EUROPEAN SOUTHERN OBSERVATORY
Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral
Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

Reflex Workflow Development Guide

Author: Cesar Enrique Garcia Dabo (cgarcia@eso.org)
Department: Pipeline Systems Department
Version: 1.0
Date: June 19, 2013

Contents

1 Introduction And Scope

`Reflex` is the ESO Recipe Flexible Execution Workbench, an environment to run ESO VLT pipelines which employs a workflow engine (Kepler[1]) to provide a real-time visual representation of a data reduction cascade, called a workflow, which can be easily understood by most astronomers.

It is important that workflows present the overall data flow of a pipeline in a way that is intuitive and self- explanatory. Workflows are complex programs and designing them in such a way that they meet these high-level requirements takes significant planing and effort.

This document is a guide for those who want to write a workflow that uses VLT pipelines, which are based in CPL. Altough not all the components in Reflex are specific to the VLT pipelines most of them are and what we describe here is based on our experience designing some VLT workflows. Using Reflex for other purposes is very limited.

We will present the basic blocks of a pipeline workflow, a step by step procedure to create a simple workflow and some hints on specific aspects of the workflow development. Finally we show some of the guidelines used by the current VLT workflows and a section to transisition from Reflex 1 workflows to Reflex 2.

This guide assumes that you have already some knowledge of Reflex at least from the user point of view. Also, this tutorial does not cover installation issues. It is recommended that you read the Reflex User Manual (TODO), the Kepler user manual (TODO) and maybe some of the pipeline workflow tutorials (currently UVES tutorial and Xshooter tutorial are available) (TODO).

User support for this software is available by sending enquiries to `usd-help@eso.org`.

2 Overview on the development of a Reflex workflow

Developing a workflow is usually not as easy as dragging and dropping some actors in the canvas. In order to avoid some mistakes, we recommend to follow these guidelines, during the development:

- Workflows are most useful for pipeline modularized with an appropriate granularity. Older monolithic pipelines should be broken up into individual recipes. The modularization of pipeline should be done from an astronomers point of view. This means that independent steps which might need to be redone are contained in a single recipe. Intermediate data products should be useful for monitoring of the progress in data processing and for diagnostic purposes. Recipe parameters should be independent from each other, i.e. situations where the setting of a parameter in one recipe implies a particular value for a parameter in a different recipe have to be avoided.

- Collect all the supported observing modes that the workflow should support. A workflow might support different observing modes or calibratiion strategies, altough sometimes it is recommended to develop different workflows for substantially different observing modes.

[1] https://kepler-project.org

In some cases the calibration strategy has different calibration chains, for instance when a given step is optional (flux calibration is an example). It is important to collect all the possible calibration chains in order to design the workflow with those cases in mind. Therefore, the modes to be supported have to be carefully considered.

- It is recommended to start a workflow with a fairly stable pipeline, at least in terms of interface. This means that the pipeline recipes should have well-defined inputs and outputs frames, with corresponding PRO.CATG tags as used by *esorex*. The recipe chain depends strongly on this interfaces and therefore a valid design for some inputs/outputs might not be valid if these are changed. There is however some flexibility, for instance, adding a product created by a recipe which is not going to be used anywhere else by other recipe doesn't impact the workflow design.

  It is also strongly recommended that the list of recipe parameters are well defined before creating a workflow. If the parameters are changed, once a workflow already exists could be tedious and therefore not recommended. Also, the order in which the parameters are defined by the recipe should be kept.

- Reflex workflows are driven by the data files, which are sorted and routed based on the file tags. It is therefore important that the file tags uniquely describes the purpose of a file. For example, if science data and calibration data use different kind of sky observations, these sky observations should get different tags.

- Start designing your workflow with the different calibration chains of the previous step in paper. Rather than starting the workflow design with the Reflex tool itself, it is sometimes better to start a design in paper, so that

- Design the interactive points in the workflow. TODO

- Create a OCA rules file which mimics the relationships outlined in previous steps.TODO. More information about OCA rules in section **??**.

- Create workflow following the layout explained in TODO

- Check TODO

3 HOWTO: Creating a simple workflow

In this section we will show how to create a simple working workflow. Take into account that the complexity of workflow creation depends very much on the complexity of the calibration cascade. In this case we will target a workflow with only two recipes, where one recipe creates a calibration needed by the next recipe.

This step by step procedure will create a workflow similar to the template workflow distributed by Reflex (see section **?**)

1. It is recommended that Reflex is installed using the manual method rather than the `install_reflex` script. See http://www.eso.org/sci/software/reflex/ for details.

2. A CPL-based pipeline has to be created and made it available via esorex. This guide doesn't cover how to create a pipeline or esorex configuration. Please refer to the **?** for more details. For this HOWTO, it is assumed that a pipeline with recipes named `rrrecipe` and `rrrecipe_calib` are visible via esorex.

3. The first thing is to create a OCA-rule file that will be used to classify, group and associate the proper files together.

   The classification part of the OCA rules will look like this:

```
if DPR.CATG like "%SCIENCE%" and DPR.TYPE like "%OBJECT%" then
{
  DO.CATG = "RRRECIPE_DOCATG_RAW";
  REFLEX.CATG = "RRRECIPE_DOCATG_RAW";
  REFLEX.TARGET = "T";
}

if DPR.CATG like "%CALIB%" and DPR.TECH like "%IMAGE%" and DPR.TYPE like "%STD%" the
{
  DO.CATG = "RRRECIPE_CALIB_DOCATG_RAW";
  REFLEX.CATG = "RRRECIPE_CALIB_DOCATG_RAW";
}
```

   This basically specifies that based on some keywords of the main header of the files, the files should be assigned some classification keywords. Reflex will use `REFLEX.CATG` mainly.

