFDL schema when applied to a PGM file is shown below.

```
<ex:file xmlns:ex="http://example.com">
  <ex:header>
    <ex:type>P2</ex:type>
    <ex:dimensions>
      <ex:width>16</ex:width>
      <ex:height>16</ex:height>
    </ex:dimensions>
    <ex:depth>255</ex:depth>
  </ex:header>
  <ex:pixels>
    <ex:pixel>136</ex:pixel>
    <ex:pixel>136</ex:pixel>
    <ex:pixel>136</ex:pixel>
    ...
    <ex:pixel>136</ex:pixel>
    <ex:pixel>136</ex:pixel>
  </ex:pixels>
</ex:file>
```

The Data Tilling Service (DTS)

The Data Tilling Services handles data extractions. If your code, tool, or software extracts information such as keywords from a file or its contents then it should be included in the DTS as a Medici extractor. If your code, tool, or software extracts a signature from the file's contents which in turn can be compared to the signatures of other files via some distance measure to find similar pieces of data, then, it should be included in the DTS as a Versus extractor.

Medici Extractors

Medici extractors typically serve to automatically extract some new kind of information from a file's content when it is uploaded into Medici. These extractors do this by connecting to a shared RabbitMQ bus. When a new file is uploaded to Medici it is announced on this bus. Extractors that can handle a file of the type posted on the bus are triggered and the data they in turn create is returned to Medici as derived data to be associated with that file. The extractors themselves can be implemented in a variety of languages. Examples of these extractors in different languages can be found in the extractors-templates code repository.

Java

Java extractors can be created using the amqp-client jar file. This allows you to connect to the RabitMQ bus and received messages. The easiest way to get up and running is to use maven with java to add all required dependencies. An example of an extractor written in java can be found at Medici Extractor in Java.

Python

Python extractors will often be based on the packages pika and requests. This allows you to connect to the RabittMQ message bus and easily send requests to medici. A complete example of the python extractor can be found at Medici Extractor in Python

Calling R Scripts from Python

Coming soon**...**

Versus Extractors

Versus extractors serve to extract a signature from a file's content. These signatures, effectively a hash for the data, are typically numerical vectors which capture some semantically meaningful aspect of the content so that two such signatures can then be compared using some distance measure. Within Versus extractors operate on a data structure representing the content of a file, produced a Versus adapter, and the returned signatures compared by either a Versus similarity or distance measure. The combination of these adapters, extractors, and measures in turn compose a comparison which can be used for relating files according their contents.

Java

The main class sets up the comparison, this is done by adding the two files that need to be compared, as well as the adapter to load the file, the extractor to extract a feature from the file, and a measurement to compare the two features.

Main

```
    static public void main(String[] args) {
        PairwiseComparison comparison = new PairwiseComparison();
        comparison.setId(UUID.randomUUID().toString());
        comparison.setFirstDataset(new File("data/test1.txt"));
        comparison.setSecondDataset(new File("data/test2.txt"));
        comparison.setAdapterId(TextAdapter.class.getName());
        comparison.setExtractorId(TextHistogramExtractor.class.getName());
        comparison.setMeasureId(LabelHistogramEuclidianDistanceMeasure.class.getName());

        ExecutionEngine ee = new ExecutionEngine();
        ee.submit(comparison, new ComparisonStatusHandler() {
            @Override
            public void onStarted() {
                System.out.println("STARTED : ");
            }

            @Override
            public void onFailed(String msg, Throwable e) {
                System.out.println("FAILED  : " + msg);
                e.printStackTrace();
                System.exit(0);
            }

            @Override
            public void onDone(double value) {
                System.out.println("DONE    : " + value);
                System.exit(0);
            }

            @Override
            public void onAborted(String msg) {
                System.out.println("ABORTED : " + msg);
                System.exit(0);
            }
        });
    }
```

The text adapter will take a text file, and load all the file, splitting the text into words and return a list of all words in the text. The words are still in the right order, and it is possible to read the original information of the file by reading the words in the order as they are returned by getWords().

