Laboratory of Neuro Imaging Debabeler, version 2.0.2

Concepts: Scott C. Neu, Daniel J. Valentino, and Arthur W. Toga
Design: Scott C. Neu
Implementation: Scott C. Neu, Keith R. Ouellette, Bryan Vann, and Craig Schwartz
Documentation: Scott C. Neu

TABLE OF CONTENTS

BACKGROUND

A NOTE FROM THE AUTHOR

The concept for the Debabeler was inspired from my experiences while I was working for the UCLA hospital radiology department. It was my job to write source code for a PACS workstation that was used by the radiology staff to view medical images (MR, CT, CR) at various sites in the hospital. The images were acquired at scanners located at UCLA and Santa Monica and sent to the workstations as well as to a local PACS for long-term storage. The workstations cached images acquired within the last couple of days and allowed users to retrieve older studies from the PACS when needed.

The main screen of the workstation GUI displayed a list of the patients having images cached locally, and when a patient was selected, the GUI displayed a table of the image studies and series associated with that patient. It surprised me to see the tables were not as homogeneous as I had expected. Many of the values in the same table column (such as "series description") gave different ways of saying the same thing (like "AX T1 GD" and "AXIAL T1 POST GADO") and worse, some values were empty. Also, I was informed by the radiologists that I had mixed scout images with the axial slices they were viewing and this disrupted their readings on the workstation. Complaints also came in about PD images being mixed with T2 images for some image series.

Well, I thought I had followed the rules. I had downloaded the voluminous DICOM specs and painfully read through the standard. The DICOM tag for the Series Instance UID (0020, 000E) gives a string that uniquely identifies the series in the world. So I grouped all the DICOM images sent to each workstation by their Series Instance UID and displayed images in that group together. The DICOM tag for the Series Description (0008, 103E) gives a string that describes the image series. So I labeled each group of images with that string.

What went wrong? After getting samples of the workstation images and comparing their metadata, I found that I had indeed done everything correctly. So I determined who was sending me the problematic DICOM images and went to the scanners to find out more. I went up to one technician and said, "Would you mind not typing in 'AX T1 GD'? It looks better on the workstation if you type 'AXIAL T1 POST GADO'." "Well, " the technician replied, "this is an old scanner and it only allows me to type at most 10 characters." I didn't quite have an answer for that one, so I went on to a Siemens MR scanner and said, "Why are you are assigning the same unique series id to both the scouts and the axial slices? I can't distinguish the scout images from the slices." The technician there answered, "The scanner assigns those ids, I don't have any control over that." Disheartened, I went to a GE MR scanner and said, "Why are you grouping the proton density and T2 weighted images into the same series? The radiologists want to view them as two series." The technician there argued, "but the images are acquired during the same time, so they belong to the same series." At that point I realized I was taking the wrong approach.

I was on the receiving end of a lot of different metadata variations. For various reasons (not under my control) people were acquiring images in different situations for different purposes. I had to put them all together into some kind of uniform presentation. But I couldn't. The more metadata values I depended upon resulted in more "if-then" statements needed to handle all the variations I was encountering. So either I limited the functionality of the workstation, or I wrote more "if-then" statements into my source code. I worried about the workstation receiving images from a new scanner or an upgraded one -- what new variations would lead to bad workstation behavior?

But I just wanted to concentrate on building workstation features--why should I also have to be a data manager? And was I rediscovering exceptions known already to more experienced workstation programmers? What if I want to handle other formats besides DICOM? Why can't we take the data management layer out of my application so I don't have to handle all these unreliable fields?

These experiences inspired the Debabeler, a flexible tool for visually programming translations that convert image files between the producers of data (e.g., scanners) and the consumers of the data (e.g., viewing applications).

DEBABELER PHILOSOPHY

When most software designers sit down to design a medical informatics system, they typically start by defining a model into which they plan to organize their data. This design evolves into software components that are structured according to the data model. The source code for these systems have "hard-coded data models." Changing or modifying the data model requires rewriting the source code, which can be a time-consuming process. But having a structural data model is essential for good performance and non-complex source code.

However, the data produced by scanners, written by computer executables, or created by hand intrinsically contains the methodology and "view" the producers used when acquiring the data. The choices of what data to store and what data values describe the data are all dependent upon what was just done. There is an inherent data model in each image file and implicit meanings attached to data values that are not captured by the file format alone.

File formats commonly used in neurology (ANALYZE, MINC, DICOM) encode data into a file, but they do not always define what the data means. In many cases, the data may have a meaning subject to who is using it. What is an image series? It is a group of images. But does it contain scout images? It depends upon what you are doing with it. It is part of the Debabeler philosophy to separate out the subjective data attributes from the non-subjective. For example, the width and height of the image encoded in a file must be correct in order to decode the image (non-subjective) whereas the format for the patient name or a series id (do not affect the image decoding) tend to vary depending upon who has defined them (subjective).

So having a set of DICOM image files does not completely define the data. It's more relevant to ask: what KIND of DICOM? Siemens? From the group at UCLA? It's perhaps most pertinent to ask: who did you get this from? And for what purpose did they acquire this?

In a nutshell, Debabeler philosophy says that you can't represent all the data you get with one data model. Everyone has there own way of doing things and many have different goals and different reasons for modeling their data using the definitions they do. If you try to construct a "universal" data model into which you plan to organize everyone's data you will ultimately fail. With each new exception you encounter that doesn't fit, you can either add a new case to your universal data model (increasing its complexity) or generalize one or more of its definitions (weakening its usefulness). Eventually, after encountering many new exceptions, the universal data model becomes too complex to use or becomes to general to rely upon.

How do you get any software application to work with any image file? You translate the image file from its implicit data model into the data model assumed by the application. You make sure the data read by the application is in a form the application can understand. If data is placed in the wrong field, you move it to the right field. And if necessary, you combine the data with user input to fill in the gaps. There isn't a "correct" data model that everyone should be following. Everyone's doing things in a way that best suits their needs.

The Debabeler defines the term "debabel" as "identify, group, and translate." To identify an image file is to determine the data model used to create it. This may involve parsing through the data to find key data like the scanner name or the name of the institution or group that created the file. It may also depend upon the file format or recognizing a special format of the file name. Grouping a set of image files refers to associating a string id to each file that is used to regroup the file set into different sets. This allows you to separate PD and T2 images by echo time or remove scouts by finding the key word "SCOUT" in the series description. And finally, translating means to read the data stored in an image file and write it to another file, which may possibly be in a different file format.

GETTING STARTED

UNCOMPRESSING THE DOWNLOADED FILE

From the LONI software download web site, you should be able to download the zip file:

release_2_0_2_alpha.zip (1585764 bytes)

Uncompress the file into its contents (e.g., jar xvf release_2_0_2_alpha.zip). This should create the directory "release_2_0_2_alpha" containing the following files:

File	Description
debabeler_16October2003.jar	The main Debabeler jar file
lib/keithsProcessors_04August2003.jar	Plugin of Debabeler processors contributed by Keith Ouellette
lib/scottsProcessors_29September2003.jar	Plugin of Debabeler processors contributed by Scott Neu
lib/stdlib_proc_24June2003.jar	Plugin of Debabeler processors also contributed by Keith Ouellette
lib/vannsProcessors_28July2003.jar	Plugin of Debabeler processors contributed by Bryan Vann
plugins/afniPlugin_08October2003.jar	Java Image I/O plugin for writing AFNI files
plugins/analyzePlugin_08October2003.jar	Java Image I/O plugin for reading and writing ANALYZE files
plugins/dicomPlugin_08October2003.jar	Java Image I/O plugin for reading and writing DICOM files
plugins/gePlugin_08October2003.jar	Java Image I/O plugin for reading and writing GE files
plugins/medxPlugin_08October2003.jar	Java Image I/O plugin for reading MedX bitmap files Contour I/O plugin for reading MedX contour files
plugins/mincPlugin_08October2003.jar	Java Image I/O plugin for reading and writing MINC files
plugins/rawPlugin_08October2003.jar	Java Image I/O plugin for reading "raw" image files

plugins/sqlPlugin_08October2003.jar	Text I/O plugin for writing SQL text files
plugins/ucfPlugin_08October2003.jar	Contour I/O plugin for reading and writing LONI UCF contour files
plugins/xmlPlugin_08October2003.jar	Text I/O plugin for reading and writing XML text files
run.bat	Use to run the Debabeler in Windows
run.sh	Use to run the Debabeler in UNIX
xml/	Contains various Debabeler translations
xml/contrib	Contains Debabeler translations contributed by collaborators
xml/sources	Contains source trees used to build translations from scratch

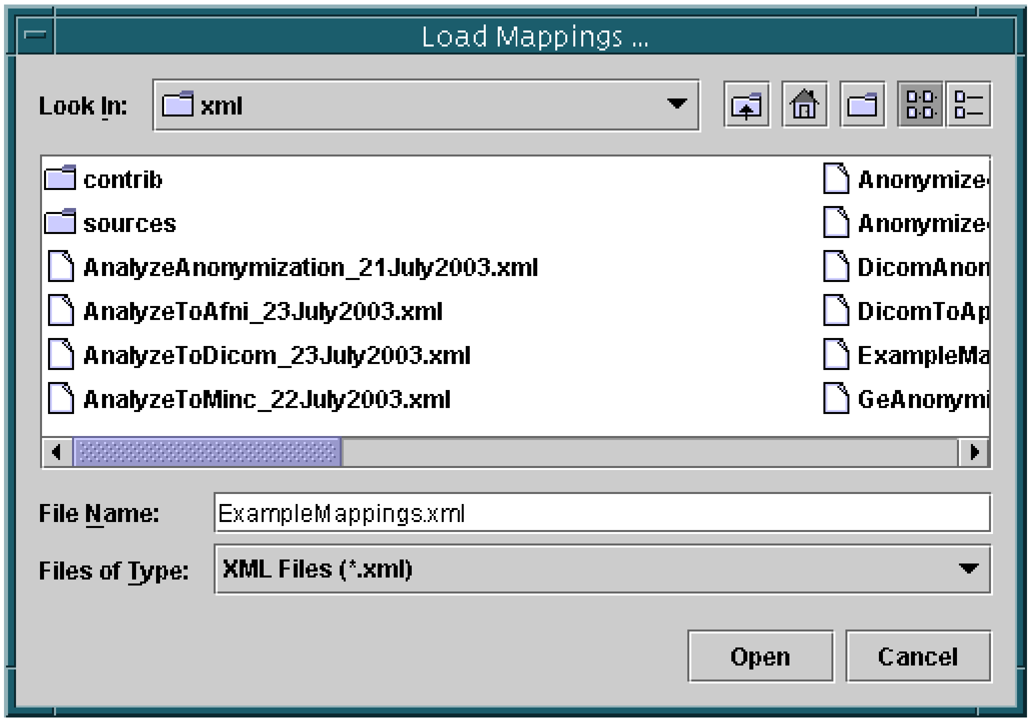
The Debabeler supports two types of plugins: processor libraries and decoders/encoders. Debabeler processors are created and executed inside of Debabeler translations. They perform operations like string concatenation and image sorting. Users may write their own processors and use them in the Debabeler; go here for more information on how to add your own processor library. Decoders and encoders read and write images, contours, and text from and to files. For image decoding/encoding, the Java Image I/O plugin architecture is utilized.

STARTING THE DEBABELER

In order to run the Debabeler, you must have Java 1.4 or later installed on your machine. The Debabeler has been tested and successfully run on Windows, Solaris, Linux, IRIX, and OS X platforms. On Windows, double click the run.bat file. On the other platforms, open a command line shell and execute the run.sh script (it may be necessary to "chmod a+x run.sh" first if it is not given executable permissions).

