PPARC

Particle Physics and Astronomy Research Council

Isaac Newton Group

TCS-GEN-1

Telescope Control System Build and Release Procedures

Marion Fisher and Frank Gribbin

Issue 2.1; 9th May 2024

Isaac Newton Group,
Apartado 321, 38780 S/C La Palma,
Tenerife , Canary Islands

Telephone	+34 922 425400
Fax	+34 922 425401
Internet	fjg at ing.iac.es

Document History

Document Location	Printed on Thursday, 09 May 2024. The document can be found at : http://www.ing.iac.es/~docs/wht/tcs/wht-tcs-9/wht-tcs-9.pdf The source is held in https://ingbitbucket.ing.iac.es/projects/SOFT/repos/tcsdocs/browse/tcs-gen-1.docx

Revision History

Revision date	**Version**	**Summary of Changes**	**Changes marked**
14/11/05	1.1	Documented TCS build procedures for the CAMAC based TCS. Earlier history not recorded.	MPF
10/03/22	2.0	Updated for the replacement TCS at the WHT	FJG
09/05/24	2.1	Added chapter on TCS Alphastation assignments	FJG

Contents :

1. INTRODUCTION

1.1 Purpose

This document describes how to build and release versions of the Telescope Control System. It also describes how to build and release other related software packages and utilities.

1.2 Scope

1.3 Definitions, acronyms and abbreviations

CMS	Code Management System
MMS	Module Management System
TCS	Telescope Control System

1.4 Overview

There are currently five TCS Alphastations at the Isaac Newton Group of Telescopes (ING); All are on the mountain-top . On the mountain-top, one Alphastation is used as the development machine (lpas3) two other Alphastations are used to control the two ING telescopes. The two remaining machines are spares. The development machine stores the source code for the TCS and related software. New versions of this code are developed, built and tested on the development machine; if the tests are successful the built system is copied to the relevant telescope control Alphastation. The built system is then tested at the telescope and released for general use when the telescope manager is satisfied with its performance.

2. SOURCE FILES

2.1 Configuration Management

The VMS Code Management System (CMS) is used for configuration management. The source files for each of the 28 TCS subsystems are stored in separate CMS libraries; the source files for the utilities PLOT and CAMTEST are also held in CMS libraries. Table 1 lists the CMS libraries.

CMS Library	Brief description	Used at INT	Used at WHT
ALTAZLIB	Altazimuth mount specific routines		X
BUILD	Builds the TCS	X	X
CAMACDB	Produces file of valid CAMAC operations	X	
CAMTEST	Program to test CAMAC, aid to diagnosing CAMAC faults	X	
COMMS	TCS process; real-time process that controls the telescope via the Bridge Computer		X
DISPLAY	TCS process; displays state of telescope and associated hardware	X	X
EQUATLIB	Library of equatorial mount specific routines	X	
GSEXAM	Utility to examine and alter global variables on running TCS	X	X
INCLIB	Library of Include files that define global variables and constants	X	X
INT	Library of INT specific routines	X	
INTINIT	Initialises global variables for INT	X	
JKT	Library of JKT specific routines (obsolete)		
JKTINIT	Initialises global variables for JKT (obsolete)		
MONITOR	TCS process; starts TCS, stops TCS when terminating flag is set	X	X
MSG	Error messages	X	X
NETSERVER	DECnet interface between TCS and VAX instrument control computer (Obsolete)		
OUTLOG	TCS process; logs encoder data	X	X
PLOT	Utility to fit simple models and plot data logged by TCS	X	X
POINT	TCS process; generates pointing information	X	X
RUN	Menu interface for starting and stopping TCS, and setting various options	X	X
SHARE	Management of global COMMON blocks and inter-process ASTs	X	X

