NRAO AIPS++ Users Group	NAUG-002 Revision:1.1 2006-04-11 *UIWG Report* S. Myers

NAUG User Interface Working Group Report

UIWG Report

NRAO AIPS++ Users Group (NAUG)

Keywords: Requirements, Offline, Software, Science, Interface	
Author Signature: S. Myers	Date: 2006-04-11
Approved by: S. Myers Institute: NRAO	Signature: Date:
Released by: Institute:	Signature: Date:

Change Record

Revision	Date	Author	Section/ Page affected	Remarks
0.0	2006-04-03	Myers	All	Start draft
1.0	2006-04-07	Myers	All	First draft
1.1	2006-04-11	Myers	All	UIWG comments

`$Id: uiwg_report.tex,v1.1,2006/04/11 myers Exp myers $`

Contents

1 Executive Summary

The Common Astronomy Software Applications (CASA) software package is the evolution of the Astronomical Information Processing System (AIPS++). The AIPS++ project, which despite more than 10 years of development, never gained significant user acceptance, and the project was reorganized in 2002. Following the recommendations of a Technical Review held in 2003, the core software libraries were moved to work under a new Python-based framework, resulting in CASA. AIPS++, and now CASA, has many user requirements to fulfill from the ALMA and EVLA projects, and is regularly monitored for progress and compliance through a testing program. However, it was felt that the important issue of user acceptance, particularly as regards the more subjective criteria of usability, package organization, documentation and interface look and feel, should be addressed at least at first by a special focus group representing users.

The User Interface Working Group (UIWG) is a subgroup of the NRAO AIPS++ User Group (NAUG) that was charged by the AIPS++/CASA project and representatives of the ALMA and EVLA computing groups to review the current state of the CASA interface and make recommendations for the future development directions of the user interface and documentation and of the package look and feel from a user perspective. The UIWG meet from 27–31 March 2006 in Socorro. This report presents a number of recommendations for the project, along with some suggestions for prototype tasks, and some discussion of unresolved issues.

These recommendations address the usability and presentation of functionality of the software to the user. They fall in the general areas:

- Parameter Interface
- Tools & Tasks
- Documentation
- Package Distribution

The look and feel of the interfaces, the manner in which users can be expected to interact with the package, the scope of the documentation, the organization of the package, and the means for the user to obtain the package were addressed. In addition, some recommendations for specific features and functionality were made where thought important. Although it is difficult to capture the breadth of the recommendations in a few points, the general key issues can be summarized thusly:

- There should be a parameter interface that lets user set the parameter values for a function and execute it. This interface is expected to be similar to that found in standard packages such as AIPS, Miriad, and IRAF.
- The interface to all functions, tasks and tool methods alike, should be the same, and provide the same level of error handling, and inline documentation. Users should not have to learn multiple ways of running things inside the package, be they novice or expert.
- Parameters, tasks, and tool methods should be named and organized in a consistent and intuitive manner (expected to be understood by astronomers) throughout the package.
- Documentation must be complete, accurate, readable, useful, and uniformly obtainable for all tasks and tools.
- Beyond the functionality provided by the CASA toolkit, tasks should be provided that enable users to more easily carry out common and important operations on their data.

The specific recommendations address these issues in more detail. Some of the high priority key tasks are also presented in the report, along with a proposed parameter interface organization scheme.

The panel was naturally not able to come to consensus on all points debated, particularly where personal look and feel preferences and complexity of package use were concerned. In addition, this panel was drawn

for logistical reasons from NRAO staff, and thus is unlikely to represent all potential users of ALMA and EVLA. Finally, there was insufficient time to explore all the options and to draw up entire lists of tasks and parameters. Thus, it is expected that over the next couple of years the UIWG will continue to monitor the progress of the CASA package and user interface, and that exposure of the package to a wider group of users outside NRAO for input on these issues will occur.

In summary, the UIWG was pleased with the progress to date on the CASA package, and looks forward to seeing working user interfaces, complete documentation, a well-organized task and tool kit, and a set of initial tasks. We hope that the project finds our recommendations pertinent and and our discussions helpful.

2 Background

A subgroup of the NRAO Aips++ User Group (NAUG) met at the Array Operations Center in Socorro the week of 27–31 March 2006 to discuss issues related to the user interface of the aips++ package as it transitions to become the Common Astronomy Software Applications (CASA) environment under a new Python-based framework. This User Interface Working Group (UIWG) was charged by the Aips++/CASA Project Manager and the ALMA and EVLA projects to review the current progress to improve the usability of the software in the new CASA framework and to provide guidance to the project on future developments in this area. The UIWG consisted of 8 NAUG members from the AOC and CV sites: Walter Brisken (AOC), Crystal Brogan (CV), Ed Fomalont (CV), John Hibbard (CV), Steven Myers (AOC), Frazer Owen (AOC), Michael Rupen (AOC), David Whysong (AOC). Also participating was Eric Greisen (AOC), who provided insights and advice from the perspective of the developer of AIPS, which is currently the flagship astronomical image processing software at NRAO and is a de-facto standard within the radio interferometry community. Debra Shepherd (AOC) was unable to participate during the bulk of the meeting, but attended the final day of deliberations.

In Joe McMullin's UIWG introduction four areas of concern for the project were outlined, that he wished addressed by the UIWG:

1. **CASA Interface**
2. **CASA Tools & Tasks**
3. **CASA Documentation**
4. **CASA Distribution**

In addition, the following specific topics were suggested for discussion by the group (presented here along with the actual topics discussed during the week):

IPython Interface	—	capabilities, CLI environments, GUI forms
Tool Migration Priorities	—	toolkit rationalization, reorganization, porting schedule to casa
Documentation	—	use of wiki, inline help, online help, maintenance
Logging	—	scripting, formats, history
Task Definition	—	priorities, scope, parameters, defaults
Task/Parameter Handling	—	CLI interface, hierarchical menus, "smart" defaults
Qtviewer	—	progress, priorities, features.

The NAUG has been involved since 2000 in monitoring the progress of the aips++ package towards fulfilling user requirements of ALMA and EVLA and testing and evaluating its performance on VLA data. In the 2003 NAUG audit of the aips++ package for VLA processing, the NAUG established three criteria for the fulfillment of requirements:

Functionality	—	whether the tool works, accuracy
Usability	—	ease-of-use, speed
Documentation	—	clearness, accuracy

The UIWG discussions centered around the *usability* of the package as enabled by the interface, tool and task organization, and documentation. However, the *functionality* of the software, particularly in how it is presented to the user and in what manner it is expected to be used, impacts the priorities for task selection. The extent to which the aips++ software historically failed to provide well-documented usable functionality contributed towards the 2002 reorganization of the project, and its current reformulation as the CASA package after implementing the plans presented to and the recommendations of the 2003 Aips++ Technical Review. The aim of the UIWG is to guide the project in the setting of the look-and-feel of CASA so as to gain user acceptance after this critical transition.

In the discussions, comparisons were made with existing software packages. These included the radio astronomy packages AIPS, Miriad, and Difmap. We also considered Pyraf as an example of existing software (in this case the optical reduction package IRAF) ported to work under Python.

2.1 Suggested Priorities and Timescales

We adopt the 2 dimensional priority scheme used for the EVLA User Requirements documents, which encompasses both the importance and the timescale for a particular requirement.

The importance can have the following values:

1 = essential
2 = important
3 = desirable, but not critical

It is intended that all Priority 1 items must be present and work with high efficiency and usability. It is expected that the package interface will be widely deemed unacceptable if Priority 1 capabilities are missing or poorly implemented. Priority 2 items are highly desirable, and will add enough value to the package interface that these should be accommodated if at all possible. It is expected that most, if not all, Priority 2 items will be implemented in some manner also. Priority 3 items should be considered for add-on options, upgrades or future development.

The time-frame of deployment is not matched to either the ALMA or EVLA timescales in those SSR documents, but to an expected deployment of CASA to the user community, with noted tie-ins to the key projects. The timescale phases are:

A initial CASA beta deployment (2006 Q3)
B second beta deployment, initial ALMA and EVLA commissioning (2007 Q3)
C mature CASA deployment, full commissioning support for ALMA and EVLA (2008 Q3)
D full user support for ALMA and EVLA (2010 Q1)
E "eventually" sometime after completion (ongoing)

For some of the timescale **A** and **B** targets for which we want to see and test a prototype before deployment we use timescale codes of **A+** and **B+** respectively.

