EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

HOS / Broker for Observation Blocks

User Manual

Doc.No. VLT-MAN-ESO-17220-1332

Issue 7

Date 24/04/2007

DRAFT - FOR ESO INTERNAL USE ONLY

	Name	Date	Signature
Prepared	E. Allaert	24/04/2007	
Approved	K. Wirenstrand		
Released	M. Peron		

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0	29/04/1997	All	First version
1.1	20/11/1997	Chapter 2	Adapted to current version (NOV97)
1.2	12/10/1998	Chapter 2	Inclusion of MOBS (OCT98)
1.3	18/11/1999	2.3 2.4 2.7.2 2.7.6 2.13	Added configuration options (SPR 980658) New section: BOB's preferences Added *setCallBack* (SPR 980658) and *sendObsKeys* (SPR 990355) New section: Search paths New section: Technical Templates
1.4	06/12/2000	2.2 2.7.2 3	Added automatic logging (SPR 20000234) Added pafReady (new version of [1]) New chapter: Troubleshooting
1.5	06/03/2001	2.3	Added OH-communication option to configuration
2	27/03/2002	2.7.2 2.11 2.6	Added *finishOB* (SPR 20010506) Added programmatic access Added FITS-logs (SPR 20010098)
3	24/03/2003	2.2 2.3 2.6 2.7.1 2.7.2 2.15 3.1.8	Added description of labels (SPR 20020527) Added comboboxes (SPR 20020526) Control flow persistence (SPR 20010762) Use of keyword-arrays (SPR 20020304) Added *seeBobVars* (SPR 20020304) Added section (SPR 20020511) Described additional warnings (SPR 20020304)
4	19/04/2004	2.3 2.7.2 2.11.1 2.15 4.2	Added details about OB label (SPR 20030145) Extended seeBobVars (SPR 20030518) Added *bobSave* and *bobSaveAs* (SPR 20040011) Added snapshot details (SPR 20030520) Added manpages for several bob* commands
5	27/10/2005	2.3 2.4	Cmd skip-list syntax revised (SPR 20040269), added config param call-back (SPR 20040363) Modified preferences panel (SPRs 20040205)
6	23/01/2007	2.6, 2.10	Multiple selection of templates (SPR 20050396)

1 INTRODUCTION

1.1 PURPOSE

This document is the User Manual for the HOS/Broker for Observation Blocks version 2.9 and is intended to provide all the necessary information for the installation and use of this module from an end-user's point of view.

1.2 SCOPE

This document describes how to configure and use BOB as a high level tool for executing observation blocks, showing primarily the VLT Control Software side. It does not consider in any detail the creation, modification or scheduling of observation blocks, nor the lower level modules (DCS, ICS, TCS) which are actually acquiring the astronomical data belonging to the observation blocks. Neither does it deal with the subsequent archiving or pipeline-reduction of these data.

1.3 APPLICABLE DOCUMENTS

[1] VLT-ICD-ESO-17240-19200, 2 31/07/2002 -- ICD between VCS and OH

1.4 REFERENCE DOCUMENTS

The following documents are referenced in this document.

[2] VLT-MAN-ESO-17220-0737, 3 28/03/2002 --- VLT SW HOS/Sequencer User Manual

[3] VLT-MAN-ESO-17220-1349, 1.1 28/08/1997 --- VLT SW HOS/BOB SW Maintenance Manual

[4] VLT-SPE-ESO-17240-0385, 4 13/01/2005 --- VLT SW INS Common Software Specification

[5] VLT-MAN-ESO-17220-3676, 2 25/01/2007 --- VLT SW HOS/Audio SW User Manual

1.5 ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used in this document:

CCS	Central Control Software
GUI	Graphical User Interface
HOS	High Level Operating Software
HW	Hardware
ICS	Instrument Control Software
OBD	Observation Block Description
OH	Observation Handling
OS	Observation Software
OT	Observing Tool
P2PP	Phase II Proposal Preparation System
SW	Software
TBD	To Be Defined
VCS	VLT Control Software

VLT	Very Large Telescope
WS	Workstation

1.6 GLOSSARY

Observation Block
: A high level view of VLT operations. An observation block is the smallest schedulable observational unit for the VLT. It is a rather complex entity, containing all information necessary to execute sequentially and without interruption a set of correlated exposures, involving a single target (i.e. a single telescope preset). It contains a.o. one or more template calls, i.e. it describes what templates to call with which parameters. Consequently, during Phase II Proposal Preparation, observers will have to build observation blocks, using the tools provided for that purpose, by selecting templates, defining parameters, and giving additional parameters for scheduling and data reduction.
The contents of an Observation block is reflected in ASCII format by a so-called *Observation Block Description*.

Phase II Proposal Preparation
: The P2PP system assists the user in preparing Observation Blocks.

sequence, Sequencer script
: a set of commands in Sequencer language, generally intended to define and execute a series of related observations. These sequences are to be interpreted by a sequencer shell.

Tcl/Tk
: **T**ool **c**ommand **l**anguage / **T**ool**k**it. A general purpose scripting language designed and implemented by Dr. John Ousterhout of the University of Berkeley, CA (presently working at Sun Microsystems Laboratories).

Template
: An entity containing a sequence (Sequencer script), dealing with the setup and execution of one or more exposures. It is literally a “template”, to which a set of parameters belong whose specific values determine the exact behaviour of its execution.
Technically, a template is an object belonging to the `Template` class, and has several components, one of the being the Sequencer script defined above. Other components are the *template signature file* and the *template parameter GUI*.
Templates should generally have the complexity of setting up and executing one or a few exposures.

Template call
: The name of the template to execute, with its parameter values.

Template signature file
: This is a description of a template and its parameters. It contains a.o. information about the type and allowed ranges of the parameters, so that a trivial validity check can already be performed when parameters are entered via the template parameter GUI.

1.7 STYLISTIC CONVENTIONS

The following styles are used:

bold
: in the text, for commands, filenames, pre/suffixes as they have to be typed.

italic
: in the text, for parts that have to be substituted with the real content before typing.

`teletype`
: for examples.

`<name>`
in the examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

`$` indicates a UNIX shell prompt

2 USER'S GUIDE

2.1 OVERVIEW

BOB is the high-level interface towards the VLT Control Software. It operates on Observation Block Descriptions (OBDs), which it either reads from a disk file or receives from the Scheduling tool (OH/OT). These OBDs contain (or refer to) all the information needed by VCS as a whole to execute individual exposures. It is BOB's task to break these OBDs down, ending up with "units" of individual exposures, sending sequences of commands which are understood by its VCS lower level partner, i.e. OS. At the same time, BOB's GUI keeps the user informed of the contents of Observation Blocks plus Templates, and their status of execution, as high-level units.

If the OBDs have been received from the Scheduler, the status of execution of the individual Templates etc. is fed back to the Scheduler as these events occur. This allows the Scheduler to take internal decisions, and also to inform its fellow processes on the OH side about relevant events.

2.2 STARTING BOB UP

BOB requires communication with the OH-scheduler and with the OS of the instrument it is controlling. For this communication it uses the CCS message system. Therefore, BOB needs to be started from a shell running on a WS equipped with CCS and running environments.

As the messages exchanged between BOB and the OH-scheduler do not include the contents of template scripts nor template signature files, but rather only a reference to their names, these files need to be present on the system where BOB runs. BOB will look for these files under specific paths, including `$INS_ROOT/SYSTEM/COMMON/SEQUENCES`[1]. So it is vital to have the `INS_ROOT` environment variable set before starting BOB.

The command to start BOB is

```
$ bob ?-display <hostName>? -- ?-configFile <fileName>? \
                               ?-password <passwd>? \
                               ?-ccsProcName <name>?
                               ?<scriptFile>? &
```

The options before the double dash are the same ones as the options permitted for the standard wish (see the manpage of Tcl). The ones after the double dash are the name of the configuration file under the home-directory (defaulting to `.bobrc` - see also below), the password to switch to the engineering interface (see section2.10) and the name under which to register BOB with CCS.

BOB will at start-up automatically configure itself, reading in the last saved configuration file (see below). When BOB starts up successfully, a main panel will appear on the screen (see Figure 4), and BOB will start executing <scriptFile> (if specified; see also section 2.11). The various constituent parts of this GUI are:

- a window title bar, which contains a.o. the name with which this instance of bob is registered with CCS, and the environment under which it is running. If the *ccsInit* failed, this part within brackets will simply contain *bob@<hostname>*
- at the top a menu bar, with the **File**, **Configure**, **Interface**, **Errors** and **Help** menus.

1. The search path also include paths which depend on the setting of the INS_USER and the instrument mode, combined with INS_ROOT, in full compliance with the INS Common Software Specification.

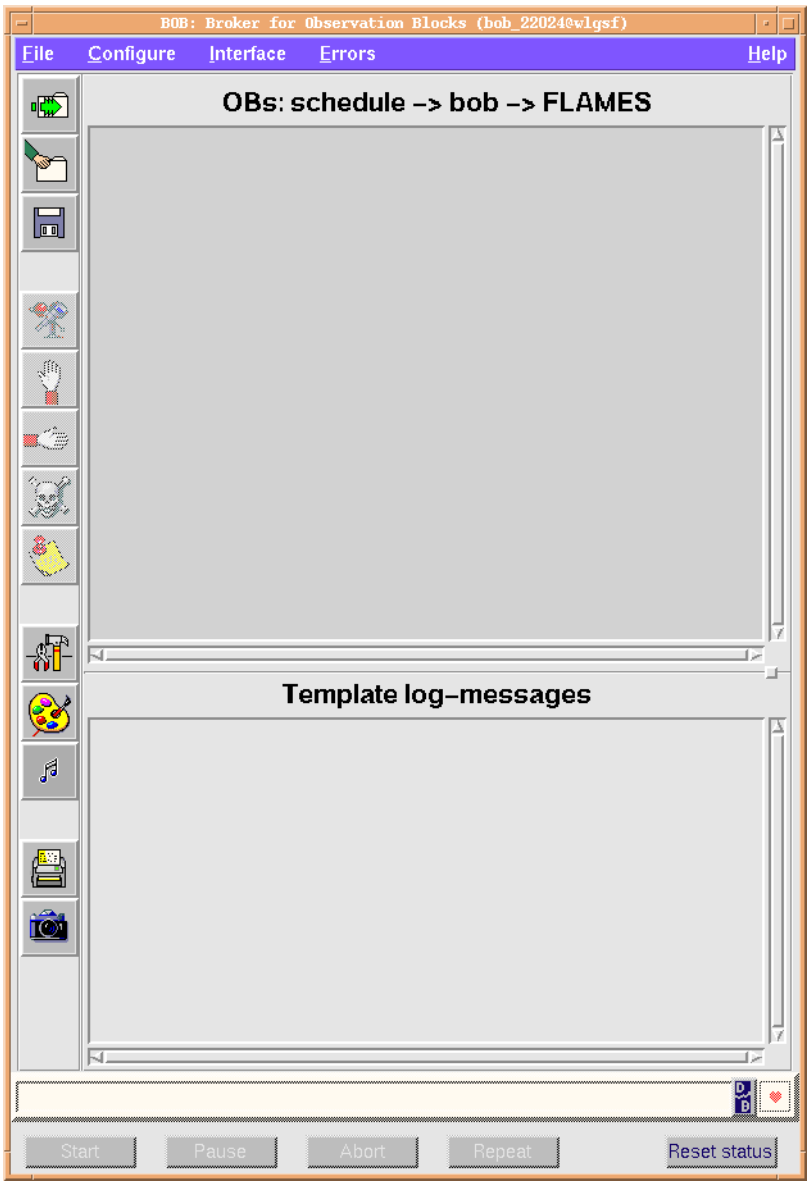

Figure 1: BOB's main panel at start-up

- a vertical toolbar on the left side with buttons which have icons in them. These buttons have balloon-help, and are typically the only buttons to press for a successful execution of Observation Blocks. Inactive buttons appear shady.
- a short-help line (white background) at the bottom, with the heartbeat indicator. Depending on where the mouse cursor is, this line will show a single line of text, showing what the effect will be if a mouse button (typically the left button) gets pressed.
- below the short-help line, an *action area*, with labelled buttons. Inactive buttons are shaded out.
- a canvas area for information about loaded Observation Blocks
- a canvas area for information about executing or executed Templates.

Although the main panel by itself is not resizable, the relative distribution of the area occupied by the two latter panes can be modified by dragging the squared sash at the right side on the horizontal separation line between the two panes. Also, these panes are equipped with vertical and horizontal scrollbars, which can be used to see information which is off-screen.

The messages which appear in the Template Log-area are also automatically stored on disk. OB-activity will be written into `$VLTDATA/tmp/bob_<procNameOS>.log`, when an OB terminates or when it is aborted. If `<procNameOS>` is changed (see 2.3 below), the log-file is changed correspondingly. If the logfile `$VLTDATA/tmp/bob_<procNameOS>.log` already exists, bob will move it to `$VLTDATA/tmp/bob_<procNameOS>.log.old`, before opening the former one.

Apart from these template logs, bob will also generate FITS ops-logs, which are archived. See also section 2.6.

2.3 CONFIGURING BOB

To be able to communicate with OS and the OH-scheduler, BOB needs to have some information which reflects its environment. Although BOB at start-up configures itself to the configuration which was last saved, it is of course necessary to set these configuration parameters when the environment (process names, etc.) change, or initially when BOB is run for the first time.

The configuration panel (see Figure 2) is invoked by either hitting the "tools" button (toolbar, ninth button from top) or by clicking on the **Configure** menu, then going to **Environment...**.

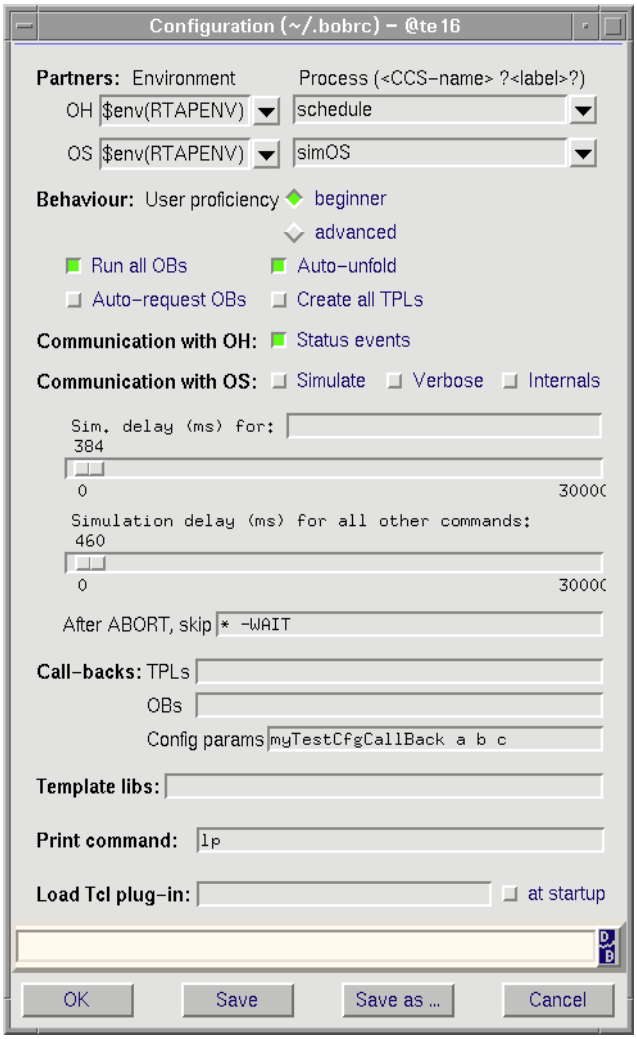

Figure 2: BOB's configuration panel

This panel contains a.o. the environment name and process name of the OH-scheduler[1], which are obviously

essential if communication with OH is needed, i.e. when the OBDs will not be retrieved from a disk file. The environment name / process name for OS also need to be filled out. Remark that Tcl-variables can also be referenced in these latter entry fields (and further down, in the *Call-backs* and *Print command* entry fields as well). This is particularly useful for environment variables, e.g. **`$env(RTAPENV)`** stands for the value of the environment variable `RTAPENV`.

The fields for the process names can contain a second list-element, in which case this element will be used as process-identifier part of the label for the OB-canvas (see Figure 1). If for OS there is no such label given, bob will try to find out the instrument-id by interrogating the OS process: first it will send a "`STATUS -function INS.ID`" command, and if this fails, a "`FORWARD -subSystem ICS -command STATUS -arguments 1,INS.ID`" command. If and only if both these commands do not return any useful information, the OS process-name itself will be used in this label for the OB-canvas. Remark that new entries in all these comboboxes can be saved in their respective listbox (showing up when the arrow is pressed) by pressing the <Enter> key. An entry can be removed from a listbox by selecting it (so it appears in the entry field), preceding it with a minus sign and then pressing <Enter>.

It should be noted that the label of the OB-canvas reflects the status of the currently displayed OB. I.e. changing the OH process name via the configuration panel will not be reflected by this label until a new OB is fetched from OH. On the other hand, changing the OS process name will be shown immediately, as this does affect the currently displayed OB. Remark that the short help-text for the vertical toolbar top button (to fetch an OB) contains the information where the OB will be fetched from, and the fourth button (to start an OB) will say where it sends the template commands to.