LOADING A MAPPING

When the Debabeler is first started, the main Debabeler window appears as well as a smaller dialog box.



The list of XML files seen in this dialog include the supplied mappings as well as a "demo" mapping used for tutorial purposes. If you would like the Debabeler to execute a more robust translation than those in the "ExampleMappings.xml" file, go here. Otherwise, click the "Open" button to load the demo mappings.

In the Debabeler console you should see the following output:

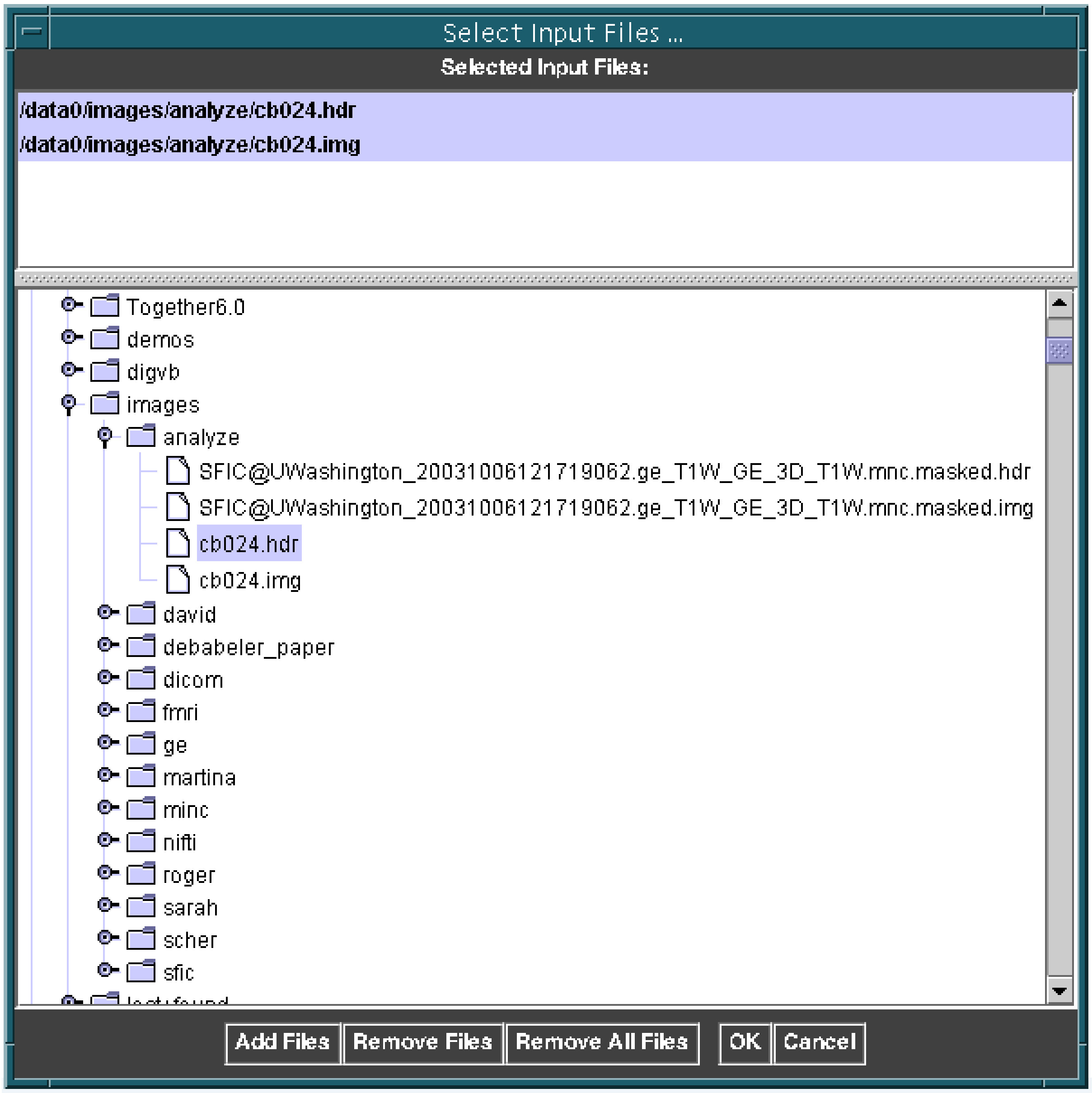
```
LONI Debabeler 2.0.2 alpha
Registered the mapping "Source Identification".
Registered the mapping "ANALYZE Group Assignment".
Registered the mapping "DICOM Group Assignment".
Registered the mapping "MINC Group Assignment".
Registered the mapping "MedX Group Assignment".
Registered the mapping "UCF Group Assignment".
Registered the mapping "ANALYZE to MINC Translation".
Registered the mapping "DICOM to ANALYZE Translation".
Registered the mapping "MINC to ANALYZE Translation".
Registered the mapping "MedX/UCF to ANALYZE Translation".
Registered the mapping "DICOM to XML Translation".
Registered the mapping "DICOM to SQL Translation".
```

There are 3 types of Debabeler mappings. The first is called "Source Identification" and it is run by the Debabeler on each input file to determine the name of the source that produced the file. There is always and only one such mapping. If you load another XML mapping file into the Debabeler the "Source Identification" in this file will replace the currently loaded "Source Identification." So be careful! The second type of Debabeler mapping is called a "Group Assignment." After the Debabeler has identified the source of an input file, it uses the source name to find a "Group Assignment" for the file. In this case, there is a "Group

Assignment" for ANALYZE, DICOM, MINC, MedX, and UCF input files. The appropriate "Group Assignment" is run by the Debabeler to produce a string identifier for each input file. After all the "Group Assignments" have been run, the Debabeler organizes the input files into groups, with each group having its own string identifier. Lastly, the third type of Debabeler mapping is called a "Translation." There are 6 "Translations" loaded into the Debabeler. Using the source names and multiplicity (how many files of each source type) in each group, the Debabeler determines the appropriate "Translation" to run on each group of input files. This produces a set of output files written in the requested file format.

CHOOSING THE INPUT FILES

You can select the files to use as input to the Debabeler by navigating through the menus: File->Select Input Files ... This will display a dialog box similar to:



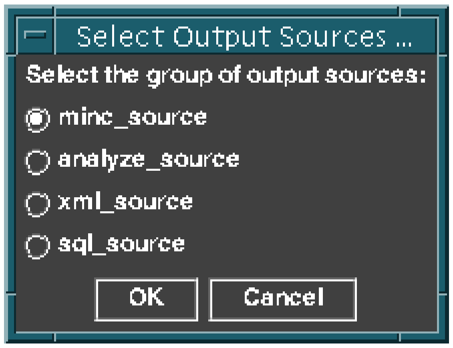
/data0/images/analyze/cb024.hdr
/data0/images/analyze/cb024.img

You can select an input file by double-clicking the file name in the directory tree, or by single-clicking the file name and hitting the "Add Files" button. Single-clicking a directory and selecting the "Add Files" button will select all the files in that directory (but not select any files in its subdirectories). The selected input file names are displayed in the top box labeled "Selected Input Files:." In the above picture, an ANALYZE file pair has been selected. Once you have selected the input files, hit the "OK" button. The number of input files selected will appear in the lower left corner of the main Debabeler window.

You can select one or more input files of the same type (e.g., DICOM) or select input files of different types (e.g., ANALYZE, MINC, DICOM). The Debabeler is designed to identify all the input files and process them as groups which you can define. If you select an input file that the Debabeler does not recognize (can not decode using the I/O plugins), it will not be translated.

CHOOSING THE OUTPUT TYPE

The output type can be selected by navigating through the menus: File->Select Output Sources... This will display the dialog box:

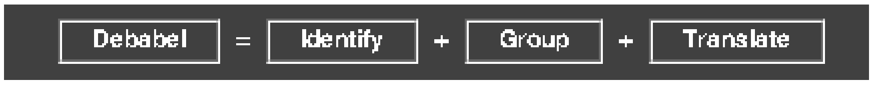

Select Output Sources ...
Select the group of output sources:
minc_source
analyze_source
xml_source
sql_source
OK
Cancel

Although the dialog box suggests that the output files can be either MINC, ANALYZE, XML, or SQL, there must be a "Translation" loaded into the Debabeler that matches the input type (chosen to be an ANALYZE file pair above) and the selected output type. For example, in the list of registered "Translations" above, there is an ANALYZE to MINC translation but not a DICOM to MINC translation. This means DICOM files cannot be converted to MINC files using only the above loaded translations. However, there do exist translations to convert DICOM and MINC images to ANALYZE file pairs.

You have to select the output type because the Debabeler does not identify the input files before it debabels. Select the "minc_source" toggle button and hit "OK" to specify the output as MINC.

DEBABELING THE INPUT FILES

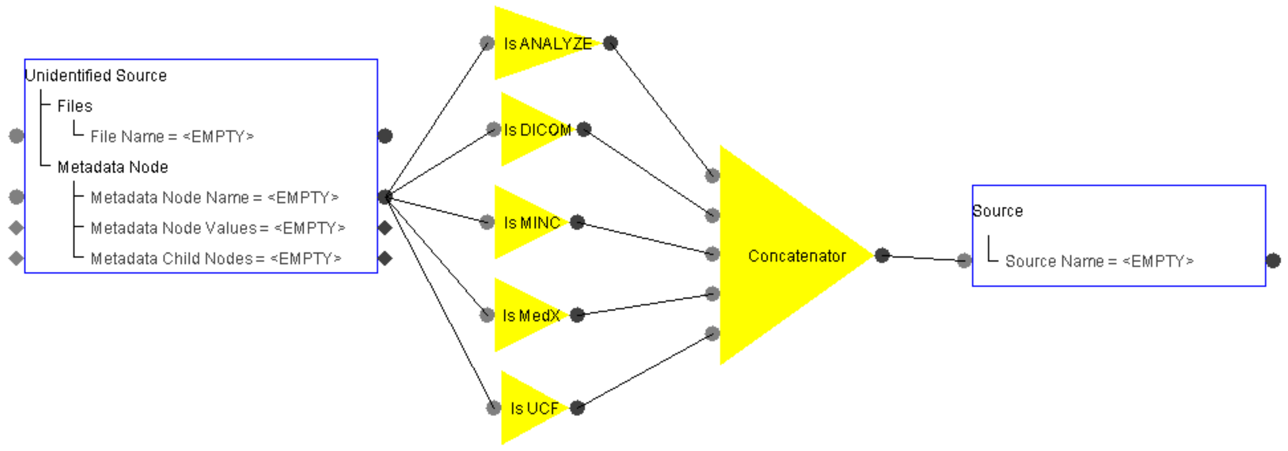
The following steps (identify, group, translate) can all be accomplished at once by clicking the "Debabel" button:



However, to illustrate how the Debabeler works, each step will be described in detail.