Text Adapter

```
public class TextAdapter implements FileLoader, HasText {
    private File         file;
    private List<String> words;

    public TextAdapter() {}

    // ------------------------------------------------------------------------
    // FileLoader
    // ------------------------------------------------------------------------
    @Override
    public void load(File file) {
        this.file = file;
    }

    @Override
    public String getName() {
        return "Text Document";
    }

    @Override
    public List<String> getSupportedMediaTypes() {
        List<String> mediaTypes = new ArrayList<String>();
        mediaTypes.add("text/*");
        return mediaTypes;
    }

    // ------------------------------------------------------------------------
    // HasText
    // ------------------------------------------------------------------------
    @Override
    public List<String> getWords() {
        if (words == null) {
            words = new ArrayList<String>();
            try {
                BufferedReader br = new BufferedReader(new FileReader(file));
                String line;
                while((line = br.readLine()) != null) {
                    String[] w = line.split(" ");
                    words.addAll(Arrays.asList(w));
                }
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return words;
    }
 }
```

The extractor will take the words returned by the adapter and count the occurrence of each word. At this point we are left with a histogram with all words and how often they occur in the text, we can no longer read the text since the information about the order of the words is lost.

Text Histogram Extractor

```
public class TextHistogramExtractor implements Extractor
{
    @Override
    public Adapter newAdapter() {
        throw (new RuntimeException("Not supported."));
    }

    @Override
    public String getName() {
        return "Text Histogram Extractor";
    }

    @Override
    public Set<Class<? extends Adapter>> supportedAdapters() {
        Set<Class<? extends Adapter>> adapters = new HashSet<Class<? extends Adapter>>();
        adapters.add(HasText.class);
        return adapters;
    }

    @Override
    public Class<? extends Descriptor> getFeatureType() {
        return LabelHistogramDescriptor.class;
    }

    @Override
    public Descriptor extract(Adapter adapter) throws Exception {
        if (adapter instanceof HasText) {
            LabelHistogramDescriptor desc = new LabelHistogramDescriptor();

            for (String word : ((HasText) adapter).getWords()) {
                desc.increaseBin(word);
            }

            return desc;
        } else {
            throw new UnsupportedTypeException();
        }
    }

    @Override
    public boolean hasPreview(){
        return false;
    }

    @Override
    public String previewName(){
        return null;
    }
}
```

To compare two texts we use the euclidian distance measure of two histograms. First we normalize each histogram, so we can compare a large text with a small text, next we compare each big of the two histograms. If the bin is missing from either histogram it is assumed to have a value of 0.

Euclidian Distance Measure

```
public class LabelHistogramEuclidianDistanceMeasure implements Measure
{
    @Override
    public SimilarityPercentage normalize(Similarity similarity) {
        return new SimilarityPercentage(1 - similarity.getValue());
    }

    @Override
    public String getFeatureType() {
        return LabelHistogramDescriptor.class.getName();
    }

    @Override
    public String getName() {
        return "Histogram Distance";
    }

    @Override
    public Class<LabelHistogramEuclidianDistanceMeasure> getType() {
        return LabelHistogramEuclidianDistanceMeasure.class;
    }

    // correlation

    @Override
    public Similarity compare(Descriptor desc1, Descriptor desc2) throws Exception {
        if ((desc1 instanceof LabelHistogramDescriptor) && (desc2 instanceof LabelHistogramDescriptor)) {
            LabelHistogramDescriptor lhd1 = (LabelHistogramDescriptor) desc1;
            LabelHistogramDescriptor lhd2 = (LabelHistogramDescriptor) desc2;

            // get all possible labels
            Set<String> labels = new HashSet<String>();
            labels.addAll(lhd1.getLabels());
            labels.addAll(lhd2.getLabels());

            // normalize
            lhd1.normalize();
            lhd2.normalize();

            // compute distance
            double sum = 0;

            for (String s : labels) {
                Double b1 = lhd1.getBin(s);
                Double b2 = lhd2.getBin(s);

                if (b1 == null) {
                    sum += b2 * b2;
                } else if (b2 == null) {
                    sum += b1 * b1;
                } else {
                    sum += (b1 - b2) * (b1 - b2);
                }
            }

            return new SimilarityNumber(Math.sqrt(sum), 0, 1, 0);
        } else {
            throw new UnsupportedTypeException();
        }
    }
}
```