   Next, with all the files that have been classified with the same keywords, there is the need to group them and trigger a specific action. This part will look like this:

```
select execute(ACTION_CALIB_IMG) from inputFiles where DO.CATG == "RRRECIPE_CALIB_DO
  group by INS.FILT1.NAME, OBS.ID, OBS.NAME, OBS.TARG.NAME, TPL.START as (TPL_A,tpl)
select execute(ACTION_COMBINE_IMG) from inputFiles where DO.CATG == "RRRECIPE_DOCATG
  group by INS.FILT1.NAME, OBS.ID, OBS.NAME, OBS.TARG.NAME, TPL.START as (TPL_A,tpl)
```

   The next thing is to define the actions, and within the actions, the calibrations or dependencies needed by each of the actions:

```
action ACTION_CALIB_IMG
{
minRet = 0; maxRet = 1;
select file as STATIC_MASK from calibFiles where DO.CATG == "STATIC_MASK";
minRet = 0; maxRet = 1;
select file as IMG_STD_CATALOG from calibFiles where DO.CATG == "IMG_STD_CATALOG";

recipe rrrecipe_calib;
product IMG_CALIBRATED { REFLEX.CATG = "IMG_CALIBRATED"; PRO.CATG = "IMG_CALIBRATED"
}

action ACTION_COMBINE_IMG
{
minRet = 0; maxRet = 1;
select file as STATIC_MASK from calibFiles where DO.CATG == "STATIC_MASK";
minRet = 1; maxRet = 1;
```

```
select file as IMG_CALIBRATED from calibFiles where PRO.CATG == "IMG_CALIBRATED";

recipe rrrecipe;
product IMG_OBJ_COMBINED { PRO.CATG = "IMG_OBJ_COMBINED"; PRO.EXT="tpl_0001.fits";}
}
```

The `minRet, maxRet` keywords specify constraints on the number of matching files. If at least `minRet` files are not found the dataset will be incomplete.

The `product` clause defines the products created by a given action, in order to associate it later (like the `IMG_CALIBRATED` file in the example).

4. Next thing will be to define some directories that will be used bu the workflow. There are three types of directories:

   (a) Input directories. These directories will be scanned by the DataOrganizer using the OCA rules to define the datasets to process. It usually contains the user data and the specific pipeline calibration files.

   (b) Working directories. These directories are used internally by the Reflex actors. The user might have to look at them only for debugging purposes. The tt TMP_PRODUCTS_DIR is likely to grow very quickly in size.

   (c) Output directory. This directory will contain the final reduced data in a easy to browse directory structure.

   The way this is created is via variables in Reflex whose value points to the desired directory. Use the `FileParameter` Kepler standard actor to define these variables. To change the name of the variable, right click on the variable and select `Customize Name`. Figure 1 show how the standard directories implementation looks like.

Figure 1: *Standard setup directories.*

Take into account that this is the standard for workflows but you can use other directories setup for your workflows if needed. However, the working directories must be always be there and with those names, since they are used by several standard Reflex actors. See section 4.1 for more information about the standard directories.

5. Now we start putting actors which represent the real workflow. The first thing is to combine the calibration directory and the input directory in a single input that will be fed into the data organization. For that we put two `StringConstant` actors with the values `$RAWDATA_DIR` and `$CALIB_DATA_DIR`. Make sure that the `firingCountLimit` parameter is set to 1 in both cases. Then we connect the output of these two actors to an `ElementsToArray` actor. Figure 2 shows the result of that.

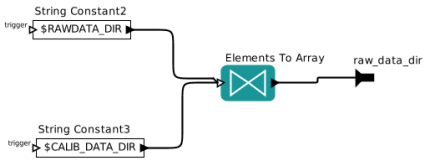

Figure 2: *Directory preparation for the DataOrganizer.*

6. The next thing to do is to connect the output of the array to the input of `DataOrganizer`, i. e., the port `input_data`. The `DataOrganizer` has to be configured with a proper path to the OCA rules. For that, use the parameter `OCA File` of the `DataOrganizer`. The output port of the `DataOrganizer` has to be connected then to the input port `datasets_in` of the `DataSetChooser` and finally the output poty `datasets_out` of the `DataSetChooser` has to be connected to the input port `in` of the `FitsRouter`.

   After all these connections, the workflow supports the creation of datasets using the definitions of the OCA rules, displaying the datasets for selection and inspection and feeding the selected datasets to the `FitsRouter`.

7. The next thing is to setup the `FitsRouter` to deliver the proper data to different connections. Out simple workflow has two main recipes, one calibration and one science. With this scheme, it would be enough to create two channels: one with all the input needed by the calibration recipe and one with science frames. We first create two ports in the `FitsRouter`: `CALIB` and `SCIENCE`. Then, we create two parameters of the `FitsRouter` using the `Add` button in the `Edit Parameter` window called `CALIB_config` and `SCIENCE_config`. By default, the parameter type is generic, so we will have to quote the list of PRO.CATGs assigned to that port. Other option is to change the parameter type to `StringParameter` and then the quotes are not needed.

   The calibration port will just need one type of files: RRRECIPE_CALIB_DOCATG_RAW. The science port however will redirect several types of files: STATIC_MASK and RRRECIPE_DOCATG_RAW. The result is the configuration shown in figure 4. See section **??** for more details.

8. Instantiation of Recipes. How to create a subworkflow. SofSliptter and SofAccumulator. TODO

9. Finally, adding the

3.1 The TEMPLATE workflow

When Reflex is installed using the *install_reflex* scritpt as explained in the Reflex User Manual, you will be presented with a list of pipelines to install. One of them is called TEMPLATE and contains a basic pipeline with a basic workflow that can be used as a template or an example to build other pipelines/workflows. The template pipeline can also be retrieved from this link: ftp://ftp.eso.org/pub/dfs/pipelines/iiinstrument/.

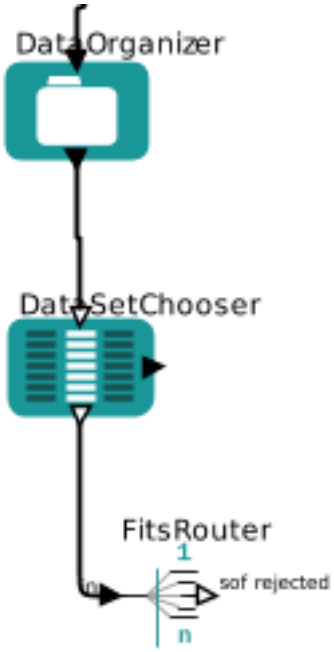

Figure 3: *Connections of the DataOrganizer, DataSetChooser and FitsRouter.*

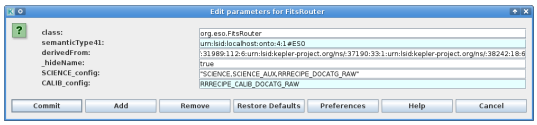

Figure 4: *Confguration of the FitsRouter.*

Figure 7 shows how this basic workflow looks like. The OCA rules delivered with this workflow are also very basic and just contain one level of dependency: a science recipe just needs the result of a calibration recipe.

4 Main Components of a Reflex Workflow

In this section we will describe more in detail which are the main components used most commonly in a Reflex workflows.

4.1 Setup directories

Some actors require that some variables are set in order to work properly. In particular, the *RecipeExecutor* actor requires the following variables:

- `BOOKKEEPING_DIR`: a directory where each pipeline recipe execution will create a subdirectory to use as a working directory. Useful for debugging purposes, since it contains the input sof for each

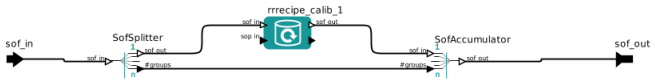

Figure 5: .