SYNC	TCS process: real-time process that controls the telescope via CAMAC	X	
SYSCOMP	TCS process; sends TCS information at 1Hz to NETSERVER and TELD	X	X
TCSLIB	Library of general routines	X	X
TELD	DRAMA interface to Unix-based Observatory Control System	X	
TELS	TCS process; receives commands from the Bridge Computer and returns command completion status.		X
TV	TCS process; reads and processes guiding information from autoguider and TV	X	X
USER	TCS process; User Interface	X	X
WHT	Library of WHT specific routines		X
WHTINIT	Initialises global variables for WHT		X

Table 1

The logical name `SOURCE_DIR` defines the root directory of the whole CMS library tree; each CMS library is a subdirectory of the root directory. To set the CMS library of subsystem *aaa* to be the current CMS library, issue the command

```
> CMS SET LIB SOURCE_DIR:[aaa.CMS]
```

or use the `SETCMS` symbol:

```
> SETCMS aaa
```

Each CMS library has a reference directory which holds the most recent version of each file in that CMS library; for subsystem *aaa* the reference directory would be SOURCE_DIR:[*aaa*.REF]. The reference directory is automatically updated whenever an amended file is replaced in CMS. The reference directory is provided as a convenience to the software developer, as it is very useful to have all the code available for inspection when planning the addition of new functionality or when trying to find a software bug. However, it must be stressed that these files should not be altered, nor should any other files be placed in the reference directories. If you suspect that the reference directories have been changed in any way, then they should be verified. You can verify all the CMS libraries as follows:

```
>ALLCMS VERIFY/REPAIR
```

This will check the reference copies and repair them if they have been altered but it will not report any 'foreign' files that should not be there. If a reference directory contains many 'foreign' files, it is best to delete all files in that directory and then verify the CMS library as described above. This will recreate each reference file.

2.2 MMS files

Each CMS library contains an MMS file which is used to build specified targets. Typical targets are executable images, object libraries, command files and data files. The MMS files follow a standard format and contain at least some of the sections described below.

- **Suffices** (sic) This section clears the default suffixes precedence list and replaces it with only those suffixes necessary to build the specified targets for that particular CMS library.
- **Update rules** A list of MMS files that contain user-defined rules that will be used instead of the MMS built-in rules to update a target. These files are kept in the BUILD CMS library. For instance, the file BUILD_DIR:FOROBJ.MMS contains the user-defined rule to update an object file from a Fortran file. It compiles the Fortran file and adds the name of the object file to the list of files to be deleted at the end of the build process.

- **Tidying up**
  - .FIRST defines the action to be performed before any other actions. It simply deletes the file DELETE_FETCHES.COM if it exists.
  - .LAST defines the actions to be performed after all other action lines. If the file DELETE_FETCHES.COM exists, it will be run and then deleted.
- **Top-level targets** All MMS files have the top-level target SYSTEM which is the default MMS target. SYSTEM lists all the targets that should be present in the built system. There is also a target named VERSION which consists of the file VERSION.NUM; this file lists the name and CMS class of the built system, the date it was built, who built it, and the CMS class of all subsystems that contain object and option files that can be linked in to produce an image file.
- **Lower-level targets** This section contains the dependency rules which describe how to build the targets listed under the SYSTEM target. These targets may depend on files which are themselves dependent on other files. The order of the dependency rules in an MMS file is important; if a target is dependent on other files, then the rules that specify how to build those other files must come after the dependency rule for the target. For this reason, the lower-level targets are ordered as follows:
  - Executable image dependent on object libraries and other files from other subsystems
  - Object library dependent on library modules
  - Library modules dependent on object files
  - Object files dependent on source files and include files
  - Explicit actions to fetch files from CMS that need to stay in the built directory. If the user-defined rules contained in the MMS files were to be used, then the file would be added to the list of files to be deleted at the end of the build. Use of an explicit action avoids this.
  - Explicit actions are also used to fetch files from other CMS libraries. An example is:

    ```
    TV.INC : SOURCE_DIR:[TV.CMS]TV.INC~/GEN="$(TV_VERSION)"
      DEFINE/USER CMS$LIB SOURCE_DIR:[TV.CMS]
      $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).INC -
       /GEN="$(TV_VERSION)" $(CMSCOMMENT)
    ```

    The first line specifies that `TV.INC` depends on the file `TV.INC` in the TV CMS library, in the class defined by the symbol `TV_VERSION`.