2.2 Glossary

The following are terms used in this document and their intended meanings. We do not guarantee that these are the official definitions for the project.

package — Usually, the software package being discussed. In the case of CASA, what is available to the user upon entering the IPython environment, via `casapy`. In CASA, the tool methods and tasks, plus other available features such as Python commands, special parameters, and procedures.

function — Usually here to mean the generic call to a tool method, task, procedure, or command in order to do something.

toolkit — In aips++ or CASA, the set of object-oriented tools and methods made available to the user.

tool — In the aips++ or CASA toolkit, the name of the base tool to which the actual functional methods below, e.g. the `imager` tool or `calibrater` tool.

method — In the aips++ or CASA toolkit, the method or function that the user will call to do something, e.g. `imager.clean` or `calibrater.solve`. In CASA the tool methods can be accessed via a function call, or using the standard CLPI, and includes full inline documentation and error handling.

task — In the CASA package, the non-toolkit functions available to the user, usually built out of the toolkit at either the C++ or Python level. In CASA the tasks can be accessed via a function call (same as for tool methods), or using the standard CLPI, and includes full inline documentation and error handling.

CLI — Command-Line Interface, a text-based interface to the package (e.g. that can be run on an xterm). In aips++, the most basic CLI is the glish interface, while in CASA this is the IPython environment.

CLPI — Command-Line Parameter Interface, a text-based interface to querying and setting the parameter values and executing functions within the package. These are usually a separate environment to the basic CLI of the package, e.g. invoked via `inp <task>` or similar command.

GUI — Graphical User Interface, a graphical environment for setting parameter values and executing package functions. For example, the `viewer` in aips++ or the `qtviewer` in CASA.

GPI — Graphical Parameter Interface, a graphical environment for the user to set parameter values and execute package functions. For example, the autoguis in aips++, and the epar environment in Pyraf.

inline — referring to help or documentation that can be activated from the command line without a connection to the Web.

online — referring to help or documentation that is posted at the package or affiliated websites or is downloadable.

3 Recommendations of the NAUG UIWG

During the week of discussion during the UIWG meeting, the group considered a number of issues based on the charges to the panel outlined above. There were also debates that ranged outside the specific charges, but that were felt to be of relevance for the usability of the software. The items discussed and decisions or recommendations are given below broken down by topical area. Where appropriate, the recommendations are given in the form of User Requirements, along with priorities and timescales. More descriptive recommendations are presented as descriptive items.

3.1 CASA Interface

UI-1-R1 There must be an environment, which we call the Command Line Parameter Interface (CLPI) in which the user can set parameter values for all available task or tool method parameters, and then

execute the function.

Priority: 1 Timescale: A+

UI-1-R2 The CLPI shall have, at minimum, a mode that works over a simple text terminal window, including an xterm.

Priority: 1 Timescale: A+

UI-1-R3 The CLPI shall include mechanisms to set parameter-value pairs, preferably through simple `<parameter>=<value>` statements.

Priority: 1 Timescale: A+

UI-1-R4 There should be a mechanism in the basic command-line (Python) interface for setting the task and tool method arguments outside the CLPI and without executing the function. Note that the setting of parameters in this manner would be "sticky" as the values would be remembered until reset (or the tool destroyed).

Priority: 2 Timescale: B+

Comments: For example, a mechanism such as in Pyraf for assigning values to method arguments as attributes, such as `<tool>.<method>.<parameter> = <value>` *would be extremely valuable in script writing, and would alleviate the need for many* `set*` *methods as these parameters in these methods could be arguments to other methods which you can set ahead of time.*

UI-1-R5 There should be a mode of CLPI operation where the user can scroll through the inputs with arrow/page keys and set values, perhaps using CURSES (e.g. as in Pine and IRAF epar).

Priority: 2 Timescale: B+

Comments: It was thought that this would be a more user-friendly approach than AIPS or Miriad style CLPI.

UI-1-R6 The CLPI shall have a command or mechanism to execute the tool method or task function, e.g. through a `GO` command.

Priority: 1 Timescale: A

UI-1-R7 The CLPI shall have a command or mechanism to show the list of available parameters and their current values, e.g. through a `SHOW` or `INP` command. In addition, a short description of the parameter purpose and choices (for enumerated parameter values) should be shown.

Priority: 1 Timescale: A+

UI-1-R8 The CLPI shall have a command or mechanism to save the values of all currently set parameters (similar to AIPS SAVE) for the current function, e.g. through a `SAVE` command. The user shall be able to specify the filename.

Priority: 1 Timescale: A

UI-1-R9 The CLPI shall have a command or mechanism to read in the values of previously saved parameter values (similar to AIPS GET), e.g. through a `GET` command.

Priority: 1 Timescale: A

UI-1-R10 The CLPI shall automatically save the parameter values used when last executed using the `GO` facility and there shall be a command or mechanism (similar to AIPS TGET) to restore those values and to optionally save to this default (similar to AIPS TPUT).

Priority: 2 Timescale: B

UI-1-R11 The parameter value `SAVE` and `GET` file or variable storage formats should be such that they are editable by users outside the package, e.g. with a text editor, or a provided or publically available tool such as an XML or database editor.

Priority: 3 Timescale: C

UI-1-R12 The CLPI shall have a command or mechanism to reset all the parameters for the function to their default values, e.g. through a `DEFAULT` command.

Priority: 1 Timescale: A

UI-1-R13 The CLPI shall have a command or mechanism to reset an individual parameter to its default value without affecting the values of other parameters, e.g. through a `DEFAULT <parameter>` command.

Priority: 1 Timescale: B

UI-1-R14 The CLPI shall have a command or mechanism to quit or escape the CLPI environment without executing the function or saving the the current values of the parameters, e.g. through an `ESC` command.

Priority: 1 Timescale: A

UI-1-R15 The CLPI shall have a command or mechanism to exit the CLPI environment without executing the function but saving the current parameter values (as if using a *TPUT* action).

Priority: 2 Timescale: B

UI-1-R16 The CLPI shall have a command or mechanism to provide a basic "info" level of inline help (similar to what is seen in AIPS INPUTS), if the level of information provided in the standard SHOW/INP list is not sufficient, e.g. through an `INFO` command.

Priority: 1 Timescale: A+

UI-1-R17 The CLPI shall have a command or mechanism to provide a more detailed level of inline help (similar to what is seen in AIPS HELP), e.g. through a `HELP` command.

Priority: 1 Timescale: A

UI-1-R18 The CLPI shall have a command or mechanism to provide a highly detailed explanation level of inline help (similar to AIPS EXPLAIN), e.g. through an `EXPLAIN` command.

Priority: 1 Timescale: B

UI-1-R19 There shall be a mechanism in the basic command-line (Python) interface to retrieve "info", "help", and "explain" level help on tasks and methods, e.g. via `help <taskname>`, similar to that in AIPS. If a parameter name is given instead, e.g. `help <param>` since its outside the task CLPI this would be generic help for that parameter across tasks and tools.

Priority: 1 Timescale: A

Comments: Note that the latter requires uniformity of parameters across the package.

UI-1-R20 The CLPI shall output to the package session logger and/or scripter the scriptable function call used upon invocation (upon a `GO` action).

Priority: 1 Timescale: B

UI-1-R21 The CLPI shall have a command or mechanism to output to a user-selected file the scriptable function call that would be enacted upon invocation (upon a `GO`) but does not actually invoke the function, e.g. through a `SCRIPT` command.

Priority: 2 Timescale: B

UI-1-R22 The CLPI should have a command or mechanism to optionally output a scriptable toolkit version of the function call when appropriate (when the task or procedure is a Python script built from the toolkit), e.g. through a `TOOLSCRIPT` command or a qualifier to the `SCRIPT` action such as `SCRIPT/TOOL`.

Priority: 3 Timescale: D

Comments: Some panelists thought this was higher priority. However, it is not clear whether this is of general imporatance. Many tasks will likely be in C++.

UI-1-R23 The interface shall only present to the user the parameters relevant for the task operation. This may mean that the CLPI parameter menu will be hierarchical, with the setting of one or more parameters determining which other parameters are visible. If the menu is hierarchical, it should not be excessively deep (should be as "flat" as possible) to reduce the complexity of the operation.

Priority: 1 Timescale: A+

Comments: Some panelists liked the idea of deeply hierarchical parameters, while others wanted only a single parameter that changed the visible list. See the discussion below on prototype tasks (§ 4). In any event we would like to see some mock-up of this, if possible.