The action corresponding to the top button on the toolbar is influenced by the settings for OH: if either the environment name or process name is left blank, BOB will assume you want to retrieve files from disk, instead of retrieving them from OH.

Part of the configuration can be set directly via the **Configuration** menu of BOB's main panel, i.e. without pressing **Environment...** and invoking the configuration panel. All of the following configuration menu options can be selected directly:

- **Run all OBs**: if more than one OB has been downloaded, this option will make sure that they are all executed one after the other, unless there is some error or operator intervention.
- **Auto-request OBs**: when set, this will cause BOB to request automatically for the next OB when the current one is finished. Remark that if an OBD is loaded from a file, while the OH procname is blank and the auto-request option is set, then BOB will reset the OBs when the last one has been executed, and start all over again (in other words, BOB will loop infinitely).
- **Auto-unfold**: when set, the information shown on the canvas about the OB and its templates will automatically unfold as they are executed, and fold-back when finished - unless the OB/ templates were explicitly unfolded before the start (in which case they remain as they were).
- **Create all TPLs**: this will create and check (to some extent) all the templates of all OBs. This is likely to be a "waste of energy", as a normal operation of BOB plus scheduler foresees to use only the first of the OBs received, and then after execution to request the next.
- **Verbose TPL communication**: this will log in the "Template Messages" text-box all commands + parameters sent by the template script to OS, plus corresponding replies. Commands are logged in blue, OK replies in green, error messages and error replies in red. Messages logged explicitly by the template-script itself and the "`starting exposure ...`" message remain in black (the standard foreground colour), and are always underlined.

1. Currently, this process initializes itself with the name **schedule**.

- **Simulate TPL communication**: this will short-circuit the communication between BOB and OS. This means that commands sent by the template script to OS will always return successfully, with some configurable delay. Remark that this type of simulation is without the intervention of any external simulation process: all short-cuts are taken by BOB itself.

All these configuration menu options appear also on the Configuration panel (see Figure 4). Configuration items which are in the configuration panel, but not under the Configure menu of BOB's main panel are:

- The **Environment** and **Process names** of both OH and OS (see above). These are the names under which they run in the CCS environment.
- The **Behaviour** of bob; this includes the **User proficiency** (*beginner* or *advanced*), which allows bob to decide how verbose and technical some messages need to be. This section includes also the first four checkbuttons as available under the Configure menu (see above).
- The **Communication with OH**: contains the **Status events** checkbutton. When this button is activated (which it is by default), all template- and OB-status messages as defined in [1] will be sent to OH. Otherwise, none of the automatically generated status events will be transmitted, and OH will be told it should not expect any of these events either. Remark that the `PAFReady` event, which is not generated by BOB itself but rather by an explicit call within a template script (see section 2.7.2) is always transmitted to OH.
- The **Communication with OS**: contains the **Simulate**, **Verbose** and **Internals** checkbuttons, and two sliders for simulation delays, allowing to set how many ms a reply should artificially be delayed when in simulation mode. The top one is for specific commands which are listed in the entry field close to this slide bar. The bottom one is similar, but for all commands which are not listed in the entry field of the "specific" OS commands. The **Internals** checkbutton is for obtaining additional information about keyword use - i.e. additional info/warnings, logged in the TPL-messages text-box (see section 2.7.1).
  Finally, this section has an entry field to list all OS-commands which should be skipped after the Abort flag has been set (by pressing the Abort button). This can speed up considerably the response time to the Abort button (see 2.7.3). The character * can be entered as a wildcard for all commands, while commands preceded by a dash indicate an exclusion from this skip-list.
- **Call-backs**: the scripts to execute at termination of an OB, template, or when a configuration parameter is modified. The definition of the template call-back can be overridden programmatically in e.g. the template script itself via a call to the *setCallBack* convenience procedure (see section 2.7.2). For the OB call-back on the other hand there is no similar convenience procedure.
  These OB/template call-backs will be executed independent from the exit-status of their OB resp. template, i.e. even when the OB/template fails, the call-back will be evaluated.
  Remark that these call-backs should not be seen as an extension of the template's functionality (e.g. to prepare/start exposures). Instead, they should be considered as internal housekeeping functions. Their typical use is to log the template's or OB's termination or to put the instrument in a safe state. They will be executed *after* the template's resp. OB's termination status has been communicated to the Scheduler (OH/OT). The template call-backs will *not* skip any commands after the ABORT key has been pressed, independent of the configuration setting explained in the section above on *Communication with OS*. This allows to put the instrument into a safe state, independent of the termination status of the template/OB.
  The call-back for the modification of a configuration parameter is executed at global level, and gets at invocation automatically four more arguments than what is declared in the corresponding entry field:
  a. the name of the modified configuration parameter.
  b. a boolean indicating if this call-back is executed during the initialization phase (value 1) or during normal operation (value 0)
  c. the previous value of that configuration parameter

d. the new value of that configuration parameter

E.g. if in the configuration panel the entry field for this call-back is set to '`myCfgCB a b`', the command `myCfgCB` could be defined in a library as

```
proc myCfgCB {fixed1 fixed2 configParam initFlag oldValue newValue} {
    # here we process the input params
    ...
}
```

- **Template libraries**: a space-separated list of names of VLT/Tcl-libraries which contain declarations of auxiliary procedures used by some templates. Remark that these names do not include the "lib" prefix nor the ".tcl" suffix of the directory they are stored in, so in the example of Figure 4 BOB would look for procedures under directories called `lib/libmySeqLib.tcl` and `lib/libmyTclLib.tcl`. The search path for these directories will be:
  e. the current working directory
  f. the parent directory of the current working directory
  g. the directory pointed to by the environment variable INTROOT
  h. the directory pointed to by the environment variable VLTROOT

  This scheme coincides with what is common practice for the VLT Common Software, and is supported by a.o. vltMakefile.
- **Print command**: command into which logs will be piped when printing is requested. As this is actually the trailing part of a Tcl exec-command, Tcl-variables can be used
- **Load Tcl Plugin**: to import a Tcl script (e.g. a VLT panel), within the same interpreter as BOB. This means such a Tcl script can have access to e.g. variables set by the template script. If the **at start-up** checkbutton is ticked, this script will be executed during start-up, immediately after bob is initialized. Otherwise this script can be activated via the **Configure** menu.

The configuration settings can be saved by pressing the **Save** or **Save as ...** action buttons in the Configuration Panel. In the former case (**Save** button) these settings will be stored in the file as indicated in the Configuration Panel title bar, i.e. the currently loaded configuration file. In the latter case (**Save as ...** button), a file selection panel will pop up to enter a filename. After saving the configuration file, the window title showing the name of the configuration file will be updated. Remark that the default name for the configuration file is `.bobrc`, and this can be overridden at BOB's start-up with the `-configFile` option. If this is a relative pathname, it will be interpreted as relative to the home-directory. If this file exists, it will be read by BOB at start-up, i.e. BOB will start-up with the configuration that was last saved into that file. Otherwise, BOB will try to give some "reasonable" default values to these parameters.

A similar scheme is applied for the **Preferences**. The Preferences panel can be triggered via the Configuration menu or the "palette" button on the toolbar. The purpose and contents of the Preferences panel is explained in section 2.4.

2.4 BOB'S PREFERENCES

There are several settings of BOB which do not require any adjustment in order to have BOB operating and behaving correctly. These have been grouped in a *Preferences* panel, which will pop-up when the toolbar-button below *Configure* is pressed. The panel which will then be displayed is shown in Figure 3.

There are currently 3 sections in this panel: font definitions, sound related settings and log-texts for the OS WAIT command. In the former, one can set the fonts for the labels of the OB canvas and TPL-messages textbox, the text within the OB canvas (the upper part of BOB's main panel), the text within the template log area (the lower part of BOB's main panel), the dialog boxes showing warning or error messages, and finally the editor panels used during editing of OBs/templates (within engineering mode - see section 2.10).

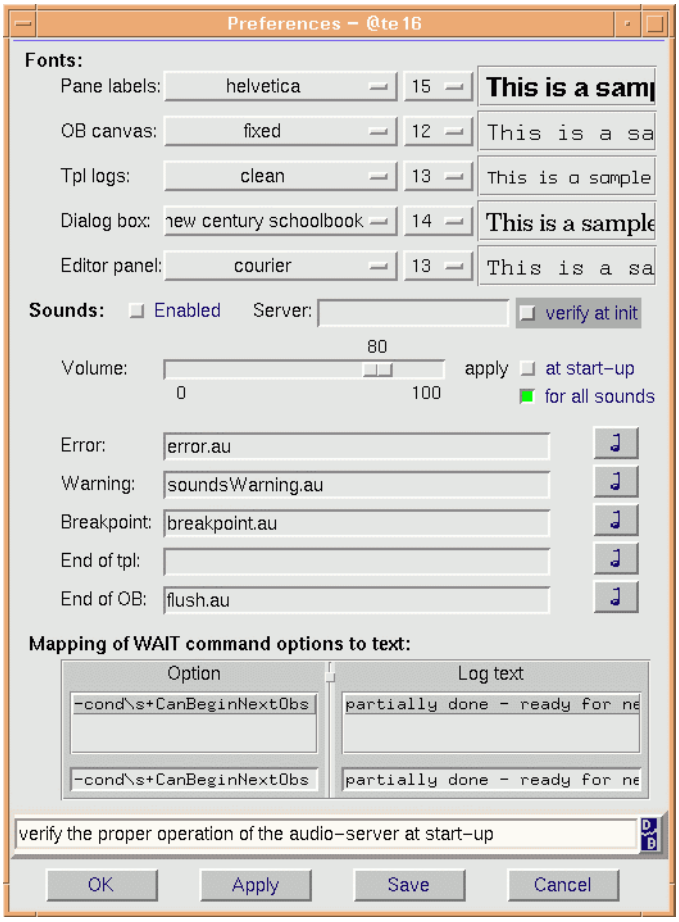

Figure 3: BOB's preferences panel

In the middle part of the preferences panel, one can select the audio-server, indicate whether its proper functioning should be tested at start-up or not, set the audio-volume and determine at what occasion this volume should be applied. Remember though that it is necessary to press the "Apply" or "Save" button to enforce these settings - e.g. merely moving the volume slider will have no effect on the volume until either of these latter buttons has been pressed. In this part of the panel one can also point to the sound files to be played when particular events occur. Remark that this requires the sound to be enabled. This can be done via either the "Enabled" checkbutton at the top of this section, or the BOB toolbar-button below *Preferences*. Of course that on its turn requires that the audio-server pointed to (by default or via the corresponding entry field) is properly functioning; for details on audio see [5]. The filenames of the sound-files need to have their extension included in these entry-fields. Current supported file formats are WAV, MP3, AU, SND, AIFF, SMP, and RAW binary; as BOB relies on the *audio* module for playing sounds, this list of formats is actually determined by the *audio* module.

The bottom section of this preferences panel allows to configure the messages to log when a reply is received on a `WAIT` command sent to OS, according to the option that is passed along with this `WAIT` command. The options are specified in the left column, while the text is entered in the right column. The resulting log-text for a matching option will be "`exposure <exposureNr> of <nrOfExposures> <text> (<timeStamp>)`" whereby `<text>` is as defined in the right column. If a `WAIT` command is issued for which there is no match as far as its accompanying options are concerned, the log-text upon receiving a reply will be "`ended exposure <exposureNr> of <nrOfExposures> (<timeStamp>)`". Remark that the option-matching is done using regular expressions, so in order to match any amount of whitespace one should specify "`\s+`". To edit the entries in this widget, some key-bindings have been defined. In short, you first click on an entry in the left or right listbox to select which option/text you want to edit; you validate this editing by pressing the

<Enter> key while the cursor is in the left or right entry field. To add an entry, first click on the left or right listbox, and then click on the <Insert> key. Now you can edit this new entry in the fields below. To remove an entry, select the line first in the left or right listbox, then press the <Delete> key. See also the manpage of `BobPanedComboBox.n in` section 4.2, in particular for additional key-bindings, and the `re_syntax.n` manpage for the syntax of regular expressions.

The different action buttons for this preferences panel are standard: **OK** will apply the values and remove the panel, **Apply** will force BOB to immediately take the shown values into account while leaving the preferences panel displayed, **Save** will apply the values and save them in the configuration file, leaving the panel on the display, while **Cancel** will pull out of this preferences setting, keeping the values as they were when applied last.

2.5 LOADING AND SAVING OB-DESCRIPTIONS

OBs can be loaded via the **File** menu - **Load OBs**. A sub-menu will allow to either request one or more OBs from the OH-scheduler, or to load them from a file (penultimate menu option). In the latter case a file-selection box will appear. The file pointed to needs to start with a so-called PAF header (see [1]), followed by the OBD(s), in the exact same format as sent by OH. It can contain more than one OB; in that case a next OB is in that file marked by the OBS.ID keyword.

If a set of OBs was loaded via a file, it can afterwards also be reloaded via the **Reload file** menu option under **Load OBs**. This is useful e.g. if one has edited some parameters via the Engineering interface, and wants to discard the changes. This button can normally also be activated during execution of an OB. Although this may look like a graceful recovery for a hanging script, there is presently for technical reasons no complete clean-up of internal tables and stacks associated with this action. I.e. if a reply was pending when a file was reloaded, this reply will still be pending and can actually disturb BOB if it would still come in at a later time. So care has to be taken with the use of this menu option.

The next OB can also be solicited via the top button of the toolbar. This will then lead to a request of a single OB to OH. If in the Configuration panel the environment name or process name of the OH partner is left blank, BOB will assume you want to retrieve a file from disk, instead of retrieving an OB from OH.

The OBs which are received from OH can also be saved locally, either by the **Save** command under the **File** menu, or via the third button of the toolbar. In either case a file-selection box will appear to get a filename. If the filename given does not have a `.obd` extension, BOB will append one to the filename.

When the OBD is loaded, BOB will show its contents, and allows you to perform several actions on it, like:

- fold/unfold the information; an arrow pointing to the right indicates there is more information hidden underneath, and clicking on it will unfold to the next level.
- flag that a particular OB or template should be skipped[1], or that BOB must pause when it gets to that point.
- select one or more particular OBs or templates to display their logs; logs are empty at the beginning and get filled as execution proceeds.

An example of this is shown in Figure 4.

Remark that the label of an OB and its templates contains information which is taken from the OBD and the template signature files. This includes the ID of OBs and templates.

1. This is an engineering mode, which is not to be used as a standard observation practise, as it actually renders the OB useless for further dataflow-pipeline processing. For that reason, this feature may not be globally available in future releases of BOB.

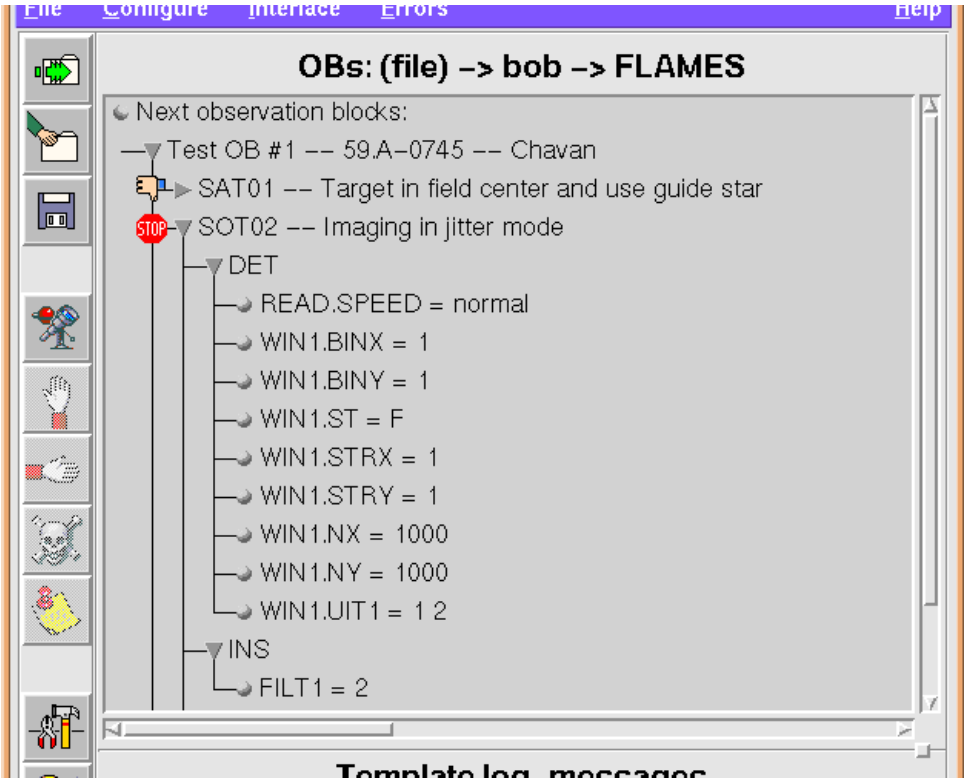

Figure 4: An example OBD loaded into BOB. It is (partially) expanded, and some flags have been set.

2.6 RUNNING AN OB

Press the “Start” action button, or the telescope button in the toolbar (fourth from top). An icon will appear next to the OB and template, showing its state (“running”). In general, an OB inherits the state of the currently executing template. If this template returns an error, or is aborted, its OB will get the same status.