    The second line defines the default CMS library to be TV, this definition only applies to the next command, after which the default CMS library reverts to its original value.

    The third line fetches the file TV.INC from the TV CMS library, from the class defined by the symbol `TV_VERSION`.

2.3 BUILD subsystem

2.3.1 Contents

The BUILD subsystem comprises the common MMS files and the command procedures used in building the TCS software. With the exception of BUILDSUB.COM, none of the command procedures should be used directly. Use of the command procedures is described in Section 3.

2.3.2 MMS files

The MMS files contain the user-defined rules that are used to update a target, they have filenames of the form *ab*.MMS where *a* defines the source file and *b* defines the target file type. For example, the file CMSDAT.MMS contains the rule to fetch a file with an extension of .DAT from the default CMS library, and the file COBJ.MMS contains the rules to produce an object file from a C source file. Intermediate files such as object files and files fetched from CMS are added to the list of files to be deleted at the end of the build process.

2.3.3 Command procedures

BUILD_SETUP.COM

Usage: `@BUILD_DIR:BUILD_SETUP [version-file]`

BUILD_SETUP sets up the Fortran command symbol and FORT$LIBRARY logical name for compiling TCS Fortran files; checks whether a parameter was given, if it was then the specified version file is executed, if not then it only proceeds if the logical name BUILD_DIR is defined; creates the BUILD subsystem itself in the directory BUILD_DIR if it doesn't exist already; and sets up symbols for the command procedures BUILDSUB and BUILDALL.

BUILDSUB.COM

Usage: `@BUILD_DIR:BUILDSUB <subsys> <class> [dir] [skip] [debug] [supersede]`

`<subsys>` is the name of the subsystem to be built

`<class>` is the name of the CMS class on which to base the build

`[dir]` is the directory in which the subsystem is to be built. This will be created if it doesn't exist already. If this parameter is omitted, the subsystem will be built in the current default directory. To specify the current default directory explicitly if further parameters are being given, use [].

`[skip]` is a flag that controls the behaviour of MMS, and must be either NOSKIP or SKIP. If SKIP is specified MMS will be invoked with the /SKIP qualifier. SKIP is the default.

`[debug]` is a flag that controls whether the subsystem is built in debug mode, and must be either NODEBUG or DEBUG. NODEBUG is the default. DEBUG is not currently supported, and the build will fail if it is specified.

`[supersede]` is a flag that controls the overwriting of pre-existing software, and must be either NOSUPERSEDE or SUPERSEDE. If NOSUPERSEDE is specified, the build will not go ahead unless the target directory is empty at the outset. The default is SUPERSEDE.

BUILDSUB builds a single TCS subsystem. Note that the logical names identifying the versions of the subsystems required to build the target subsystem must be defined before this procedure is executed - you must have already executed a version file at some stage.

Although you must specify a CMS class from which to build the subsystem, BUILDSUB will search the build directory for any source files which are newer than those in CMS and use any that it finds in preference. Therefore, for development work, you can use BUILDSUB without having to save your code changes in CMS.

BUILDALL.COM

Usage: `@BUILD_DIR:BUILDALL [version-file]`

BUILDALL builds all the subsystems that constitute the TCS (including the BUILD subsystem but not the operational data directories). For each subsystem, it calls BUILDSUB with the flags NOSKIP, NODEBUG and NOSUPERSEDE.

Note that BUILDALL will not make any changes to a subsystem's build directory unless it finds it empty. This is designed to help avoid the accidental overwriting of pre-existing software, and it is also useful when a full build fails owing to a problem with some particular subsystem. Once the problem has been fixed, a subsequent BUILDALL will resume the build at the first subsystem whose directory is empty, and thus avoids repeating builds that succeeded.