UI-1-R24 Parameter names shall not be case-sensitive inside the CLPI.

Priority: 1 Timescale: A

UI-1-R25 Command and parameter names shall be "min-matchable" in the CLPI, e.g. specifiable by the minimum number of unique characters without necessarily spelling out the entire parameter name. The CLPI shall issue an appropriate error message if the min-match is in fact ambiguous.

Priority: 1 Timescale: B

UI-1-R26 The commands and parameters in the CLPI shall use "tab-completion", allowing the TAB key to complete the command if min-match is acheived, or showing possible options if ambiguous.

Priority: 2 Timescale: B

Comments: This is highly desirable and should be done, but is not absolutely necessary for operation of the interface.

UI-1-R27 There shall be Graphical User Interfaces for interactive tasks and tools. These should include:

UI-1-R27.1 A task or tool for plotting data (an X-Y plotter, currently aips++ `msplot`).
Priority: 1 Timescale: A

UI-1-R27.2 A task or tool for viewing ms data (currently the aips++ `viewer`).
Priority: 1 Timescale: B

UI-1-R27.3 A task or tool for viewing images (now included in the aips++ `viewer`).
Priority: 1 Timescale: B

UI-1-R27.4 A task or tool for editing and flagging of ms data (currently integrated into the aips++ `viewer`).
Priority: 1 Timescale: B

UI-1-R27.5 A task or tool for graphically defining regions, boxes, and/or masks (currently part of the aips++ `viewer`, the `interactivemask` function, and incorporated into `imager.clean`.
Priority: 1 Timescale: B

Comments: It is recognized that porting of the aips++ `viewer` *to* `qtviewer` *in CASA will take some additional time, but it is important that key functions are available soon.*

UI-1-R28 These GUIs may have operations that are not reproducible using the CLI or CLPI (such as some flagging and complex mask drawing operations) but these functions should to the maximum extent possible provide outputs, such as tables or equivalent script commands, that allow later reproduction or application of the action from a script.

Priority: 1 Timescale: A

UI-1-R29 It is desirable that there be a Graphical Parameter Interface (GPI) that provides all the same functionality as the CLPI, such as through a form interface (as in Pyraf).

Priority: 2 Timescale: C

UI-1-R30 The GPI must be operable in a "clickless" manner, via use of keyboard arrow, page, and enter keys, in addition to mouse navigation and operation.

Priority: 2 Timescale: C

UI-1-R31 It is desirable that a GPI provide extra functionality over the CLPI, such as direct access to online help, extra menu options (such as the "wrench" menu in the old aips++ autoGUI).

Priority: 3 Timescale: E

3.2 CASA Tools & Tasks

These recommendations deal with the general nature and features and organization of the tasks and toolkit in CASA. Suggested specific tasks are presented in § 3.5.

The last recommendations in this section are on specific task and tool features and functionality that were thought to be needed. This is not a complete list and is somewhat out of place in this document, but were thought to be of enough importance for the usability of the package that we include them here anyway.

UI-2-R1 The parameters in the tasks and toolkit shall be uniformly named, in that parameters for different functions that mean the same thing shall be named the same (or as similar as possible).

Priority: 1 Timescale: A

UI-2-R2 The tasks and tool methods, and the parameters in the tasks and toolkit, shall be named in an intuitive manner.

Priority: 1 Timescale: A

Comments: It was thought by the panel that two-letter tool names, such as `im` *and* `cb`*, although efficient for typing, were too uninformative.*

UI-2-R3 The tasks and tool methods shall be accessible by the user in the same manner and with the same interface properties.

Priority: 1 Timescale: A

Comments: There shall not be another level of commands which only experts know how to use. All commands should be accessible in the standard fashion, and should be documented and maintained in the same way.

UI-2-R4 The tasks and tool methods shall be organized within the package by (scientific or user) operation performed. One possibility is to organize the package by operational classes, such as:

imaging	—	uv(and single dish) data to images via Fourier Transform
calibration	—	calibration of data, resulting in improved data (or calibration tables for application)
uv analysis	—	analysis of uv data, not via images
sd analysis	—	analysis of single-dish data, not via images
image analysis	—	analysis of images, not involving uv or single-dish data
data handling	—	manipulation of data (uv,sd) including editing and flagging
image handling	—	manipulation of images including transforms and masking
visualization	—	the `viewer`/`qtviewer`, general plotting (those facilities not included in other classes)
simulation	—	where the instrument data simulators live
utilities	—	misc things that are needed (measures,quanta,math)

Ideally, this organization should be reflected in the naming of tasks and tools where appropriate. At the minimum, there shall be inline listings (e.g. via `help imaging`) or apropos that will list the tasks and tool methods that fall within these classes.

Priority: 1 Timescale: A

UI-2-R5 In the tasks, the user should be insulated from carrying out "toolkit maintenance" activities such as tool constructors and destructors, explicit `open` and `close` operations in tools. The user should be concerned only with filling in the information needed to carry out the astronomical data reduction process.

Priority: 1 Timescale: A

UI-2-R6 There should be modes or methods for tasks and tools to output in user-understandable terms the state of the task or tool with regards to the current parameter settings and what would result if the function were executed.

Priority: 2 Timescale: C

Comments: Given the complexities implied by some of the following requirements, such tasks may all benefit from a `STATE` *command (to be optionally executed before* `GO`*) which would, in English, describe the operation to be done, and which could issue various warnings that the user can choose to heed or ignore (e.g. "WARNING: Time-average smearing will be significant beyond 1.4 degrees radius).*"

UI-2-R7 To as great an extent as is practical, the toolkit should be structured so as to reduce the number and obscurity of maintenance operations such as tool constructors and destructors, ms and image `open` and `close` operations, and the calling of extraneous `set*` functions. This might be effected by having single `open/close` methods (e.g. `ms.open`, `im.open`) that open (possibly multiple) ms and images for all default tools that open on those types of data, consolidation of `set*` methods, introduction of a mechanism for setting method arguments as attributes.

Priority: 2 Timescale: C

UI-2-R8 The selection of data in ms and images shall be made as uniform, flexible, and efficient as possible in the tasking and toolkit environments. This should include:

UI-2-R8.1 Method names, parameter names, and meaning of methods and parameters of same name across tools, methods, and tasks, shall be uniform.
Priority: 1 Timescale: A

UI-2-R8.2 The meaning of selection parameters and operations shall be clear and intuitive. The selection of data for restriction purposes (e.g. before calibration) and for transformational mapping purposes (e.g. mapping in ms channels to image channels during imaging) should be distinct and clear.
Priority: 1 Timescale: A

UI-2-R8.3 The ability to easily transfer selection parameters or objects between tasks and tools where appropriate (e.g. within the same operational class).
Priority: 1 Timescale: A

UI-2-R8.4 The ability to set global selections across multiple data sets in the case of multi-ms or multi-image inputs (e.g. to `imager`).
Priority: 1 Timescale: A

UI-2-R8.5 The ability to associate ms or image selections with particular data sets in the case of multi-ms or multi-image inputs (e.g. to `imager`).
Priority: 1 Timescale: B

UI-2-R8.6 The property that a selection, once made for a particular ms or image, would hold until changed.
Priority: 1 Timescale: A

UI-2-R8.7 Selection should only be changed when needed. If selection has not changed, the tasks or tool methods should not spend time re-selecting.
Priority: 1 Timescale: A

UI-2-R8.8 Selection should only be made when needed. For example, tasks or tool methods should not actually carry out (time-consuming) selection operations until a task or method that will use that selection is executed, unless there are good reasons to do so upon the selection specification.
Priority: 2 Timescale: C

UI-2-R8.9 The user shall be able to explicitly choose to make a new ms after selection, e.g. for cases where this will greatly improve efficiency (where selection results in a large reduction in dataset size). The user should still have the ability to associate further actions (e.g. flagging, calibration) on the scion ms to the parent ms through some scope transfer mechanism.
Priority: 1 Timescale: B

Comments: Note that this implies that there are tasks and tool methods to split up data (e.g. AIPS SPLIT). It is expected that with the very large ALMA and EVLA datasets that many operations can be more efficiently done on separate files made from data subsets, and that it will be necessary to then apply results from the subsets to the original data.