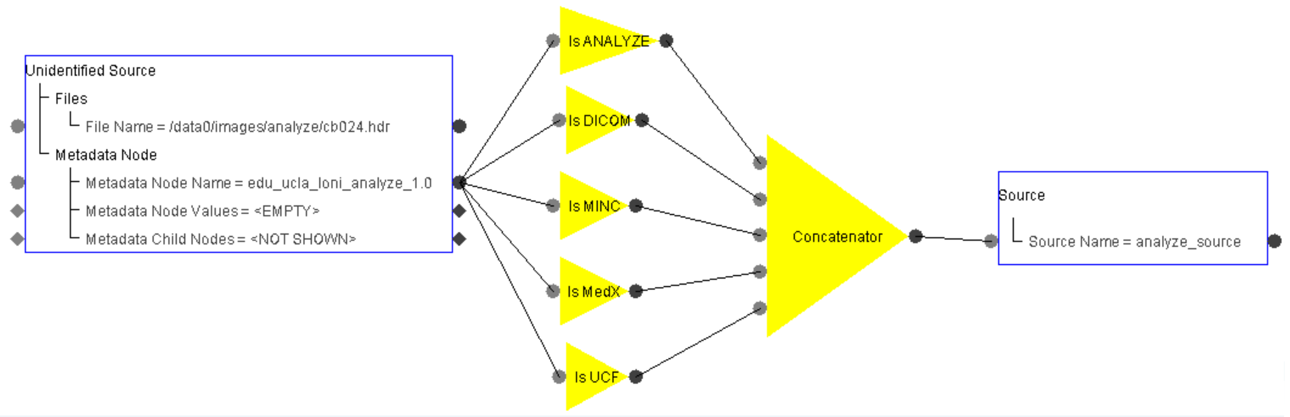
Identify the input files

To view the "Source Identification" mapping, navigate through the menus: Views->Source Identification. The processor panel of the Debabeler will display:



The yellow triangles represent Debabeler processors, and the blue-outlined rectangles are the source and target trees. The source tree (leftmost rectangle) displays information obtained from each input file. This information is sent to the processors via their connections, and the output of the processors is sent to the target tree (rightmost rectangle). The target tree has only one input node (circle that accepts input values from connections) which is used to hold the name of the source that produced the input file.

After hitting the "Identify" button at the bottom of the Debabeler, the processor panel displays:


Unidentified Source
Files
File Name = /data0/images/analyze/cb024.hdr
Metadata Node
Metadata Node Name = edu_ucla_loni_analyze_1.0
Metadata Node Values = <EMPTY>
Metadata Child Nodes = <NOT SHOWN>
Is ANALYZE
Is DICOM
Is MINC
Is MedX
Is UCF
Concatenator
Source
Source Name = analyze_source

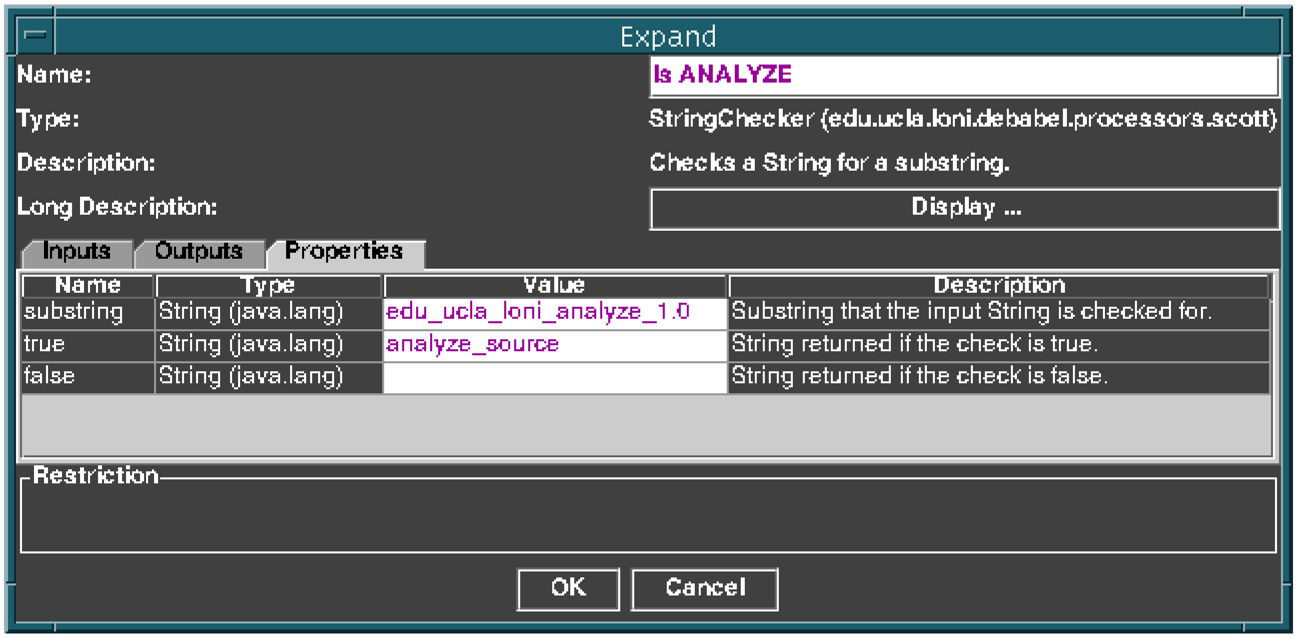
As shown in the source tree of the processor panel, there are 3 ways to obtain information about the source that produced an input file. The first is from the file name itself, the second is from the name of the Debabeler decoder that discovered how to decode the file, and the third is from a general tree of metadata read from the file. In the above picture, only the name of the Debabeler decoder (edu_ucla_loni_analyze_1.0) is used to determine the source names.

The Debabeler console shows the following output:

BEGIN: Determine the names of the sources that produced the input files.
Identified the source "analyze_source" for the files:
/data0/images/analyze/cb024.hdr
/data0/images/analyze/cb024.img
Number of successfully identified sources = 1.
END: Determine the names of the sources that produced the input files.

which reports that the Debabeler was successful in identifying the source name of the ANALYZE file pair (cb024.hdr, cb024.img) as "analyze_source."

The logic involved in producing the source name can be seen by double-clicking the "Is ANALYZE" processor. In the dialog box that appears, clicking the "Properties" tab pane shows:



Name:	Is ANALYZE
Type:	StringChecker (edu.ucla.loni.debabel.processors.scott)
Description:	Checks a String for a substring.
Long Description:	Display ...

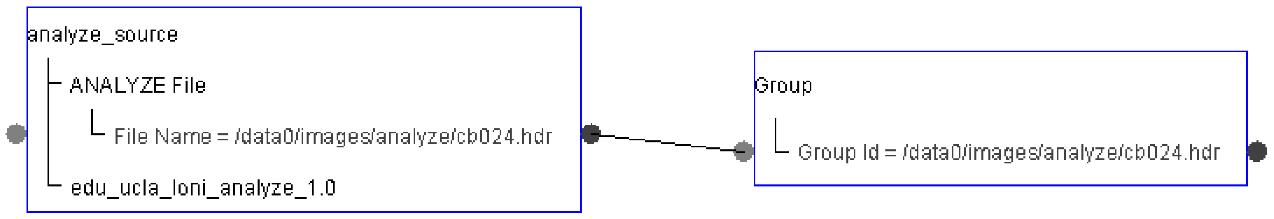
Name	Type	Value	Description
substring	String (java.lang)	edu_ucla_loni_analyze_1.0	Substring that the input String is checked for.
true	String (java.lang)	analyze_source	String returned if the check is true.
false	String (java.lang)		String returned if the check is false.

What this processor does is check its input string for an occurrence of the substring "edu_ucla_loni_analyze_1.0." If the substring is found, it returns the value "analyze_source." Otherwise, it returns the empty string "". Therefore, when the name of the Debabeler decoder is passed into the 5 StringChecker processors, only 1 (if any) will return the source name associated with its decoder name while the other processors will return the empty string. The large processor that receives all these output strings concatenates them together, essentially passing the identified source name into the target tree. After all the input files are processed through the "Source Identification," the Debabeler has associated a source name with each input file.

Why not just program the Debabeler to automatically identify input files by their file format? Because (see Debabeler Philosophy) in practice you can't identify every input file by just it's file format. You may have a DICOM file, but you don't know where it came from based solely on knowing its a DICOM file. Practically, there are *types* of DICOM and ANALYZE and MINC in which certain metadata fields are used differently for different purposes. Sometimes you need to further distinguish the source type from the file name or by looking for a key word in the metadata associated with the file.

Group the input files

With the ANALYZE input file pair being identified as "analyze_source" above, the Debabeler searches through the registered "Group Assignments" until it finds one whose source tree has the name "analyze_source." The "Group Assignment" it finds can be shown in the processor panel by navigating through the menus: Views->ANALYZE Group Assignment. After clicking the "Group" button near the bottom of the Debabeler, the processor panel shows:


analyze_source
ANALYZE File
File Name = /data0/images/analyze/cb024.hdr
edu_ucla_loni_analyze_1.0
Group
Group Id = /data0/images/analyze/cb024.hdr

The method for assigning group ids to ANALYZE files is simple here: make the group id the file name of the ANALYZE file that contains the header information. Assigning the group id in this way means that each ANALYZE file pair will be assigned to its own group. Therefore, if you choose multiple ANALYZE files as input, each ANALYZE file pair will be translated separately.

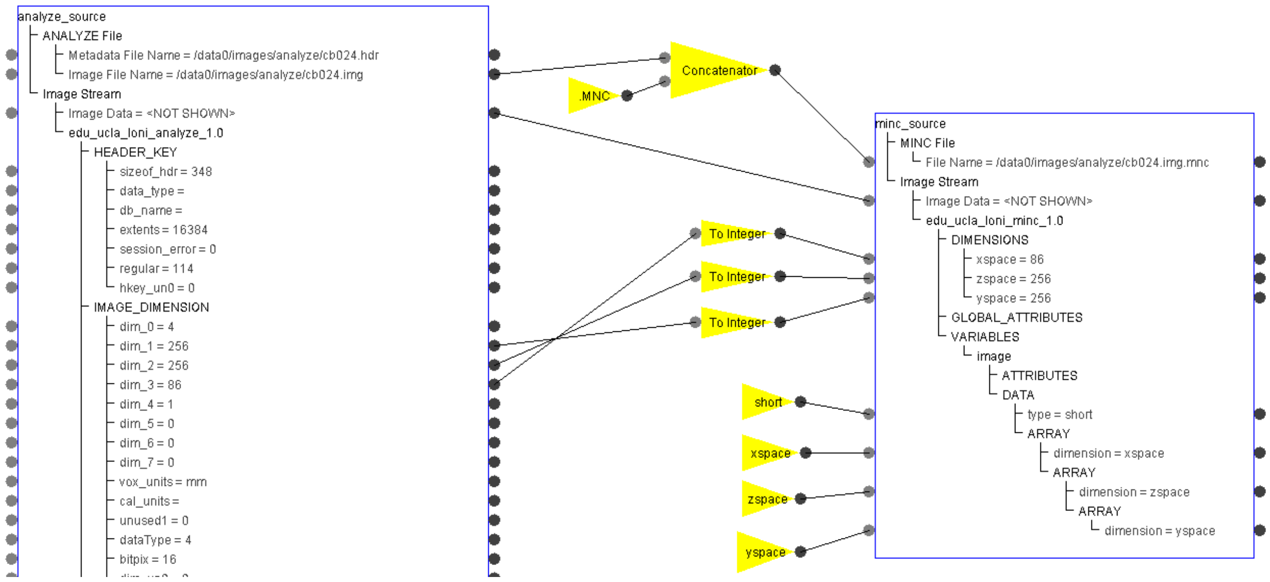
The output in the Debabeler console reads:

BEGIN: Determine how the input files are grouped together.
Assigned the group id "/data0/images/analyze/cb024.hdr" to the files:
/data0/images/analyze/cb024.hdr
/data0/images/analyze/cb024.img
Number of group ids successfully assigned = 1.
END: Determine how the input files are grouped together.