BUILD_UPDATE.COM

Usage: `@BUILD_DIR:BUILD_UPDATE [version-file]`

BUILD_UPDATE builds all TCS subsystems where source files have been updated, either in the current CMS class or in the build directory. It will also relink any subsystems where an associated library has been updated. For each subsystem, it calls BUILDSUB with the flags SKIP, NODEBUG and SUPERSEDE.

BUILD_UPDATE is most useful when developing a new major TCS version and the source code needs to be developed iteratively.

NOTE_VERSION.COM

Usage: `@BUILD_DIR:NOTE_VERSION <subsystem> <class>`

This file is used in MMS files to build the file VERSION.NUM. VERSION.NUM lists the name and CMS class of the built subsystem, the date it was built, who built it, and the CMS class of all subsystems that contain object and option files that can be linked in to produce an image file.

UPDATE_FETCHES.COM

Usage: `@BUILD_DIR:UPDATE_FETCHES.COM <filename>`

This file is used by some MMS files in the BUILD subsystem, containing the user-defined rules for updating a target. UPDATE_FETCHES.COM appends a command to the file DELETE_FETCHES.COM to delete <filename>. This provides a method whereby intermediate files are deleted from the target directory, leaving only the higher-level targets, a log file of the build and the MMS file itself in the built directory.

3. TCS VERSIONS

3.1 Version Files

TCS versions are defined in *version files*, which are kept in the directory [<tel>_LOGIN.VERSION]. A version file sets up the logical names and symbols required to build and run a given version of the TCS. This serves the purposes of:

documentation	the components of a TCS version are defined explicitly in a single place;
building	the version file contains all the information required to build the given version of the TCS;
executing	the version file contains all the information required to run the given version of the TCS.

The names of the version files, which are the same as the names of the versions they define, are of the form **ti-j-k**. The following naming scheme is used:

t	is used to identify the telescope, e.g. W=WHT, I=INT.
i	is the base revision id. This identifies the root directory under which all the built software comprising the corresponding version of the TCS lives; it is changed as a result of a change to any subsystem which requires one or more other subsystems to be rebuilt. Examples of such subsystems are the library of common include files, any of the object libraries, or a piece of infrastructure such as the interprocess-communication package.
j	is the major revision id, something which is changed as a result of a modification to any other subsystem except the program OWNINIT.FOR, which initialises the installed COMMON blocks.
k	is the minor revision id, something which is changed as a result of a modification to the program OWNINIT.FOR, which initialises the installed COMMON blocks.

A version file defines the following -

- the telescope name;
- the type of its mount (altaz or equatorial);
- the name of the disk which accommodates the built software;
- the base revision id (and therefore the root directory for this TCS version);
- the version (CMS class) of each of the TCS subsystems;
- directories containing proprietary software;
- the set of logical names defining the built subsystem directories;
- a logical name which defines the version itself;
- logical names pointing to the shareable images which are used to hold the global COMMON blocks, to send ASTs to other TCS processes, and to communicate with the observing system;
- the data directories which the running TCS uses.

3.2 Defaults file

The file DEFAULTS.COM defines defaults which are not specific to a particular TCS version. It defines the following -

- the telescope name;
- the default TCS version;
- the default bridge computer host name for status broadcasts (WHT only)

- the default syslog host for logging. This will normally be the local machine and is specified for WHT only.

3.3 Modifying the TCS

From time to time the TCS will need to be modified. The usual reasons for this are

- to update variables such as IERS parameters or pointing coefficients;
- to add new functionality;
- to fix bugs.

Once the modifications to the source files have been identified, a new CMS class must be created for each of the subsystems to be modified. CMS class names are of the form V*m-n*, where *m* is the major version id and *n* is the minor version id. If the change to the CMS subsystem is major, then increment the major version id, otherwise increment the minor version id. Each source file to be changed must be reserved, which will copy the source file to your default directory so that it can be edited. It is advisable to reserve the source file from the previous class, otherwise the latest generation of the source file will be used. This may be a version under development for a future class and it could contain untested code. This is more likely to happen if more than one person is working on the TCS code, but it is easy to forget about files under development. The command:

```
CMS SHOW GEN/MEMBER/DESC <filename>
```

can be used to list all generations of a filename and show the classes each generation belongs to.