OBs and templates can be paused or aborted while running. The “pause” action button will only pause in between 2 templates. An “abort” will pass the proper flags on to the executing template, but the template only looks at these flags on particular moments, so an “abort” is usually not immediate. Anyway, BOB itself does not issue an abort-command towards OS, it merely sets a flag which can be picked up by the template. It is up to the template to decide to issue an abort command. BOB will not permit to send specific commands to OS for this particular template (see section 2.3).

OBs and templates can also get a “breakpoint” as long as they are idle (i.e. not running nor finished). This is achieved by clicking with mouse-button-3 on the label of the particular OB/template on the canvas. This clicking works as a toggle, i.e. the breakpoint is removed by clicking again. Putting a breakpoint on an OB means it will enter into “pause” before any of its templates starts. Putting a breakpoint on a template means it will enter into pause before the template script is started.

Pressing again mouse-button-3 on an OB or template with a breakpoint will change this breakpoint into a “skip”-flag. I.e. if during execution such a flag is seen, the OB resp. template will be skipped. A third mouse-button-3 press on such an item will put it back into its “normal” idle state.

Note that if more than one OB/template is currently selected, this toggling between “normal” , “breakpoint” and “skip” will be applied to *all* selected OBs/templates, if the right mouse button is clicked while the mouse

pointer is over *any* of these selected objects. The instructions on how to select multiple OBs/templates can be found in section 2.10.

When a template is executing, it can write some log messages to the template text-box (lower part of panel). It should at least indicate what the important actions are. The start of an exposure is automatically intercepted by BOB and a corresponding log-message is displayed. Every template keeps record of its own text, and this can be displayed by clicking with mouse-button-1 on the template label.

If during the execution of the template script an error occurs within this script, a message dialog box will pop up indicating this. If the user-level is set to "advanced", this message will also contain an error stack trace.

Foldable items (i.e. items which contain more viewable information) are indicated on the canvas with a small triangle, which at the same time indicates the status of folding. The (un)folding is simply performed by clicking with the mouse-button-1 on this triangle (see Figure 4).

OBs and templates can be marked with:

- a STOP sign; only possible if not yet executed
- a reject sign: to flag that the OB/template needs to be skipped
- an arrow in flashy colours: the OB/template is being executed
- a dark blue arrow: the OB/template has been paused.
- a green diamond: all went fine - full completion.
- a skull: the OB/template was aborted. Remark that aborting a template leads immediately to the abortion of its OB.
- a red flag: some error was returned by one of the templates. This is usually because OS returned an error. When this happens, an error dialogue box will pop up signalling this problem.

Once an OB is finished (either successfully or with error), it cannot be executed again. For the time being, there is a "Reset status" action button, which will reset the status of all OBs and templates on the canvas, so they can be re-executed, while maintaining the flow control settings (breakpoints and skip flags). To start an OB fully afresh, i.e. without flow control flags as set before, one should instead re-load the OB.

If there are more than 1 OBs loaded, and when BOB is idle, the next OB to execute can be selected by clicking mouse-button-1 on its label.

There can be several FITS ops-logs during the running of an OB, e.g. when it starts and stops, and also when a new OB gets loaded. Remark that this requires the OBS and TPL dictionaries to be loaded into logManager - this is not done by bob itself. On top of the ops-logs generated automatically by bob at particular occasions, template scripts themselves may also create ops-logs - see section 2.7.2.

2.7 RULES, TIPS AND TRICKS FOR TEMPLATE SCRIPTS

As all the other files and formats involved in the execution of OBs, template scripts need to comply with a certain set of rules which are outlined in documents [1] and [4]. Based on these assumptions, the search for a template script that gets called for a certain template takes place as follows:

1. The keyword `TPL.ID` in the OBD allows to determine the name of the template signature file. The latter needs to have a `.tsf` extension. This template signature file is then searched for in a number of directories under `$INS_ROOT`. Remark that the setting of the keyword `TPL.MODE` in the OBD can influence this search - see [4].
2. The template signature file must contain the keyword `TPL.PRESEQ`. This keyword contains the filename of the template script, excluding the pathname, and with or without the `.seq` extension. Remark that the

template script needs to be stored in a file which has a `.seq` extension. Although most of the times the basename contained in `TPL.PRESEQ` will be the same as the one from `TPL.ID`, this is not mandatory. The label of the template shown on BOB's canvas will contain both if they are different.

3. The script file is located searching a set of directories based on the environment variable `INS_ROOT`. Again, the value of `TPL.MODE` has an impact on the paths searched.
4. This script file is sourced in before a template starts its execution; the basename of that file is assumed to be identical to the procedure-name contained in it.
5. To execute the template, the procedure with name as given in 4. gets executed

2.7.1 Template calling procedure / access to variables

Templates must be written as Tcl-procedures. There are no arguments passed to them[1]; BOB takes care that these procedures have automatically access to the keyword-variables as local arrays, as explained below.

The information contained in the Observation Block Description in so-called Parameter File Format is parsed by BOB, and put into arrays for easy access. E.g. the OBD-line

```
DET.WIN1.BINX    "1";
```

will result in a variable `DET(WIN1.BINX)` with value 1.

These arrays are automatically made accessible to the template script[2] (but not its library procedures it uses - see below). Remark that only the arrays for which keywords are within this OB defined for the current template are made accessible, plus the **DPR** array. The **TPL** array contains the TPL parameters set by the Observation Block Description plus the ones defined in the signature file (e.g. **TPL(REFSUP)** for the instrument reference setup file).

The following rules/guidelines should be observed when accessing these arrays:

1. While template scripts have automatic access to these arrays, library procedures must use the convenience procedure `seeBobVars` (see section 2.7.2 item 10.) before dealing with these arrays as local variables.
2. Keywords specified in the OBD (as received from OH or as read from a disk-file) and/or the template signature file should be considered strictly read-only, i.e. the template script should not write into the corresponding keyword elements, nor should it unset any of these keyword elements.
3. Same for a specific list of keywords generated by BOB itself: TPL(START), TPL(EXPNO).
4. Template scripts are allowed to add elements to these keyword-arrays and write into them.

BOB sets traces on all these category-arrays when in engineering mode (see 2.10). This permits to check if all keywords declared in the OBD and/or TSF are effectively used by the template script, and do not get overwritten or unset. Remark that the category-arrays created by BOB are only guaranteed to exist during template execution, i.e. one cannot keep a reference to such an array after the template script finishes, in particular indirect references created via `upvar` commands.

By means of the mentioned tracing mechanism, BOB will report within the Template-log window the following "infractions" against the above rules:

---

1. BOB-versions up to 1.108 required 2 arguments, namely first the list of category-names and second a reserved argument. It also required that each template script takes care itself of making these category-arrays (which are passed by reference) locally accessible, by means of `upvar` instructions. Newer versions of BOB still honour this scheme, for backward-compatibility reasons, although its use is no longer recommended.
2. Only if no arguments are passed to the template script - see the previous footnote.

1. Keywords in the OBD/TSF which are not used by the template script
2. Infractions to rules 2.. and 3. above
3. Keyword-arrays that have been made accessible via an own, internal `upvar` scheme (i.e. not using `seeBobVars`), whereby the original category array-names were linked to variables of a different name.

When the `tplInternals` flag is set (see sections 2.3 and 2.11), there will be additionally the following messages, if applicable:

1. Warning that keywords are linked from a different namespace. Although not forbidden, this requires imperatively that these links are removed before the template finishes - otherwise there will be persistent links to variables of deleted objects.
2. The list of category-array elements created by the template script.

Although BOB can in principle ensure that keywords are read-only, it will not enforce this (for backward compatibility reasons); apart from logging messages as described above, BOB will continue as normal.

It is mandatory for every template script that contains some hardcoded exposures (i.e. the amount of which is unknown to BOB), to set the total number of exposures in the variable **TPL(NEXP)**, *before starting the first one*, and to adjust it during the execution as this value gets affected by runtime conditions. E.g. for a template which loops SEQ(NEXPO) times over an exposure setup and execution, and has three more hardcoded exposures, we need

```
set TPL(NEXP) [expr $SEQ(NEXPO)+ 3]
```

This is for BOB the only way to know how many exposures there will be in total. BOB uses this value to show it in the message next to the OB-identifier ("(exposure X of nnn)") and in the Template Log when it starts executing an exposure ("Starting exposure X of nnn"). By default, BOB assumes there is only one exposure in a template script.

2.7.2 Communication and logging

There are basically 10 convenience-procedures available from BOB:

1. **sendCmd**

   This has to be used for whatever communication with OS. A direct call via seq_msgSendCommand is not allowed. Remark that BOB tries to filter out three particular command-names, as they require and receive a special treatment:

   a. START
   When BOB sees this command, it will - before forwarding this START to OS - insert a SETUP command to pass on all OBS and TPL keywords that should be written into the FITS header of each exposure. One of these keywords is TPL.EXPNO, i.e. the sequential number of this exposure within the template (with the counter starting from 1).

   b. START_NO_OS
   If for particular applications (like engineering) the above mentioned insertion of an additional SETUP command would be disturbing, the START_NO_OS command should be used instead of START.
   A typical scenario for this is when BOB is used in engineering mode straight with DCS. Of course DCS does not have the OBS and TPL keywords in its dictionary, and consequently would complain if it receives them. Using the START_NO_OS instead, there is no such problem. One could actually say that BOB together with the proper OBs and templates behave in this case (to some extent) as a replacement for OS.

c. WAIT
This command is simply filtered to detect when to display messages in BOB that the current exposure was finished (i.e. after the final reply to WAIT came in).

2. **tplLog**

   This allows to log some messages in the template text-box at the lower part of the BOB GUI. Remark that this will also be recorded into the automatic log (stored in `$VLTDATA/tmp/bob_<procNameOS>.log`), like all other messages that end up in the Template Log-area .

3. **checkAbortFlag**

   A procedure to check if BOB's ABORT button was pressed. It will automatically generate an "ACK ABORT" error message if the abort flag was set, otherwise it returns with an empty string

4. **setResult**

   A procedure which allows to set a variable to a certain value, so that it can be picked up by another template in the same OB, using the companion procedure *getResult* (see below). Remark that all such variables are erased when the first template of an OB starts it execution. In other words, with this procedure result-passing from one template to another is only possible within a single OB.

5. **getResult**

   The companion procedure of *setResult* - see above.

6. **setCallBack**

   To set the script to be evaluated upon termination of the current template, thereby overriding - for this template only - the setting in the configuration panel of the default template call-back script.

7. **sendObsKeys**

   To send the OBS and DPR keywords, plus some TPL keywords, to OS. This is normally done automatically by bob when a "START" command is given within the template; however, there are cases where data can be produced without starting an exposure, i.e. without a START command. The "sendObsKeys" procedure provides for these cases a means to still have the proper OBS-, DPR- and TPL-keywords recorded into the FITS header. This proc should only be called once per data file, and only if there is no START command involved.

8. **finishOB**

   To indicate that the entire OB should be finished immediately after the current template terminates, i.e. that the remaining templates should be skipped. Optionally, the OB termination status sent to OH (default: TERMINATED) can be indicated as an argument to this procedure. Of course, this needs to be consistent with what OH can expect for this instrument, i.e. with what is written in the instrument summary file.

9. **pafReady**

   To send an event to OH, signalling the readiness of some PAF-files. OH can after receiving this event decide to retrieve these files at a later point in time. Remark that this event will always be sent to OH, independent from BOB's configuration setting for the sending of the automatically generated OB/TPL status events.

10. **seeBobVars**

    This makes a series of variables, created/owned by BOB, accessible as local variables to the invoking template library procedure. Currently, the only variables are the category arrays, as described in section 2.7.1. Remark that these variables are made automatically accessible at the level of the template script itself, so calling `seeBobVars` at the level of the template script itself will lead to errors, complaining about already existing variables.

This procedure can also take the optional argument "-localcopy <categories>", in which case the list of <categories> will appear as local arrays.

This procedure can be called from any namespace, thereby making these BOB-vars available in that namespace; however, if called in a different namespace, the template script will have to ensure that these links do not exist beyond the execution of the template script (to avoid the existence of references to variables of deleted objects), e.g. by deleting this namespace; this problem does not exist if using local variables within a procedure, as these are non-persistent.

The technical details about these ten procedures and how to use them (i.e. the number and meaning of their argument(s)) are given in the bobTPL manpage (see chapter 4, under the heading *CLASS PROCEDURES*.

If a parameter-value to be sent to OS is a string containing quotes, it has to be passed on to sendCmd with the quotes well protected, by either surrounding them with braces or escaping them with backslashes. Examples:

```
sendCmd $timeout SETUP -expoId $id -function CAT.PAR {"my value"}
```

or

```
sendCmd $timeout SETUP -expoId $id -function CAT.PAR \"my value\"
```

The above list of procedures does not consider "basic" Sequencer commands or library procedures which are available to any Sequencer script, and which may be very useful for logging purposes (e.g. `seq_logFitsAction(n)` - see [2]).

2.7.3 Aborting a running template script

As BOB is not fully aware about the detailed status of the execution of a template and its exposures, BOB cannot decide when to abort an exposure after the "Abort" button has been pressed. For this purpose, a variable is accessible within the template, flagging the pressing of this abort button. This variable is accessible only after issuing the following command in the template script:

```
upvar abort localAbortFlag
```

The template script should then at appropriate times test this localAbortFlag (0 means no abort, 1 is abort pending), and take appropriate action if necessary. When it has finally take action, it should also return an error as follows:

```
error "ACK ABORT"
```

Alternatively template script can use the `checkAbortFlag` convenience procedure to deal with the abort-flag (see item 3. in section 2.7.2). As this method is for template developers far easier to use and less error prone, it is clear that this is the preferred way to test the abort-flag. The above mentioned direct use of the abort-flag is still supported, but only for backward-compatibility reasons.

Remark that BOB allows to configure the set of commands that should no longer be sent to OS once the abort flag has been raised. This can indeed speed-up the abortion of the template script. At the same time, it allows to still send some critical commands that always should be sent to guarantee a graceful termination (leaving the instrument/telescope in a safe state).

2.7.4 Return values

Template scripts should return the following way, with proper values:

- **`return "<RA> <Dec>"`:** for successfully terminated acquisition templates, with <RA> and <Dec> the coordinates of the acquired object (both in degrees, as a float number, identical to how the primary FITS keywords RA and DEC are written)

- **`return {}:`** if all went OK, for normal templates; acquisition templates should return a non-empty string, containing the coordinates of the acquired object.
- **`error "message"`**: if there is some error within the template, or an error returned from the OS-level (e.g. ABORT). The setting of the ABORT-flag within BOB should be acknowledged by an **`error "ACK ABORT"`** message at the first opportunity.
  If a template returns an error, BOB will try to pop-up the CCS error display, to allow error-stack browsing, provided the user-level is configured to "advanced". The error display can under any circumstance be forced to pop-up, via the *Errors* menu item.

2.7.5 Library scripts

Application/instrument specific Tcl libraries can be declared via the configuration panel (see 2.3), provided they have been built according to the VLT software standards, i.e. using `vltMake`. This means template scripts can be kept relatively simply, putting the more complicated and frequently used sections into library procedures by default unknown to BOB.

Remark however that these library procedures will of course be called from bob's template class namespace (i.e. ::BobTPL), while the template script is executed; to avoid namespaces problems, in particular when these procedures try to call on their turn public template class procedures like `sendCmd`, it is best to declare them within the ::BobTPL namespace. Example:

```
namespace eval ::BobTPL {
    proc myProc {arg} {
        ....
        sendCmd $timeout $cmd
        ....
    }
}
```

2.7.6 Search paths

BOB needs to access a series of auxiliary files, most notably the template scripts and signature files. The locations where BOB scans through to find them, are listed in this section.

2.7.6.1 Template signature files

An OBD can include for every template the TPL.MODE keyword, which is in fact extracted from the template signature file at the time an OBD is constructed. If it does, i.e. if the instrument mode is defined, the search path will be:

1. `$INS_ROOT/$INS_USER/<insMode>/TEMPLATES/TSF`
2. `$INS_ROOT/$INS_USER/<insMode>/SEQUENCES`
3. `$INS_ROOT/$INS_USER/COMMON/TEMPLATES/TSF`
4. `$INS_ROOT/$INS_USER/COMMON/SEQUENCES`

In case the environment variable INS_USER was not set, $INS_USER will be replaced by SYSTEM. Paths 2. and 4. are there for backward compatibility purposes only: users are encouraged to put their template signature files in either path 1. or 3.

2.7.6.2 Template scripts

A template signature file can contain the filename of the associated template script in the variable TPL.PRESEQ. If this field is blank, the name of the sequencer script will be derived from the TPL.ID. Also, the signature file should include the TPL.MODE keyword, describing what the instrument mode for this template is. The search path for the template script will be:

1. `$INS_ROOT/$INS_USER/<insMode>/TEMPLATES/SEQ`
2. `$INS_ROOT/$INS_USER/<insMode>/SEQUENCES`
3. `$INS_ROOT/$INS_USER/COMMON/TEMPLATES/SEQ`
4. `$INS_ROOT/$INS_USER/COMMON/SEQUENCES`

In case the environment variable INS_USER was not set, $INS_USER will be replaced by SYSTEM. Paths 2. and 4. are there for backward compatibility purposes only: users are encouraged to put their template script files in either path 1. or 3.