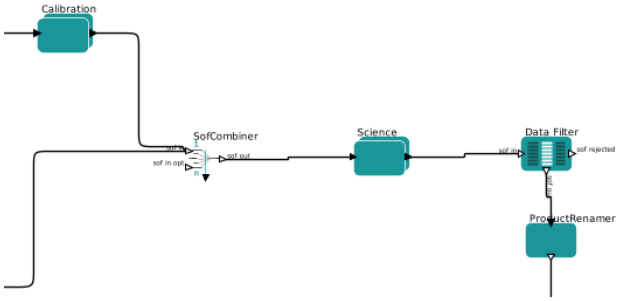

Figure 6: .

call of the recipe, the output sof created, the parameters used and the esorex command executed.

- `LOGS_DIR`: a directory where the recipe logs will be saved. In particular, the esorex log will be stored here.
- `TMP_PRODUCTS_DIR`: a directory where the workflow intermediate products will be saved
- `ESORexArgs`: additional parameters passed to esorex by the RecipeExecuter

It is actually possible to use a *RecipeExecuter* which does not use those variables, defining the corresponding actor parameters. However, being the default parameters, when the actor is instantiated, it requires them to be present. Moreover, it is convenient to have them as a global variable in the main canvas, so that they can be accessed by all the *RecipeExecuter* actors, including the subworkflows underneath.

A very convenient way to setup a workflow is to create a variable that defines a root directory and define the rest of the directories as subdirectories fof that. For instance, create a variable.

TODO. $ROOT_DATA\ ROOT_DATA_PATH_TO_REPLACE$.

Other variables that we recommended to define are:

- `END_PRODUCTS_DIR`: a directory where the workflow final products will be saved. This variable is used by the *ProductRenamer* actor.
- `FITS_VIEWER`: executable used to visually inspect FITS files. This variable is used by the *DataFilter* actor.

4.2 Data Organization

One of the most useful tasks of a workflow is the automatic organization of large lists of files available on the disk. The organization of the data is provided by the `DataOrganiser` actor, which actually uses a OCA file that specifies the organisation logic.

The OCA file can be very complex

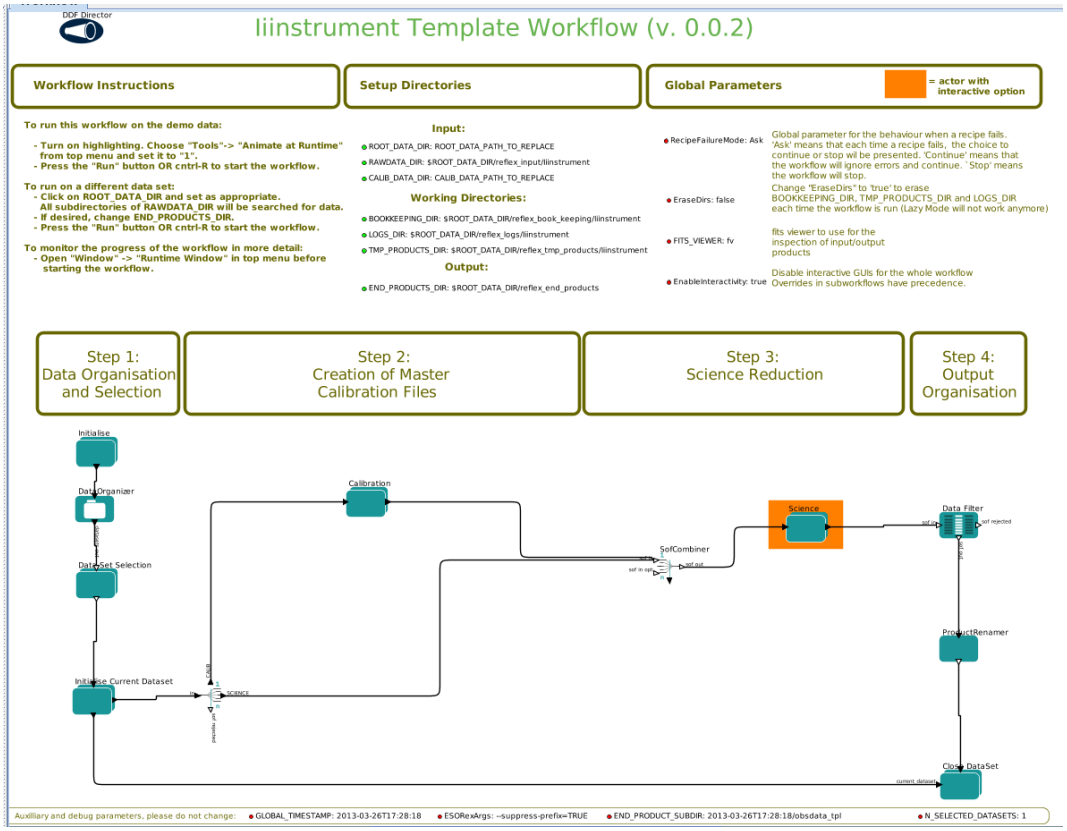

Figure 7: *The layout of the template workflow.*

4.2.1 OCA rules syntax

4.2.2 File Purpose

In Reflex 2.0, each file which is part a Sof carries around a *Purpose*. It represents what a file is needed for. The purpose of a file describes its position in the whole reduction cascade that will make use of the file. For example, a raw BIAS might have purpose MASTER_BIAS/SCIENCE if it's going to be used to produce the master bias that will be used to reduce the science frames.

4.2.3 Executing Recipes

The relations in a Reflex workflows contain XML files that describes a Sof. This Sof are used as input to execute recipes in general. However, through a single relation there could be several files with different purposes. For instance, the input of the bias recipe could have biases used for the flat and biases used for the science. In order to properly use the relevant files, the actor *SofSplitter* has been created. The *SofSplitter*, as its name implies, will create several Sof with the same purposes, one at a time. This way, the *RecipeExecuter* will receive a coherent Sof for only one purpose. The *SofAccumulator*, on the other hand, waits until all the products of the recipes have been produced and joins them all together.

The proper way to use the *RecipeExecuter* will be to place a *SofSplitter* in front of it and a *SofAccumulator* after it. This two actors have to be connected each other by the *# grous* port. Figure 8 shows how to

do this in general.

