sourceIdentify

Architectural Design Document

Version 1.0
21 November 2004

Jürgen Knödlseder
Centre d’Etude Spatiale des Rayonnements
knodlseder@cesr.fr
http://www.cesr.fr/~jurgen/index.html
Note to the reader

This software has been written to analyse data of the LAT telescope onboard GLAST.
Contents

1 Introduction 1

2 Architectural design 2
   2.1 Overview .................................................. 2
   2.2 Interfaces ................................................. 2
   2.3 Structure of the executable .............................. 4
   2.4 Parameter file ............................................. 5
   2.5 Catalogue arithmetics ..................................... 6
   2.6 Counterpart catalogue list ................................ 7
   2.7 Probability assignment ................................... 7

3 Error codes 7
1 Introduction

The aim of the executable sourceIdentify is the identification of possible counterparts for newly found GLAST sources using potential counterpart source catalogues.

Counterpart identification is a complex process. In general, counterparts can be identified from a large number of different catalogues, each having its own precision, specific information and biases. Counterparts can be identified in a hierarchical scheme, searching for example first for X-ray counterparts, then for radio counterparts of the X-ray counterparts, and finally for optical or infrared counterparts of the radio counterparts. They could also be searched in a parallel scheme, where counterparts in two radio catalogues taken at different frequencies should be found simultaneously in order to use the radio spectral index to evaluate the counterpart probabilities (c.f. Fig. 1).

Combining all this complexity in a single-run executable would lead to a huge and complex program with an overwhelming number of parameters that will be difficult to handle. Also, adding functionalities will in general add to the complexity of the program, and will also require software modifications.

We pursue here a different approach for sourceIdentify which we believe is more flexible and versatile. sourceIdentify consists of a simple and solid executable that identifies counterparts for a list of sources from a single catalogue. As such, sourceIdentify has a limited number of well defined task parameters. The required complexity for source identification is reached by executing sourceIdentify subsequently on different catalogues. This subsequent execution may be either programmed using a scripting language, or could be coded in specific analysis executables.

![Diagram](image)

Figure 1: Usage of the executable sourceIdentify. The left panel illustrates a hierarchical scheme where the uncertainty in the counterpart location is refined at each run of sourceIdentify. The right panel illustrates a parallel scheme where first two counterpart catalogues are combined (for example two radio catalogues for observations at frequency $\nu_1$ and $\nu_2$) and then the resulting counterpart catalogue is confronted with the GLAST source catalogue.
For example, in the hierarchical identification scheme cited above, 3 runs of \texttt{sourceIdentify} are required, the output of each run being the input of the next run:

1. Finding counterparts for GLAST sources in a X-ray catalogue results in the counterpart candidate catalogue 1 (CCC1)
2. Finding counterparts for CCC1 objects\footnote{A counterpart candidate catalogue object is the association between the sources of various input catalogues, found by \texttt{sourceIdentify}.} in a radio catalogue results the counterpart candidate catalogue 2 (CCC2)
3. Finding counterparts for CCC2 objects in a optical or infrared catalogue results the final counterpart candidate catalogue.

In the parallel scheme cited above, 2 runs of \texttt{sourceIdentify} are required:

1. Finding counterparts for a catalogue of objects observed at radiowavelength \( \nu_1 \) in a catalogue of objects observed at radiowavelength \( \nu_2 \) results in a radio counterpart catalogue (RCC) including spectral information (here the spectral radio index which will be computed by \texttt{sourceIdentify}).
2. Finding counterparts for GLAST sources in the RCC will result in the final counterpart candidate catalogue.

\section{Architectural design}

\subsection{Overview}

The executable \texttt{sourceIdentify} finds for a list of sources specified in an input catalogue the possible counterparts from a counterpart catalogue. Each source from the input catalogue may have a number of counterpart candidates. \texttt{sourceIdentify} assigns counterpart probabilities for each of the counterpart candidates.

\subsection{Interfaces}

The interfaces of the executable \texttt{sourceIdentify} are illustrated in Fig. 2.

On input the executable \texttt{sourceIdentify} takes two catalogues

- a source catalogue for which counterparts should be searched (this may typically be the GLAST source catalogue)
- a counterpart catalogue which provides the list of potential counterparts (this catalogue is typically taken from a database, such as CDS or HEASARC)

On output the executable produces a catalogue of counterpart candidates, with one row (entry) for each counterpart candidate. This means that each counterpart candidate will be considered as a distinct source in the output catalogue; this logic allows to treat the output counterpart candidate catalogue as source catalogue for a further run of \texttt{sourceIdentify} (c.f. Fig. 1).