Each reserved source file is modified, replaced in its CMS library and inserted into the new CMS class. If a new file is created, a CMS element must be created from it, and the element must be inserted into the new class. A new entry must be made in the file <subsys>_CHANGES.TXT to describe the changes made for the new class.

The new class must then be completed by inserting all unmodified files from the previous class. It is important to insert unmodified files from the previous class or from the class that the new version is based on, otherwise by default the latest generation of all files contained in the CMS library that are not currently present in the new class will be inserted. This could cause obsolete files to be inserted in the new class; these obsolete files may be part of an older class and still need to be stored in CMS to enable the older class to be rebuilt if necessary. Alternatively, the latest generation of a file may be a version under development for some future class, and it should not be inserted into the new class currently being completed.

If a file from the previous class has become obsolete, it should be removed from the new class.

The CMS commands necessary to create and populate a new CMS class as described above are listed below. These commands are described in detail in VMS HELP and in the CMS manual.

- CMS SET LIBRARY <subsystem>
- CMS CREATE CLASS <class> "date: reason for new class"
- CMS RESERVE/GEN=<previous-class> <filename> "Reason for modification"
- CMS REPLACE <filename> ""
- CMS INSERT GEN <filename> <class> "Reason for modification"
- CMS REPLACE/INSERT=<class> <filename> “Reason for modification”
- CMS CREATE ELEMENT <new-filename> "Very brief description of file"
- CMS INSERT GEN <new-filename> <class> "Very brief description of file"
- CMS INS GEN/IF_ABSENT/GEN=<previous-class> *.* <class> "Unchanged"
- CMS REMOVE GEN <filename> <class> "Reason for removal"

3.4 Full TCS Release

3.4.1 Building

You will need to build a new base TCS version if any of the modified subsystems necessitate the rebuild of any unmodified subsystem. For example, if one of the object library subsystems is modified, any unmodified subsystem that links against that object library will have to be relinked, and that subsystem will thus have a different executable image but the same version number (as it is built from the same class).

To build a new base version of the TCS, perform the following steps on the development machine:

1. Go to the directory [<tel>_LOGIN.VERSION] and create a new version file ti-0-0 based on the version file for the previous release.
2. In the new version file, update the base revision id and update the version number of each modified subsystem that is to be included in the new TCS version. Note that the build process interprets the version of each subsystem to be the name of a CMS class, from which it will attempt to build the software. It is not possible to build the TCS from scratch unless the version specified for each subsystem has a corresponding CMS class.
3. Go to the [RELEASE] directory and check that the file BUILDALL.COM is the latest version in the BUILD subsystem.
4. Issue the command:

   ```
   @[RELEASE]TCSBUILD <TCS-version>
   ```

   This will submit a batch job which executes BUILDALL.COM and writes the output to the log file [RELEASE]<TCS-version>_BUILD.LOG .
5. Once the build has completed, issue the command:

   ```
   @CHECK_BUILD.COM <TCS-version>
   ```

   This procedure checks both the build log file and each individual subsystem's log files for reported errors with the following severity codes: E(rror), F(atal) and W(arning). If the build was successful, CHECK_BUILD.COM will output six identical lines as follows:

   ```
   %SEARCH-I-NOMATCHES, no strings matched
   ```

   If the build was unsuccessful, then one or more of the six lines will contain an error message. In this case, examine the build log file <TCS-version>_BUILD.LOG to determine at which subsystem the build process failed. The error messages in this file, and in the log file for the failed subsystem, give the reason for the failure. Once the fault has been fixed, go to the subsystem's directory and run BUILDSUB as follows:

   ```
   BUILDSUB <subsys> <class>
   ```

   This will continue building the subsystem, repeat if necessary until the build is successful.
6. Once the subsystem builds successfully, put the corrected file(s) into CMS using the command:

   ```
   CMS INSERT GEN/SUPER <filename> <class> ""
   ```

   Then delete all files in the failed subsystem's directory, and return to step 4. This will continue the build starting at the first subsystem whose directory is empty.
7. Add a description of the version to [<tel>_LOGIN.VERSION]VERSIONS.TXT.
8. Update the TCS version number in the file [<tel>_LOGIN.VERSION]DEFAULTS.COM
9. If there were more than a couple of failures when building the TCS, it is advisable to delete the whole version tree `[<tel>.S<n>...]` and build the TCS again.