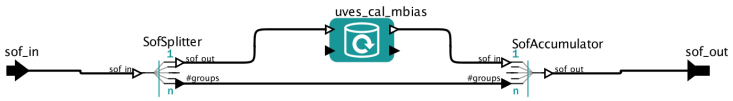

Figure 8: *This figure shows how to properly use a RecipeExecuter actor in combination with a SofSplitter and a SofAccumulator.*

Take into account that in order to help for the readibility of the workflow, it is recommended to place these three actors inside a *CompositeActor*.

One important question is which is the *Purpose* of the products created by a given recipe.

4.2.4 Using the SofCombiner

4.2.5 How to Write Workflows which do not Use Purposes

From the description above it is clear that the *Purposes* are a crucial feature in the design of the workflows. However, if one is not interested in them and just want to process with a given recipe all the files which come from a given relation, it is possible to do it. This is not, however, the recommended way to create workflows.

Take into account that the behaviour might not be what you expect. If for instance a given calibration set has been associated several times in the association tree, you would get duplicated files into your recipe execution. One possible solution for that is to associate calibration files in the OCA rules only in the action which has `REFLEX.TARGET = T`.

4.2.6 Introducing a Purpose-less Recipe in the Middle of a Workflow

There are cases where one is interested in introducing a recipe in the middle of two already existing recipes. In order to work properly with *Purposes*, that change has to be reflected in the OCA rule, possible through an intermediate action. However this might not work, if the new recipe does not have raw frames as an input (this limitation of OCA rules might disappear in the future).

The option is to configure the *RecipeExecuter* with the option *Do nothing* in the parameter **File Purpose Processing**, like it is shown in figure 9.

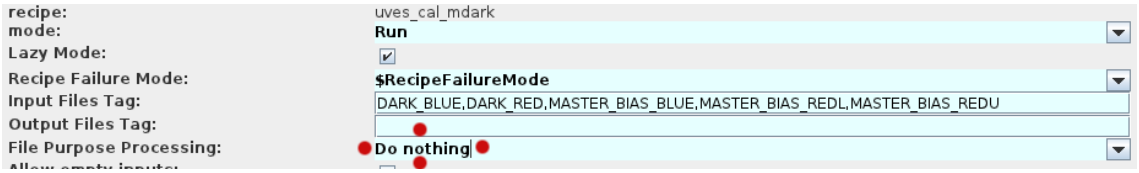

Figure 9: *Configuration of a RecipeExecuter to act as a pass-through recipe.*

4.3 Bookkeeping and Auxiliary Tasks

4.3.1 Initialize Composite Actor

4.3.2 Close Data Set Actor

4.4 RecipeExecuter

4.4.1 Update to a change in the recipe

Sometimes, a change in the recipe interface cannot be avoided and a given parameter is renamed, deleted or added. Reflex stores in the workflow the order of the parameters. For this reason, even a change in the order in which the parameters are defined is strongly discouraged. However, if none of the parameters of the recipe have been changed from the defaults, the workflow probably didn't store any value of the parameters and nothing should be changed.

If a recipe has been changed, the best way to incorporate the changes is simply to delete the current actor and reinstantiate it: Reflex will read the recipe parameters again and will create a new configuration. In other cases, reinstantiating will cause too much trouble and the only change was maybe a parameter renaming. In that case, the workflow .xml could be edited and searched for strings like `<property name="recipe_param_`. Once the right parameter has been found (make sure that the recipe is the correct one: `<entity name="name_of_recipe_1"`), you can change the value of the *Reflex* parameter, which is in fact the name of the *recipe* parameter and the value together.

4.5 The OCA rules and Reflex

Reflex uses the OCA rules of the purpose of classification, grouping and association. The OCA syntax is explained in detail in **??** documentation, and here we will just point out the details specific to the use of OCA rules by the Reflex application.

OCA rules written for other applications (Gasgano, DO, ABbuilder) could in principle be used with Reflex, provided that some additions are included for the rules to work within Reflex. In fact Reflex uses the same library to parse the rules as all the other applications

4.5.1 Classification

For the classification part of the OCA rules, Reflex will use the special keyword `REFLEX_CATG`. Other applications, for instance DO, use the keyword `DO_CATG`, which Reflex will simply ignore. Usually, the value of both keywords would be the same.

Each of the classification entries look like the following:

```
 if DPR.CATG like "%SCIENCE%" and DPR.TYPE like "%OBJECT%" and INSTRUME=="TEMPLATE" ther
{
  DO.CATG = "RRRECIPE_DOCATG_RAW";
  REFLEX.CATG = "RRRECIPE_DOCATG_RAW";
}
```

The *like* operand will match for keyword values which contain the string between percentages. Any Java regular expression can be used.

In contrast, the == operator will do an exact matching of the values of the keyword. Note that it is recommended to match always the *INSTRUME* keyword, since it is a common use case that a user has data for different instruments in the same directory and for a given workflow, only the supported instruments should be matched. The use of *INSTRUME* is recommended also when matching static calibrations in the

OCA rules, using the syntax *inputFile.INSTRUME==INSTRUME*. Even for virtual product associations this is a good practice.

Use _WKF not to match precomputed calibrations in the OCA rules.

4.5.2 Target of the workflow

Reflex uses the OCA rules to define the calibration/association cascade. However, there must be a way to inidicate where the chain should be started. For instance, many workflows aim to reduce science frames, while some other specialized workflows will just create calibration files.

The way to specify the target is using the `REFLEX_TARGET` classification:

```
 if DPR.CATG like "%SCIENCE%" and DPR.TYPE like "%OBJECT%" and INSTRUME=="TEMPLATE" ther
{
  REFLEX.TARGET = "T";
}
```

A workflow can have more than one target, for instance science images and standard star images, although the rest of the workflow has to support both reduction chains, of course.

4.5.3 Grouping and action triggering

Reflex doesn't need any special syntax or additions to the grouping and select commands in OCA. The usual synatx is like

```
  select execute(DARK) from inputFiles where RAW.TYPE=="DARK"
  group by DET.READ.CLOCK,DET.CHIP1.ID,DET.WIN1.BINX,DET.WIN1.BINY,TPL.START as (TPL_A,t
```

Note that the grouping usually contains `TPL.START`, which will group all the images of the same template together, or `ARCFILE`, which will create just one group per file, since this keyword is unique to each file.

The clause *where* can use any of the classification keywords defined in the classification part. If compatibility with other software (DO, ABbuilder), the same keyword used there could be used here.

If a group should contain at least a minimum number of files (for instance a raw bias group should have at least 5 raw biases), then the clause *minRet* can be added before the select *clause*, like here:

```
 minRet = 5;
 select execute(BIAS_BLUE) from inputFiles where RAW.TYPE=="BIAS_BLUE"
  group by DET.CHIPS,DET.WIN1.BINX,DET.WIN1.BINY,DET.READ.SPEED,TPL.START as (TPL_A,tpl)
```

5 Subworkflows

A subworkflow can be created with a `CompositeActor`. To open the subworkflow canvas, right click on the actor and select `Open Actor`. Note that it is handy to use the arrows toolbox to create ports directly for the subworkflow. The name of the ports, however, should be changed in the usual way for the composite actor.