"Group Assignments" can be useful in two respects. First, if you have a series of images saved in multiple files (like a DICOM series that has one image per file), you can decide how to group them yourself. For example, you can remove scout images or separate MR images by echo time. Also, you can put image files of different file formats into the same group. This is useful if you would like to combine the contents of different image files together, or maybe read a contour and an image and save the image with the contour ovelayed on top of it.

Translate the input files

Now that the input files have identified as being produced from the source "analyze_source," the input files have been grouped, and the output type has been chosen as "minc_source," the Debabeler searches for a "Translation" in its list of "Translations" that has a source tree with the name "analyze_source" and a target tree with the name "minc_source." This "Translation" can be displayed in the processor panel by navigating through the menus Views->ANALYZE to MINC Translation. Clicking the "Translate" button near the bottom of the Debabeler produces the following display:



```
analyze_source
├ ANALYZE File
│ ├ Metadata File Name = /data0/images/analyze/cb024.hdr
│ └ Image File Name = /data0/images/analyze/cb024.img
└ Image Stream
  ├ Image Data = <NOT SHOWN>
  └ edu_ucla_loni_analyze_1.0
    ├ HEADER_KEY
    │ ├ sizeof_hdr = 348
    │ ├ data_type =
    │ ├ db_name =
    │ ├ extents = 16384
    │ ├ session_error = 0
    │ ├ regular = 114
    │ └ hkey_un0 = 0
    ├ IMAGE_DIMENSION
    │ ├ dim_0 = 4
    │ ├ dim_1 = 256
    │ ├ dim_2 = 256
    │ ├ dim_3 = 86
    │ ├ dim_4 = 1
    │ ├ dim_5 = 0
    │ ├ dim_6 = 0
    │ ├ dim_7 = 0
    │ ├ vox_units = mm
    │ ├ cal_units =
    │ ├ unused1 = 0
    │ ├ dataType = 4
    │ ├ bitpix = 16
```

```
minc_source
├ MINC File
│ └ File Name = /data0/images/analyze/cb024.img.mnc
└ Image Stream
  ├ Image Data = <NOT SHOWN>
  └ edu_ucla_loni_minc_1.0
    ├ DIMENSIONS
    │ ├ xspace = 86
    │ ├ zspace = 256
    │ └ yspace = 256
    ├ GLOBAL_ATTRIBUTES
    └ VARIABLES
      └ image
        ├ ATTRIBUTES
        └ DATA
          ├ type = short
          └ ARRAY
            ├ dimension = xspace
            └ ARRAY
              ├ dimension = zspace
              └ ARRAY
                └ dimension = yspace
```

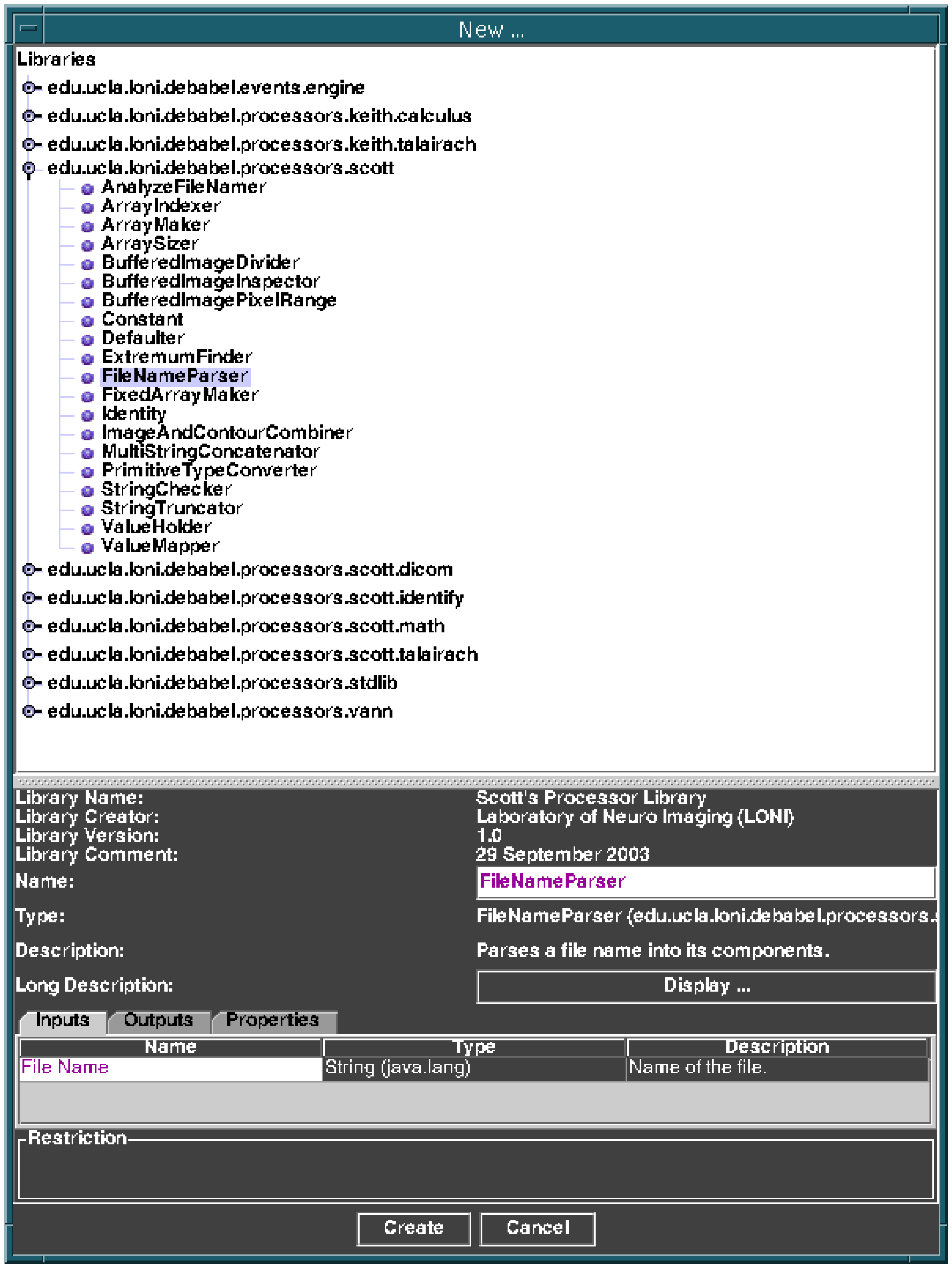
And the Debabeler console has printed:

BEGIN: Convert the input file groups into output files of the target.
Number of groups to translate = 1.
Decoded and read data for "analyze_source" from the files:
/data0/images/analyze/cb024.hdr
/data0/images/analyze/cb024.img
Wrote translated data for "minc_source" to the files:
/data0/images/analyze/cb024.img.mnc
Successfully translated (analyze_source) to (minc_source).
Number of successful translations = 1.
END: Convert the input file groups into output files of the target.

which reports that the ANALYZE file pair has been successfully converted into the MINC image "cb024.img.mnc." To many people this "Translation" mapping looks complex and its connections are not immediately obvious. Actually, many people would argue the "Translation" shown above isn't a sufficient mapping at all (there are missing fields). If you were going to create a "Translation" from scratch, you would necessarily have to be an expert in both the input and output file formats in order to create a meaningful mapping.

So what can be gained by representing the translation this way? First of all, you can see all the information in the input files. Often producers of image files place information in non-obvious or "hidden" places that you'd like access to. And second, most people want to do something slightly different from the hard-coded translations they have, and this representation of the translation allows you to tweak or add to it. In fact, a majority of requests center around just changing the output file name. The Debabeler comes with many supplied mappings, and it is recommended that you start by tweaking those.

As the mapping stands, the name of the output file is the name of the ANALYZE image file concatenated with the string ".mnc". To illustrate how to tweak this mapping, go to the top of the Debabeler and click on the "New ..." button to create a new processor. This will launch a dialog box that displays a list of the processor libraries to choose from. Select the processor "FileNameParser" in the library "edu.ucla.loni.debabel.processors.scott":

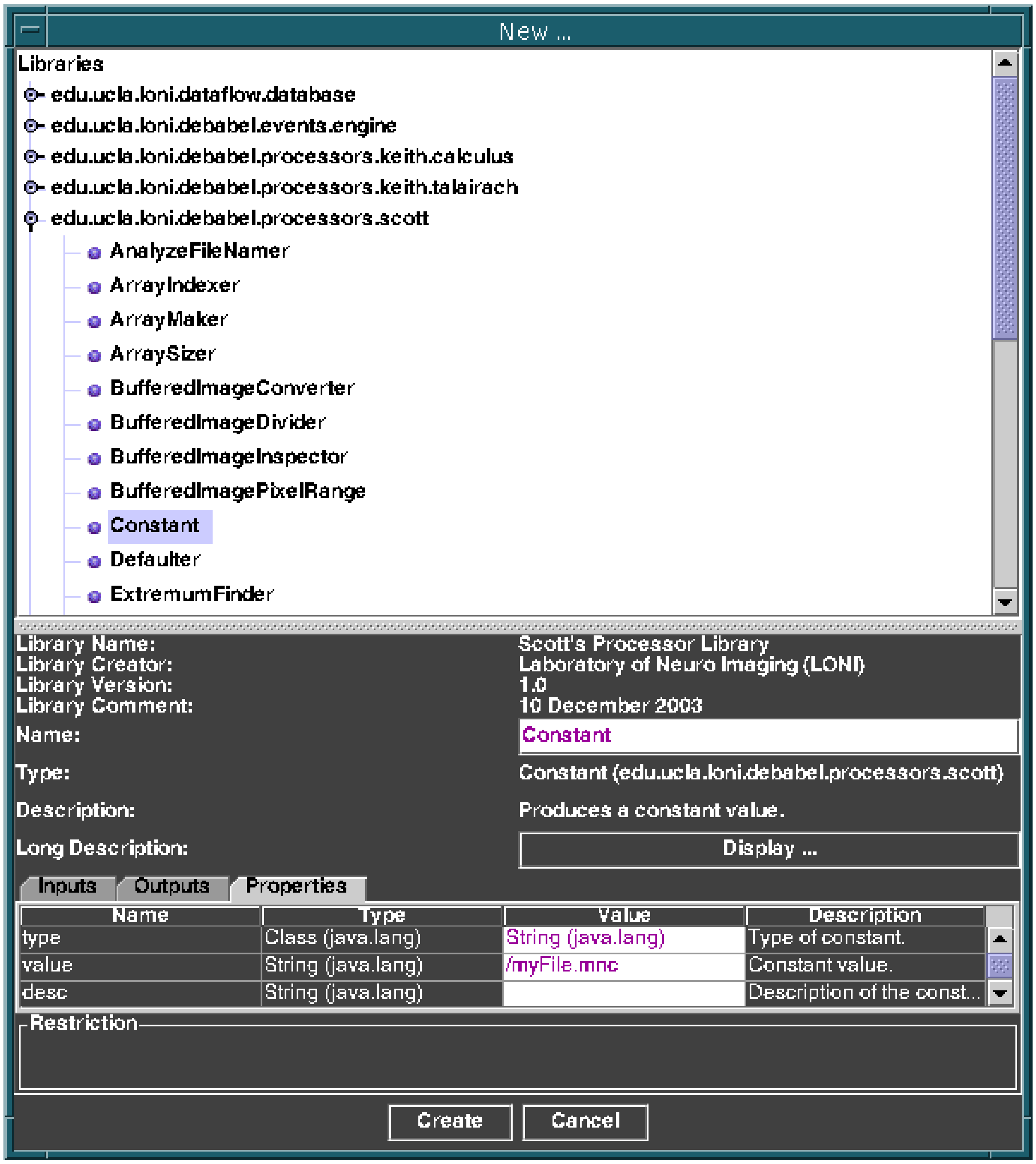


Libraries
- edu.ucla.loni.debabel.events.engine
- edu.ucla.loni.debabel.processors.keith.calculus
- edu.ucla.loni.debabel.processors.keith.talairach
- edu.ucla.loni.debabel.processors.scott
  - AnalyzeFileNamer
  - ArrayIndexer
  - ArrayMaker
  - ArraySizer
  - BufferedImageDivider
  - BufferedImageInspector
  - BufferedImagePixelRange
  - Constant
  - Defaulter
  - ExtremumFinder
  - FileNameParser
  - FixedArrayMaker
  - Identity
  - ImageAndContourCombiner
  - MultiStringConcatenator
  - PrimitiveTypeConverter
  - StringChecker
  - StringTruncator
  - ValueHolder
  - ValueMapper
- edu.ucla.loni.debabel.processors.scott.dicom
- edu.ucla.loni.debabel.processors.scott.identify
- edu.ucla.loni.debabel.processors.scott.math
- edu.ucla.loni.debabel.processors.scott.talairach
- edu.ucla.loni.debabel.processors.stdlib
- edu.ucla.loni.debabel.processors.vann

Library Name:	Scott's Processor Library
Library Creator:	Laboratory of Neuro Imaging (LONI)
Library Version:	1.0
Library Comment:	29 September 2003
Name:	FileNameParser
Type:	FileNameParser (edu.ucla.loni.debabel.processors.
Description:	Parses a file name into its components.
Long Description:	Display ...