UI-2-R8.10 The actual selection implementation mechanism should be transparent to the user and chosen in the most efficient manner available (e.g. virtual ms creation, narrowing of selection after a previous selection rather than redoing a whole new selection, on-the-fly selection during use of selection such as in gridding). Ideally, the user might also be able to explicitly choose the mechanism.
Priority: 2 Timescale: C

UI-2-R8.11 The ability to make and manipulate selection "objects", perhaps tied to or even independent of ms or image objects (from an ms or image server).
Priority: 2 Timescale: C

UI-2-R9 The manner in which channel selection and mapping is specified should be as clear and straightforward as possible. Ideally, this should conform to the way such selections are done in other packages to reduce the learning curve.

Priority: 1 Timescale: B

Comments: It was thought that the channel selection in `imager` *was fairly obscure, and unnecessarily different to that in AIPS or Miriad in parameter naming (see separate recommendation below). For example, use of* `bchan`, `echan`, `nchav` *as in AIPS seems clearer. But we recognize that mapping of ms channels to image cube channels could be complex, particularly for multiple ms cases where the data shapes are different. Note that it is critical to get this right as channel operations will be used frequently in ALMA and EVLA.*

UI-2-R10 The ability to calibrate data weights is essential.

Priority: 1 Timescale: A

Comments: Many further operations, such as modelfitting and analysis, rely on getting the weights correct (and not just relatively correct).

UI-2-R11 FITS image support in the package, particularly in the image analysis class tasks and tools, and in the `viewer`, must be robust. Users should be able to import and process any valid FITS image (particularly WCS compliant images), and if possible the CASA image handling should be forgiving for some deviations from orthodoxy.

Priority: 1 Timescale: B

Comments: It is extremely annoying to not be able to view or process FITS images that other software, such as ds9*, can handle easily. We can think of no good excuse so make it Priority 1 even though its not a show-stopper. Note that NAUG members should point the CASA team to examples of such files so that this can be fixed.*

UI-2-R12 The mechanism for specification of imaging weights should be improved and consolidated. This includes robust parameters, tapers, uvrange (which can be thought of as weighting rather than selection). The implementation of weighting should be efficient and sensible (e.g. computing weights only when needed or changed, virtual columns only when needed, trivial weights should be on-the-fly). As always, parameter and method names should reflect the function and should follow common radio astronomy usage.

Priority: 2 Timescale: B

Comments: It was thought that there were many confusing was of specifying weighting in the `imager` *tool. For example, tapering was in the* `imager.filter` *method, which should be called taper, and only lets you set the output Gaussian image beam size rather than allowing specification of the uv taper (in klambda). Also, there were many robust parameters in* `imager.weight` *dealing with Briggs weighting that seemed strangely named. It was also thought that maybe natural and uniform weighting could be handled as robust weighting only, cutting the number of parameters.*

UI-2-R13 There should be a mechanism to handle the many calibration tables that can be produced during the calibration process. Ideally, these should be easily associated with the ms(s) either by naming convention or by inclusion as special cal-tables within the ms.

Priority: 2 Timescale: C

Comments: There was some worry that the current mode of operation where the calibration tables are external to the ms and the choice of naming and responsibility of bookkeeping is up to the user, while maximally flexible, might lead to too many problems for users more used to the Miriad (internal sub-tables) and AIPS (extension tables) approaches. There was once talk about calibration ms sub-tables, is that still in the plan?

UI-2-R14 A mechanism for the handling of ASCII "boxfiles", e.g. in the definition of clean boxes, fields, or facets, similar to and if possible compatible with those used in AIPS, should be available.

Priority: 2 Timescale: C

Comments: The handling of AIPS boxfiles is a reasonable request. Whether this is the preferred mechanism for defining clean boxes etc. is debatable (mask might be more flexible for complex regions), but the simplicity of boxfiles is compelling. Compatibility with AIPS is a big bonus.

3.3 CASA Documentation

This covers the contents and maintenance of the CASA documentation system (not the mechanisms for using it).

UI-3-R1 All officially supported tasks and tool methods in the CASA package must have a full set of documentation that is correct, useful, and uniform in both inline and online versions.

Priority: 1 Timescale: A

UI-3-R2 All tasks and tools shall have inline "info" level help (similar to AIPS INPUTS), including basic task or tool purpose, description of expected inputs and results, and for each parameter the name, current value, type, and brief description (including options for enumerated types).

Priority: 1 Timescale: A

UI-3-R3 All tasks and tools shall have inline "help" level documentation (similar to AIPS HELP), including the "info"-level information plus further discussion of all choices for parameters and some of the basic uses of the function.

Priority: 1 Timescale: A

UI-3-R4 All tasks and tools shall have inline "explain" level documentation (similar to AIPS EXPLAIN), including all "info" and "help" information, plus more detailed explanations of the operation and use of the function, and possibly some examples.

Priority: 1 Timescale: B

UI-3-R5 All inline documentation shall be available online also, in the form of a User Reference Manual (URM). The URM may contain additional information also.

Priority: 1 Timescale: A

UI-3-R6 There shall be a CASA Cookbook that provides tutorial-level guidance on the use of the package for astronomical processing. This should be suitable for both savvy users of other radio astronomy image processing packages and for astronomers of graduate student level or above in other disciplines to learn to reduce radio data using the CASA package. This cookbook shall be provided online and in a printable format.

There should be at least one comprehensive version of the Cookbook, although there may be separate sections for the reduction of ALMA, EVLA, or VLBA-specific data, or additionally there may be separate versions of the cookbook for these audiences.

Priority: 1 Timescale: B

UI-3-R7 Examples shall be provided online for each task and tool method in the User Reference Manual. These examples should be useful in showing the user how to use the function, preferably in several modes of operation.

Priority: 1 Timescale: A

Comments: It has long been argued that most of the examples in the current aips++ URM are trivial and do not give the user sufficient information to figure out how to use the methods. We do realize that the examples will likely be added to as the URM as the package grows, but keep the timescale to the first "release" to highlight its importance.

UI-3-R8 Use of the GUIs and GPIs shall be adequately documented in the online documentation and cookbook(s). Where appropriate, examples (e.g. screen shots) should be provided.

Priority: 1 Timescale: C

3.4 CASA Distribution

UI-4-R1 Released versions of CASA shall be complete (for functionality required by the projects at particular milestones), robust, stable, and well-documented.

Priority: 1 Timescale: A

Comments: This was one of the biggest problems with aips++, and we cannot afford to further alienate users by releasing woefully incomplete, buggy, poorly performing, and undocumented software. We recognize the need to get some alpha and beta versions out there for testing, but this must be weighed against the backlash from impatient potential users.

UI-4-R2 The CASA package shall be downloadable on the web.

Priority: 1 Timescale: A

Comments: No coasters. Unless for PR purposes (but beware).

UI-4-R3 The CASA package shall be installable using some standard mechanism (e.g. through rpms on linux platforms).

Priority: 1 Timescale: A

UI-4-R4 CASA shall be supported for the standard NRAO and ALMA platforms and operating systems, such as a standard Linux (e.g. RHE4) on Intel-based machines.

Priority: 1 Timescale: A

UI-4-R5 The CASA package must be installable by users without root permission.

Priority: 1 Timescale: B

Comments: This is one of the biggest user complaints heard by the panel from those who have participated in recent tests. Please make this horror end!

UI-4-R6 CASA shall be supported for use under Mac OSX (G4+ series, plus new Intel series).

Priority: 1 Timescale: B

Comments: This has become enough of a de-facto standard among astronomers now that this is Priority 1.

UI-4-R7 All libraries and software for full CASA installation shall be provided with the CASA installation.

Priority: 2 Timescale: B

Comments: There may be reasons to leave out certain things from the installation, for example to allow non-root installations. But it is highly desirable.

UI-4-R8 There should be a simple mechanism for obtaining the most recent, but possibly less stable (e.g. the aips++ `daily` or `weekly` versions for testing purposes. These versions should be installable in the same manner as the officially released versions, and able to co-exist with the released versions.

Priority: 2 Timescale: A

UI-4-R9 There should be a mechanism for advanced users to obtain and install a version of CASA that will allow development (e.g. has source code and a `make` facility).

Priority: 2 Timescale: D

3.5 Suggested Tasks and Package Features

Joe McMullin tasked the UIWG with coming up with a list of high-priority tasks, their parameter list, and defaults. The UIWG did not succeed on all these counts. However, during the week's discussions, a number of prototype tasks, and some ideas on how they should look and feel to the user, were explored. Note that these are only a subset of the tasks that could be usefully implemented and thus serve as a guide to kick off the task definition process. It might be reasonable to ask the ALMA and EVLA SSR committees to help define tasks also.