During the overall design of the workflow it should be considered which parts should be grouped into sections, and which parts should go into a subworkflow. As a rule of thumb, most workflows should not

contain more than 10 to 15 actors or subworkflows. Subworkflows should be kept simple and have clean and simple input and output ports. For example, if a recipe or other actor is called several times with different input or in different modes which depend on available data, these actors and the associated logical elements should typically be moved to a dedicated subworkflow.

TODO: Use a director in it? Example: the initialize subworkflow.

6 Interactive Composite Actors

7 Interactivity in the workflow

7.1 Python Interactive Framework

8 General Tips

8.1 Workflow layout

Overall layout The workflow structure has several parts:

- Top area. Contains annotations about the workflow and the main parameters to setup the workflow. Note that not all the parameters of the workflow are set here: each actor has its own parameters that are set individually.
- Bottom area. This contains all the actors that actually execute or perform a workflow action. It is also composed of several parts:
  - Initialisation actors on the left side. These actors prepare the rest of the workflow to start with the data reduction. It includes setup of required intermediate variables, the data organisation, the data selection and the routing of the data. Please note that the data organisation and selection is performed only once, while the routing is performed once for each data set that has been selected.
  - Recipe execution. The middle part of the bottom area contains the actors that execute pipeline recipes. This is basically where the logic of the data reduction chain is implemented. Closing of the reduction for this data set. The right side of the workflow contains actors that perform the last actions needed after the reduction of a data set. This includes renaming of the final science data, an interactive data display (disabled by default) and some housekeeping of variables.

8.2 Other tips

- Keep in mind that the OCA rules and the workflow layout are intimately related. Many times changing one without changing the other won't work. This is specially true when a new calibration recipe is introduced in the workflow for instance, however more subtle cases may appear. As a rule of thumb, do not change the workflow layout unless you understand the changes you have to do at OCA rules level.
- Use the DDF director all over. The parallel director is known to give some problems.
- The Python actor is meant to provide visualization/interactive hooks in the workflow. Of course it can be used for other purposes, but at your own risk.

- If the number of parameters of a recipe changes, the recipe actor won't be able to automatically use the new parameters and merge them with the saved parameters at the workflow. It is highly recommended that you remove the actor and reinstantiate it again. For this reason, we recommend adding recipes to the workflow once its interface/parameters are unlikely to change.
- Use subworkflows to hide complexity. Remeber however that to change the recipe parameters you will have to open the subworkflows where the actor actually is, which may not be clear to your users.
- Use a global RecipeFailureMode.

8.3 Saving Workflows

Save in XML

In the course of your data reductions, it is likely that you will customise the workflow for various data sets, even if this simply consists of editing the `ROOT_DATA_DIR` to a different value for each data set. Whenever you modify a workflow in any way, you have the option of saving the modified version to an `XML` file using `File -> Save As` (which will also open a new workflow canvas corresponding to the saved file). The saved workflow may be opened in subsequent `Reflex` sessions using `File -> Open File`.

8.4 Workflow Canvas Parameters

The workflow canvas displays a number of parameters that may be set by the user (see Figure **??**). Under "Setup Directories" the user is only required to set the `ROOT_DATA_DIR` to the working directory for the data set(s) to be reduced, which, by default, is set to the directory containing the standard UVES test data. Raw data should be stored in a subdirectory of `ROOT_DATA_DIR`, defined by the parameter `RAWDATA_DIR`, which is recursively scanned by the `Data Organiser` actor for input raw data. If required, the user may edit the directories `BOOKKEEPING_DIR`, `LOGS_DIR`, `TMP_PRODUCTS_DIR`, and `END_PRODUCTS_DIR`, which correspond to the directories where book-keeping files, logs, temporary products and end products are stored, respectively (see the Reflex User Manual for further details; Forchì 2010).

Under the "Global Parameters" area of the workflow canvas, the user may set the `FITS_VIEWER` parameter to the command used for running his/her favourite application for inspecting FITS files. Currently this is set by default to `fv`, but other applications, such as `ds9` and `gaia` for example, may be useful for inspecting image data.

By default the `EraseDirs` parameter is set to `false`, which means that no directories are cleaned before executing the workflow, and the recipe actors will work in Lazy mode (see Section 8.5.4), reusing the previous pipeline recipe outputs where input files and parameters are the same as for the previous execution, which saves considerable processing time. Sometimes it is desirable to set the `EraseDirs` parameter to `true`, which forces the workflow to recursively delete the contents of the directories specified by `BOOKKEEPING_DIR`, `LOGS_DIR` and `TMP_PRODUCTS_DIR`. This is useful for keeping disk space usage to a minimum and will force the workflow to fully rereduce the data each time the workflow is run.

The remaining two global parameters are set automatically by the workflow itself and should not be modified.

8.5 Workflow Actors

8.5.1 Simple Actors

Simple actors have workflow symbols that consist of a single (rather than multiple) green-blue rectangle. They may also have a logo within the rectangle to aid in their identification. In the UVES workflow,

the following actors are simple actors: Access to the parameters for a simple actor is achieved by right-clicking on the actor and selecting `Configure Actor`. This will open an "Edit parameters" window (see Figure **??**). Note that the `Product Renamer` actor is a jython script (Java implementation of the Python interpreter) meant to be customized by the user (by double-clicking on it).

8.5.2 Composite Actors

Composite actors have workflow symbols that consist of multiply-layered green-blue rectangles. They generally do not have a logo within the rectangle. A composite actor represents a subworkflow of more simple or composite actors which hides over-complexity from the user in the top-level workflow. In the UVES workflow, the following actors are composite actors: Access to the parameters for a composite actor is achieved by right-clicking on the actor and selecting `Configure Actor`. This will open an "Edit parameters" window (see Figure **??**).

Composite actors may also be expanded to inspect the subworkflow that they represent. To do this, right-click on the actor and select `Open Actor`, which will open the subworkflow in a new `Reflex` canvas window. If the composite actor corresponds to a pipeline recipe, then the corresponding recipe executor actor will be present as a simple actor within the subworkflow, and its parameters are accessible as for any other simple actor.