Name	Type	Description
File Name	String (java.lang)	Name of the file.

After you click the "Create" button located near the bottom of the dialog box, a new processor appears in the Debabeler processor panel. Click the "New..." button again but this time choose the processor "Constant" in the library "edu.ucla.loni.debabel.processors.scott". Select the "Properties" tab and change "Integer (java.lang)" to "String (java.lang)" using the pull down menu and change "0" to "/myFile.mnc" (change to "\myFile.mnc" on Windows):

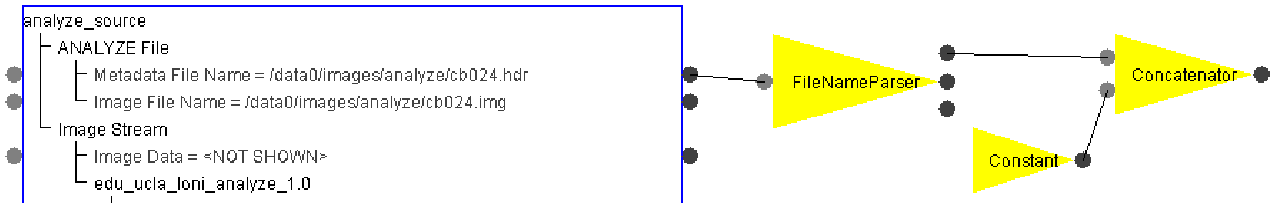


Libraries
- edu.ucla.loni.dataflow.database
- edu.ucla.loni.debabel.events.engine
- edu.ucla.loni.debabel.processors.keith.calculus
- edu.ucla.loni.debabel.processors.keith.talairach
- edu.ucla.loni.debabel.processors.scott
  - AnalyzeFileNamer
  - ArrayIndexer
  - ArrayMaker
  - ArraySizer
  - BufferedImageConverter
  - BufferedImageDivider
  - BufferedImageInspector
  - BufferedImagePixelRange
  - Constant
  - Defaulter
  - ExtremumFinder

Library Name:	Scott's Processor Library
Library Creator:	Laboratory of Neuro Imaging (LONI)
Library Version:	1.0
Library Comment:	10 December 2003
Name:	Constant
Type:	Constant (edu.ucla.loni.debabel.processors.scott)
Description:	Produces a constant value.
Long Description:	Display ...

Name	Type	Value	Description
type	Class (java.lang)	String (java.lang)	Type of constant.
value	String (java.lang)	/myFile.mnc	Constant value.
desc	String (java.lang)		Description of the const...

Hitting the "Create" button adds a new Constant processor to the Debabeler processor panel. If you remove the connections to the Concatenator (click the connection, then hit the "Remove" button located near the top of the Debabeler), you can connect the new processors like this:


analyze_source
ANALYZE File
Metadata File Name = /data0/images/analyze/cb024.hdr
Image File Name = /data0/images/analyze/cb024.img
Image Stream
Image Data = <NOT SHOWN>
edu_ucla_loni_analyze_1.0
FileNameParser
Concatenator
Constant

which produces the file name "/data0/images/analyze/myFile.mnc" in UNIX.

CREATING AN EXECUTABLE JAR

Once you have loaded, edited, and successfully run your Debabeler mappings, you can save them either as an XML file (which can be loaded back into the Debabeler for further editing and running) or as an executable jar file. By navigating through the menus: File->Save Mappings ..., you can save your mappings as a Debabeler XML file. If you instead choose the menus: File->Save as Executable Jar File ..., you can create a stand-alone Java jar file that can be executed independently of the Debabeler. If the name of your executable jar file is "myExecutable.jar", then entering on the command line:

java -jar myExecutable.jar

gives the following informative output:

```
LONI DEBABELER EXECUTABLE JAR FILE

Usage:  java -jar myExecutable.jar -input <args> -args <args> -target <args> -suppress -recursive

where:  java -> Java 1.4 or higher
        -input -> Input files and/or directories
        -args -> Command line arguments
        -target -> Output source names
        -suppress -> Use to suppress verbose messages
        -recursive -> Use to recursively search input directories
```

The "-input" flag specifies the files and/or directories of files that are to be debabeled. White space is not allowed for a file or directory name (if there is whitespace, put quotes around the name). For example, "-input myFile1.dcm myDirectory myFile2.dcm" would specify as input the files "myFile1.dcm" and "myFile2.dcm" as well as all the files in the directory named "myDirectory". If the "-recursive" flag is specified, then all subdirectories contained in "myDirectory" are specified as input files. If the "-recursive" flag is not present, the subdirectories are not searched.

The "-args" flag specifies Debabeler command line arguments whose meaning is dependent upon the mappings being run. The "-target" flag gives the name of the target and must be the name of a target tree in the list of registered mappings.

If the "-suppress" flag is present, no messages are printed while the Debabeler executable runs. Otherwise, messages are printed about the activity in progress.

WORKING WITH THE SUPPLIED MAPPINGS

The Debabeler download comes with some useful mappings developed at LONI. The mappings are logically separated as "translations" and "anonymizations." Each "translation" is a mapping between two different file formats and each "anonymization" is a mapping that removes patient identification metadata from one file format. Each translation and anonymization was written for a particular purpose with specific assumptions (e.g., in all cases image pixel data is assumed to be represented by 16 bit unsigned shorts). They are offered here not as all-purpose solutions, but as templates upon which other translations and anonymizations can be built. If you find yourself wanting the output file name to be constructed differently or today's date added to a metadata field, please follow the example here to make your own changes. However, each mapping by itself does offer useful functionality, so we are offering them here "as is."

The following is a list of the translations and anonymizations supplied with the Debabeler. Each translation and anonymization is saved in the "XML File" specified below. In order to create the executable jar file needed to run the examples, load the XML file into the Debabeler and save it as an executable jar file (name the jar file as in the example). The maximum heap size (the "-Xmx flag") should be adjusted according to the size of the images being converted.

- **ANALYZE TO ANFI TRANSLATION**
  - XML File: AnalyzeToAfni_23July2003.xml
  - Example: java -Xmx200M -jar analyzeToAfni.jar -input myFile.hdr -target afni_source
  - Result: Converts the ANALYZE files "myFile.hdr" and "myFile.img" to the AFNI files "myFile.hdr.BRIK" and "myFile.hdr.HEAD".
- **ANALYZE TO DICOM TRANSLATION**
  - XML File: AnalyzeToDicom_23July2003.xml
  - Example: java -Xmx200M -jar analyzeToDicom.jar -input myFile.hdr -target "DICOM source"
  - Result: Converts the ANALYZE files "myFile.hdr" and "myFile.img" to the DICOM files "myFile.hdr_<N>.dcm", where <N> is from 1 to the number of images in the ANALYZE file.

- **ANALYZE TO MINC TRANSLATION**
  - XML File: AnalyzeToMinc_22July2003.xml
  - Example: java -Xmx200M -jar analyzeToMinc.jar -input myFile.hdr -target minc_source
  - Result: Converts the ANALYZE files "myFile.hdr" and "myFile.img" to the MINC file "myFile.hdr.mnc".

- **DICOM TO ANALYZE TRANSLATION**
  - XML File: DicomToApproxAnalyze_29September2003.xml
  - Example: java -Xmx200M -jar dicomToAnalyze.jar -input mySrcDirectory -target approx_analyze_source
  - Result: Groups all the DICOM files in the directory "mySrcDirectory" into series and converts each group into an ANALYZE file. Each ANALYZE file pair is named "<myDicomFile1>.hdr" and "<myDicomFile1>.img", where <myDicomFile1> is the name of the first DICOM file in the series.

- **DICOM TO MINC TRANSLATION**
  - XML File: AnonymizedDicomToSficMinc_29September2003.xml

  - Example: java -Xmx200M -jar dicomToMinc.jar -input mySrcDirectory -args myDestDirectory -target sfic_minc
  - Result: Groups all the DICOM files in the directory "mySrcDirectory" into series and converts each group into a MINC file. The MINC files are written to the directory "myDestDirectory" and named according to their series.

- **GE TO ANALYZE TRANSLATION**
  - XML File: GeToApproxAnalyze_09September2003.xml
  - Example: java -Xmx200M -jar geToAnalyze.jar -input mySrcDirectory -target approx_analyze_source
  - Result: Groups all the GE files in the directory "mySrcDirectory" into series and converts each group into an ANALYZE file. Each ANALYZE file pair is named "<myGeFile1>.hdr" and "<myGeFile1>.img", where <myGeFile1> is the name of the first GE file in the series.

- **GE TO DICOM TRANSLATION**
  - XML File: GeToDicom_21July2003.xml
  - Example: java -Xmx200M -jar geToDicom.jar -input myFile.ge -target dicom
  - Result: Converts the GE file "myFile.ge" to the DICOM file "myFile.ge.dcm".

- **GE TO MINC TRANSLATION**
  - XML File: AnonymizedGeToSficMinc_29September2003.xml
  - Example: java -Xmx200M -jar geToMinc.jar -input mySrcDirectory -args myDestDirectory -target sfic_minc
  - Result: Groups all the GE files in the directory "mySrcDirectory" into series and converts each group into a MINC file. The MINC files are written to the directory "myDestDirectory" and named according to their series.

- **MINC TO ANALYZE TRANSLATION**
  - XML File: MincToApproxAnalyze_31July2003.xml
  - Example: java -Xmx200M -jar mincToAnalyze.jar -input myFile.mnc -target approx_analyze_source
  - Result: Converts the MINC file "myFile.mnc" to the ANALYZE files "myFile.mnc.hdr" and "myFile.mnc.img".

- **MINC TO DICOM TRANSLATION**
  - XML File: MincToDicom_28July2003.xml
  - Example: java -Xmx200M -jar mincToDicom.jar -input myFile.mnc -target dicom_source
  - Result: Converts the MINC file "myFile.mnc" to the DICOM files "myFile.mnc_<N>.dcm", where <N> is from 1 to the number of images in the MINC file.