Each counterpart candidate in the output catalogue is specified by

- a unique name
the coordinates of the counterpart

- the position uncertainty specified as the 95% error circle around the counterpart position

- the name of the source in the source catalogue

- the name of the counterpart in the counterpart catalogue

The coordinates and positional uncertainty of the counterpart are those of the object that has the smaller positional uncertainty in either of the two input catalogues. In that way, the positional accuracy for a counterpart generally improves with respect to the positional accuracy in the source catalogue. This logic allows to set up sourceIdentify in a hierarchical counterpart identification scheme, where sourceIdentify is run in a sequence using counterpart catalogues with smaller and smaller error circles (c.f. left panel of Fig. 1).

The name of the sources in the output catalogue will be constructed from the coordinates of the source. The names of the objects in the source and the counterpart catalogue will also be saved in the output catalogue for tracability.

Additional information can be added to the output catalogue from either of the two input catalogues, such as for example source fluxes.

New information can be derived by combining information from both input catalogues. In that way, flux ratios or spectral indices may be calculated (c.f. section 2.5).

For reading and writing of catalogues, sourceIdentify makes use of the interface routines provided by the library catalogAccess (99). Consequently, the catalogues can be read and written in the formats that are implemented in catalogAccess.

The parameters for sourceIdentify are specified in a so-called task parameter file, which is defined in section 2.4. The task parameter file is given in the IRAF format and will be read using either the PIL library or the HOOPS class provided in the GLAST software package.

sourceIdentify will also create an ASCII log-file in the directory where the executable sourceIdentify is executed. The purpose of the log-file is
Figure 3: Structure of the executable sourceIdentify.

- to log errors that occurred during task execution
- to log the actions that sourceIdentify performed for the counterpart search in order to control the proper execution of the task
- to provide a detailed report about the task execution in case that the result looks suspicious to the user

The level of information provided in the log-file can be controlled by a task parameter.

2.3 Structure of the executable

The structure of the executable is illustrated in Fig. 3. sourceIdentify is broken up into a driver code sourceIdentify, which calls all functions that are necessary to perform the counterpart identification, and 3 packages that do the main work:

- the Parameters package that handles the access to the task parameters that are specified in the parameter file (c.f. section 2.4).
- the Log package that handles information logging into the ASCII log file
- the Catalogue package that handles the loading and writing of the catalogues, and that performs the counterpart identification

The 3 packages interface with the outside world (the interface is indicated by the thick red line in Fig. 3). Parameters interfaces to PIL and/or HOOPS to read the task parameters. Log interfaces to an ASCII file into which information will be logged. Catalogue interfaces to the catalogAccess (US) library for reading and writing catalogues.
This structure allows to easily extend the executable. One may also replace `sourceIdentify` by codes that perform a hierarchical or a parallel source identification (in the actual design it is anticipated that complex identification schemes are realised by scripts that combine multiple runs of `sourceIdentify`, c.f. Fig. 1).

### 2.4 Parameter file

The task parameter file has the following structure:

```plaintext
# # Catalogue information
#
srcCatName, s,a, "glast.fits", "Source catalogue name"
cptCatName, s,a, "Veron", "Counterpart catalogue name"
outCatName, s,a, "result.fits", "Output catalogue name"
srcCatQty, s,a, "*", "Source catalogue quantities to be written"
cptCatQty, s,a, "*", "Counterpart catalogue quantities to be written"
outCatQty01, s,a, "R=SRC_FLUX/CPT_FLUX", "New output catalogue quantity 1"
outCatQty02, s,a, "S=SRC_FLUX+CPT_FLUX", "New output catalogue quantity 2"
#
# Task parameters
#
probMethod, s,a, "POSITION", "Probability method"
probThres, r,a, "1.0e-6", "Probability threshold"
maxNumCtp, i,a, "100", "Maximum number of counterpart candidates"
select01, s,a, "1.0e20 < SRC_FLUX", "Selection criterion 1"
select02, s,a, "1.0 < R < 5.0", "Selection criterion 2"
#
# Standard parameters
#
clobber, b,a, "yes", "Overwrite existing files?"
verbose, i,a, 3, "Verbosity level"
```

The task parameters have the following meaning:

- **srcCatName** Filename of the source catalogue.
- **cptCatName** Name of the counterpart catalogue. The name may be either the filename of a counterpart catalogue that resides locally on the disk, or the name of a counterpart catalogue available at one of the data centres and specified in the catalogue list given in section 2.6.
- **outCatName** Name of the output counterpart candidate catalogue that will be created by `sourceIdentify`.
- **srcCatQty** specifies the quantities from the source catalogue that should be copied into the output catalogue. If * is specified, all quantities of the source catalogue will be copied to the output catalogue. If the parameter is left blank, only the source name will be copied into the output catalogue. To avoid duplication of quantity names in the output catalogue, the quantities that will be copied from the source catalogue will be prefixed by `SRC_`.
- **cptCatQty** specifies the quantities from the counterpart catalogue that should be copied into the output catalogue. If * is specified, all quantities of the counterpart catalogue will be copied to the output catalogue. If the parameter is left blank, only the source name will be copied into the output catalogue. To avoid duplication of quantity names in the output catalogue, the quantities that will be copied from the counterpart catalogue will be prefixed by `CPT_`. 
• **outCatQtyn** specifies new quantities that will be created in the output catalogue by combining quantities from both the source and the counterpart catalogues. A typical example would be the determination of flux ratios, colors, or spectral indices. For the syntax of this parameter the user is referred to section 2.5. *nn* runs from 01 to 99: hence a maximum of 99 new output columns may be created.

• **probMethod** specifies the method that is used to assign counterpart probabilities. The following methods exist (c.f. section 2.7):

  - **POSITION**: probability based on positional proximity.

• **probThres** specifies the minimum required probability for a counterpart candidate to make it a member of the output catalogue. Counterpart candidates with smaller probabilities will be discarded and will not be written to the output catalogue.

• **maxNumCtp** specifies the maximum number of counterpart candidates for each of the sources in the source catalogue that will be written to the output catalogue. If 0 is specified, no limitation will be applied. The counterpart candidates will be written with descendent counterpart probability, i.e. specifying for example 5 will provide the 5 most likely counterparts for each of the sources.

• **selectnm** specifies selection criteria that should be applied for counterpart identification. Only counterpart candidates will be considered that meet these selection criteria. The general selection criterion is:

  \[ {\text{MIN}} < \text{QUANTITY} < {\text{MAX}} \]

  where **MIN** and **MAX** specify the minimum and maximum value that the quantity **QUANTITY** may take. For lower or upper limits, one of the two boundaries may be omitted, i.e.

  \[ {\text{MIN}} < \text{QUANTITY} \text{ or } \text{QUANTITY} < {\text{MAX}} \]

  *nn* runs from 01 to 99: hence a maximum of 99 selection criteria may be specified. To identify to which catalogue the quantity refers, quantities from the source catalogue have to be prefixed with **SRC_**, while quantities from the counterpart catalogue have to be prefixed with **CPT_**. Selection criteria may be also applied for newly created quantities.

• **clobber** specifies if existing output data structures should be overwritten or not. If **yes** is specified, the executable will notify the user about the deletion of any file. If **no** is specified and the executable attempts to overwrite existing data, the task will exit with an error message.

• **verbose** specifies the verbose level of the executable:

  - **verbose=0**: no information will be logged in case of an error.
  - **verbose=1**: only errors will be logged.
  - **verbose=2**: errors and actions will be logged.
  - **verbose=3**: detailed report about the task execution.

### 2.5 Catalogue arithmetics

The parameter **outCatQtyn** allows the creation of new quantities in the output catalogue by combining information from both the source and the counterpart catalogues. To signal from which of the input catalogues the information should be taken, the prefix **SRC_** has to be added for source catalogue quantities, while the prefix **CPT_** has to be added for counterpart catalogue quantities. In decoding the parameter string white spaces are ignored. The parameter string is case sensitive.

The following basic arithmetic operations are provided: addition (*+*), subtraction (*-*), multiplication (*\*\*), division (*/\*), and the logarithm (basis 10) (**log**). The operations may be combined as needed.

Here come some examples to illustrate the syntax:

```plaintext
outCatQtyn01,s,a,"SUMFLUX = SRC_FLUX + CPT_INTENSITY",,"Add fluxes"
```
adds the flux from the source catalogue quantity FLUX to the intensity of the counterpart catalogue quantity INTENSITY and creates the new quantity SUMFLUX in the output catalogue.

\[
\text{outCatQty01},s,a,"FLUXRATIO = SRC\_FLUX / CPT\_INTENSITY",,"Divide fluxes"
\]
divides the flux from the source catalogue quantity FLUX by the intensity of the counterpart catalogue quantity INTENSITY and creates the new quantity FLUXRATIO in the output catalogue.

\[
\text{outCatQty01},s,a,"INDEX = \log(90.0/53.0) * \log(SRC\_FLUX53 / CPT\_FLUX90)"","
\]
determines the spectral index from two microwave catalogues, taken at 53 GHz (the source catalogue) and at 90 GHz (the counterpart catalogue). The fluxes are given in the input catalogues by the FLUX53 and FLUX90 quantities. A new quantity INDEX is created in the output catalogue.

2.6 Counterpart catalogue list

The list of available counterpart catalogues is given by the functionality of catalogAccess (U9).

2.7 Probability assignment

TBD

3 Error codes

TBD