8.5.3 Recipe Executor Actors

A recipe executor actor is used in the workflow to run a single UVES pipeline recipe (e.g: the "Master Bias Creation" actor runs the `uves_cal_mbias` pipeline recipe). In Figure **??** we show the "Edit parameters" window for a typical recipe executor actor, which can be displayed by right-clicking on the actor and selecting `Configure Actor`. In the following we describe in more detail the function of some of the parameters for a recipe executor actor:

- The "recipe" parameter states the UVES pipeline recipe which will be executed.
- The "mode" parameter has a pull-down menu allowing the user to specify the execution mode of the actor. The available options are:
  - `Run`: The pipeline recipe will be executed, possibly in Lazy mode (see Section 8.5.4). This option is the default option.
  - `Skip`: The pipeline recipe is not executed, and the actor inputs are passed to the actor outputs.
  - `Disabled`: The pipeline recipe is not executed, and the actor inputs are not passed to the actor outputs.
- The "Lazy Mode" parameter has a tick-box (selected by default) which indicates whether the recipe executor actor will run in Lazy mode or not. A full description of Lazy mode is provided in the next section.
- The "Recipe Failure Mode" parameter has a pull-down menu allowing the user to specify the behaviour of the actor if the pipeline recipe fails. The available options are:
  - `Stop`: The actor issues an error message and the workflow stops. This option is the default option.
  - `Continue`: The actor creates an empty output and the workflow continues.
  - `Ask`: The actor displays a pop-up window and asks the user whether he/she wants to continue or stop the workflow.

- The set of parameters which start with "recipe param" and end with a number correspond to the parameters of the relevant UVES pipeline recipe. By default in the recipe executor actor, the pipeline recipe parameters are set to their pipeline default values. If you need to change the default parameter value for any pipeline recipe, then this is where you should edit the value. For more information on the UVES pipeline recipe parameters, the user should refer to the UVES pipeline user manual (Larsen et al. 2010[2]).

The description of the remainder of the recipe executor actor parameters are outside the scope of this tutorial, and the interested user is referred to the Reflex User Manual for further details (Forchì 2010). Any changes that you make in the "Edit parameters" window may be saved in the workflow by clicking the `Commit` button when you have finished.

8.5.4 Lazy Mode

By default, all recipe executor actors in the UVES workflow are "Lazy Mode" enabled. This means that when the workflow attempts to execute such an actor, the actor will check whether the relevant pipeline recipe has already been executed with the same input files and with the same recipe parameters. If this is the case, then the actor will not execute the pipeline recipe, and instead it will simply broadcast the previously generated products to the output port. The purpose of the Lazy mode is therefore to minimise any reprocessing of data by avoiding data rereduction where it is not necessary.

One should note that the actor Lazy mode depends on the contents of the directory specified by `BOOKKEEPING_DIR` and the relevant FITS file checksums. Any modification to the directory contents and/or the file checksums will cause the corresponding actor when executed to run the pipeline recipe again, thereby rereducing the input data.

The forced rereduction of data at each execution may of course be desirable. To force a rereduction of all data for all recipe executor actors in the workflow (i.e. to disable Lazy mode for the whole workflow), set the `EraseDirs` parameter under the "Global Parameters" area of the workflow canvas to `true`. To force a rereduction of data for any single recipe executor actor in the workflow, right-click the actor, select `Configure Actor`, and uncheck the Lazy mode parameter tick-box in the "Edit parameters" window that is displayed. For a composite actor, you will first need to open the subworkflow by right-clicking on the composite actor and selecting `Open Actor`.

8.6 Workflow Steps

8.6.1 Step 1: Data Organisation And Selection

On clicking the `Run` button on the `Reflex` canvas, the workflow will highlight and execute the `Initialise` actor, which among other things will clear any previous reductions if required by the user (see Section 8.4). The `Data Organiser` will be executed next which recursively scans the directory specified by `RAWDATA_DIR` and constructs the set of LoSOs from the data it finds.

The next action in this step is the execution of the `Data Set Chooser` which displays the set of LoSOs available in the "Select LOSOs" window, activating a vertical scroll bar on the right if necessary (see Figure **??**). Sometimes you will want to reduce a subset of these LoSOs rather than all LoSOs, and for this you may individually select (or de-select) LoSOs for processing using the tick boxes in the first column, and the buttons `Select All` and `Deselect All` at the bottom left.

You may also highlight a single LoSO in blue by clicking on the relevant line. If you subsequently click on `Inspect Highlighted`, then a "Select Frames" window will appear that lists the set of files that make up the highlighted LoSO including the full filename and path for each file, the file category (from the FITS header), and a selection tick box in the first column (see Figure **??**). The tick boxes allow you to edit

[2] Available from http://www.eso.org/sci/facilities/paranal/instruments/uves/doc/index.html

the set of files in the LoSO which is useful if it is known that a certain calibration frame is of poor quality (e.g: a poor raw flat-field frame). The list of files in the LoSO may also be saved to disk as an ASCII file by clicking on Save Selected As and using the file browser that appears.

By clicking on the line corresponding to a particular file in the "Select Frames" window, the file will be highlighted in blue, and the file FITS header will be displayed in the text box on the right (see Figure **??**), allowing a quick inspection of useful header keywords. If you then click on Inspect, the workflow will open the file in the selected FITS viewer application defined by the workflow parameter FITS_VIEWER.

To exit from the "Select Frames" window, click Continue, and to exit from the "Select LOSOs" window, click either Continue in order to continue with the workflow reduction, or Stop in order to stop the workflow.

On continuing the workflow, the Fits Router actor will be executed which simply sends the relevant files from the Data Set Chooser for the current LoSO along the correct paths in the workflow.

8.6.2 Product Renamer

After having processed the input data for a data set, the workflow highlights and executes the Product Renamer actor, which, by default, will copy the most important final products of the UVES pipeline recipe uves_obs_scired to the directory specified by END_PRODUCTS_DIR and rename them with names derived from the values of certain FITS header keywords. Specifically, final products are renamed by default with names of the form <HIERARCH.ESO.OBS.NAME>_<HIERARCH.ESO.PRO.CATG>.fits, where <HIERARCH.ESO.OBS.NAME> and <HIERARCH.ESO.PRO.CATG> represent the values of the corresponding FITS header keywords. These names are fully configurable by double-clicking on the Product Renamer actor and editing the string as appropriate.

9 Frequently Asked Questions

1. **Why does the actor highlighting jump around the workflow so much?**

   As mentioned in the quick start Section **??**, this is a perfectly normal consequence of the workflow director scheduling the workflow execution.

2. **Where are my intermediate pipeline products?**

   Intermediate pipeline products are stored in the directory <TMP_PRODUCTS_DIR>/UVES and organised further in directories by pipeline recipe.