2.7.6.3 Sound files

1. The subdirectory sounds of the current working directory (i.e. when BOB was started)
2. The subdirectory sounds of the parent of the current working directory
3. The subdirectory sounds of `$MODROOT` (if defined)
4. The subdirectory sounds of `$INTROOT`
5. The subdirectory sounds of `$VLTROOT`

2.7.6.4 Template library scripts

These are directories named as `lib<name>.tcl`, which contain a set of Tcl-procedures and a corresponding `tclIndex` file, as generated by the VLT Makefile (see sections 2.3 and 2.7.5). A list of such libraries can be specified via BOB's configuration panel. Remark however that in this entry field only `<name>` must be given, excluding the `lib`-prefix and `.tcl`-extension. In this case, the corresponding Tcl-library directory `lib<name>.tcl` will be searched for in:

1. The subdirectory lib of the current working directory (i.e. when BOB was started)
2. The subdirectory lib of the parent of the current working directory
3. The subdirectory lib of `$MODROOT` (if defined)
4. The subdirectory lib of `$INTROOT`
5. The subdirectory lib of `$VLTROOT`

2.8 SIMULATION

For the purposes of the implementation of template scripts, it is useful to have some simulation, resulting on the one hand in BOB being independent from the other hard- and software, and on the other hand in a predictable, prompt reaction when a command is sent. There are actually two levels of simulation provided by BOB, as described below.

2.8.1 Internal simulation

This level is set by setting the "Simulating TPL communication" to true (see Configuration - section 2.3). In this case the template will no longer communicate with OS. Instead, BOB will short-circuit this communica-

tion in the sendCmd procedure, and normally return after a certain delay with a reply equal to "`OK SIM`".

This delay is also configurable: a first group of commands can have a certain delay, and all the other commands get a different delay. Typically the commands that reflect the execution of an exposure would have a somewhat longer delay (set in first group), and most other commands can get a much shorter one. See also section 2.3.

If any of these delays is set to a value larger than the time-out value the template passes on to sendCmd, a time-out error will occur. In this case the error reply will contain "simulated reply timeout".

2.8.2 External simulation

This involves two external processes: simOS and simOH. This level of simulation is intended for development purposes of BOB itself, and is normally not needed by template developers. See [3] for more details.

2.9 STATUS RETURNED

If in the configuration panel both the process name and environment name of the OH-partner are filled in, BOB will return information to this partner, telling in what state the Observation Block and/or Template are. The following events are fed back:

2.9.1 Observation Block Events

All messages have the format

```
<OBSid> yyyy-mm-ddThh:mm:ss <status> <add_params>
```

where <OBSid> is the observation block identifier; <status> is one of the following:

- **ABORTED**; if the abortion is due to a user-intervention via BOB's GUI, `<add_params>` is a string containing a description of the motive, as entered in the abort dialogue-box. If the abortion is due to an error returned by the template script, `<add_params>` will contain "`template error: <errorText>`".
- **STARTED**
- **PAUSED**
- **CONTINUED**
- **TERMINATED**: in case of successful termination
- **MUSTREPEAT** followed by a message why the OB needs to be repeated (BOB's Repeat button was pressed).

2.9.2 Template Events

All messages have the format

```
<OBSid> <TPLid> yyyy-mm-ddThh:mm:ss <status> <add_params>
```

where <OBSid> and <TPLid> are the observation block resp. template identifier; <status> is one of the following:

- **ABORTED**; if the abortion is due to a user-intervention via BOB's GUI, `<add_params>` is a string containing a description of the motive, as entered in the abort dialogue-box. If the abortion is due to an

error returned by the template script, `<add_params>` will contain `"template error: <errorText>"`.

- **STARTED**
- **TERMINATED**; for acquisition templates this is followed by the coordinates (right ascension and declination, in that order) of the acquired object, (both in degrees, as a float number, identical to how the primary FITS keywords RA and DEC are written).

2.10 ENGINEERING INTERFACE

BOB's menubar contains an item to change the interface from *Observation* (default) to *Engineering*. In the latter mode, one still has access to the normal buttons, but there is more functionality available: one can edit the OB and the template parameters, partially in a graphical cut-and-paste way (see Figure 5). The function

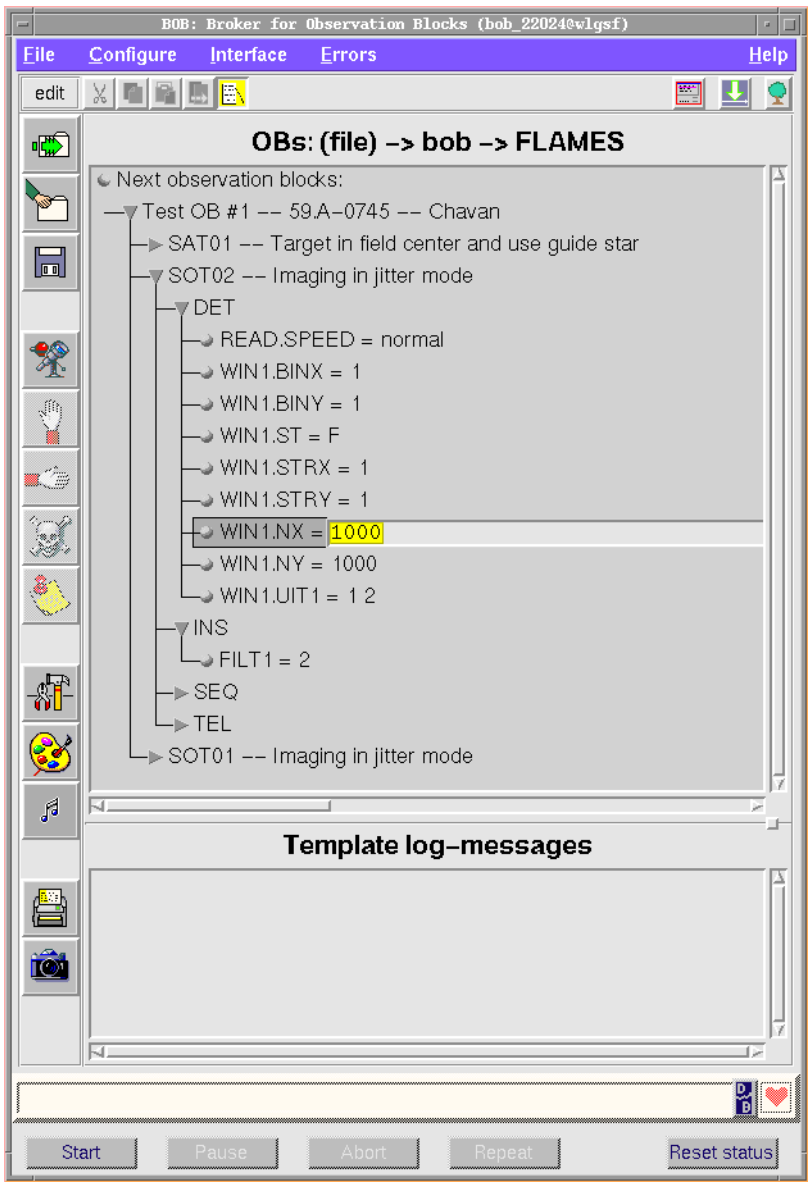

Figure 5: An example OBD loaded into BOB, displayed via the Engineering interface.

to execute is pre-selected by clicking one of the small buttons below the menubar, and this action will be per-

formed on *all* currently selected OBs/templates which are marked by a shaded reactangle, by pressing the middle mouse button on *any* selected item. Alternatively, if one wants to affect only one particular OB, template or section of parameters, there is no need to select it first, i.e. one can immediately press the middle mouse button on this unselected object.

The selection of operands (OBs or templates) is done with the left mouse button, in conjunction with the <Shift> and <Ctrl> modifier keys: a simple click will select the object under the mouse pointer, after de-selecting all other currenlty selected objects (if any); pressing the left mouse button while the <Ctrl> key is pressed will add the object under the mouse pointer to the selected objects, if it wasn't selected yet, or de-select it when it was already selected; when the <Shift> button is pressed, clicking the left mouse button will remove or add all objects between the previous selection and the one under the mouse pointer, depending on whether they were already selected or not. Note that all selected objects need to be of the same type, i.e. BOB will not permit to select OBs together with templates.

If parameter-editing is on, clicking this middle mouse button on a single parameter will allow to edit the value straight on the canvas, via a standard entry widget with the usual key-bindings. Pressing the `<Enter>` key is required to accept the value, while `<Esc>` will cancel this editing. If on the other hand the middle mouse button is clicked over an OB or template label, or on a keyword-category such as `TEL`, a new mega-widget will pop up allowing to modify the entire set of corresponding values (see Figure 6). When the edited values are

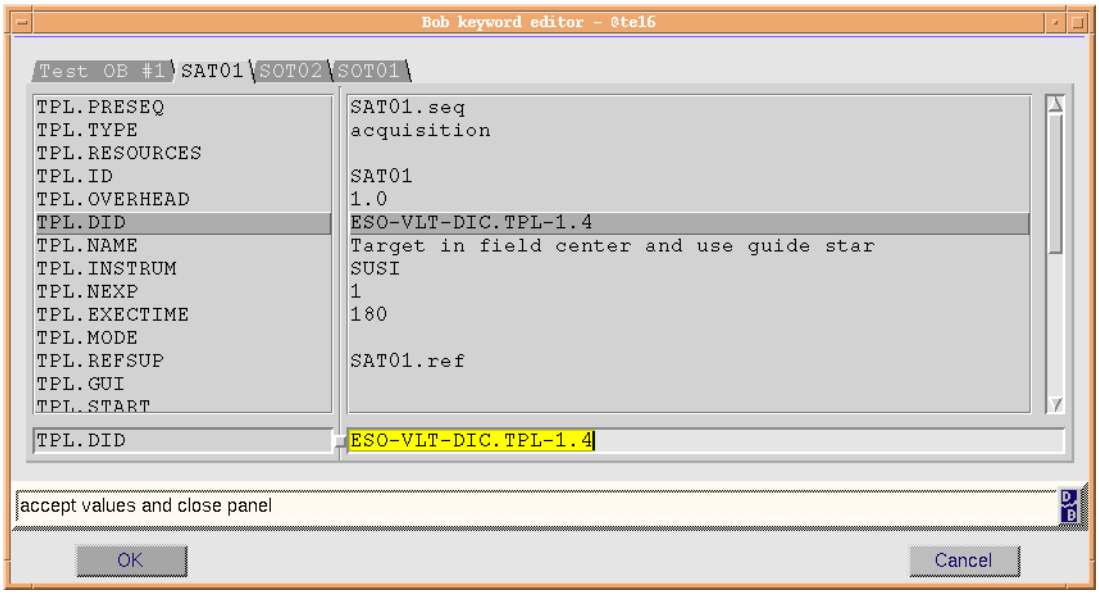

Figure 6: The megawidget for editing template-parameters.

accepted, this section of the OBD will again be verified using the same procedures as for a newly read-in OBD, labels on the canvas will get updated, etc.

The selection of this Engineering interface will be password protected once the *Trusted mode* option has been disabled. This is to provide some basic level of security against accidental modification of OB parameters by people who are not (yet) very comfortable with this engineering interface; such parameter modification, in particular for science OBs at the telescope, can invalidate the entire data-flow process for this OB. On the other hand, engineers building templates and exercising their software and instrument may find the engineering interface a useful tool, providing a convenient and direct means to edit an OB.

The password can also be given on the commandline when BOB is started up (see 2.2). This password is requested every time a switch is made from Observing mode to Engineering mode, once the "*Trusted mode*" under the *Interface* menu has been disabled (it is enabled by default). A newly installed BOB will have an

empty password, which should after installation be modified to a proper value, using the "`Change passwd..`" menu-item.

Under this interface, the template log can contain error messages about some keywords which have not been touched by the template script, although they are defined in the Observation Block Description and/or the Template Signature File, so they should be sent to OS. This is of course to be seen as an error in the template script.

More explanation about this Engineering interface can be found in [3].

2.11 PROGRAMMATIC ACCCESS

The initial implementation of BOB considered its interactive use as a GUI only. It became quickly clear however that there is a definite interest in programmatic access to BOB's functionality, be it by sending it CCS messages, or alternatively by starting it up as a shell passing it a script to execute, or even running it interactively in prompt-mode. Consequently, there was first a (somewhat limited) shell implementation of BOB, called *bobWish*, followed by enabling the sending of commands (e.g. to start an OB) via CCS to the standard BOB, and the addition of the <`scriptFile`> run-string parameter (see also section 2.2).

This means that alternatively to using the standard BOB GUI, commands can be given in a typed, interactive form or grouped together in a script, using a set of specific *bob(Wish)*-commands. This enables the creation of different applications with a BOB-like behaviour (based on these *bob(Wish)*-commands), or even to execute OBs via modem connections without having to deal with the X-traffic generated by BOB's GUI - as explained below. A useful and typical application of bob(Wish) is a bob(Wish)-script dealing with the automatic execution of one or more OBs as part of a non-interactive maintenance procedure.

In the following section a list of publicly available commands are listed, giving access to most of BOB's settings. This is the only supported API for retrieving or modifying these settings.

2.11.1 Commands

The following list contains the commands available at script level from bob and bobWish; calling these procedures has the same effect as interactively executing the corresponding GUI-function (like e.g. pressing the "Start OB" button or calling the configuration panel):

- **configuration/preferences commands**:

  `bobProcNameOH, bobProcNameOHlist, bobEnvNameOS, bobEnvNameOSlist, bobEnvNameOH, bobEnvNameOHlist, bobProcNameOS, bobProcNameOSlist, bobUserLevel, bobRunAll, bobAutoUnfold, bobGetNext, bobAutoCreateTpl, tplEvents, tplSim, tplVerbose, tplInternals, tplSimList, tplSpeDelay, tplDefDelay, tplSkipList, tplCallBack, bobCallBack, bobCfgCallback, tplLibList, bobCfgFile, bobPrintCmd, bobPlugin, bobPluginFlag, bobTitleFont, bobCanvasFont, bobLogFont, bobDialogFont, bobEditorFont, bobSoundOn, bobSoundVolume, bobSoundHeadphones, bobSoundSpeaker, bobSoundLineOut, bobErrorSound, bobWarningSound, bobBreakpointSound, bobTplSound, bobObSound, bobWaitOptions`

  These commands correspond with the fields in BOB's configuration and preferences panel (see 2.3 resp. 2.4). When given without arguments, they return the current setting. They can also be given a single argument, in which case this value will be *set* into the corresponding configuration variable.

  Remark that modifying a configuration setting can trigger a call-back (see section 2.3). The name of the configuration variable that will be passed to this call-back corresponds to the commandname used to

retrieve or set this parameter, e.g. using '`bobProcNameOH myScheduler`' to modify the OH process name will trigger the configuration call-back with additional argument '`bobProcNameOH`'.

- **action commands**:

  `bobStart, bobPause, bobContinue, bobAbort, bobRepeat, bobReset, bobNextObsBlocks, bobSave, bobSaveAs, bobSoundOn, bobPrint, bobSnapshot, bobInfo, bobSaveConfig, bobSaveConfigAs`

  Most of these commands correspond to GUI buttons, and do not require any arguments. There are three notorious exceptions to the latter rule. First the `bobNextObsBlocks` command, which can have as arguments either an integer number (for the number of OBs to download from OH) or the word "`file`" optionally followed by a filename (to avoid the file-selection box to pop-up). Second resp. third exception are the `bobSaveAs` and `bobSaveConfigAs` commands, which can have a filename as argument (also to avoid that a file-selection window pops up).

- **status commands**:
  - `audioFailed`: boolean, indicating whether an attempt to play an existing sound-file has failed
  - `bobVersion`: version number of bob. Remark that this is not necessarily a decimal number, as it is the version number manipulated by the cmm-tool (i.e. when dealing with a branched off version, it will be of the form i.j.k.l)
  - `configFile`: filename of the current configuration file
  - `interface`: enumerator, showing the type of selected interface; 0 = Observation mode, 1 = Engineering mode
  - `interpreter`: gives the Tcl-interpreter name, i.e.. `bob` or `bobWish`; this permits scripts to distinguish between a graphical and non-graphical interface
  - `obdFile`: filename (absolute path) of the loaded technical OB, or empty if the OB was retrieved from OH.
  - `obdSource`: string indicating the source of the OB; can have the values *OH* or *file*

These commands are all declared in the namespace `::bob`. To have them available in the global namespace, one should call the procedure `bobMakeUifProcsGlobal`. Remark however that this is not recommended: to avoid name-clashes and avoid pollution of namespaces (in particular the global one), it is better to use the fully qualified commandnames, with the `::bob::` prefix.

2.11.2 bobWish

All runstring parameters which can be passed to BOB (see section 2.2) are also accepted by bobWish; bobWish retrieves its configuration the same way as BOB does (i.e. the file `~/.bobrc`, or the file specified with the `-configFile` option), and these settings can also be modified via commands. An overview of the special bobWish-commands are given at the start-up of an interactive run (try *bobWish* at your terminal to see, or have a look at the bobWish manpage in section 4.2).

The command *bobSetup* within bobWish shows the list of modifiable parameters (listed with the command name to change them). Remark that bobWish does not provide an *engineering mode*, nor any command related to this mode: as the purpose is exactly to avoid the use of GUIs and their associated X-traffic, this is considered beyond the scope of bobWish.

Using the bob(Wish)-commands listed in section 2.11.1, one can easily write a script to load and execute an OB. These scripts should be terminated with an *exit* command.