General recommendations on how the tasks should work and should be organized are given in § 3.2.

To delineate tasks from tools, we use the style `TASKNAME`. This does NOT imply that task names in CASA will be capitalized!

UI-5-R1 `CLEAN` — carry out (clean-based) deconvolution of an ms or multiple ms and produce one or more images.

Priority: 1 Timescale: A

Comments: There was debate on whether `CLEAN` *should encompass other deconvolution methods such as MEM, or whether there should be a separate* `MEM` *task. There was no consensus, but a slim majority thought that it would simplify operation for the user if there were a separate* `MEM` *task, particularly since MEM implementation does not currently offer all the options (e.g. multi-scale) as CLEAN and this could make the parameter hierarchy less transparent. See § 4.1.*

UI-5-R2 `CALIB` — carry out calibration of interferometer ms or multiple ms and produce one or more calibration tables, which can then be applied (outside this task) to the data.

Priority: 1 Timescale: A

Comments: It was thought that a `CALIB` *task (such as in AIPS) where a single pass through the task does one type of calibration (e.g. bandpass, gain, phase-only gain, d-term) was best to start with. It was also thought that application of the calibration tables produced in this task should be separate. It was recognized that a more complex task that carries out the full calibration process in the proper order could be built from this basic task plus the toolkit, but the heuristics would take much work to define and that this bordered on what the pipelines would do anyway (so the pipeline work could be borrowed instead). See § 4.2.*

UI-5-R3 `CLCOR` — for want of a better name (this is the AIPS name), one or more tasks that carry out various data and calibration table adjustments commonly needed in data processing. Will probably be broken into multiple tasks (e.g. `OPAC`, `ANTGAIN`, etc.).

Priority: 1 Timescale: A

Comments: These are extremely useful and could be used right away. See § 4.3.

UI-5-R4 `MSSELECT` — select the ms dataset(s) and apply selection criteria on them for further processing in tasks and tools.

Priority: 1 Timescale: A

Comments: This one was somewhat controversial, but it was generally thought that having a single-point task for ms selection would simplify things for the user, and also remove the burden of carrying many extra parameters in subsequent tasks like `CLEAN` *and* `CALIB`*. Some thought that the more traditional approach (like in Miriad and AIPS) of having selection in each task would be more familiar to users, though it was also noted that Difmap uses separate selection. We suggest that this approach be at least prototyped and tested to see if it will work.*

UI-5-R5 `MAKEMASK` — make a mask file for use in `CLEAN` or toolkit processing, from user-specified boxes, from images, other masks, or interactively.

Priority: 2 Timescale: B

Comments: It was thought this would be a useful utility task that would bring together the fairly large number of methods in the toolkit that do this. Note that some others thought masks should be replaced by boxfiles etc.

4 Tasking and Toolkit Issues

During the week there were a number of discussions and debates about tasking and the toolkit. We present some of these here, along with our attempts to prototype a few tasks. Note that some of these items are reflected in the recommendations of § 3, but are presented here in the context of the larger discussion on tasking.

4.1 A prototype `CLEAN` Task

`CLEAN` has a number of parameters (arguments) as inputs. The difference is that the choice of algorithm (the first argument `alg`) changes the lists of parameters shown in the interactive param setting environment. Note that this first parameter can be reset at anytime during the interactive setting session, which will then readjust the parameter list accordingly (returning any previously set values or hiding but not erasing no longer relevant ones). Furthermore, the parameters are grouped by function. For example, using Joe's prototype "inp" command:

```
inp clean
```

You can also prime the task by selecting the first argument, e.g.

```
inp clean ['wf','mosaic','sd']
```

which should bring up something like:

```
[CLEAN]
------------------------------------------------------------------------
Task to deconvolve (multiple) ms using CLEAN.
NOTE: input ms(s) must have been selected using SELECT task.
------------------------------------------------------------------------

   alg          = 'wf','mosaic','sd'    | Choice of algorithm(s)
------------------------------------------------------------------------
imagenames      :: Names of input/output images
   model        =                       | model image (I/O)
   complist     =                       | component list (I/O)
   mask         =                       | mask image (I/O)
   image        =                       | restored image (O)
```

```
   residual     =                       | residual image (0)
   psf          =                       | psf (beam) image (0)
--------------------------------------------------------------------
setimage        :: Set position and stokes shape of output images
   nx           =
   ny           =
   cellx        =
   celly        =
   stokes       =
   doshift      =
   phasecenter  =
   shiftx       =
   shifty       =
--------------------------------------------------------------------
setchannels     :: Set frequency axis shape of output images
   mode         =                       | Chan mode (mfs,channel,velocity)
   nspw         =                       | Number of spectral windows
   nchan        =                       | Number of channels for each window
   chwid        =                       | Channel width (GHz,kms) per window
--------------------------------------------------------------------
weighting       :: Set controls for visibility gridding weights
   type         =                       | Weight mode (briggs,uniform...
   rmode        =                       | Mode for Briggs weighting
   robust       =                       | Factor for Briggs weighting
   noise        =                       | Noise for Briggs weighting
   ...
   uvmin        =                       | Min uvradius for nonzero weight
   uvmax        =                       | Max uvradius for nonzero weight
   taper        =                       | Taper type (none,Gaussian)
   tapbmaj      =                       | Gaussian taper bmaj
   ...
--------------------------------------------------------------------
clean           :: Set common controls for cleaning
   algorithm    =                       | 'clark'|'hogbom'
   niter        =                       | Max number of iterations
   ...
--------------------------------------------------------------------
wf              :: Set controls for wide-file imaging
   wplanes      = 1                     | if > 1 then wprojection
   nfacets      = 1                     | if > 1 then uv faceting
--------------------------------------------------------------------
mosaic          :: Set controls for mosaicing
   gridtype     =                       | 'image','uv'
   ...
--------------------------------------------------------------------
sd              :: Set controls for single-dish data inclusion
   useac        = F                     | Use interferometer autocorrelations
   ...
--------------------------------------------------------------------
interaction     :: Set controls for interaction
   interactive  = F                     | Interactive clean?
   async        = F                     | Run asynchronously?
```

```
-------------------------------------------------------------------------
commands: GO, SAVE, GET, DEFAULT, ESC, EXIT, SCRIPT, HELP, EXPLAIN
```

In this prototype, it is what is set in param `alg` that determines what is shown. In the first example above, you see sections for `'wf'`, `'mosaic'`, and `'sd'` because those were set as `alg=['wf','mosaic','sd']`. Note that these sections will be missing if you use the default (single-field) mode `alg=''`. For example, if the user were to reset the `alg` parameter, say to only `alg = 'wf'`, then the interface would become:

```
[CLEAN]
-------------------------------------------------------------------------
Task to deconvolve (multiple) ms using CLEAN.
NOTE: input ms(s) must have been selected using SELECT task.
-------------------------------------------------------------------------

   alg          = 'wf','mosaic','sd'    | Choice of algorithm(s)
-------------------------------------------------------------------------
imagenames      :: Names of input/output images
   model        =                       | model image (I/O)
   complist     =                       | component list (I/O)
   mask         =                       | mask image (I/O)
   image        =                       | restored image (O)
   residual     =                       | residual image (O)
   psf          =                       | psf (beam) image (O)
-------------------------------------------------------------------------
setimage        :: Set position and stokes shape of output images
   nx           =
   ny           =
   cellx        =
   celly        =
   stokes       =
   doshift      =
   phasecenter  =
   shiftx       =
   shifty       =
-------------------------------------------------------------------------
setchannels     :: Set frequency axis shape of output images
   mode         =                       | Chan mode (mfs,channel,velocity)
   nspw         =                       | Number of spectral windows
   nchan        =                       | Number of channels for each window
   chwid        =                       | Channel width (GHz,kms) per window
-------------------------------------------------------------------------
weighting       :: Set controls for visibility gridding weights
   type         =                       | Weight mode (briggs,uniform...
   rmode        =                       | Mode for Briggs weighting
   robust       =                       | Factor for Briggs weighting
   noise        =                       | Noise for Briggs weighting
   ...
   uvmin        =                       | Min uvradius for nonzero weight
   uvmax        =                       | Max uvradius for nonzero weight
   taper        =                       | Taper type (none,Gaussian)
```

```
   tapbmaj      =                       | Gaussian taper bmaj
   ...
-------------------------------------------------------------------------
clean           :: Set common controls for cleaning
   algorithm    =                       | 'clark'|'hogbom'
   niter        =                       | Max number of iterations
   ...
-------------------------------------------------------------------------
wf              :: Set controls for wide-file imaging
   wplanes      = 1                     | if > 1 then wprojection
   nfacets      = 1                     | if > 1 then uv faceting
-------------------------------------------------------------------------
interaction     :: Set controls for interaction
   interactive  = F                     | Interactive clean?
   async        = F                     | Run asynchronously?
-------------------------------------------------------------------------
commands: GO, SAVE, GET, DEFAULT, ESC, EXIT, SCRIPT, HELP, EXPLAIN
```

Note that the parameters for `'mosaic'` and `'sd'` have disappeared. They would return if you reset (again) `alg = ['wf','mosaic','sd']` etc. Thus the user has control over what they see at any time without restarting the task.