3. **I have many LoSOs in my data set (or PI-Pack). How can I reduce them interactively without having to wait a long time between interactive windows being displayed?**

   Reduce all the LoSOs at once with the interactive windows disabled for all interactive actors (right-click the corresponding composite actor, select Configure Actor, set the "EnableInteractivity" parameter to false, and click Commit to save the change). When this reduction has finished, you should re-enable the interactive windows that you require, and run the workflow again. The workflow will run in Lazy mode and no time will be spent on pipeline reductions, unless you specifically change a parameter in one of the interactive windows.

   Note that Lazy mode will not work if the workflow parameter EraseDirs is set to true.

4. **Can I use different sets of bias frames to calibrate my flat frames and science data?**

   Currently, the Reflex platform only supports the creation of a single masterbias frame (or any other calibration) per LoSO. This means that the same masterbias frame will be used to calibrate the flat frames and science data from the same LoSO. Future releases of Reflex will aim to address this limitation.

5. **Can I launch `Reflex` from the command line?**

   Yes, use the command:

   ```
   reflex -runwf -nocache -nogui <workflow_path>/<workflow>.xml
   ```

   Note that this mode is not fully supported, and the user should be aware of two points. Firstly, the execution prompt is not returned after the workflow finishes, and therefore `Reflex` must be manually killed. Secondly, all the interactive windows will still appear (if activated in the workflow), so it is not suitable for batch processing.

6. **How can I add new actors to an existing workflow?**

   You can drag and drop the actors in the menu on the left of the `Reflex` canvas. Under `Projects -> ESO` you may find all the actors relevant for pipeline workflows, with the exception of the recipe executor. This actor must be manually instantiated using `Tools -> Instantiate Component`. Fill in the "Class name" field with `org.eso.RecipeExecuter` and in the pop-up window choose the required recipe from the pull-down menu. To connect the ports of the actor, click on the source port, holding down the left mouse button, and release the mouse button over the destination port. Please consult the Reflex User Manual (Forchì 2010) for more information.

7. **How can I broadcast a result to different subsequent actors?**

   If the output port is a multi-port (filled in white), then you may have several connections from the port. However, if the port is a single port (filled in black), then you may use the black diamond from the toolbar. Make a connection from the output port to the diamond. Then make connections from the input ports to the diamond. Please note that you cannot click to start a connection from the diamond itself. Please consult the Reflex User Manual (Forchì 2010) for more information.

8. **Which instruments will be supported in the future?**

   The upcoming workflow releases for next year include AMBER, MIDI, HAWK-I, XSHOOTER and FORS.

10 General Tips

11 Troubleshooting

1. **The workflow reports to me that there are no LoSOs in my data directory. How can this be?**

   Remember that the UVES workflow supports data for point source observations only. It is possible that your data consists entirely of extended source, image slicer or multi-object spectrograph observations, in which case the `Data Organiser` actor will not construct any LoSOs.

2. **The "Select LoSOs" window displays my LoSOs, but some/all of them are greyed out. What is going on?**

   If a LoSO in the "Select LoSOs" window is greyed out, then it means that the LoSO that was constructed is missing some key calibration(s) (i.e. the LoSO is incomplete). To find out what calibration(s) are missing from a greyed out LoSO, click on the LoSO in question to highlight it in blue, and then click on the button `Inspect Highlighted`. The "Select Frames" window that appears will report the file category of the calibration(s) that are missing.

3. **I am getting an error message when I start a second instance of `Reflex`. What should I do?**

   When a second instance of `Reflex` is started on the same machine (under the same account), a window with the title "Command failed" will appear. This will also happen if a previous execution of `Reflex` did not finish cleanly. To fix this, close the other instance of `Reflex`, or if it is not alive anymore, kill java.

4. **When I click on the "Configure subplots" button in the interactive window (see Section ??, step 6), the window that appears is empty. Can this be corrected?**

   This is a known problem that will addressed in future releases.

12 OCA rules used by Reflex

1) The associations are now recursive. If science A needs calibration B which in turn needs calibration C, all those files will be associated together. This also means that one can for instance use different biases for the flat and the science

2) There is now an implicit triggering of the whole cascade using the keyword REFLEX.TARGET="T"; Previously, the implicit target were the science data, but now it is possible to start the association cascade with any kind of data, no more implicit triggering.

3) The DO.CATG is ignored by Reflex. Use REFLEX.CATG. For compatibility reasons it is recommended to define both, it is harmless and nesures that the same OCA file can be used by Reflex and the Paranal DataOrganizer.

13 Browse actor documentation

Right click on the actor....

14 Using the common Python module

How to get the dataset and files: files, datasetname =parseSof(insof) print "Dataset: ", datasetname for fits in files: print print file print "Name: ", file.name print "Category: ", file.category print "Purpose: ", file.purpose

15 Reflex 1.x to 2.x transition

Structural changes ========== - Put Sof Combiner before composite actors (where before there was simply a RecipeExecutor. This will ensure that only common purposes to all the channels go through. - Change the EmptySof string. Use the SofCreator with an empty string as an input - Use the new IsSofEmpty actor - Put PurposeSeriazlied/De before each recipe - Set StripLast Purpose - Change all the ports editing manually the XML. See port conversion sheet. - Change the ProductRenamer, since the code is new and the old one is directly embedded in the workflow. - Set the widths of all the channels which go to the SofSplitter to Auto

Design changes ========== - In the FITSRouter, if two (or more) DO.CATG have several purposes, one has to separate them in different ports, otherwise the Purpose matching of the SOFCombiner won't work. - In the OCA rules, if recipe1 generates 2 virtual products, and recipe2 needs them, only one virt product can be associated. If you associate both, then there is a duplication of files. This will be fixed

by Vincenzo. - Do not connect the output of two recipes to the same SofCombiner unless one is really sure that they share exactly the same purposes. Otherwise the non-matching purposes will be lost.

- Set the Variable Setters not te be delayed. This is not specific to Reflex2, but it should go into the guidelines.

Testing ======= - Check that the executed purposes match the action tree (presented by the Data Set Chooser): grep -i action reflex_book_keeping/Uves/uves_cal_mbias_1/*/*xml — uniq - For each recipe, make sure that all the input SOFs correspond to all the presences of that action/recipe in the action tree

Tips ====

- For massive reduction of data. Put everything to Continue 'Failure mode' and non-interactivity. Run everything. Then put back to Ask and enable interactivity, to review the problems.

15.1 When to use the sof_opt port in the SofCombiner

Expose the case of the optional input bias for the format check.

16 How to packk a workflow with a pipeline

- Use reflex directory - Create Makefile.am which install everything in wkf.. - Add the wkfdir to acinclude.m4 - Add the workflows .in to configure.ac - Use something like parse_wkf_for_cvs

- Set all the paths relative to the installation of the pipeline (OCA rules, python actors,...) with a global root variable like $PIPELINE_INST_ROOT.