- **SIEMENS MOSAIC DICOM TO ANALYZE TRANSLATION**
  - XML File: SiemensMosaicDicomToApproxAnalyze_15September2003.xml
  - Example: java -Xmx200M -jar siemensMosaicDicomToAnalyze.jar -input myFile.dcm -target approx_analyze_source
  - Result: Converts the DICOM file (in the Siemens Mosaic DICOM format, e.g., for FMRI) to the ANALYZE files "myFile.dcm.hdr" and "myFile.dcm.img".

- **ANALYZE ANONYMIZATION**
  - XML File: AnalyzeAnonymization_21July2003.xml
  - Example: java -Xmx200M -jar analyzeAnonymize.jar -input myFile.hdr -args <anonFile> <id> -target anonymize

  - Result: Anonymizes the metadata of the ANALYZE files "myFile.hdr" and "myFile.img", replaces the patient id with <id>, and writes the anonymized ANALYZE files "<anonFile>.hdr" and "<anon.File>.img".

- **DICOM ANONYMIZATION**
  - XML File: DicomAnonymization_22July2003.xml
  - Example: java -Xmx200M -jar dicomAnonymize.jar -input myFile.dcm -args <anonFile> <id> -target anonymize
  - Result: Anonymizes the metadata of the DICOM file "myFile.dcm", replaces the patient id with <id>, and writes the anonymized DICOM file "<anonFile><current date>.dcm", where <current date> is the current date and time.

- **GE ANONYMIZATION**
  - XML File: GeAnonymization_05August2003.xml
  - Example: java -Xmx200M -jar geAnonymize.jar -input myFile.ge -args <anonFile> <id> -target anonymize
  - Result: Anonymizes the metadata of the GE file "myFile.ge", replaces the patient id with <id>, and writes the anonymized GE file "<anonFile><current date>.ge", where <current date> is the current date and time.

- **MINC ANONYMIZATION**
  - XML File: MincAnonymization_21July2003.xml
  - Example: java -Xmx200M -jar mincAnonymize.jar -input myFile.mnc -args <anonFile> <id> -target anonymize
  - Result: Anonymizes the metadata of the MINC file "myFile.mnc", replaces the patient id with <id>, and writes the anonymized MINC file "<anonFile>".

- **DICOM AND SIEMENS MOSAIC DICOM TRANSLATION**
  - XML File: RogerCombined_29September2003.xml
  - Example: java -Xmx200M -jar contrib/rogerWoodsDicomToAnalyze.jar -input mySrcDirectory -target analyze_source
  - Result: Searches through all the DICOM files in the directory "mySrcDirectory" and separates the Siemens Mosaic DICOM files from the other DICOM files. Each Siemens Mosaic DICOM file is converted into an ANALYZE file, and the other DICOM files are grouped into series and each group is converted into a ANALYZE file. The ANALYZE files are written to a directory structure consisting of the patient name, study description, and series description.

WRITING YOUR OWN PROCESSOR LIBRARY

Users are free to develop their own processors for use in the Debabeler. The Debabeler was designed with a processor plugin architecture to support this. You write your own Java code, compile it and put it in a jar file, and place the jar file on the Debabeler classpath. At runtime, the Debabeler automatically searches the classpath for processor plugin jar files and loads the processor classes. When you create a Debabeler executable jar file, your processor classes are copied into the executable jar file.

Follow the steps below to create your own processors for use in the Debabeler:

CREATE A NEW DEBABELER PROCESSOR

Every Debabeler processor must be a subclass of *edu.ucla.loni.flow.process.Processor* and must implement two constructors. The first constructor has one string argument for the processor name, and the second constructor has one string argument for the processor name and one string argument for the properties of the processor. Processor properties allow Debabeler users to edit configurable parameters of a processor. Each processor must also implement the method *generateOutputValues* to generate values for the processor's output nodes using the values of the processor's input nodes. Source code is available for the Processor, Processor Factory, and Processor Property classes.

Here is an example (Adder. java) of how to create a simple Debabeler processor that takes as input two integers and returns as output the sum of the two integers.

And here is an example (MultiAdder.java) of how to create a more complex Debabeler processor that uses a processor property to specify how many integers to sum.

These examples can be compiled using the main Debabeler jar file:

javac -classpath debabeler_16October2003.jar Adder.java MultiAdder.java

CREATE A SERVICE PROVIDER INTERFACE

A processor SPI (service provider interface) is needed to link each library of processors to the Debabeler. Each processor SPI must be a subclass of *edu.ucla.loni.moduleio.spi.ProcessorSpi* and must implement the method *getProcessorClassNames* in order to give the Debabeler a list of processors in the library. Source code is available for the ProcessorSpi class.

Here is an example (MyProcessorSpi.java) of how to add the two processors above to a library.

This example can be compiled with the processors using the main Debabeler jar file:

javac -classpath debabeler_16October2003.jar Adder.java MultiAdder.java MyProcessorSpi.java

CREATE A PROCESSOR PLUGIN JAR FILE

In order for Java to recognize a service provider interface in a jar file, you must add a small file to the jar file's "META-INF/services" directory. For a Debabeler processor plugin jar file, the name of this file must be "edu.ucla.loni.moduleio.spi.ProcessorSpi" and its contents must be the names of the SPI subclasses.

In the above example, the file "META-INF/services/edu.ucla.loni.moduleio.spi.ProcessorSpi" contains:

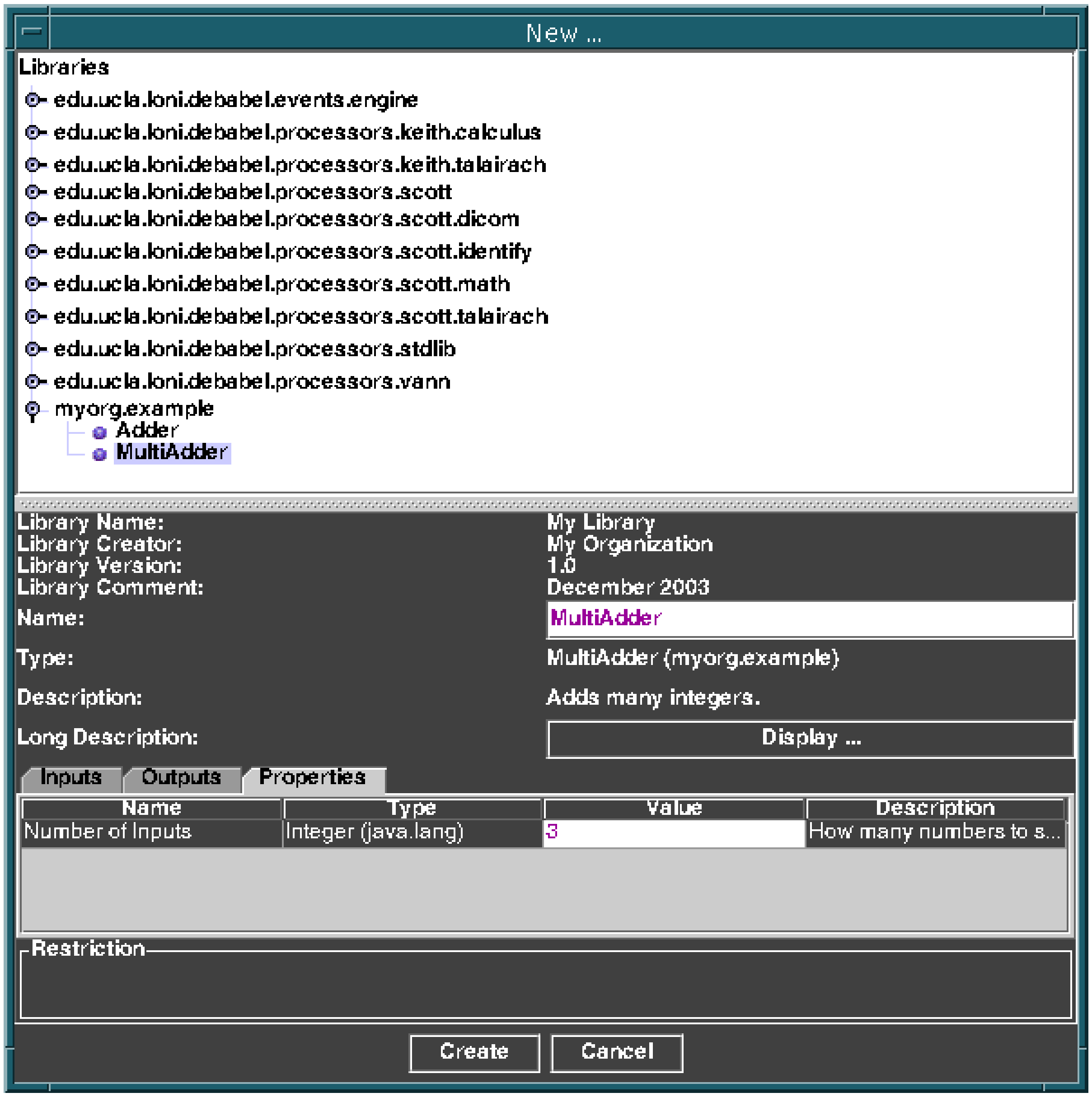
myorg.example.MyProcessorSpi

The processor plugin jar file (MyProcessorLib.jar) is created by combining the Debabeler processor classes, the processor SPI class, and the above META-INF services file:

jar cf MyProcessorLib.jar myorg/example/Adder.class myorg/example/MultiAdder.class myorg/example/MyProcessorSpi.class META-INF/services/edu.ucla.loni.moduleio.spi.ProcessorSpi

ADD THE PROCESSOR PLUGIN JAR FILE TO THE DEBABELER

New processor plugin jar files are added to the Debabeler by editing the "run.sh" and "run.bat" scripts; just add the new jar file names to the classpath. After the processor plugin jar file created above is added, the new processor dialog box looks like:



Libraries
- edu.ucla.loni.debabel.events.engine
- edu.ucla.loni.debabel.processors.keith.calculus
- edu.ucla.loni.debabel.processors.keith.talairach
- edu.ucla.loni.debabel.processors.scott
- edu.ucla.loni.debabel.processors.scott.dicom
- edu.ucla.loni.debabel.processors.scott.identify
- edu.ucla.loni.debabel.processors.scott.math
- edu.ucla.loni.debabel.processors.scott.talairach
- edu.ucla.loni.debabel.processors.stdlib
- edu.ucla.loni.debabel.processors.vann
- myorg.example
  - Adder
  - MultiAdder

Library Name:	My Library
Library Creator:	My Organization
Library Version:	1.0
Library Comment:	December 2003
Name:	MultiAdder
Type:	MultiAdder (myorg.example)
Description:	Adds many integers.
Long Description:	Display ...

Name	Type	Value	Description
Number of Inputs	Integer (java.lang)	3	How many numbers to s...

