NOTE 249 – The AIPS++ Recipes Repository

George Moellenbrock, NRAO-Socorro

30 January 2002

Contents

1 Introduction

The Glish Command-Line Interface (CLI) in AIPS++ is a very powerful and flexible environment for data manipulation and analysis. In addition to providing a concise means of access to and operation of the standard astronomy reduction tools in AIPS++ an impressive array of generic tools are available. One of the greatest strengths of AIPS++ is the ability to combine standard astronomy reductions with ad hoc analyses devised by individual users.

Owing to this exceptional flexibility, and a tendency among users to document standard reduction sequences as glish scripts, a tremendous wealth of user-designed Glish scripts exists all over the world. As glish scripts are the language of AIPS++ it is therefore desirable to collect useful recipes in one place where than can be used as written, or as practical examples from which unique data analysis tools can be easily developed by users. This document describes the revitalized AIPS++ Recipes Repository, and how to contribute to it.

More information about how to use Glish itself can be found in AIPS++ Note 195, Getting Started in Glish, and the Glish User's Manual. The AIPS++ User Reference Manual describes how to use all AIPS++ modules and tools in Glish, and is often referred to in individual recipes.

1.1 How to Contribute a Recipe

It is now a fairly simple matter to contribute a recipe. Just follow the instructions described in the next few sections for formatting your script file (a template `.g` file is available here: recipe.g). Give it a short descriptive filename that ends in `rec.g`, and send it to aips2-recipes@nrao.edu.

2 Basic Principles

A useful recipe is not simply a random glish script. The purpose of the Recipes Repository is to supplement the conventional AIPS++ documentation with concrete examples that will prove instructive for novice glish script-writers. To make the most of the AIPS++ Recipes Repository, some guiding principles are required:

- The best recipes are short and of limited scope. A recipe should provide a quick and useful function, or concisely illustrate a clever glish mechanism or set of closely related operations. Such recipes will make it easy for a user to understand how to implement a particular operation and leverage it to his/her own purposes.
- Scripts intended for the Recipes Repository should generate their own data, or make use of data in the Data Repository. When appropriate, datasets of reasonable size may be added to the Data Repository.
- Scripts submitted for the Recipes Repository must follow the standard format (described in detail below). The format is self-contained, comment-rich, executable, and has been designed for readability.

- Markup of scripts for the Recipes Repository should be simple and require a minimum of additional information. This is necessary for maintenance of the Recipes Repository web pages. In most cases, the additional information should be limited to links to relevant documentation.

- The recipes page should be organized hierarchically according to different recipe categories. For now, there are three categories: General, Synthesis, and Single Dish. Each of these categories may be divided as necessary. In short, it should be easy to find scripts of different types, even as the Repository grows.

3 The Recipe Script Format

Scripts submitted for the Recipes Repository must follow a standard format. A template `.g` file is available here: recipe.g). The format consists of four main sections, Titles, Script, Output, and Credits. These are described in turn:

3.1 Titles

The Titles section provides comment keywords for general information about the script, its goals and its results. It should be possible to recognize a particular recipe as useful to one's own purposes from a quick scan of this section. The keywords in the Titles section are as follows:

`#RECIPE:` A descriptive title for the recipe (not the filename)

`#CATEGORY:` The category to which the recipe belongs (General, Synthesis, or Single Dish)

`#GOALS:` The specific goals of the recipe, in one line

`#USING:` The AIPS++ modules and/or tools used in the script

`#RESULTS:` A one-line description of the output of the script

`#ASSUME:` A one-line description of assumptions made by the script, in particular, which external data files may be required

`#SYNOPSIS:` A paragraph describing the novelty of the script; this is the only Titles keyword which may extend beyond a single line. Each line should be prepended by "`# `".

`#SCRIPTNAME:` The .g filename of the script (used for markup purposes)

3.2 Script

The Script section is where the script itself resides, and it begins with the keyword "`#SCRIPT:`" on a line by itself. The script should written following these rules:

- A recipe script has two columns, each 40 characters wide. The left column consists of the executable commands, and the right column consists of short descriptive comments (prepended by "`# `") for each line. If the executable commands extend beyond 40 characters, the comment should appear on the previous line, properly indented.
- All `include` commands should appear at the top of the script, and each should be specifically commented.
- Use mnemonically useful names for AIPS++ tools and other variables. For example, a fitter tool should be called "`myfit`" or similar. Above all, avoid single-letter names for tools.
- Terminate each glish command with "`;`". This will aid in debugging typos (e.g., unbalanced delimiters).
- Double-space the glish commands unless neighboring lines are very closely associated (e.g., a sequence of simple glish algebra, a simple loop, etc.).
- Comment every line of glish. This rule may be relaxed when two or three closely associated lines of rudimentary glish are fully described by a single comment, or when a short loop is executing a single operation repeatedly. Every double-spaced line of glish should have its own descriptive comment.
- For glish commands containing tool function executions, the parameters of the function should be listed with parameter names included, one per line, left-justified according to the indentation of the first parameter. Standard glish functions (e.g. `array()`, `min()`, `max()`, and others which do not require an `include`) need not have their parameter names included. Parameters used with default values may be omitted. When a function parameter consists of a function execution, that function's parameters should follow the same rules. In cases where

this leads to poor readability (do not nest functions in more than two layers), opt for storing values returned by functions in glish variables, and use these variables in subsequent function parameters.