17 autotools and workflows

Use fors-@VERSION@ for the calibration directory.

18

Poner todo lo del subworkflow para el initialise current dataset (PIPE-3859). Aniadir tb lo de requiredFiringsPerIteration (mail from Wolfram).

19

The FitsRouter _config option: You have to set it to a StringParameter and do not use quotes.

20

Multiports: explain that multiports in Reflex actors work the same as SofCombiner. In other Kepler actors (including Composite actors), you get however two tockens.

21 SofCombiner

Explain its behaviour very well!!!

22 Supporting optional calibrations

Strategies:
-allow empty inputs - Python actors -...

23 The Reflex Python framework

24 Reflex and the CalSelector

CalSelector is a tool to provide all the needed raw frames used to reduce a given type of data. In order to accomplish this, it uses OCA rules that define the calibration cascade. ESO sanctioned workflows use a similiar set of OCA rules, allowing the workflow to be fed with the data retrieved using CalSelector. However, it has to be kept in mind that this is achieved only if the OCA rules do actually provide the same calibration cascade.

25 Deploying and delivering a workfow

If a workflow has to be delivered together with the pipeline, the recommended way is to incorporate it in the usual autotools setup of it. The workflow itself is installed under `$prefix/share/reflex/workflows/$pipe-$vers` where *pipe*−`vers` is the pipeline name and version. This place is where a user should look for a workflow. Additionally, the workflow might need extra files (OCA rules, python scripts...) which are installed under `$prefix/share/esopipes/$pipe-$vers/reflex`.

The template pipeline implements these behaviour. We list here the steps to do:

- Put all your Reflex related files under `reflex` directory in the pipeline source tree.
- Edit file `acinclude.m4` in the pipeline source directory and under the function `[PIPE_SET_PATHS]` add the following:

```
if test -z "$wkfextradir"; then
    wkfextradir='${datadir}/esopipes/${PACKAGE}-${VERSION}/reflex'
fi

if test -z "$wkfcopydir"; then
    wkfcopydir='${datadir}/reflex/workflows/${PACKAGE}-${VERSION}'
fi


AC_SUBST(wkfextradir)
AC_SUBST(wkfcopydir)
```

- Edit file `configure.ac` and in the `AC_CONFIG_FILES` clause add the workflow:

```
reflex/Instrument.xml)
```

- Create a `reflex/Makefile.am` file with information about he workflow files, OCA rules and Python scripts. For example, for VIMOS, which contain two workflows but not Python scripts would be like this:

```
AUTOMAKE_OPTIONS = foreign
WORKFLOWS = VimosIfu.xml VimosMos.xml
OCAWKF =  vimos_ifu_wkf.oca vimos_ifu_wkf.dvd.oca vimos_mos_wkf.oca
PYTHONWKF =
wkfextra_DATA = $(WORKFLOWS) $(OCAWKF) $(PYTHONWKF)
EXTRA_DIST = $(WORKFLOWS).in $(OCAWKF) $(PYTHONWKF)
wkfcopy_DATA = $(WORKFLOWS)
```

- Make sure that in the toplevel `Makefile.am` the `reflex` subdirectory is in the `SUBDIRS` variable.
- The workflow should actually be named Instrument.xml.in (i. e., the name put in file `configure.ac` appended with .in). The paths to the OCA rule has to be set to `@prefix@/share/esopipes/pipe-@VERSION@/refl` where `pipe` is the name of the pipeline. Searching for string `OCA File` in the .xml will help.
- Also the paths to the python scripts should be updated accodingly. Search for string `Python script` and substitute the path with `@prefix@/share/esopipes/pipe-@VERSION@/reflex/script.py`.
- It is also convenient to set the title of the workflow to something descriptive plus the version of the pipeline. To do that, put the placeholder `@VERSION@` instead of the version.
- Also convenient is to substitute the data paths with the placeholder `ROOT_DATA_PATH_TO_REPLACE`. This is not substituted by autotools, but the `install_reflex` script will do it. Also the calibration directory should be `CALIB_DATA_PATH_TO_REPLACE/pipe-@VERSION@`.

The template pipeline/worflow contains a script called `reflex/parse_wkf_for_cvs` that will do all the workflow changes for you, usually recommended just before archiving the workflow in the SCM system. Most likely the script should be edited to suit the particular needs.

25.1 File association

TODO

25.2 Virtual products

TODO _WKF..

25.3 Recipe specification

The recipe specification part in the OCA rules, used by other tools is completely ignored by Reflex. Reflex uses the OCA rules to get the dependency tree of the raw frames, but the recipes used the reduce the data are simply defined in the workflow itself, with the `RecipeExecutor` actors. Obviously, the recipe parameters defined in the same section are also ignored.

26

What happens if a static calibration is needed by science and calibrations. You have to create a _extra in the FitsRouter and join it afterwards.

27 Graphical design considerations

- The top level of a workflow should represent an overview of the whole data reduction flow. Actors should be grouped in sections. Each section represents a subjects relevant to the astronomer. The sections should be be labeled accordingly. The first and last section should be Data Organisation and Out- put Organisation. Other sections describe top level processing steps, such as Master Calibrations, Preprocessing or Image Combination.
- Direction of Data Flow. The overall data flow in any workflow should be either from left to right, or from top to bottom. Actors and subworkflows should be placed along this overall direction of the workflow in such a manner that their position reflects the logical order of the flow.
- The connections between actors should also be grouped and labeled. Similar data should flow in parallel connections. It should be easy to trace back each input to an actor to its origin.
- Logical Elements The implementation of relative easy logical decisions sometimes require fairly complex constructs in Kepler. Such constructs should be hidden in a subworkflow. An example of this is the Flat Combiner in the UVES workflow.
- To draw the attention of the user to the most important steps (e.g. interactive actors), actors should be marked with colored boxes. The color scheme should be explained on the workflow

Acknowledgements

The `Reflex` team in alphabetical order consists of Pascal Ballester, Daniel Bramich, Vincenzo Forchì, Wolfram Freudling, César Enrique García, Maurice Klein Gebbinck, Andrea Modigliani, Sbine Möhler & Martino Romaniello.

References

Forchì V., Reflex User Manual, VLT-MAN-ESO-19000-5037, Issue 0.7

Horne K., 1986, PASP, 98, 609

Kaufer A. et al., UV-Visual Echelle Spectrograph User Manual, VLT-MAN-ESO-13200-1825, Issue 86

Larsen J.M., Modigliani A. & Bramich D.M., UVES Pipeline User Manual, VLT-MAN-ESO-19500-2965, Issue 14.0