JAVA FILES

ADDER.JAVA

```
package myorg.example;

import edu.ucla.loni.flow.InputNode;
import edu.ucla.loni.flow.OutputNode;
import edu.ucla.loni.flow.exceptions.ProcessingException;
import edu.ucla.loni.flow.process.Processor;
import edu.ucla.loni.flow.process.ProcessorProperty;

public class Adder extends Processor
{
  public Adder(String name)
   {
    super(name, "Adds two integers.");

    // Create the Input Node for the first and second integers
    Class type = Integer.class;
    _addInputNode( new InputNode("Number 1", "First number.", type) );
    _addInputNode( new InputNode("Number 2", "Second number.", type) );

    // Create the Output Node for the sum
    _addOutputNode( new OutputNode("Sum", "1 + 2", type) );
   }

  public Adder(String name, ProcessorProperty[] properties)
   {
    this(name);
   }

  public void generateOutputValues() throws ProcessingException
   {
    try {

           // Get the two integers
           Integer number1 = (Integer)_getInputValue("Number 1", null);
           Integer number2 = (Integer)_getInputValue("Number 2", null);

           // Calculate the sum
           if (number1 != null && number2 != null) {
             int sum = number1.intValue() + number2.intValue();
             _setOutputValue("Sum", new Integer(sum));
           }
    }
```

```
      catch (Exception e) {
             String msg = "Unable to generate output values for \"" + this + "\".";
             throw new ProcessingException(msg, e);
      }
    }
}
```

MULTIADDER.JAVA

```
package myorg.example;

import edu.ucla.loni.flow.InputNode;
import edu.ucla.loni.flow.OutputNode;
import edu.ucla.loni.flow.exceptions.ProcessingException;
import edu.ucla.loni.flow.process.Processor;
import edu.ucla.loni.flow.process.ProcessorProperty;

public class MultiAdder extends Processor
{
  public MultiAdder(String name)
    {
      this(name, _getDefaultProperties());
    }

  public MultiAdder(String name, ProcessorProperty[] properties)
    {
      super(name, "Adds many integers.");

      // Add the Properties
      _addProperties(_getDefaultProperties(), properties);

      // Get the number of inputs
      Integer n = (Integer)_getPropertyValue("Number of Inputs");

      // Create the Input Nodes
      Class type = Integer.class;
      for (int i = 0; i < n.intValue(); i++) {
            _addInputNode( new InputNode("Number " + i, "Input #" + i, type) );
      }

      // Create the Output Node for the sum
      _addOutputNode( new OutputNode("Sum", "Sum of the inputs.", type) );
    }

  public void generateOutputValues() throws ProcessingException
    {
      try {

            // Get the number of inputs
            Integer n = (Integer)_getPropertyValue("Number of Inputs");

            // Add the integers
            int sum = 0;
            for (int i = 0; i < n.intValue(); i++) {
              Integer number = (Integer)_getInputValue("Number " + i, null);
              if (number != null) { sum += number.intValue(); }
            }

            // Set the sum
            _setOutputValue("Sum", new Integer(sum));
      }

      catch (Exception e) {
            String msg = "Unable to generate output values for \"" + this + "\".";
            throw new ProcessingException(msg, e);
      }
    }

  private static ProcessorProperty[] _getDefaultProperties()
    {
      ProcessorProperty[] properties = new ProcessorProperty[1];

      // Number of inputs (default of 3)
      properties[0] = new ProcessorProperty("Number of Inputs",
                                            "How many numbers to sum.",
                                            new Integer(3));
      return properties;
    }
}
```

MYPROCESSORSPI.JAVA

```
package myorg.example;

import edu.ucla.loni.moduleio.spi.ProcessorSpi;
import java.util.Locale;

public class MyProcessorSpi extends ProcessorSpi
{
  public MyProcessorSpi()
    {
      super("My Organization", "1.0", "My Library", "December 2003");
    }

  public String getDescription(Locale locale)
    {
      return "My Library";
    }

  public String[] getProcessorClassNames()
    {
      String[] classNames = new String[2];

      classNames[0] = Adder.class.getName();
      classNames[1] = MultiAdder.class.getName();

      return classNames;
    }
}
```

PROCESSOR.JAVA

```
/*
COPYRIGHT NOTICE
Copyright (c) 2003  Scott C. Neu, Daniel J. Valentino, and Arthur W. Toga

See README.license for license notices.
 */

package edu.ucla.loni.flow.process;

import edu.ucla.loni.flow.InputNode;
import edu.ucla.loni.flow.Module;
import edu.ucla.loni.flow.OutputNode;
import edu.ucla.loni.flow.exceptions.IllegalNameException;
import edu.ucla.loni.flow.exceptions.InvalidNameException;
import edu.ucla.loni.flow.exceptions.ProcessingException;
import java.util.Enumeration;
import java.util.Vector;

/**
 * Module that reads input values and produces output values.  Each derived
 * Class must implement two constructors.  The first constructor must take one
 * String argument for the Processor name.  The second constructor must take
 * one String argument for the Processor name and one Processor Property array
 * argument that configures the Processor according to the values in the array.
 *
 * @version 12 May 2003
 */
public abstract class Processor extends Module
{
  /** Input Nodes of the Processor. */
  private Vector _inputNodes;

  /** Output Nodes of the Module. */
  private Vector _outputNodes;

  /**
   * Properties of the Processor.
   *
   * @associates <{edu.ucla.loni.flow.process.ProcessorProperty}>
   * @link aggregation
   * @clientCardinality 1
   * @supplierCardinality 0..*
   */
  private Vector _properties;
```

```
/**
 * Constructs a Processor with the specified name and description.
 *
 * @param name Name of the Processor.
 * @param description Description of the Processor.
 *
 * @throws IllegalStringException If the name or description contains illegal
 *                                characters.
 */
protected Processor(String name, String description)
  {
    super(name, description);

    _inputNodes = new Vector();
    _outputNodes = new Vector();
    _properties = new Vector();
  }

/**
 * Gets the long description.
 *
 * @return A verbose description of what the Processor does.
 */
public String getLongDescription()
  {
    return "";
  }

/**
 * Gets the properties of the Processor.
 *
 * @return Copy of this Processor's properties.
 */
public ProcessorProperty[] getProperties()
  {
    ProcessorProperty[] props = new ProcessorProperty[ _properties.size() ];

    // Clone the Properties
    for (int i = 0; i < props.length; i++) {
        props[i] = (ProcessorProperty)_properties.elementAt(i);
    }

    return props;
  }
```

```
/**
 * Gets the Input Nodes.
 *
 * @return Input Nodes of the Module.
 */
public Enumeration getInputNodes()
 {
   return _inputNodes.elements();
 }

/**
 * Gets the Output Nodes.
 *
 * @return Output Nodes of the Module.
 */
public Enumeration getOutputNodes()
 {
   return _outputNodes.elements();
 }

/**
 * Generates values for the Output Nodes using the current values of the
 * Input Nodes.
 *
 * @throws ProcessingException If an error occurs during processing.
 */
public abstract void generateOutputValues() throws ProcessingException;

/**
 * Adds a Processor Property to the Processor.
 *
 * @param property Processor Property to add.
 *
 * @throws IllegalNameException If the name of the Property matches the name
 *                              of a Property already added to the Processor.
 */
protected void _addProperty(ProcessorProperty property)
 {
   // Check that the Property has not already been added
   if ( _properties.contains(property) ) {
        String msg = "The Processor Property named \"" + property + "\" has " +
                     "already been added to the Processor \"" + this + "\".";
        throw new IllegalNameException(msg);
   }

   // Add a Property clone
```

```
    _properties.addElement( property.clone() );
  }

 /**
  * Adds the default Processor Properties to the Processor and updates their
  * values with the given Processor Property values.  Two Properties are
  * considered equal if they have the same name.  If a value cannot be set
  * (e.g., does not satisfy a Restriction), it is ignored.
  *
  * @param defaultProperties Default Processor Properties to add.
  * @param newProperties Processor Properties with new values.
  *
  * @throws IllegalNameException If any Property cannot be added because its
  *                              name matches the name of a Property already
  *                              added to the Processor.
  */
 protected void _addProperties(ProcessorProperty[] defaultProperties,
                                         ProcessorProperty[] newProperties)
  {
    // Add all the default Properties
    for (int i = 0; i < defaultProperties.length; i++) {
         String name = defaultProperties[i].getName();

         // Look for a new Property with the same name
         for (int j = 0; j < newProperties.length; j++) {
           if ( newProperties[j].getName().equals(name) ) {
             Object value = newProperties[j].getValue();

             // Try to set the new value
             try { defaultProperties[i].setValue(value); }
             catch (Exception e) {}
           }
         }

         // Add the Property
         _addProperty(defaultProperties[i]);
    }
  }

 /**
  * Gets the value of the Processor Property with the specified name.
  *
  * @param propertyName Name of the Processor Property.
  *
  * @return Value of the named Processor Property.
  *
 * @throws InvalidNameException If the named Processor Property cannot be
 *                              found.
 */
protected Object _getPropertyValue(String propertyName)
  {
    // Search for a Processor Property with the specified name
    Enumeration enum = _properties.elements();
    while ( enum.hasMoreElements() ) {
           ProcessorProperty property = (ProcessorProperty)enum.nextElement();

           // Found a Property with the specified name
           if ( property.getName().equals(propertyName) ) {
             return property.getValue();
           }
    }

    // No Processor Property found
    String msg = "Unable to find a Processor Property with the name \"" +
                     propertyName + "\".";
    throw new InvalidNameException(msg);
  }

/**
 * Adds an Input Node to the Processor.
 *
 * @param inputNode Input Node to add to the Processor.
 *
 * @throws IllegalNameException If the name of the Input Node matches the
 *                              name of an Input Node already added to the
 *                              Processor.
 */
protected void _addInputNode(InputNode inputNode)
  {
    // Check that the Input Node doesn't already exist
    if ( _inputNodes.contains(inputNode) ) {
           String msg = "The Input Node named \"" + inputNode +
                        "\" has already been added to the Processor \"" +
                        getName() + "\".";

           throw new IllegalNameException(msg);
    }

    // Add the Input Node
    _inputNodes.addElement(inputNode);
  }
```

```
/**
 * Adds an Output Node to the Processor.
 *
 * @param outputNode Output Node to add to the Processor.
 *
 * @throws IllegalNameException If the name of the Output Node matches the
 *                              name of an Output Node already added to the
 *                              Processor.
 */
protected void _addOutputNode(OutputNode outputNode)
  {
    // Check that the Output Node doesn't already exist
    if ( _outputNodes.contains(outputNode) ) {
        String msg = "The Output Node named \"" + outputNode +
                     "\" has already been added to the Processor \"" +
                     getName() + "\".";

        throw new IllegalNameException(msg);
    }

    // Add the Output Node
    _outputNodes.addElement(outputNode);
  }

/**
 * Gets the value of the Input Node with the specified name.
 *
 * @param inputNodeName Name of the Input Node to get a value from.
 * @param defaultValue Value to return if the Input Node value is not of the
 *                     Input Node type.
 *
 * @return Value of the named Input Node, or the default value if the Input
 *         Node value is not of the Input Node type.
 *
 * @throws InvalidNameException If the Input Node name cannot be found.
 */
protected Object _getInputValue(String inputNodeName, Object defaultValue)
  {
    // Search the Input Nodes for a Node with the specified name
    Enumeration enum = getInputNodes();
    while ( enum.hasMoreElements() ) {
        InputNode inputNode = (InputNode)enum.nextElement();

        // Found a matching Input Node
        if ( inputNodeName.equals( inputNode.getName() ) ) {
          Object value = inputNode.getValue();
```

```
            // Return the value if it is of the Input Node type
            if ( inputNode.getType().isInstance(value) ) { return value; }

            // Otherwise return the default value
            return defaultValue;
          }
      }

      // No Input Node found
      String msg = "Unable to find an Input Node with the name \"" +
                   inputNodeName + "\".";
      throw new InvalidNameException(msg);
    }

  /**
   * Sets the value of the Output Node with the given name.
   *
   * @param outputNodeName Name of the Output Node to set the value of.
   * @param value Value to set.
   *
   * @throws IllegalNodeValueException If the value cannot be given to the
   *                                   Node.
   * @throws InvalidNameException If the Output Node name cannot be found.
   */
  protected void _setOutputValue(String outputNodeName, Object value)
    {
      // Search the Output Nodes for a Node with the specified name
      Enumeration enum = getOutputNodes();
      while ( enum.hasMoreElements() ) {
          OutputNode outputNode = (OutputNode)enum.nextElement();

          // Found a matching Output Node, set its value
          if ( outputNodeName.equals( outputNode.getName() ) ) {
            outputNode.setValue(value);
            return;
          }
      }

      // No Output Node found
      String msg = "Unable to find an Output Node with the name \"" +
                   outputNodeName + "\".";
      throw new InvalidNameException(msg);
    }
}
```