- In cases where additional annotation or explanation is required, these may be added before or after the relevant line(s) of glish as full-width (80 characters) comments, prepended by "`# `". In the interest of succinctness, such comment blocks should be kept to a minimum, but they are often useful to express the finer points and lore of a particular operation.

3.3 Output

The output section begins with "`#OUTPUT:`" (on its own line) and contains sample output from the script. This section is very important since it enables a novice user to determine if the recipe (or their adaptation of it) is running properly. Rules for including recipe output are as follows:

- Prepend all output lines with "`# `" so that the script remains executable.
- Only include output unique to the script. It is not necessary, for example, to include the messages generated by `include` commands.

3.4 Credits

The Credits section provides for any last words of advice for using the script, and for attribution. The following keywords are supported:

`#CONCLUSION:` Like the synopsis, this section is useful for describing details of the recipe, in this case, those appropriate after a user has run the recipe. All lines must be prepended with "`# `".

`#SUBMITTER:` The full name of the submitter should be supplied here.

`#SUBMITAFFL:` The affiliation of the submitter should be supplied here.

`#SUBMITDATE:` The date of submission should be supplied here, formatted as yyyy-mmm-dd, e.g., 2002-Jan-29.

3.5 The Recipe Script Format: An example

An example of a properly formatted recipe script can be found here: plotrec.g.

4 The Recipe LaTeX Markup

This section describes the LaTeX markup of recipes for the Recipes Repository web-page. Users submitting recipes for the Repository need not concern themselves with this section since AIPS++ project staff will handle this part of the recipe submission if the contributed script is properly formatted as described above.

The recipe script format described above is designed to make markup for the Recipes Repository web-page very simple. LaTeX macros for each of the recipe script format keywords are defined in ahrecipe.tex. Via these macros, the Titles and Credits sections of the script format are duplicated in the LaTeX markup, with commands for linking to the script itself and to the relevant module or tool documentation the only added information. For simplicity's sake, the Script and Output sections of the script format are included as `verbatim` blocks. In this way, the script looks like a script (and is readable if the prescribed format is followed), and lines of the script may be copied directly to the Glish CLI with mouse-driven cut-and-paste.

A template `.latex` file is available here: recipe.latex).

4.1 The Recipe LaTeX Markup: An example

An example of a proper recipe LaTeX markup can be found here: plotrec.latex.

5 Recipes Repository Implementation

The Recipes Repository consists of two parts. First, the recipes themselves are to be checked in as well-commented glish scripts (see above for details of the required format) in `code/contrib/recipes`. The build system copies the scripts to `libexec` so that they may be accessed directly.

The second part of the Repository is the documentation of each recipe. The directory structure for the documentation is as follows:

`code/doc/recipes`	root directory for Recipe docs
`code/doc/recipes/Recipes.latex`	LaTeX markup for main Recipes web-page
`code/doc/recipes/makefile`	makefile for `Recipes.latex`
`code/doc/recipes/index.html`	forwarding html for main Recipes web-page
`code/doc/recipes/General/`	directory for General recipes markup
`code/doc/recipes/General/General.latex`	LaTeX markup for General category web-page
`code/doc/recipes/General/makefile`	makefile for General recipes
`code/doc/recipes/General/index.html`	forwarding html for General category web-page
`code/doc/recipes/Synthesis/`	directory for Synthesis recipes markup
`code/doc/recipes/Synthesis/Synthesis.latex`	LaTeX markup for Synthesis category web-page
`code/doc/recipes/Synthesis/makefile`	makefile for Synthesis recipes
`code/doc/recipes/Synthesis/index.html`	forwarding html for Synthesis category web-page
`code/doc/recipes/SingleDish/`	directory for SingleDish recipes markup
`code/doc/recipes/SingleDish/SingleDish.latex`	LaTeX markup for SingleDish category web-page
`code/doc/recipes/SingleDish/makefile`	makefile for SingleDish recipes
`code/doc/recipes/SingleDish/index.html`	forwarding html for SingleDish category web-page

The LaTeX markup file for each recipe (see above for details of the required format) is to be deposited into the markup directory appropriate to the recipe's category. The LaTeX markup file for the category should be updated with the recipe name and title, and the appropriate line added to the category's makefile. The build system will then process the recipe's LaTeX markup file and generate `.ps.gz` and html files in the appropriate locations in the docs tree.