One important caveat: although during normal execution you will not see GUIs on any screen, the use of bobWish requires the definition of the environment variable DISPLAY to a sensible value, i.e. to an X-server for which you have authorization to connect (typically your X-terminal is OK, if you are already logged into it).

Remark that for some error messages a dialog window will still pop-up, and some commands, depending on the options given, may require input widgets, like a file selection box.

Once the DISPLAY environment variable is set, proceed as follows for a programmatic background session from a remote site:

- dial in to your institution and connect to your workstation.
- start your script as follows, redirecting *stdout*:

```
bobWish -- theFileNameOfYourScript > theFileNameForOutput &
```

  where:
  - *theFileNameOfYourScript* is self-explanatory
  - *theFileNameForOutput* is the file to send messages to that would normally appear on the screen during an interactive session
  - *&* is needed to make the thing run in the background and not having it abort when logging off before the script is finished.
- log off
- verify at any later stage the output in the file *theFileNameForOutput*

Remark that by default the *bobStart* and *bobContinue* commands will make an OB run in the background. In other words, control is returned to your terminal or script immediately after this command has been accepted, and actually before the OB is running. This of course permits full asynchronous control of the OB's execution (e.g. it permits to give the *bobAbort* command before the OB finishes its execution), but on the other hand requires a mechanism to check and verify the state of execution, e.g. within a script. WIthout such synchronization one may give a subsequent command too early, particularly when running non-interactively, via a script. One of these mechanisms is the OB-termination call-back (see section 2.3). For non-interactive use however, this asynchronous behaviour is normally not required. For that reason, the *bobStart* and *bobContinue* have a *-foreground* option. If this option is given, these commands will only return control as the OB finishes.

2.12 MOBS AND GENERIC TEMPLATES

2.12.1 What MOBS and Generic Templates are

MOBS (**M**ultiple **Ob**servations **S**oftware) has been defined as a common part of the VLT INS software (see [4]). Its was initially presented as a separate software module, interacting with the OS of each instrument. While the task of OS is to coordinate and supervise the execution of a single exposure, MOBS's task was to offer an "easy" way to define, modify and execute a *sequence* of such exposures. The flexibility and "intelligence" offered by relatively complicated templates is many times not required at all, in particular if the goal is to execute unconditionally a set of exposures, simply one after the other, changing a couple of parameters in between. In ESO's pre-VLT software this feature was offered in the form of so-called *sequence tables*, where each row in the table contained the definition of a single exposure, and the columns contained the values for specific parameters. The usefulness and appreciation of this feature was translated into a requirement towards the VLT software to offer something similarly simple and powerful.

Templates on the other hand are of course very powerful tools to deal with a certain predefined sequence of commands required to execute one or more exposures. But their scope is limited in the sense that they are designed for very specific modes of observation. To overcome this limitation, the ICD between VCS and OH (see [1]) mentions so-called *generic templates*. This sort of templates has been inspired by the specification of MOBS: it must be possible to specify/modify *any* parameter within a given mode of observation. [1] refers to MOBS as the tool to achieve this, by means of a table editor as described above. Remark however that the use of generic templates imposes certain restrictions on the *Data Flow*: as for a generic template any instrument/ telescope parameter can be specified, it is in practice impossible to classify a priori such exposures and to ap-

ply e.g. a pre-defined reduction recipe on the resulting images.

In the implementation of MOBS within BOB there is of course a clear mapping between the MOBS-table and the generic templates: a MOBS Observation Block corresponds to a MOBS table, and each line of the table corresponds with a generic template. Through this mapping the features already implemented in BOB can be exported to the execution of a MOBS-table (alias a set of generic templates), e.g.:

- the table can be expanded (unfolded) or folded as any OB containing templates
- a breakpoint can be inserted before a particular line in the table is executed
- any line of the table can be skipped during execution

Of course the availability of these and other features has been one of the prime driving factors to integrate the MOBS concepts into BOB.

2.12.2 The MOBS Engineering Interface Editor

The engineering interface editor for a MOBS-table differs substantially from the one for “normal” templates (see section 2.10). This is because a MOBS Observation Block can (largely) be presented by a table, and a table-widget is much more compact and surveyable than a set of pages in notebook.

BOB will pop up this MOBS Editor (see Figure 7) instead of the standard one if the following conditions are

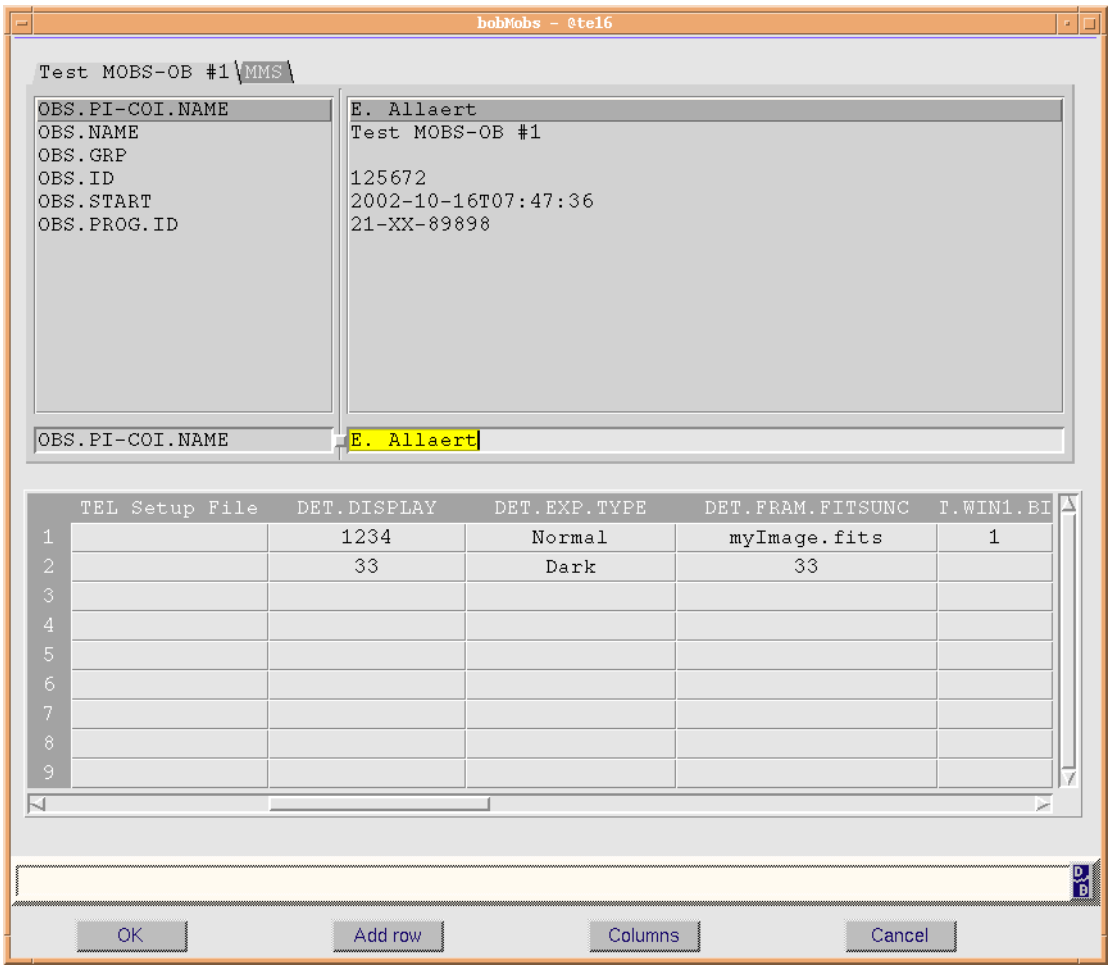

Figure 7: The MOBS-OB editor with sample data

met:

- The first template of the OB can be either an acquisition template or a generic template
- all the other templates of the OB are generic templates

By extension of the concept of *generic templates*, we will call further on such an OB a *generic OB* or a *MOBS-OB*. Such an OB is dealt with by BOB in a generic fashion, independent of the instrument.

The MOBS-OB editor contains 2 distinct parts, and the top one is already familiar: it is the tab-notebook style widget as used for editing “normal” templates (see section 2.10). The parameter which can be edited in this part are:

1. The OB parameters like OBS.ID
2. The parameters for the acquisition template, if and only if the first template is of this type

Below this is a table widget, with at as many lines as the MOBS-OB contains generic templates, complemented with a few empty (available) lines. The table has a minimum of 6 columns, in fixed positions on the left of the table. These columns are, from left to right:

- A loop counter. It tells the generic template script how many times the exposure defined on this line needs to be repeated, and its default value is 1. Remark that the full SETUP command, including reference to the various setup-files, is normally sent only once, before the first exposure starts. Subsequent exposures will be preceded by a SETUP command without any arguments, in order to retrieve an exposure-id. This standard behaviour can be modified by using a negative value for the loop counter; in this case, the full SETUP command will be repeated for every exposure.
- The timeout value in seconds for the setup command to OS which the generic template script will construct from the collection of all parameters on this line. If for a line this field is left blank, the timeout value taken by the generic template script will be the one previously defined within the same OB (possibly also in the acquisition template - see section 2.12.4 for an example). If no such parameter has been defined, a default of 30 seconds is applied.
- The name of the reference setup file for this observation. This will determine the list of parameters which can appear as additional columns (see further).
- The partial setup files for INS, DET and TEL setup.

The existence of any other columns beyond these six, and their meaning, depends on the contents of the Observation Block Description. The MOBS-OB editor will show every keyword it has found in this description as a separate column. It is possible to add new columns, or remove some already defined by pressing the *Columns* action button, and selecting the set of additional keywords to display. When the *OK* button is pressed, only the keywords and values of the columns displayed are saved, i.e. not the ones of the columns which were removed.

2.12.3 The Generic Template Script

The execution of a MOBS-OB is based on the generic template script. This script is not instrument specific, and is therefore picked up form a central place rather than from a directory under *$INS_ROOT*. It is supplied and installed together with its template signature file as part of the bob software module. Once bob was installed, the generic template script is accessible.

The tasks executed by the generic template script are as follows (in the order as listed):

- determine a timeout value (see section 2.12.2)

- determine the name of the reference setup file. If for this generic template no reference setup file was given, use the one defined before within the same OB.
- construct a SETUP command, including the name of the reference and partial setup files, and the other keyword-value pairs which are part of the observation block description for this template.
- send this SETUP command to OS. For this purpose the *sendCmd* is used (see section 2.7.2), with a timeout as calculated before. OS will return an exposure-Id.
- Use this expoId to START the exposure, using the same timeout
- Use this expoId to WAIT for the exposure to finish - this time with a sufficiently large timeout, and return

2.12.4 An example acquisition template script for MOBS

The first template of a MOBS-OB can be either an acquisition template or a generic template. One possibility is to define within such acquisition template a mode of observation, i.e. a reference setup file, and not worry about it any further for the rest of the OB. On top of that, we could at that stage already define a default value for the timeout of the SETUP command.

The contents of a template which does nothing more than that is given below. This template, called MMS, has 2 parameters of category SEQ, namely SEQ.SETUP.FILE and SEQ.SETUP.TIMEOUT. The corresponding template signature file and script are both installed under $VLTROOT/templates/forBOB/mobs/ as part of the installation of the bob software module.

```
# Example acquisition template script
proc MMS {} {
    setResult refSetupFile $SEQ(SETUP.FILE)
    setResult timeout $SEQ(SETUP.TIMEOUT)
    tplLog "MMS finished"
    return {}
}
```

2.13 TECHNICAL TEMPLATES

Up to this point we have considered the use of BOB as a tool to execute "real" OBs (imported from the Data Flow world). OBs contain templates, and each of these templates contain instructions for the OS process of the instrument it is controlling. This communication with OS is taken care of by BOB.

BOB is actually not interested in knowing whether the OS-process it is talking to is really an OS, or any other process that responds properly to the commands it receives. This means we can have templates designed e.g. to talk to ICS, combine them together and declare their arguments in an OBD, and have them executed by BOB. Of course the rules imposed by BOB for use of templates, and template signature files, for dealing with hierarchical keywords, return values etc. must be respected (see also section 2.7). As long as this is the case, one can benefit from the features offered by BOB also for tasks which in reality have very little to do with Data Flow and "real" OBs. For that reason, this type of templates is qualified as *technical templates*.

As an example, let's assume we want to send a single SETVAL command to a process. The parameter of this command is supposedly a single integer, in the range 100-1000, and we want to pass this parameter to the template as an argument. We can easily create a template signature file[1] for that purpose:

```
PAF.HDR.START            ;        # Marks start of header
```

1. see document [1] for more detailed info on the syntax, structure and content of template signature files.

```
.....
PAF.HDR.END

# The next set of keywords (up to TPL.RESOURCES) are mandatory.
# The assigned values need to be adjusted for each signature file.
TPL.INSTRUM             "TEST";     # This is a template for WORKSHOP
TPL.MODE                "";             # Not applicable for WORKSHOP
TPL.VERSION             "0.1 alpha";    # (any string)
TPL.REFSUP              "";             # Reference setup file
TPL.PRESEQ              "waTemplate.seq";       # Sequencer script
TPL.GUI                  "";            # GUI panel
TPL.TYPE                "test";         # science, calib, ...
TPL.EXECTIME            "computed";     # can be "computed" or a number
TPL.OVERHEAD            "1.0";
TPL.RESOURCES           "";

# Here starts the description of the various arguments of the
# template script. In this particular case, we have only one
# parameter. We need to give its type, range and default value.
TPL.PARAM                       "SEQ.VALUE";
SEQ.VALUE.TYPE                  "integer";
SEQ.VALUE.RANGE                 "100..1000";
SEQ.VALUE.DEFAULT               "555";
```

The above signature file shows there is a unique argument for the script, and it is defined with the name VALUE in the SEQ category. On top of that, it will make BOB look for a corresponding template script called `waTemplate.seq` (see value assigned to the keyword `TPL.PRESEQ`). So to keep things simple, the template signature file can be named `waTemplate.tsf` (the template script and signature files share the same basename). The template script `waTemplate.seq` could look as follows:

```
# Example Technical Template script
proc waTemplate {} {

    checkAbortFlag;  # Check if bob's ABORT button has been pressed

    # Make it look like we're talking about real OBs, so we don't
    # get any complaints from bob (in engineering mode) about
    # unused variables
    set TPL(REFSUP);  set TPL(EXPNO) 1

    # Timeout value (ms) for replies on commands to partner process
    set timeout 10000


    #  send command SETVAL
    tplLog "About to send SETVAL command ..."
    set reply [sendCmd $timeout "SETVAL $SEQ(VALUE)"]

    return {}
}
```

We can put this technical template into an OBD. In fact, the example below calls this template twice, with dif-

ferent arguments:

```
# Standard parameter file header
PAF.HDR.START   ;               # Marks start of header
PAF.TYPE                "OB Description";  # Type of parfile
PAF.ID                  "";     # Unused
PAF.NAME                "";     # Unused
PAF.DESC                "";     # Unused
PAF.CRTE.NAME         "BOB";    # Broker for OBs
PAF.CRTE.DAYTIM     "1999-04-20T11:54:34"; # Date+time of creation
PAF.LCHG.NAME           "";     # Unused
PAF.LCHG.DAYTIM     "1999-04-20T11:54:34"; # Date+time of last change
PAF.CHCK.NAME           "";     # Unused
PAF.CHCK.DAYTIM         "";     # Unused
PAF.CHCK.CHECKSUM       "";     # Unused
PAF.HDR.END             ;       # Marks end of header

# Observation Block description follows; first come the keywords
# belonging to the OB itself (up to OBS.PROG.ID).
OBS.ID                          "125672"
OBS.PI-COI.NAME                 "Allaert"
OBS.GRP                         ""
OBS.NAME                        "Workshop OB #1"
OBS.PROG.ID                     "ThisIsMyId"

# 1st TPL
TPL.ID                          "waTemplate"
TPL.NAME                        "Workshop demo template"
SEQ.VALUE                       "444"

# 2nd TPL
TPL.ID                          "waTemplate"
TPL.NAME                        "Workshop demo template"
SEQ.VALUE                       "555"
```

The only remaining task now is to configure BOB properly with its partner data: the name under which this partner process registered itself with CCS, and its environment name need to be inserted into BOB's configuration panel, in the entry fields that corresponds to a "normal" OS (see section 2.3). After this has been done, we can activate the usual features of BOB:

- load the OBD into BOB
- execute the OB, thereby sending the proper SETVAL command to the partner process
- modify the value assigned to SEQ.VALUE within the Engineering Interface (see section 2.10); in other words, this permits to edit the parameter of the SETVAL command.
- save the OBD with these modified values, giving it a filename of our choice.

etc.

2.14 PARAMFILE KEYWORDS

There has been a special keyword-type defined, called paramfile. This allows keywords to absorb the contents of a PAF-file. I.e. the value of such keyword will be ASCII text, taken from a disk file. When bob gets such a

parameter as part of an OB, it will store this text in a file with name as given by the text itself: the basename is taken from the included `PAF.NAME` keyword, while the path is taken from the mandatory `TPL.FILE.DIRNAME` keyword. If the latter is an empty string, the path `$INS_ROOT/INS_USER/MISC` will be used. For security reasons, storage of these PAF-files will only happen if the resulting path points to a directory under `$INS_ROOT`.