Comments and clarifications on `CLEAN`:

1. There needs to be a column with the type, although there wasn't room in the above prototype.
2. The only param whose choice changes the further params shown is the first one (alg) for simplicity. One could think of making this possible for others (e.g. weight.type='briggs')
3. We COULD take this further. Obvious tree branches include `weight.type='briggs'` which then shows the `rmode`, `robust`, and `noise` params. It is possible that if one were to graph out this list even for single-field clean we will have to include these branches to keep the param list manageable.
4. Since you can input multiple ms, which may have different channel or velocity mappings, we wanted to divorce the output frequency axis shape choice from the input (which probably is in the `SELECT` task. You could also choose to map from the first ms and then use standard bchan,echan,nchav style params. We made up some params — there is probably a better way to break these down.
5. It would be best if the names here corresponded directly to method arguments in the toolkit, e.g. weighting::`type` maps to `im.weight.type` which might guide the toolkit reorganization.
6. At its most basic, this is just a delineated set of AIPS INPuts.
7. It would be good to restrict paging of the (long) input list, which might move you to a deeper hierarchy.
8. It would be nice to have a command-line IRAF epar scrolling inputs and additionally a Pyraf epar form window inputs (where commands are buttons).
9. The `SCRIPT` command will write the current task call (w/arguments) to a specified .py file, with something like a `SCRIPT/TOOL` option to write the series of toolkit calls to a .py file.
10. There will have to be some "hidden" parameters/arguments not visible in the interactive interface to handle return variables (error, rms, cleanflux, and such) plus to allow writing of .par files when used in script (parinit, parsave).
11. Some "return" variables might appear in the interface if they can be set in a previous run of the task and are relevant for subsequent runs.

4.2 A prototype CALIB Task

An example CALIB task would look similar to that of the prototype CLEAN, maybe looking like:

```
[CALIB]
------------------------------------------------------------------------
Task to solve for calibration tables.
NOTE: input ms(s) must have been selected using SELECT task.
------------------------------------------------------------------------

   what         = 'g'                   | Type: 'g'|'t'|'d'|'b'
------------------------------------------------------------------------
setapply        :: Calibration to apply before solving
...
------------------------------------------------------------------------
commands: GO, SAVE, GET, DEFAULT, ESC, EXIT, SCRIPT, HELP, EXPLAIN
```

Comments on CALIB:

1. Note that this task really is equivalent to an AIPS CALIB or BPASS or PCAL (though writing a table not putting in AN table) and only does one, outputting a table. There needs to be another task, maybe CALCORRECT to apply calibration if desired (or do on-the-fly in imaging?).
2. Since we envision that tasks have the same interface as tool methods, tasks can also be built from tasks as well as the toolkit (mix and match). For example, multiple CALIB calls can be assembled into a guided full calibration task. maybe AUTOCAL. But this gets dangerously close to a pipeline and maybe is best left to the pipeline or its emulator in the offline package.

   NOTE: Since the pipeline was being developed (and paid for) it seems inefficient to recreate it in our tasks. The finite programmer time might be best spent on the levels distinctly between pipeline and toolkit (not too close to either).
3. It was also suggested that one should move autoflag up to the task level as AUTOFLAG and keep flag or flagger (including auto options) at the toolkit level. Note this mostly behind the scenes as whether its a task or tool should be invisible to the user.
4. While talking about calibration, it was apparent that AUTOFLAG and CALIB or AUTOCAL plus voluminous plot outputs are insufficient and there needs to be some sort of SNIFFER (using VLBA terminology) to summarize relevant info for diagnostics. An example suggested was some sort of CALVERIFY task, that attempts to check the calibration using some basic heuristic and notify the user if there are obvious trouble areas. This could be combined (in a procedure) with a separate calibration-based flagging task.
5. Is there a better way to organize calibration tables, which now are standalone tables with user-supplied names? Associate in/with ms? Should we use a default naming (e.g. keyed off of ms name)?

4.3 Possible CLCOR functions

One hybrid task that should be considered is the equivalent of the AIPS CLCOR and UVFIX. These are tasks which apply corrections to the visibility amplitude and phase for antenna position errors, source position errors, additional extinction corrections, etc. UVFIX actually changes the visibility data, CLCOR

produces calibration tables. The use of CLCOR and UVFIX is more common for VLBA data than for VLA data.

How should these correction tasks be implemented and accessed in CASA? Should they be applied to the ms main table columns, ie. 'DATA' to 'CORRECTED_DATA' like UVFIX, or should cal tables be made for each type?

A NEW WRINKLE for these methods is that the ms table needs modification. For example, if you change the position of a source (for example, an incorrect source position was used for a calibrator), how should the table with the source positions be modified? Updated? Duplicated with the new position? What happens if you make a mistake and want to redo the source position shift and also keep accountability?

CLCOR and UVFIX have the following options, separated for convenience into 'general' and 'specialized' ones. These opcodes have been adhoc additions over the years, and some of the methods could be combined and a few outdated. Clearly each method (eg, OPAC) would see a small number of input parameters in CASA however it is implemented.

```
OPCODE: General
---------------
POLR: R-L phase difference
PHAS: RATE: Change phase and rate of an antenna
CLOC, Antenna residual delay
OPAC: Change atmospheric opacity
GAIN: Polynomial form of gain vs elevation
ADEL: ATMO, TROP:  Various atmosphere phase corrections
PANG: Parallactic angle correction (in casa already)
IONS: Apply ionospheric phase for input GPS data
ANTP, ANTC; Correction antenna position and/or source position
UVFIX: Recompute u,v,w, phases accurately for
      epoch change, position change, antenna change, time-offset

OPCODE: Specialized
-------------------
PONT: Gross pointing error for object a field center
PCAL, PCFX: Specialized MKIII VLBI phase cal entries
SBLD, MBDL: Specialized MKIII VLBI delay entries
SSLO: Very VLA specific for non-identical IF pairs
ANAX: Antenna axis-offset term.  Should be part of ANTP.
SUND: Gravitational bending of solar-system objects
```

4.4 An analysis of the ms tool

There is a huge amount of functionality within the ms tool. From a user point of view making several tasks from this would be useful. This would allow much of the lower level stuff to be hidden to avoid confusion and assure that the ms is opened with correct parameters.

Functionality that the user needs to see, perhaps in the form of individual tasks, somewhat categorized with important parameters (and sensible names) shown:

```
Importers
~~~~~~~~~
```

```
fitstoms
infile [No default]
outfile [Default = <infile>.ms, with the .FITS removed]
[obstype -- if this parameter is important then it needs to
 explained]
sdfitstoms
infile [No default]
outfile [Default = <infile>.ms, with the .FITS removed]
```

In the packaging I would add the VLA, ATCA/RPFITS, BIMA, ... fillers to this list.