PROCESSORFACTORY.JAVA

```
/*
COPYRIGHT NOTICE
Copyright (c) 2003  Scott C. Neu, Daniel J. Valentino, and Arthur W. Toga

See README.license for license notices.
 */

package edu.ucla.loni.flow.process;

import edu.ucla.loni.flow.exceptions.InvalidNameException;
import edu.ucla.loni.flow.exceptions.NewProcessorException;
import java.lang.reflect.Constructor;

/**
 * Factory for creating new instances of Processors.
 *
 * @version 10 May 2003
 */
public class ProcessorFactory
{
  /** Name of the Processor to create. */
  private String _name;

  /** Class of the Processor to create. */
  private Class _class;

  /** Properties of the Processor being created. */
  private ProcessorProperty[] _properties;

  /**
   * Constructs a Processor Factory.
   *
   * @param processorName Name of the Processor to create.
   * @param className Class name of the Processor to create.
   *
   * @throws NewProcessorException If a Processor with default properties
   *                               cannot be instantiated.
   */
  public ProcessorFactory(String processorName, String className)
    throws NewProcessorException
    {
      // Create a Processor with default properties
      try {
            _name = processorName;

            // Get the Class for the Processor
            _class = Class.forName(className);

            // Get the Constructor with one String argument
            Class[] parameters = new Class[1];
            parameters[0] = String.class;
            Constructor constructor = _class.getDeclaredConstructor(parameters);

            // Create a new instance of the Processor
            Object[] arguments = new Object[1];
            arguments[0] = processorName;
            Processor p = (Processor)constructor.newInstance(arguments);

            // Get the default properties
            _properties = p.getProperties();
      }

      // Cannot instantiate the new Processor
      catch (Exception e) {
            String msg = "Unable to create a new instance of the Processor \"" +
                         className + "\".";
            throw new NewProcessorException(msg, e);
      }
    }

  /**
   * Sets the value of the specified Processor property.
   *
   * @param name Name of the property.
   * @param value Value for the property.
   *
   * @throws FlowRuntimeException If the Property value does not satisfy a
   *                              non-null Restriction.
   * @throws IllegalArgumentException If the Property value Class is not a
   *                                  Boolean, Byte, Class, Double, Float,
   *                                  Integer, Long, Short, or String Class.
   * @throws InvalidNameException If the named property does not exist.
   * @throws NullPointerException If the Property value is null.
   */
  public void setPropertyValue(String name, Object value)
    {
      // Get the named Property
      ProcessorProperty property = null;
      for (int i = 0; i < _properties.length; i++) {
            if ( _properties[i].getName().equals(name) ) {
              property = _properties[i];
            }
      }

      // Named Property not found
      if (property == null) {
            String msg = "No property named \"" + name + "\" found.";
            throw new InvalidNameException(msg);
      }

      // Set the value of the named Property
      property.setValue(value);
    }

  /**
   * Creates a new instance of the Processor.
   *
   * @return New instance of the Processor.
   *
   * @throws NewProcessorException If the Processor cannot be instantiated.
   */
  public Processor createProcessor() throws NewProcessorException
    {
      try {

            // Get the Constructor with one String and one Properties argument
            Class[] parameters = new Class[2];
            parameters[0] = String.class;
            parameters[1] = ProcessorProperty[].class;
            Constructor constructor = _class.getDeclaredConstructor(parameters);

            // Clone the properties
            ProcessorProperty[] props = new ProcessorProperty[_properties.length];
            for (int i = 0; i < props.length; i++) {
              props[i] = (ProcessorProperty)_properties[i].clone();
            }

            // Create and return a new instance of the Processor
            Object[] arguments = new Object[2];
            arguments[0] = _name;
            arguments[1] = props;
            return (Processor)constructor.newInstance(arguments);
      }

      // Cannot instantiate the new Processor
```

```
      catch (Exception e) {
             String msg = "Unable to create a new instance of the Processor \"" +
                          _class.getName() + "\".";
             throw new NewProcessorException(msg, e);
      }
    }
}
```

PROCESSORPROPERTY.JAVA

```
/*
COPYRIGHT NOTICE
Copyright (c) 2003  Scott C. Neu, Daniel J. Valentino, and Arthur W. Toga

See README.license for license notices.
 */

package edu.ucla.loni.flow.process;

import edu.ucla.loni.flow.FlowElement;
import edu.ucla.loni.flow.Restriction;
import edu.ucla.loni.flow.exceptions.FlowRuntimeException;

/**
 * Flow Element that represents a property of a Processor.  The property value
 * is limited to a Boolean, Byte, Class, Double, Float, Integer, Long, Short,
 * or String type.
 *
 * @version 10 May 2003
 */
public class ProcessorProperty extends FlowElement implements Cloneable
{
  /** Value of the Property. */
  private Object _value;

  /**
   * Restriction imposed on the Property value.
   *
   * @label restricted by
   * @link aggregationByValue
   * @supplierCardinality 0..1
   * @clientCardinality 1
   */
  private Restriction _restriction;

  /**
   * Constructs a Processor Property without a Restriction.
   *
   * @param name Name of the Property.
   * @param description Description of the Property.
   * @param value Value of the Property.
   *
   * @throws IllegalArgumentException If the Property value Class is not a
   *                                  Boolean, Byte, Double, Float, Integer,
   *                                 Long, Short, or String Class.
   * @throws IllegalStringException If the name or description contains illegal
   *                               characters.
   * @throws NullPointerException If the Property value is null.
   */
  public ProcessorProperty(String name, String description, Object value)
    {
      this(name, description, value, null);
    }

  /**
   * Constructs a Processor Property.
   *
   * @param name Name of the Property.
   * @param description Description of the Property.
   * @param value Value of the Property.
   * @param restriction Restriction imposed on the Property value.
   *
   * @throws FlowRuntimeException If the Property value does not satisfy a
   *                             non-null Restriction.
   * @throws IllegalArgumentException If the Property value Class is not a
   *                                 Boolean, Byte, Double, Float, Integer,
   *                                 Long, Short, or String Class.
   * @throws IllegalStringException If the name or description contains illegal
   *                               characters.
   * @throws NullPointerException If the Property value is null.
   */
  public ProcessorProperty(String name, String description, Object value,
                                         Restriction restriction)
    {
      super(name, description);

      _restriction = restriction;
      setValue(value);
    }

  /**
   * Gets the Property value.
   *
   * @return Value of the Property.
   */
  public Object getValue()
    {
      return _value;
    }
```

```
/**
 * Sets the Property value.
 *
 * @param value New value for the Property.
 *
 * @throws FlowRuntimeException If the Property value does not satisfy a
 *                              non-null Restriction.
 * @throws IllegalArgumentException If the Property value Class is not a
 *                                  Boolean, Byte, Class, Double, Float,
 *                                  Integer, Long, Short, or String Class.
 * @throws NullPointerException If the Property value is null.
 */
public void setValue(Object value)
  {
    // Check for a null value
    if (value == null) {
          String msg = "Property value may not be null.";
          throw new NullPointerException(msg);
    }

    // Check for a primitive Object type
    Class type = value.getClass();
    if (type != Boolean.class && type != Byte.class &&
            type != Class.class && type != Double.class &&
            type != Float.class && type != Integer.class &&
            type != Long.class && type != Short.class && type != String.class)
          {
            String msg = "Property value must be a Boolean, Byte, Class, " +
                         "Double, Float, Integer, Long, Short, or String.";
            throw new IllegalArgumentException(msg);
          }

    // Check that the value satifies a non-null Restriction
    if (_restriction != null && !_restriction.satisfiesRestriction(value)) {
          String msg = "The value \"" + value + "\" does not satisfy the " +
                       "restriction \"" + _restriction + "\".";
          throw new FlowRuntimeException(msg);
    }

    // Set the value
    _value = value;
  }

/**
 * Gets the Restriction.
 *
```

```
  * @return Restriction imposed on the Property value.
  */
 public Restriction getRestriction()
   {
     return _restriction;
   }

 /**
  * Determines whether or not the specified Object is equal to this Object.
  *
  * @param object Object for which to determine equality.
  *
  * @return True if the object is a Processor Property with the same name;
  *         false otherwise.
  */
 public boolean equals(Object object)
   {
     // Object must be a Processor Property
     if ( object instanceof ProcessorProperty ) {
         ProcessorProperty property = (ProcessorProperty)object;

         // Must also have the same name
         if ( property.getName().equals( getName() ) ) { return true; }
     }

     // Object is not equal to this one
     return false;
   }

 /**
  * Returns a clone of this Object.
  *
  * @return Clone of this Object.
  */
 public Object clone()
   {
     String name = getName();
     String description = getDescription();
     Object value = getValue();
     Restriction restriction = getRestriction();

     // Return a new Processor Property
     return new ProcessorProperty(name, description, value, restriction);
   }
}
```

PROCESSORSPI.JAVA

```
/*
COPYRIGHT NOTICE
Copyright (c) 2003  Scott C. Neu, Daniel J. Valentino, and Arthur W. Toga

See README.license for license notices.
 */

package edu.ucla.loni.moduleio.spi;

import javax.imageio.spi.IIOServiceProvider;

/**
 * Service Provider Interface (SPI) for importing a library of Processors.
 *
 * @version 6 May 2003
 */
public abstract class ProcessorSpi extends IIOServiceProvider
{
  /** Name of the library of Processors. */
  protected String _libraryName;

  /** Comment pertaining to the library. */
  protected String _libraryComment;

  /**
   * Constructs a Processor SPI.
   *
   * @param vendorName Name of the vendor who supplied this plug-in.
   * @param version Version of this plug-in.
   * @param libraryName Name of the library of Processors.
   * @param libraryComment Comment pertaining to the library.
   *
   * @throws IllegalArgumentException If the vendor name, version, or library
   *                                  name is null.
   */
  protected ProcessorSpi(String vendorName, String version, String libraryName,
                                  String libraryComment)
    {
      super(vendorName, version);

      // Check for a null library name
      if (libraryName == null) {
          String msg = "Must supply a non-null library name.";
          throw new IllegalArgumentException(msg);
      }
      _libraryName = libraryName;

      // Set the library comment to the empty String if it is null
      _libraryComment = libraryComment;
      if (_libraryComment == null) { _libraryComment = ""; }
    }

  /**
   * Gets the library name.
   *
   * @return Name of the library of Processors.
   */
  public String getLibraryName()
    {
      return _libraryName;
    }

  /**
   * Gets the comment pertaining to the library.
   *
   * @return Comment pertaining to the library, or the empty String if none is
   *         provided.
   */
  public String getLibraryComment()
    {
      return _libraryComment;
    }

  /**
   * Gets an array of fully-qualified class names of the Processors belonging
   * to the library.
   *
   * @return Fully-qualified class names of the Processors belonging to the
   *         library.
   */
  public abstract String[] getProcessorClassNames();
}
```