Remark that after storing this file to disk, bob will put the absolute pathname of this stored PAF-file into the keyword, so it becomes a relatively short string to display. This filename, rather than the complete text, can then be edited via the engineering interface. When the OBD is saved, the file pointed to will be read-in, and the complete ASCII text will be saved into the .obd file.

Bob will by default remove files stored by this mechanism, when the corresponding template gets deleted (e.g. when a new OB is loaded). In order to override this behaviour, one needs to set the boolean keyword TPL.FILE.KEEP in the paramfile-text.

Detailed explanation about the paramfile keyword-type can be found in [1].

2.15 SNAPSHOTS

The bottom toolbar button in the main panel of BOB allows to store a snapshot of the information involving BOB. When this button is pressed the following sequence of actions takes place:

- Create the directory /`vltdata/tmp/<bobProcName>.<timestamp>`
- Save the currently running or last executed *.obd file (name: `active.obd`) in this directory
- Make a copy of `/vltdata/tmp/logFile` and place it in the above directory; same for the most recent previous `logFile` (if any).
- Make a copy of /vltdata/tmp/bob_<OS>.log into the above directory
- Create a text file (name: `bobSnapshot`) containing info about the currently executing OB/template/exposure, and the template logs of the current OB.
- Add to this file the current Tcl error message stack, the complete CCS error stack, the Tcl errorInfo contents at the last template script problem.
- Prompt for a motive, and add this comment to this text file.
- Report via a popup panel the name of the created directory that contains the above snapshot

This function can also be activated via the `::bob::bobSnapshot` command (see section 2.11).

3 TROUBLESHOOTING

3.1 FREQUENTLY ASKED QUESTIONS

3.1.1 Installation

Q1 I need the latest and greatest features of bob, but I do not wish to update the entire VLT-software just for that. So can I use the current version of bob on top of a previous release of the VLT software?

It depends. New features of bob can be tightly linked to new features of the VLT Sequencer library, Tcl/Tk, the VLT CCS package or other VLT common software components, in which case a mere upgrade of bob will not give you the expected features (and may even lead to a crashing bob). So the recommendation is surely to upgrade everything at the same time, to be on the safe side. In case of doubt, consult the author.

Q2 I have a home-directory which is NFS-mounted on a HP-UX, a Solaris and a Linux box. When running bob I have some engineering mode password problems on some of these machines. How do I fix this?

This is due to the way passwords are encrypted by bob up to version 1.117 inclusive. The encryption is done with the help of the "`crypt`" utility, available only on HP-UX and Solaris, and there is no encryption used on Linux. But apart from the fact that this crypt utility is not available on Linux, its result are not identical between HP-UX and Solaris. This means that this password can only be validated on the platform it was generated.

For that reason there is since bob 1.118 a Tcl-based crypt procedure, which of course produces identical results cross-platform. So the solution is to upgrade to (at least) this version

Q3 After upgrading to bob 1.118, bob reports that I use a wrong password when attempting to enter engineering mode. Can this be fixed?

Yes - just delete the old and new password files (~/.bobpasswd resp. ~/.bob.passwd). Although bob will attempt to convert automatically the old to the new, there are some conditions under which it will fail. The easiest way out is to deleted these password files. From that point on you should have the default (blank) password.

3.1.2 Start-up

Q4 How do I run several bob-processes on one host, each with their own settings for OH and OS?

Start-up these bob-processes with the `-configFile` option, pointing to individual configuration files. See also section 2.2. You can of course use this feature as well for the case that you want to use different configurations, still talking to the same OH and OS. If you need to distinguish these various bob-processes from CCS's point of view, you should use additionally the -ccsProcName option when starting them up.

3.1.3 Template scripts

Q5 How come I cannot access OBS-keywords within my template script?

The OBS-keywords are automatically sent to OS (when a START command is given), so you should not be worried about them not getting to OS. Caring about OBS keywords is generally not the task of the template; a template can deal with all keywords which are its own (TPL) or at a lower level (DCS, TCS, ...), not with the ones which are at a higher level (OBS). Remark that giving access to these OBS keywords within the template would expose them to modification - and this is forbidden, as this risks to invalidate the entire OB. If your template really depends on an OBS-keyword, there may be something wrong either with the design of the template script, or with the classification of this keyword.

However, since bob version 1.130 one can use the "`seeBobVars -localcopy OBS`" command to get a local copy of the OBS into the context of the procedure where this is called from.

Q6 How can I find out within a template script which keywords have a .VALUE entry in the template signature file?

You cannot - at least not easily. Some keywords have indeed a `.VALUE` entry in their `.tsf` file, with the purpose of flagging to P2PP/OT that these keywords should not be edited, i.e. have a fixed value. In fact, P2PP/OT currently do not include `.VALUE` keywords into the OB they produce; upon receipt of an OB bob verifies its contents and adds some info contained only in the `.tsf` files (e.g. the `.VALUE` keywords). After this verification/combination took place, there is for a template script no way to tell the difference between a keyword which was defined in the OB by P2PP/OT versus one which had its value fixed in its corresponding `.tsf` file - unless you start of course with your own analysis of the `.tsf` file.

3.1.4 Template signature files

Q7 I want to force the user (engineer) to edit some keyword values after he loads an OBD. Can I do this by leaving the keyword out of the OBD, and by setting the DEFAULT value of a keyword to "NODEFAULT" in the template signature file?

This is an invalid approach. The value "NODEFAULT" for the `DEFAULT` keyword in a template signature file is intended to show that there is no reasonable default value (see [1]) - it is **not** a flag to anyone that this keyword should be defined after loading the OB. It has been introduced for the purposes of P2PP, which will in this case force the users to enter a value. Which means none of these keywords are left undefined in the resulting Observation Block, by the time it gets to bob.

Defining a keyword after loading a template is actually explicitly forbidden by the ICD (see [1]). All keywords must be defined in the OBD, with the possible exception of the ones with a valid (i.e. matching the corresponding TYPE) DEFAULT or VALUE keyword in their resp. template signature file. "NODEFAULT" cannot be abused as a flag to have a keyword created which is not in the OBD. The `.obd` files are there for cases where you don't have P2PP, e.g. during development/testing. Their content however *has to be identical* to what P2PP would send, i.e. their syntax has to be correct, no stories about parameters which in this case can be dropped, etc.

Bob will complain (as it should) about all parameters defined in the TSF and absent in the OBD, also if in the TSF their DEFAULT value has been declared with NODEFAULT. Bob will however create the corresponding variable, which will allow editing.

The recommended, clean solution addressing the need for an editable parameter is in any case an `.obd` file which does contain also the keywords whose value you want to change.

3.1.5 Sounds

Q8 Running bob on an audio-equipped HP X-station, bob crashed when I clicked on the "enable/

disable sounds" icon, with a segmentation fault. Is this a bug in bob?

There is a known problem on HP X-terminals with some versions of the Audio-server software, which these terminals normally download at boot-time from their server workstation. This bug makes *any* program accessing the audio-hardware core-dump. There is a patch (from Hewlett-Packard) for this Audio-server software available, which gets installed on a workstation when the normal installation procedure is followed.

Remark however that some X-terminals boot from a flash-ROM card, and consequently do not pick up this X-server update from the workstation. So make first of all sure that your X-terminal is configured to boot from an updated workstation. Ask "vltmgr" for more info / how to configure your X-terminal etc.

Q9 When I run bob from an HP-UX 10.20 box on an X-emulator or an HP X-station without audio hardware, bob crashed when I clicked on the "enable/disable sounds" icon, with a segmentation fault. Is this a bug in bob?

There is a problem with earlier versions of the "snack" audio extension (used by bob), leading to a core dump, on all HP-UX versions, when trying to access non-existing audio-hardware. This should only affect pre-MAR2002 versions, as the MAR2002 version of snack has been patched to correct this problem.

3.1.6 FITS header information

Q10 Could it be that `OBS.START` does not have the same value for all files produced by one OB?

In principle this should of course never happen. `OBS.START` is calculated when the OB is *started* (after having been initialized when the OB was *created*), and is **not** modified further on. It is completely independent from the type of templates in the OB, their sequence of execution, etc. There are however a few situations which may produce "inconsistent" values (and these are all *features* rather than bugs):

a. The acquisition template (i.e. the first template of the OB) does not send a `START` command to OS, although it does produce one or more FITS files. This `START` command is the (documented) requirement to get all these `OBS`-keywords sent automatically to OS. If you cannot live with this mechanism, you should use the convenience procedure "`sendObsKeys`" to get them across - see section 2.7.2. In the absence of this new `OBS`-info and depending on how your OS is implemented, your OS may just insert the `OBS.START` of the previous OB, which is not the value you expect.

b. You use the `START_NO_OS` command, to explicitly avoid sending of these keywords to OS. No surprises here...

c. You press the "*Reset Status*" on the bob-panel (lower right button) after executing the acquisition template, and then start the OB again, this time skipping the acquisition template. Again, the result is what has been asked for...

3.1.7 Operational states

Q11 When I abort a running exposure via the OS-panel, bob doesn't seem to realize that this happens, and flags a normal termination of the template. How can that be?

A typical template should after starting an exposure issue a WAIT command to OS. This command should only return when the exposure is finished, indicating in its reply the exposure status (as an integer, see the "`expStatus`" database attribute as defined in [4]). It is then up to the template to interpret this status, and eventually to return with an "`error`" command instead of "`return {}`". Using this "`error`" command,

the OB will be aborted. Using the "`return {}`" statement, bob will not be aware that the exposure was aborted at the OS level. bob itself does not interpret this expStatus returned by the WAIT command.

Q12 When I press the `ABORT` button on the bob-panel, the current exposure still proceeds, and only after it is finished, the template and OB get aborted. Why?

Bob's abort command aborts the currently running *template* and *OB*, it does not abort the currently active *exposure*, i.e. bob does not send automatically `ABORT` commands to OS. This is intentional, and was decided at design time. You can of course decide within a template to send ABORT commands to OS, using the `sendCmd` convenience procedure.

By the way, the template is only aborted when bob checks the abort-flag as set by pressing the ABORT button; this happens only if the "`checkAbortFlag`" convenience procedure is called within a template, and at the (normal) termination of the template - whichever comes first.

3.1.8 Engineering mode

Q13 In the template-logs, I occasionally see some messages in red complaining about keywords that have not been accessed. What does that mean?

This happens if the OB and/or template signature file contain some keywords which are not used by the template script, i.e. they are not read nor sent across to OS. This probably means there is a bug in the template script, the OB or the TSF: if a keyword gets defined, it probably needs to be accessed during the execution of the template script. Otherwise there is no sense in having this keyword.

Q14 In the template-logs, I occasionally see some messages in red complaining about keywords that have been modified or unset by the template script. What does that mean?

This happens if the template script and/or its library procedures attempt to modify or unset the keyword array element(s) listed in this message. So this message will appear whenever an existing element's value gets overwritten or unset. This points most likely to a bug in the template script: keywords are in principle read-only; they should either appear explicitly in the OBD, or be derived from DEFAULT/VALUE entries in the template signature file. The few exceptions (like TPL.NEXP) are filtered out before printing this warning message into the template-logs. See also section 2.7.1.

Q15 In the template-logs, I occasionally see some messages in red complaining about reading or writing into some gvar-variables instead of using the public interface. What does that mean?

This happens if the template script and/or its library procedures attempt to read, write or unset particular elements of the global array gvar. This array is used internally by bob to pass some information between the various panel widgets and bob itself - it is not intended for public use. To obtain the result you're after, please use the public API as documented in section 2.11.1.

Q16 In the template-logs, I occasionally see some messages in red complaining about '... keys of unknown category (due to misuse of "upvar")'. What does that mean?

This happens if the template script and/or its library procedures "hijack" the names of the categories (TPL, DET, ...), ie. if by the use of `upvar` commands these category names end up being referred to by any other name. As BOB has put traces on the original arrays, it will get a trace call-back referring to this new name; when this happens, it won't know anything about this mapping, so it won't be able to do the usual keyword

checks (see also Q13).

The template script should use the `seeBobVars` convenience procedure instead - see section 2.7.2.

Q17 I get in Engineering Mode a message in the template log, saying that the template did not use the TPL.EXPNO and TPL.NEXP keywords. Although I effectively do not use either of these explicitly in any of my templates, most templates do not complain about this. Why?

BOB takes always care of `TPL.EXPNO` (the current exposure's sequential number within the template), and sets `TPL.NEXP` (the number of exposures in this template) to a default value of 1 if it is not defined in the TSF nor OBD. These 2 keywords are passed automatically by BOB to OS when the template intends to send a START command to OS, or gives explicit instructions to forward all OBS and TPL header keywords by means of the `sendObsKeys` convenience procedure. So most likely, in the template which produces this log, there is neither a call to `sendObsKeys` nor a "`sendCmd START`".

This is all based on the assumptions for which BOB was designed, i.e. to deal with templates which start exposures, talking to an OS. This turned out to be not always the case, in particular not for technical templates and acquisition templates without exposures - which will then also produce this log. Hence, future versions of BOB could be modified to deal with this situation.

Q18 What's the purpose of editing keywords which are supposed to have a fixed value, assigned via a .VALUE entry in their template signature file?

This is for the purposes that Bob's OB-editor was created, i.e. *engineering*. So typically, if you want to tune these values, you'd edit them via the engineering mode, and after determining an optimum value you'd put this value back into the `.VALUE` entry of the template signature file.

Remark however that after the editing of an OB (or template), the resulting OB will be again checked against the `.tsf` file. Now if this modified keyword has a value which is different from what the .tsf dictates (via the `.VALUE` entry), bob will flag this by popping up a warning. This warning will appear independent of the user-level, and will stay up for 15 seconds or until acknowledged ("OK" pressed), whichever comes first. Just to make sure that this cannot ever be considered a *standard* procedure.

Q19 A technical OB with many templates failed half-way and I've restarted it after setting the already executed templates one by one to "skip". Is there a better/faster way to do that?

Yes there is: you can select an entire range of templates by first clicking the left mouse button over the first template, and then clicking over the last template while holding down the <Shift> key. This will select the entire range of templates. Release the <Shift> key, and press the right mouse button once over any of the selected templates - this will put them all into "skip". See also section 2.10.

4 REFERENCE

4.1 ERROR MESSAGES

During the operation of bob, while it analyses or tries to execute an OB and its templates, there are of course plenty of things which can go wrong: illegal syntax, values out of range, referenced files not found, etc. In the following sections you will find a list of error messages which can pop up in the form of dialogue boxes or logs; they are grouped according to the activity that is going on. Within each activity, the order is alphabetical (case insensitive).

4.1.1 OB/Template creation

Remark that some of these messages can also appear after the OB/template is edited via bob's engineering mode editor, as some of the "creation checks" are then passed through again.

Message 1

```
Could not find the sequencer script for <tplId>.
This template cannot be executed, and
you will have to reload the OB to which it belongs,
after making the sequencer script available.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says. Either the file pointed to by TPL.PRESEQ was not found, or TPL.PRESEQ was not defined and <tplId>.seq could not be found

Cure

Make the proper .seq file available and reload the OB.

Message 2

```
Could not find the template signature file
for <tplId>.  This template cannot be executed, and
you will have to reload the OB to which it belongs,
after making the template signature file available.
```

Category

error, no timeout, bob blocked till confirmed

Cause

the template signature file is searched for, based on the <tplId> name (i.e. <tplId>.tsf), eventually taking the INS.MODE into account as defined in the OB. Apparently this file is not found. It is searched for under $INS_ROOT/<insMode>/TEMPLATES/TSF and $INS_ROOT/COMMON/TEMPLATES/TSF.

Cure

Correct either the <tplId>, or make <tplId>.tsf available.

Message 3

```
Error: <message> while dealing with parameter <key>
of the <tplId> template (type: paramfile).
Please correct and re-load this OB.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the <message> says.

Cure

Let yourself lead by the <message>.....

Message 4

```
The keyword TPL <key> is missing for the template <tplId>.
It must either be in the OB description or in the template signature file.
```

Category

error, no timeout, bob blocked till confirmed

Cause

There is a set of mandatory TPL keywords, which should be defined for all templates.

Cure

Correct the .tsf or the OB, and reload the OB.

Message 5

```
The keyword TPL.MODE is not set in the template signature file,
nor in the OB Description.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says.

Cure

Correct the .tsf and reload the OB.

Message 6

```
The keyword TPL.PRESEQ is not set in the template signature file.
I will look for the template script using naming rules.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says. A search for <tplId>.seq will be undertaken, although in principle TPL.PRESEQ should be defined.

Cure

Correct the .tsf and reload the OB.

Message 7

```
The OBD-info for template <tplId> contains the keyword <key>
which is not in the template signature file <tsf>.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says.

Cure

Correct the .tsf file and reload the .obd file

Message 8

```
The PAF file with the set of OBs has an illegal format:
it contains more than one PAF.HDR.START
```

Category

error, no timeout, bob blocked till confirmed

Cause

bob has received an OB (or read an .obd file) with an illegal format, as indicated by the message.

Cure

Correct the OB and reload it.

Message 9

```
The PAF file with the set of OBs has an illegal format:
it has no PAF.HDR.START
```

Category

error, no timeout, bob blocked till confirmed

Cause

bob has received an OB (or read an .obd file) with an illegal format, as indicated by the message.

Cure

Correct the OB and reload it.

Message 10

```
The PAF file with the set of OBs has an illegal format:
there is no OBS.ID
```

Category

error, no timeout, bob blocked till confirmed

Cause

bob has received an OB (or read an .obd file) which does not contain an OBS.ID keyword.

Cure

Correct the OB and reload it.

Message 11

```
The PAF file with the set of OBs has an illegal format:
there is no TPL.ID
```

Category

error, no timeout, bob blocked till confirmed

Cause

bob has received an OB (or read an .obd file) which does not contain a TPL.ID keyword.

Cure

Correct the OB and reload it.
Correct the .tsf and reload the OB.

Message 12

```
The parameter <key> of the <tplId> template has the value "<value>",
which is supposed to be a float.
It is replaced by the default value: "<default>"
as taken from the signature file <tsf>.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout, only shown for beginners

Cause

As the message says.