```
Exporters
~~~~~~~~~
tofits
infile [No default]
outfile [Default = <infile>.FITS, with .ms removed]
column [Default = 'CORRECTED_DATA']
standard data selection params
tosdfits
infile [No default]
outfile [Default = <infile>.FITS, with .ms removed]
standard data selection params

information
~~~~~~~~~~~
lister
infile [No default]
column [Default = 'CORRECTED_DATA']
standard data selection params
summary
infile  [No default]
verbose [Default F] (Or perhaps use prtlev?)
history
  * Note -- better if this same task applied to ms or im
infile [No default]
starttime [Default = -1]
stoptime [Default = -1]

ms management
~~~~~~~~~~~~
split
infile  [No default]
outfile [No default]
column [Default = 'CORRECTED_DATA' (Note -- different from toolkit) ]
standard data selection params
standard averaging params
concatenate
infiles [No default, -- ideally accept any number of input files]
outfile [No default]
freqtol [Default = '1Hz']
dirtol [Default = '1mas']
```

```
processing
~~~~~~~~~~
continuumsub (was uvlsf)
  * Note1: this "feels" out of place in the ms tool
  * Note2: Should operate only on one spwid at a time!
infile [No default]
spwid [No default -- not not array, but single spwid!]
fitchans  [No default]
fitorder  [Default = 0]
solint [Default = 0.0]
mode [Default = 'subtract']


[standard data selection params]  (All options are ANDed together)
  * Note: defaults are all empty, ie no deselection
starttime
stoptime [Should support a delta time here, ie +10m]
antenna1
antenna2
uvmin [Note -- either in km or klambda]
uvmax
spwid [Vector of ints]
fieldid [selection by field number]
source [selection by source name]
channels [Vector of ints, ranges]
stokes
taql