Cure

Correct the OB and reload it, or edit this <key> via bob's engineering interface editor.

Message 13

```
The parameter <key> the <tplId> template has the value "<value>",
which is supposed to be a float.
Its value is either fixed (.VALUE entry) or the .DEFAULT value in the
signature file <tsf> is also not a float.
Please correct the OB and the signature file.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout

Cause

As the message says.

Cure

Correct the OB and/or TSF, reload the OB, or edit this <key> via bob's engineering interface editor.

Message 14

```
The parameter <key> of the <tplId> template has the value "<value>",
which is supposed to be a float.
```

```
There is no default value in the signature file <tsf>
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout, only shown for beginners

Cause

As the message says. Remark that this is for a <key> which has a NODEFAULT value for the DEFAULT entry in its template signature file.

Cure

Correct the OB and reload it, or edit this <key> via bob's engineering interface editor.

Message 15

```
The parameter <key> of the <tplId> template has the value "<value>",
which is supposed to be an integer.
It is replaced by the default value: "<default>"
as taken from the signature file <tsf>.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout, only shown for beginners

Cause

As the message says.

Cure

Correct the OB and reload it, or edit this <key> via bob's engineering interface editor.

Message 16

```
The parameter <key> the <tplId> template has the value "<value>",
which is supposed to be an integer.
Its value is either fixed (.VALUE entry) or the .DEFAULT value in the
signature file <tsf> is also not an integer.
Please correct the OB and the signature file.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout, only shown for beginners

Cause

As the message says.

Cure

Correct the OB and/or TSF, reload the OB, or edit this <key> via bob's engineering interface editor.

Message 17

```
The parameter <key> of the <tplId> template has the value "<value>",
which is supposed to be an integer.
There is no default value in the signature file <tsf>.
```

Category

error, 10 sec timeout, bob blocked till confirmed ror timeout, only shown for beginners

Cause

As the message says. Remark that this is for a <key> which has a NODEFAULT value for the DEFAULT entry in its template signature file.

Cure

Correct the OB and reload it, or edit this <key> via bob's engineering interface editor.

Message 18

```
The parameter <key> of the <tplId> template is set to "<value>",
which should match (any of) the pattern(s) "<patterns>".
(Warning only - no corrective actions yet in this version of bob).
```

Category

warning, 10 sec timeout, bob blocked till confirmed or timeout,

Cause

As the message says. This is for a <key> of type "filename", which should match any of the the patterns defined in the .RANGE entry of the tsf.

Cure

Correct the OB and reload it, or edit this <key> via bob's engineering interface editor.

Message 19

```
The parameter <key> for template <tplId>
is defined in the OBD as "<obd_value>",
but it is overridden to "<value>".
See <key>.VALUE in the template signature file <tsf>.
```

Category

error, 15 sec timeout, bob blocked till confirmed or timeout, only shown for beginners

Cause

There seems to be an inconsistency between what the OBD and <tsf> files contain with respect to <key>.

This message is only shown during a first pass, i.e. it will not show up if such a <key> is edited via bob's engineering mode editor.

Cure

Correct the .tsf or .obd file and reload the .obd file

Message 20

```
The parameter <key> for template <tplId>
is not defined in the OBD. It is set to the default "<default>"
as defined in the template signature file <tsf>.
```

Category

error, no timeout, bob blocked for 15 sec or till confirmed, only shown for beginners

Cause

This is for a keyword which has a default value defined in the <tsf>. This will warn beginners about this "automatic" assignment of values. Normally OBs should include this keyword, as the .DEFAULT entry in the <tsf> serves for P2PP to pick up the value and put it in the OB. This message is only shown during a first pass, i.e. it will not show up if such a <key> is edited via bob's engineering mode editor.

Cure

Correct the .obd and reload it.

Message 21

```
The parameter <key> for template <tplId>
is not defined in the OBD. There is no default value
defined in the template signature file <tsf>.
```

Category

error, 15 sec timeout, bob blocked till confirmed or timeout

Cause

This is about a key with a .NODEFAULT entry in <tsf>. Such keys should have a proper definition in the OB. This is done properly by P2PP. So we're dealing here with a local .OBD file. This message is only shown during a first pass, i.e. it will not show up if such a <key> is edited via bob's engineering mode editor.

Cure

Correct the .obd and reload it.

Message 22

```
The template signature file <tsf> assigns no value
to the <key>.RANGE keyword.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says. The .RANGE entry is mandatory, if the .VALUE entry is not given.

Cure

Correct the .tsf and reload the OB.

Message 23

```
The template signature file <tsf> assigns no value
to the <key>.TYPE keyword.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says. The .TYPE entries in the <tsf> are mandatory

Cure

Correct the .tsf and reload the OB.

Message 24

```
The template signature file <tsf> assigns no value
to the <key>.VALUE nor <key>.DEFAULT keywords.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says. Either the .DEFAULT or .VALUE entry should be defined.

Cure

Correct the .tsf and reload the OB.

Message 25

```
The template signature file <tsf> assigns the value "<type>"
to the <key>.TYPE keyword.
This is not a legal type.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says.

Cure

Correct the .tsf and reload the OB.

Message 26

```
The template signature file <tsf> has an illegal format:
there is no TPL.PARAM.
```

Category

error, no timeout, bob blocked till confirmed

Cause

As the message says.

Cure

Correct the .tsf and reload the OB.

Message 27

```
The TPL ID (<tplId>) is not in the correct format;
Check the TPL dictionary <TPL_dictionary>.
This template is illegal and cannot be coped with.
```

Category

error, no timeout, bob blocked till confirmed

Cause

the <tplId> is passed as part of the OB. It should be in the format of <context>/<name>/<version>, or <name> (see ESO-VLT-DIC.TPL).

Cure

Correct the OB and reload it.

Message 28

```
There is a problem with some <category> parameters of this template
```

```
<tplId>:
-------------------------------------------------------
<message>
-------------------------------------------------------
Please correct and reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
during population of the template tree, there were some problems with keywords of a particular <category>
Cure
Analyse <message>, correct the mistakes in the OB/tsf and reload the OB.

Message 29

```
There is a problem with some <category> parameters of <tplId>'s
signature file:
-------------------------------------------------
<message>
-------------------------------------------------
Please correct and reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
As the message says. Detected during verification of the OB with the .tsf.
Cure
Correct and reload the OB.

Message 30

```
There is something wrong with the parameters of this OB:
-------------------------------------------------------
<message>
-------------------------------------------------------
I will proceed, but you may find this OB crippled.
I suggest to check this, and eventually to reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
something went wrong with the parsing of the keywords of this OB.
Cure
Analyse <message>, correct the mistakes (in the OB) and reload the OB.

Message 31

```
There is something wrong with the parameters of <tplId>:
-------------------------------------------------------
<message>
-------------------------------------------------------
I will proceed, but you may find this template crippled.
I suggest to check this, and eventually to reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
software bug: a template object was created with arguments which lead to this error.
Cure
Analyse <message>, correct the mistakes in the code

Message 32

```
There is something wrong with this template's <tplId> parameters:
-------------------------------------------------------
<message>
-------------------------------------------------------
I will proceed, but you may find this template crippled.
I suggest to check this, and eventually to reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
something went wrong with the parsing of the keywords of this template.
Cure
Analyse <message>, correct the mistakes (in the OB and/or .tsf files) and reload the OB.

Message 33

```
<tplId>'s signature file contains keywords of category of "<category>",
which are illegal. Please correct and reload this OB.
```

Category
error, no timeout, bob blocked till confirmed
Cause
As the message says.
Cure

4.1.2 OB/Template editing

Remark that several messages which are listed in 4.1.1 can also appear after OB/Template editing, as most of the verification steps (and corresponding code) are identical.

Message 34

```
The parameter <key> for template <tplId>
is set in the TSF to the fixed value "<value>",
but you have set it to "<edited_value>".
The latter value will be used.
```

Category
warning, 15 sec timeout, bob blocked till confirmed or timeout.
Cause
Within the engineering mode editor, a <key> was edited which supposedly should have a fixed value. This message is not shown during a first pass, i.e. it will show up only if such a <key> is edited via bob's engineering mode editor.
Cure
None. This message will not appear if <edited_value> is equal to <value>.

4.1.3 OB execution

Message 35

```
Cannot abort - there is no OB active!
```

Category
error, 5 sec timeout, bob blocked till confirmed or timeout.
Cause
Apparently the "abort" button got pressed while no OB was running/paused. This should in principle not be possible.
Cure
none

Message 36

```
Cannot continue - there is no OB paused!
```

Category

error, 5 sec timeout, bob blocked till confirmed or timeout.

Cause

Apparently the “continue” button got pressed while no OB was paused. This should in principle not be possible.

Cure

none

Message 37

```
Cannot pause - there is no OB active!
```

Category

error, 5 sec timeout, bob blocked till confirmed or timeout.

Cause

Apparently the “pause” button got pressed while no OB was running. This should in principle not be possible.

Cure

none

Message 38

```
Cannot start this OB!
Its state is not idle.
Try the "Reset status" action button..
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout.

Cause

The “start OB” button was pressed, for an OB which has already (partially) been executed.

Cure

If you do need to re-start this OB, press the “Reset status” action button first.

Message 39

```
Exposure not terminated properly during execution of <tplId>.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

The OB has been aborted, due to the fact that OS has reported back an error or abort during the execution of <tplId>.

Cure

Check why OS did not manage to terminate this exposure properly.

Message 40

```
Problem during execution of $tplId:
==> <error_stack> <==
```

Category

error, no timeout, bob blocked till confirmed.

Cause

The OB has been aborted, due an error in the template script. This is most likely a bug in the template script (as it should normally be designed to trap error conditions itself, and either recover from these errors, or pass specific error conditions back to the OB).

Cure

Debug the template script.

Message 41

```
The sequencer script for <tplId> was not found
```

```
when this template was loaded.  Hence it cannot be executed,
and this is how far the OB to which it belongs gets.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

The OB has been aborted, due to the error indicated in the message. Actually, that this sequencer script was missing should have been signalled already when the OB was loaded.

Cure

Make the sequencer script for this <tplId> availabel and reload the OB.

Message 42

```
The template signature file for <tplId> was not found
when this template was loaded.  Hence it cannot be executed,
and this is how far the OB to which it belongs gets.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

The OB has been aborted, due to the error indicated in the message. Actually, that this template signature file was missing should have been signalled already when the OB was loaded.

Cure

Make the template signature file for this <tplId> availabel and reload the OB.

Message 43

```
This keyword's value is too long to edit directly on the canvas.
If you're sure you want to modify this value,
edit the entire category, template or OB instead.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

This is when bob has truncated a value on the canvas (appending trailing dots to where it truncated), because this value contained too many characters. It is error-prone to try to edit such strings directly on the canvas.

Cure

Follow the instructions contained in the message.

Message 44

```
You have to load an Observation Block before it can be started.
```

Category

error, 5 sec timeout, bob blocked till confirmed or timeout.

Cause

This is when the "start" button gets pressed before an OB has been loaded, i.e. just after bob was started.

Cure

Load an OB before attempting to press "start".

Message 45

```
You have to select an Observation Block before it can be started.
```

Category

error, 5 sec timeout, bob blocked till confirmed or timeout.

Cause

This is when the "start" button gets pressed before an OB has been selected, i.e. with multiple OBs loaded.

Cure

Select an OB before attempting to press "start".

Message 46

```
<message>
Such request is incompatible with the current OB status.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout.

Cause

Within non-interactive use (bobWish-script), there was an attempt to abort the OB while it was not in a state to be aborted.

Cure

Do not attempt to abort an OB while it is not running.

4.1.4 Selecting/Loading of OBs or .obd files

Message 47

```
Could not communicate with OH.
Check the Environment settings (under Configure menu)
and the CCS error-log.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during an attempt to retrieve an OB from OH. This is a CCS communication problem.

Cure

As the message says.

Message 48

```
Could not read file <file>. Check its access permissions.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during the selection of an OBD file. As the message says: this <file> is not readable.

Cure

Set the file-permission of this .obd file properly, and retry.

Message 49

```
Could not retrieve observation blocks from OH.
There was a problem trying to receive replies from OH.
Check the CCS error-log.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during an attempt to retrieve an OB from OH. The command requesting an OB was successfully sent, but the replies did not (all) come in.

Cure

As the message says.

Message 50

```
Got an error returned by OH. Check the CCS error-log.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during an attempt to retrieve an OB from OH. The command requesting an OB was successfully sent, but OH replied with an error.

Cure

As the message says.

Message 51

```
OH is not ready to return observation blocks.
Please prepare the schedule (select OBs) first in OH
before requesting OBs from within Bob.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during an attempt to retrieve an OB from OH. The command requesting an OB was successfully sent, but OH replied with an empty OB.

Cure

As the message says.

Message 52

```
You have selected <file>. It is of type "<fileType>",
while it should be of type "file".
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during the selection of an OBD file. As the message says: this <file> is not a plain file.

Cure

Select a proper .obd file.

Message 53

```
You have selected <file>. It seems to be a circular link.
```

Category

error, no timeout, bob blocked till confirmed.

Cause

Occurs during the selection of an OBD file. As the message says: this <file> seems to be a link pointing (ultimately) to itself.

Cure

Select a proper .obd file.

4.1.5 Various

Message 54

```
Could not locate the plug-in <plugin>".
It must be under the bin/ subdirectory of either
., .., $INTROOT or $VLTROOT (searched in that order).
Continuing without loading this plug-in.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout.

Cause

The configuration of bob tells that it should load <plugin>, but this action failed.

Cure

Make <plugin> available in one of the directories given in the message.

Message 55

```
Select OB before printing !
```

Category

error, 5 sec timeout, bob blocked till confirmed or timeout.

Cause

Apparently the "print" button got pressed while no OB was selected, i.e. there are several OBs loaded.

Cure

none

Message 56

```
You have to select an OB by clicking on it before requesting to repeat it.
```

Category

error, 10 sec timeout, bob blocked till confirmed or timeout.

Cause

The “repeat” action button was pressed, while several OBs are loaded, and none of them has been selected first.

Cure

As the message says.

Message 57

```
Your terminal either has some problem with the audio-hardware,
or doesn't have any such hardware.
Hence sounds cannot be enabled.
```

Category

error, 8 sec timeout, bob not blocked.

Cause

The “enable audio” button was pressed, but apparently there is some problem

Cure

none.

4.2 MANPAGES

bobAbort(n), bobContinue(n), bobPause(n), bobPrint(n), bobRepeat(n), bobStart(n), bobNextObsBlocks(n), bobReset(n), bobSaveConfig(n), bobSnapshot(n)

NAME

```
bobAbort, bobContinue, bobPause, bobPrint, bobRepeat, bobStart,
bobNextObsBlocks, bobReset, bobSaveConfig, bobSnapshot
    - several procedures used from GUI level to control bob
```

SYNOPSIS

```
bobAbort
bobContinue
bobPause
bobPrint
bobRepeat
bobStart ?<contFlag>?
bobNextObsBlocks <nr> ?<file>?
bobReset
bobSaveConfig
bobSaveConfigAs ?<file>?
bobSnapshot
```

DESCRIPTION

```
bobAbort: abort execution of running OB
bobContinue: equivalent to bobStart <1>
bobPause: pause execution of running OB
bobPrint: print logs of currently selected OB
bobRepeat: request a MUSTREPEAT for selected object
bobStart ?<contFlag>?: start execution of selected/paused OB; if
    <contFlag> is true, then continue after pause instead of start.
    Default for <contFlag> is false.
bobNextObsBlocks <nr> ?<fileName>?: request <nr> OBs from scheduler process.
    if <nr> is set to the word 'file', then either pop up a file-selection
    panel (if <fileName> is not set), or try to load <fileName>.
bobReset: reset objects and interface
bobSaveConfig : save the current config parameters
bobSaveConfigAs : save the current config parameters into <file>. If <file>
    is not specified, bring up a file-selection box first.
bobSnapshot : dump bob-related info and logfiles to a timestamped directory.
    The name of this latter directory is the return value.
```

- - - - - -

Last change: 23/05/05-10:02

bobSave(n), bobSaveAs(n)

NAME

```
bobSave, bobSaveAs - save the observation block description of a
      selected OB
```

SYNOPSIS

```
bobSave
bobSaveAs ?<file>?
```

DESCRIPTION

```
These procs will retrieve the OBD of the selected OB, and prepend a proper
header to it before writing it to an .obd file. There are two forms of
basically the same function:

bobSave
   saves the current OBD into a file. If the OBD was originally loaded
   from a local file, it will be saved into that same file; otherwise
   a file-selection window will pop up to ask for a filename.

bobSaveAs ?<file>?
   saves the current OBD into <file>. Remark that if <file> is given but
   does not contain an absolute path, the saving will be done with
   respect to the current working directory, which can be manipulated
   by any template.
   If <file> is not specified, bobSaveAs will pop up a file-selection
   window to select a filename.
```

- - - - - -

Last change: 23/05/05-10:02

BobPanedComboBox(n)

NAME

```
BobPanedComboBox - special paned combo-box style megawidget for bob's
specific needs
```

SYNOPSIS

```
BobPanedComboBox <widgetName> ?<option> <value>? ?<option> <value>?
```

DESCRIPTION

```
BobPanedComboBox creates an iTk-style mega-widget, designed especially
for the needs of bob. The widget consists of the following elements:
- a listbox (top left) to show typically the names of variables
- a listbox (top right) to show typically the values of variables.
  There is be a 1-to-1 correspondence between the lines of both
  listboxes, i.e. the name of a variable and its value are displayed
  in the same vertical position, and any positioning action in one
  listbox will be followed by the same action in the other.
- an entry field (bottom left), showing e.g. the name of the variable
  currently being edited
- an entry field (bottom right), showing the current value of the variable
  being edited
- optionally a scrollbar (see -scrollbar option below); moving this
  scrollbar will move both listboxes
These elements are organized in 2 paned windows, with a vertical
separation bar in the middle, and a sash between the two columns,
allowing to redistribute the available space between the name part (left)
and value part (right).
```

PUBLIC METHODS

```
fraction <left> <right>: redistribute the space of the panes as indicated
        by the numbers <left> and <right>. These 2 numbers must add up to
        100.
getEntries : returns a list with 2 sublists, namely the list of entries
        in the left resp. right listbox.
insert {<leftList> <rightList>} : append items to the end of the left and
        right listboxes; the single argument consists of 2 sublists, one
        for each listbox. Both sublists must have the same size
```

WIDGET BINDINGS

```
The normal class bindings for listboxes resp. entry fields apply, with the
following exceptions or modifications:
<Button1>: for listboxes and entry field
    get focus, select item (for listbox and entry field)
<Button1-Motion>: for listboxes and scrollbar
    drag & select
<Key-Prior> (= PageUp): for listboxes/entry fields
    show previous page in listboxes
<Key-Next> (= PageDown): for listboxes/entry fields
     show next page
<Key-Up> (= Arrow up): for listboxes/entry fields
     select previous element in listboxes
<Key-Down> (= Arrow down): for listbox/entry field
```

```
     select next element in listboxes
<Key-Home>: for listboxes/entry fields
     show top of listbox resp. beginning of line
<Key-End>: for listboxes/entry fields
     show end of listbox resp. end of line
<Key-Enter>: for entry fields
     accept entry-field value
<Key-Escape>: for entry fields
     restore entry as selected in listbox into the entry fields
<Key-Delete>: for listboxes/entry fields
     delete selection
<Key-Insert>: for listboxes/entry fields
     insert new element into the listboxes resp. paste copy buffer into
     entry field
```

INHERITANCE

```
itk::Widget<-bobPanedComboBox
```

STANDARD OPTIONS

```
cursor          font          height             width
borderWidth     relief        selectBackground   selectForeground

See the options(n) manual entry for details on the standard options.

WIDGET-SPECIFIC OPTIONS
-helpvariable (name: helpVariable, class: Variable)
    The name of the global Tcl-variable which is used to display help text
    i.e. this widget will change the contents of this variable according
    to the position of the cursor over its constituent widgets.
-leftlabel (name: leftLabel, class: Label)
    label to display on top of the left listbox; default: empty (no label)
-leftlisthelp (name: leftListHelp, class: Help)
    The short help text for the left listbox; default: empty (no help)
-leftentryhelp (name: leftEntryHelp, class: Help)
    The short help text for the left entry-field; default: empty (no help)
-leftentrystate (name: leftEntryState, class: State)
    Specifies the state for the left entry field. Can be "normal" (i.e.
    editable), "disabled" or "readonly". Default: readonly
-listvariable (name: listVariable, class: Variable)
    The name of the global Tcl-variable which contains the contents of
    the 2 listboxes, if a format as returned by the getEntries method.
    If the value of this variable changes, the widget will automatically
    update itself to reflect the new values. Attempts to assign a
    variable with an invalid list of sublists to -listvariable will cause
    an error. Attempts to unset a variable in use as a -listvariable will
    fail but will not generate an error.
-rightlabel (name: rightLabel, class: Label)
    label to display on top of the right listbox; default: empty (no label)
-rightlisthelp (name: rightListHelp, class: Help)
    The short help text for the right listbox; default: empty (no help)
-rightentryhelp (name: rightEntryHelp, class: Help)
    The short help text for the right entry-field; default: empty (no help)
-rightentrystate (name: rightEntryState, class: State)
    Specifies the state for the right entry field. Can be "normal" (i.e.
```

```
    editable), "disabled" or "readonly". Default: normal
-scrollbar (name: scrollbar, class: Scrollbar)
    Controls the presence of a vertical scrollbar to the right of the
    entry-listbox. It can take the value "on", "off" or "auto". In the
    latter case, the scrollbar will appear only if the listboxes contain
    more elements than fit in a single view. Default: auto.
```

CAUTIONS

```
When -scrollbar is set to "auto", the need for a scrollbar is evaluated
when data are inserted in the listboxed. For the listbox sizes returned
by Tk to be correct, it is needed that this widget is being displayed.
This may not be the case if it is e.g. part of a tabnotebook page. In
this case, one should either make sure this page is displayed, or opt
for the "on" or "off" value instead of "auto".
```

SEE ALSO

```
options(n)
```

- - - - - -

Last change: 23/05/05-10:02

BobTPL(n)

NAME

```
BobTPL - base class for template objects
```

SYNOPSIS

```
BobTPL <obj_name>|#auto \
       <rawData> <obdSource> <canvas> <scrolledText> \
       <initialObsInfo> <tplNr> <args>
```

DESCRIPTION

```
defines template objects - inherits BobBaseObject
```

METHODS

```
constructor: contruct the BobBaseObject, and do the store in array (always
    done at creation time for templates). Do also a first check on
    the parameters value;see [incr Tcl]
destructor: delete any saved paramfiles and proceed with base object
    destruction;
configure: see [incr Tcl]

abort: request an abort of the template's execution. This will set a
    variable called "abort", which can be upvar-ed by the template.
ackabort: acknowledge the abort (sendStatus)
unmark: extend 'unmark' to also clear the log of this template
editRawData ?<cats>? ?<pars>?: pops up the TPL editor. The set of parameters
    to be edited depends on the 2 arguments:
    <cats> : name of a single keyword-category to be edited, or the word
             "all" to include all categories. Default: all
    <pars> : name of specific keywords to edit (assuming <cats> is not set
             to a "all"). Default: all
execute <OBSinfo> <seqNr> <statVarName>: execute the template, whereby
    its sequential number in its parent OB is <seqNr>;
    <OBSinfo> contains relevant info passed to the template for execution.
    This method returns potentially messages and the exec-status in a
    variable whose name is given in <statVarName>. Possible values for
    exec-status:
    -3 : MustRepeat
    -2 : Paused
    -1 : Aborted; returns also the string "ACK ABORT".
     0 : OK - normal completion; return value will be empty
    >0 : error - the return value of the method itself will contain an
         error message
          1: Could not find the template signature file
          2: Could not find the sequencer script
          3: TPL ID has illegal format
         10: Error returned by the template
         11: Template was aborted at OS-level
    Template scripts should return the following way, with proper values:
       return "<RA> <Dec>" : for successfully terminated acquisition
            templates, with <RA> and <Dec> the coordinates of the acquired
            object (both in degrees, as a float number, identical to how
            the primary FITS keywords RA and DEC are written)
       return {} or return without value: if all went OK, for all but
            acquisition templates
```

```
        error "message" : if there is some error within the template, or an
             error  returned from the OS-level (e.g. ABORT).  The setting
             of the ABORT-flag within BOB causes the template to skip
             SETUP and START commands.  When the template sees the abort-
             flag (see above), it should return an "ACK ABORT".
getEditedData: returns the edited data in the form of "raw" template data
getExpNr: returns a list with the current exposure number and the number of
          exposures (TPL(EXPNO) resp. TPL(NEXP)).
getTplId: returns TPL(ID); used to make nice tabs label wile editing an OB
getTplType: returns TPL(TYPE)
init_editRawData <pathName> ?<cats>? ?<pars>?: initializes the TPL-editor
    widget <pathName> with the data as indicated by the two optional args:
    <cats> : name of a single keyword-category to be edited, or the word
             "all" to include all categories. Default: all
    <pars> : name of specific keywords to edit (assuming <cats> is not set
             to a "all"). Default: all
mark: 'select' this Tpl and display its in the corresponding text widget
mustRepeat: request a MUSTREPEAT of the template's execution; this implies
    an abort (see above).
paste <object> <where> : paste <object> <where> relative to this
pause: pause the template
populate: create all BobVisualTree child associated with this template
printLog: prints the log
showExpNr: shows current exposure number in canvas area display
updateRawData: builds a TPL raw-data string starting from the keyword
    arrays.
```

CLASS PROCEDURES

```
checkAbortFlag: convenience procedure to check if bob's ABORT button was
    pressed.  It will automatically generate an "ACK ABORT" error message
    if the abort flag was set, otherwise it returns with an empty string

finishOB ?-status <status>?: tell the OB that when this template finishes,
    it should skip the remaining templates and return the termination
    status <status> (default: TERMINATED) to OH.

getResult <name> : retrieve the value for the previously stored <name>
    variable. Remark that all variables are erased when the first TPL of
    an OB starts it execution. In other words, result-passing from one
    template to another is only possible within a single OB.
    Bob itself stores the execution status (0 = OK, 1 = error) of the
    last template in the variable "tplExitStatus". This variable can
    then be used by the call-back script to decide on proper actions.

pafReady <pathName> ?<keep>? : send an event to OH signalling that the
    PAF-file <pathName> is ready for download. This file will be deleted by
    bob after OH fetched it, unless the boolean variable <keep> is set to
    true (or 1). Unlike other events, this one will be sent to OH even if
    the template executing this command is part of an OB which was not
    downloaded from OH.
    If <pathName> is a directory, there will be an event generated to OH
    for every single file on that directory.

seeBobVars ?-localcopy <listOfCats>?: in the absence of the -localcopy
    option, issues the proper set of "upvar" instructions, within the
    context of the calling procedure, to make the various category-arrays
```

```
    created/owned by BOB accessible as local arrays with the same name.
    This procedure returns a list with as first element the list of
    category names which are then locally accessible.
    If the "-localcopy" option is specified, <listOfCats> must be a list of
    keyword category-arrays, for which there will be then a *copy* created
    in the context of the calling procedure (i.e. these arrays are *not*
    linked to the ones owned and accessed by BOB). The category OBS can
    be included within <listOfCats>.

sendCmd <timeout> <cmd> ?<param1>? ?<param2>? ...: convenience procedure to
    send commands to OS. This is the one and only way template-scripts are
    allowed to use for communication with OS.
    <timeout>: timeout value (in ms) for reply; 0 means return immediately
               with the commandId - do not wait for replies; this argument
               is not taken into account if the template-simulation mode
               is on.
    <cmd>    : command; this is either a commandname alone (in which case
               pararameters can be given in <paramX>) or it includes some
               parameters (more parameters can be given as <paramX>).
               Typical examples of commandnames are SETUP and START, of
               parameters are "-file <fileName>" and
               "-function <function1> <value1> [<function2> <value2> .....]"
    <paramX> : parameter for <cmd>, if not already included as part of
               <cmd>.
    This procedure returns:
    - reply on command if waiting for last reply (<timeout> # 0).  Remark
      that this reply is the new expoId value when the first SETUP command
      for a new exposure is given (with an expoId-argument equal to 0).
      Only the last reply on the command is returned.
    - cmdId if command was sent OK and <timeout> is set to 0; this can
      then be used by the caller to e.g. attach an event (see seq_evtAttach)
      to incoming replies on this command.
    - error (with message) if any error occurs or if ABORT flag is
      acknowledged
    There are a number of messages logged automatically logged by this
    procedure into BOB's template log window; the commands sent will be
    shown in green if operating in verbose mode (see configuration panel),
    while in this case normal or error replies will be shown in green resp.
    red. If <cmd> is WAIT, there will also be a message logged indicating
    the termination of the exposure, when the final reply comes in.
    Remark that sendCmd will not send any of a list of commands once the
    abort flag has been set. This list can be defined/edited via the
    configuration panel. The skipping of such commands is indicated in BOB's
    template log window.

sendObsKeys ?<timeout>? ?-array <listOfCats>?: convenience procedure to send
    the OBS and DPR keywords, plus some TPL keywords, to OS. This is
    normally done automatically by bob when a "START" command is
    given within the template; however, there are cases where data
    can be produced without starting an exposure, i.e. without a START
    command. The "sendObsKeys" procedure provides for these cases a
    means to still have the proper OBS-, DPR- and TPL-keywords recorded
    into the FITS header. The default timeout for OS to reply is 2000 ms.
    Remark that a call to sendObsKeys will automatically increase the
    exposure number (TPL EXPNO), so this proc should only be called once
    per data file, and only if there is no START command involved.
    The "-array <listOfCats>" optional pair is only legal in case OS is
    not defined (i.e. BOB is used for particular engineering purposes).
```

```
    <listOfCats> is now limited to OBS only. By using this optional
    argument, the array OBS will become available at the level that this
    procedure is called. Remark that during normal operation (with OS)
    there is no need to access the OBS keywords passed in the OB.
    (This "-array <listOfCats>" option is actually deprecated: please use
    "seeBobVars -localcopy <listOfCats>" instead).

setCallBack <script>: set the script to be evaluated upon termination of
    this template to <script>, thereby overriding - for this template
    only - the setting in the configuration panel of the default template
    call-back script.
    <script> will be invoked whether or not the template returned an error.
    It can find out the termination status of the template script via a
    "getResult tplExitStatus" command (0 = OK, 1 = error). It
    is up to the call-back to decide if and to what extent it should
    continue if the template script returned with an error.
    If the call-back script needs to get access to values the template
    script has set, the "setResult" and "getResult" mechanism can be used.
    Remark that these call-backs should not be seen as an extension of the
    template's functionality (e.g. to prepare/start exposures). Instead,
    their typical use is to log the template's termination or to put the
    instrument in a safe state. For that reason. this call-back's return-
    status does NOT influence the overall return-status of the template.

setResult <name> <value>: set the <value> for the variable <name>; this
    value can then be retrieved by another template in the same OB -
    see "getResult".

tplLog <text> ?<colour> ?<underline>??
    append <text> to the current log-text, in colour <colour> (default is
    standard default for text), and underlined if the boolean <underline>
    is set. <text> is allowed to contain newline characters.
    Remark that when "verbose communication" is on, the messages sent to
    OS are logged in blue, and OK-replies are logged in SpringGreen4.
    Errors come logged in red. The messages which are always added
    automatically by "sendCmd" (e.g. "starting exposure n of m...") are
    underlined. These colour-schemes of bob should be respected, in order
    to avoid confusion.
```

SEE ALSO

ICD between the VLT Control SW and the Observation Handling Subsystem (VLT-ICD-ESO-17240-19200)

- - - - - -

Last change: 23/05/05-10:02

bobWish(n)

NAME

bobWish - shell version of bob

SYNOPSIS

bobWish ?-ccsProcName <name>? ?-configFile <file>? ?<scriptFile>?

DESCRIPTION

This is a shell implementation of bob. The start-up parameters are similar as for bob, i.e. you can specify the <name> under which this process will register itself with the CCS environment, and pick up configuration parameters from a different <file> than the default ~/.bobrc. If you do not wish to run bobWish interactively, or need to run some script before you enter into interactive mode, you can also pass the name of a <scriptFile>.

BobWish contains dummy objects for all widget related operations. The actions accessible via buttons or menus in bob are here activated through the following set of commands :

bobAbort <reason>
abort the currently executing OB

bobContinue ?-foreground?
continue the currently paused OB

bobDisplayLog
display the log of the current OB-object

bobHelp
display a help text

bobInfo
give info about the loaded OBs and their state

bobNextObsBlocks ?<nr>|file? ?<filename>?
load <nr> OBs from the scheduler process, or an OBD from a file. In the latter case, if <filename> is not given, a file-selection widget will pop-up.

bobPause
stop the currently executing OB

bobPrint
print the log of the current OB

bobRepeat <reason>
tell the scheduler process to repeat this OB at a later stage, giving also a textual <reason>. This implies an abort.

bobReset
mark all currently loaded OBs as freshly loaded

bobSaveConfig
save the configuration parameters in the configuration file.

bobSaveConfigAs ?<file>?
save the configuration parameters into <file>, or if <file> is not specified, bring up a file selection box first.

bobSetup
display a help text, listing all the configuration commands.

bobSnapshot
dump bob-related info and logfiles to a timestamped directory.

bobStart ?-foreground?
start the OB

If the -foreground (of -fg) flag is given (valid for the bobStart and

```
bobContinue commands), control will be passed back to the shell *after*
the OB has terminated, instead of running it in the background and
returning control immediately.
```

CAUTIONS

```
Some dialogs are still kept with tk. This depends partially on the
setting of "bobUserLevel".
```

- - - - - -

Last change: 23/05/05-10:02