[standard averaging params]
chanave [Default = 1, accept -1 for full averaging]
timeave [Default = 0.0]
uvave [Average based on UV cell sizes]
```

5 Questionnaire

A set of questions were posed by the Project Scientist at the start of the UIWG session, and answered by the panel at the end of the week. These are presented here to illustrate the thinking of the panel on these issues.

5.1 Command Interface

Background: the ALMA and EVLA SSR docs require us to provide interactive command-line interfaces (CLI) and optionally some graphical user interfaces (GUI) to the package. We want to get the look-and-feel right to let the user (of any level) process their data most efficiently.

1. What part(s) of other package interfaces should we import for ours (e.g. AIPS, Miriad, difmap, IRAF, Pyraf, IDL)?

   Answer: We liked bits of IRAF (command-line epar), PYRAF (form epar), AIPS (simple INPuts), MIRIAD (short param lists, os commandline execution of tasks).

2. Is our basic CLI model right? e.g.

   (a) enter interface for tool/task
   (b) query/set parameter values
   (c) save/get parameter values (from/to file)
   (d) default parameter values
   (e) write out script
   (f) get some help (see below)
   (g) execute task
   (h) escape (quit) without doing anything or saving state
   (i) exit (quit) saving parameters

   Answer: Yes

3. What method of parameter setting is desired? e.g.

   (a) `<param> = <value>`
   (b) scrollable menu
   (c) `<tool>.<method>.<param> = <value>`

   Answer: Yes, (a) at the minimum, (b) if possible, plus a form GUI (see below). Also (c) would be very useful.

4. Should the tool and task interfaces be the same?

   Answer: Yes

5. Should hierarchical parameter lists be supported?

   Answer: Yes, at least with the first choice (or a few choices) limiting the shown parameters for subsequent processing (see our example tasks).

6. How do we best accommodate users of differencing expertise levels (novice, intermediate, expert) and needs (casual user, frequent user, heavy user)? Do we need different look/feel and if so what should these be?

   Answer: Make the tool, task, and param names intuitive as possible, order and organize them sensibly, and provide good inline help. But do not have special super-simple tasks etc (but can have procedures like VLAPROCS). Have the same look and feel across the interfaces.

7. Does the CASApy interface previewed here and in the latest ALMA test look to be on the right track? If not, what should we be doing instead? If so, what is right and what can we do better?

   Answer: Yes, if we get simple `param=value` *setting and modulo the epar interfaces that are being worked on. And be sure to get all the inp/help/explain level inline documentation into it uniformly! We are also curious how the paging of long parameter lists will work (command-line IRAF epar might help there).*

5.2 Graphical Interface

1. Some GUIs are necessary, e.g.

   (a) image/data display (viewer)
   (b) data plotting (msplot, plotcal)
   (c) graphical parameter setting (clean boxes/regions)
   (d) data flagging (in viewer and msplot)

   Are the current tools for these on the right track? What would the users like to see? Are there better models out there? Should these be simple or have extensive functionality (e.g. viewer)?

Answer: The viewer (in spite of some painful features) is well-liked (even loved) and we look forward to getting a full qtviewer. Some thought that limited use of external stuff like ds9 could be good, as long as format (FITS images) conversion were supported.

2. Is there a role for GUI parameter setting (forms, tool managers)? Should the package provide a uniform GUI interface equivalent to the CLI?

   Answer: Yes. The Pyraf epar-like form was thought to be desirable. It was strongly felt that "clickless" navigation through up/down arrows and page up/down and tab was necessary.

3. Should we provide extensive capabilities for custom GUI development by the user or is Python enough?

   Answer: No opinion.

4. What bits can or should be farmed off to standalone apps (e.g. the Miriad model)?

   Answer: We need our own viewer and x-y plotting. Can use ds9 for some images.

5.3 Tools and Tasks

1. Do we have the models and definitions right? e.g.

   tool method — bottom (fine-grain) level of functionality, important for scripting and tool/pipeline development. In c++.
   task — a more coarse-grained bundling of functionality akin to AIPS and Miriad tasks, used to carry out basic or commonly used sequences of data processing operations with astronomical knowledge built-in. Most likely in c++ or Python. Obeys standard interface with error handling and help/docs.
   procedure - a Python script that runs a sequence of tool and/or tasks. Can be provided by project or user developed. Does have a common interface

   Answer: See above. Was thought that the toolkit was the toolkit and that anything else that had the same interface and support level was a task. Procedures would also have some minimal interfacing and documentation, and probably were the sort of thing users might write (besides just Python scripts). Some also asked about IRAF style "packages" (which in aips++ were categories of tools).

2. What should be done to rationalize the toolkit (the fundamental level beneath the tasking interfaces)? These include:

   (a) make sure parameter names and usage are consistent across toolkit
   (b) consolidate `set*` methods, particularly to contain common params
   (c) `setdata` (perhaps renamed `selectdata`) uniformity (maybe through ms server?)

   Answer: YES YES YES! Maybe go further to make sensible method and argument names and organization to make task building easier.

3. What is the list of key tasks to work on first? What level of combination is needed (e.g. imager+calibrater for selfcal)? What are the parameters for these? What are the defaults?

   Answer: See above. SELECT, MAKEMASK, CLEAN, MEM, CALIB so far. Will take more work to get exhaustive param list and defaults, and to some extent requires toolkit rationalization first.

4. Is extensive feedback from tasks (e.g. return variables for pipelines) needed?

   Answer: Some feedback necessary (error codes) plus some possible return vars, but pipeline will probably use special toolkit methods etc. So what?

5. Do we have the interface models right (see above)? Are there reasons to have task interfaces distinct from tools or procedures?

   Answer: Yes. Task should look like tool.method calls to the user, who doesn't care whats beneath the hood. And no, use same interface.

6. Should the tasking model be more "data-centric" (e.g. you choose a dataset to work on up top and the tools/tasks follow, perhaps using an "attach" command)?

   Answer: Don't like open/close of ms in similar tools, so a SELECT task looks to be very helpful. No one liked "attach" BTW.

7. Are global variables desirable? What variables should be pre-defined as global (if any)?

   Answer: This was controversial among the panel, some thought this might be an easy way to pass params between tasks and set working ms lists etc., but other wanted to avoid the danger by using other ways of setting these (e.g. SELECT task). Could be offered as set of pre-defined global variables.

8. Should parameter lists for task and tools be hierarchical (e.g. setting a particular param=value implies a new set available) or flat (like AIPS and Miriad)?

   Answer: Yes, at least at one level (first argument). It was worried that too many levels of hierarchy would make it too hard to use. Some wanted deeper trees.

9. What levels of "stickiness" for parameters local to tasks is desirable? Are all params sticky (until reset) or does the user specify which ones are sticky?

   Answer: Sticky within subsequent runs of a task. Ability to save/get params. Some "global" state setting (SELECT task). It was hoped we could get command-line access to task or tool.method params via attributes, e.g. im.clean.niter = 100 and these could be sticky. Command line tool.method(param1=value1, param2=value2) not sticky.

10. Can the open/close (e.g. of ms in tools) be hidden from the user sensibly yet still not hurt efficiency and flexibility for large datasets?

    Answer: Yes, via a SELECT task, maybe in combo with a metatool/bigDO (im+cb+ia+ms+...) for the default tools in the casapy session.

5.4 Documentation

1. What are the levels of documentation needed? e.g.

   (a) quick tool/task info (e.g. INP)
   (b) more extensive help (e.g. HELP)
   (c) detailed explanations (EXPLAIN)
   (d) Cookbooks
   (e) Getting started docs (or merge with Cookbook?)
   (f) user programming docs
   (g) detailed programmer docs

   Answer: (a),(b),(c) absolutely critical for users. A good Cookbook is important and the current one is looking good. Merge (e) with (d). The (f) as User Reference Man could be useful but if (a)-(c) are good not as necessary. And (g) is for the project not users.

2. What (inline) help is needed at the CLI?

   Answer: INP level in interface or default query (?). HELP, EXPLAIN via command or button. Possibly the option to drive a browser to a webpage but I'm not fond of that...

3. What should the (online) documentation contain?

   Answer: Cookbook very important. Good User Reference Manual consistent with and including inline help (a)+(b)+(c) should be available (maybe built from inline). Want good examples also.

4. Who needs to create and maintain what documentation? (Note — there is currently no budget provided to the aips++/casa group that will allow extensive user documentation beyond basic interface info.)

Answer: A dedicated Documentation Librarian / Editor will be required. Will need dedicated scistaff input, but help will need to be written by someone very familiar with the package likely at programming level. Will be a challenge to projects to provide the necessary resources.

Comments: NOTE: John jotted down 0.1 FTE (1/2 day /wk year 'round) from NAASC for this. I would hazard to guess this is probably more than a factor of 2 too low, presuming new code is being written and new observing modes being implemented at a reasonable rate even through 2016.

5. Is there a role for consensual community documentation (e.g. wikipedia)?

   Answer: Possibly, but controversial. Good inline documentation is most critical and it is not apparent how that could be done via wiki-fu. However, it was recognized that there MUST be a way to get the documentation out there in a timely manner, and a good wiki (not twiki) if carefully controlled (group access only) could help.

5.5 Other Issues

1. Distribution (is this working for people?)

   Answer: The rpm installation is ok for now, seem to work mostly (some hiccups). Really want non-root install!

2. Known Issues (existing stuff that doesn't work right that had better be fixed)

   Answer: Needs more thought — stay tuned. Do know of issues with reading of FITS files into viewer and image tool where other software does fine. There are some issues with viewer (e.g. Crystal has some things she does not want to be inherited by the new `qtviewer`*.*

3. Existing Schedule/Plan (Query to Projects: is this schedule ok?)

   Answer: Not our call, but we do urge that the top priority issues be dealt with before an official release beyond the testers.

6 Summary and Conclusions

The UIWG reviewed the current state of the transition from aips++ to CASA and found that progress has been good, and that the IPython environment in CASA will provide a usable basic user interface to the package. In this report the UIWG has made a number of recommendations on what we would like to see in terms of command-line and graphical parameter interfaces, tasks, the task and toolkit organization, documentation, and package distribution. These mostly fall under the categories of:

1. Provide a uniform parameter interface to the tasks and tools that will allow more interactive setting of parameter values, with access to full inline documentation and efficient and effective presentation of the needed functionality to the user.
2. Provide user-friendly and uniform naming and organization of tasks and tool methods, so that use of the package is more intuitive to astronomers.
3. Provide complete, accurate, and up-to-date inline and online documentation, including tiered levels of help (info, help, explain), a complete user reference manual, comprehensive and effective examples, and cookbooks.
4. Provide a robust and straightforward mechanism for obtaining over the internet and installing the CASA package on standard supported platforms that is suitable for use by non-experts who may have have recourse to a well-staffed computing system administration group.

Many of the recommendations reflect the panel's views on the best look-and-feel for the package and its user interface, and thus are necessarily more descriptive and less objective in nature. Evaluation of implementations of these recommendations by the project will therefore also require consultation and feedback with the UIWG and the NAUG, as well as the ALMA and EVLA project representatives. However, the panel could not come to consensus on all issues and therefore some prototyping and further user testing will be required to ensure that the user interface(s) are indeed usable and functional. The UIWG therefore expects to stay involved in this process over the next couple of years and looks forward to seeing the results of this collaborative endeavor.

The UIWG panel was able to agree on some basic philosophical points regarding the interface and package organization. The highest level of these were:

- The actual system should be as "flat" as possible. All commands – tasks, tool methods, procedures – should be available directly, without (e.g.) entering a different environment. Note that this does not refer to parameter setting (CLPI, GPI) interfaces, which can be entered by user choice, but which of course must be available uniformly to tools and tasks, or to possible hierarchies of parameters presented in interfaces. It is intended that there is no need for the user to descend into a package hierarchy or to switch modes to use the "expert" toolkit, for example.
- All tasks and methods should look alike to the user: one should enter parameters in a similar fashion; those parameters should have a consistent naming convention; the commands should be similarly documented and maintained; and all commands should be usable in the same way in scripts. The user should not for instance care whether the implementation is through python, C++, or FORTRAN, or even just a simple script+interface.
- There should NOT be another level of commands which only experts know how to use. We do not say such commands should not exist — there are commands in any package that only the cognoscenti use or appreciate. But those esoteric commands should be accessible in the standard fashion, and should be documented and maintained in the same way.
- This "flat" approach for tasks and tools is IN CONTRAST to the required organization for the documentation, which must be intelligently hierarchical as well as flat. Information should be grouped in areas that make sense to the astronomer, for instance rather than according to the tool in which a method resides or who wrote the task, and there must be ways to find the right task or tool by knowing this operational hierarchy.

We finish by noting that our point of departure for the discussions and recommendations was that the interface and tasks be built on top of the toolkit, and that access to the toolkit (which we hoped itself would be somewhat reorganized) would be presented to the users uniformly with the more astronomically organic tasks. It was thought that the flexibility of the toolkit in carrying out more complex data reduction operations and in building scripts would be invaluable. On the other hand, some in and outside the panel felt that the look and feel of the toolkit, particularly as presented in the `tool.method` approach to function calls, was part of the historical problem with aips++ and thus future user acceptance would require abandonment or at least hiding of the toolkit, and that tasks should be the only part of the package that (most) users need to access. For example, there was some support for extending the "uniformity" clause of the above-mentioned philosophy to include the details of how task and tools must be used (e.g. the tool methods might have names like `<tool>.<method>` and might require some extra care in the order in which they are executed by the user). A strict enforcement might add the stricture:

- Assuming the "normal" CASA approach conceals objects (for instance) from the user, intelligently handles opens/closes so the user doesn't have to pay attention to that, etc., the "esoteric" commands should share this approach. One should not have to learn another level of complexity in style/syntax to write expert scripts — one should be able to build directly on the basic system. In this same vein the scripts (and pipelines) we provide as examples should look like what one types at the command line, not some more advanced form of OOP.

Thus, it was suggested that the toolkit, or at least tools, from the user point of view, should vanish, and that methods, again from the user point of view, should be callable directly with any requisite instantiation, opening, closing, etc., hidden. Maybe this means there are a large number of tasks corresponding to the methods which are written to wrap the toolkit, e.g. check whether the required tool is already open correctly, open it if need be, run the method, and return. Not all agreed that having the object-oriented toolkit violated the intended uniformity. The panel was also reluctant to make this recommendation, as it has some significant implications for the project, such as whether the resources needed to port all necessary parts of the toolkit to a task structure could be found without disrupting other key targets. Automatic wrappers were suggested, but might not be sufficient. The correct task model to adopt for an entire re-packaging was unclear (AIPS, Miriad, GILDAS, hybrids?). In addition, it was not clear that the look and use of the toolkit was a substantial problem — if the package was robust, functional, well-documented, and performed well, then it is probable that users could adapt to and even embrace this somewhat non-standard approach to user software (note that this is in line with the popularity of the user of exploratory tools such as IDL and Matlab in the astronomical community). The best we can suggest on this regard is that the project try to make the toolkit and select tasks as functional and usable as possible, and then evaluate whether this will work for users or whether the toolkit must be suppressed or turned into tasks.

The road from aips++ to CASA has been a rocky one. It is clear that for the first decade of the project, aips++ failed to gain user acceptance and confidence. This was for a number of reasons, which have been debated thoroughly and blame assigned in a number of reviews, most recently in the 2003 Aips++ Technical Review. Clear changes were made in the project just before that review, and the project is to be congratulated on the excellent progress made in recent years. Part of blame for lack of user acceptance was levied at the user interface to the package. This working group was convened by the ALMA, EVLA, and the aips++/CASA project in order to not make the same mistake again. The panel feels that the recommendations made in this report will, at least, satisfy a reasonable cross-section of astronomical users. The UIWG recognizes that it cannot necessarily represent the entire cross-section of future users of the package, and that there will need to be some manner of staged release and outside user evaluation as progress is made on the interface and package organization improvements.