3.4.2 Testing and Releasing

The new TCS should be tested in simulation mode on the development machine before it is copied to the relevant telescope control computer. The tests to be carried out will depend on the modifications made to the code, but you should always test the general health of the TCS by getting the simulated telescope to track a source. It is also useful to perform a CALIBRATE, applying small offsets using the Handset for each source. Any bugs that are found should be fixed, the corrected files need not be replaced in CMS until the TCS is working correctly as the build procedure will use source files from the current directory if they are newer than the source file in CMS. Each faulty subsystem can be rebuilt using the command:

```
BUILDSUB <subsystem> <class> []
```

When you are satisfied with the corrected files, they can be replaced into CMS as described in a previous section. Unless you only made a few corrections, you should delete the TCS directory tree for the new version and rebuild it completely.

Once the TCS performs acceptably in simulation mode you can copy it to the relevant telescope control computer. To make a backup saveset of the new TCS version, issue the command:

```
@[RELEASE.BACK]BACK-TCS <TCS-version>
```

This will create a backup saveset called TCS_<tel>_<base-version>.BCK in [RELEASE.BACK], containing the TCS built directories, the TCS version file and the file VERSIONS.TXT from [<tel>_LOGIN.VERSION]. Copy this saveset to [RELEASE.BACK] on the relevant telescope control computer using ftp or DECnet COPY. Log in to the telescope control computer and unpack the saveset with the command:

```
@[RELEASE.BACK]RESTORE-TCS <TCS-version>
```

This will restore all the files in the saveset, giving an exact copy of the original TCS directory tree and the associated version files in [<tel>_LOGIN.VERSION].

By default, the version of the TCS to be used is read from a file on the system computer; the file specification as seen from the TCS computer is <TEL>:[etc]tcs_version.<tel>. On the Unix file system <TEL>:[etc] equates to /<tel>/etc. The version given in DEFAULTS.COM is not used unless the system computer is unavailable.

When testing a new TCS version, use the menu option to choose the TCS version. The new TCS should be tested as much as possible during the daytime, so that the time required for testing at night is minimised. Night testing should be carried out during an S/D night, with the permission of the telescope manager and/or the Support Astronomer for that night.

Once the TCS is working satisfactorily and has been accepted by the telescope manager, it can be released for general use. To enable the new version of the TCS to be used by default, update the file on the system computer that specifies the TCS version, /<tel>/etc/tcs_version.<tel>, and also update or copy the file [<tel>_LOGIN.VERSION]DEFAULTS.COM.

A new version of the Release Notes for the released TCS version should be produced. The files *_CHANGES.TXT can be helpful here, as they detail the changes made to each subsystem, at least they do if they are kept up-to-date! The *_CHANGES.TXT files should be updated with the release date.

3.5 Partial TCS Release

3.5.1 Building

If the modified subsystems do not necessitate the rebuild of any other subsystem, then you only need to perform a partial release of the TCS. The modified subsystems are built under the same base version as the current TCS, and unmodified subsystems form part of the new TCS version. Reasons for building a partial release include updating the IERS parameters, updating the pointing model for a particular focal station, or making changes to the User Interface.

To build a new partial version of the TCS, perform the following steps on the development machine:

1. Go to the directory [<tel>_LOGIN.VERSION] and create a new version file ti-j-k based on the current version file.

2. In the new version file, update the version number of each modified subsystem.
3. Go to the [RELEASE] directory and issue the command:

   ```
   @[RELEASE]TCSBUILD <TCS-version>
   ```

   This will submit a batch job which executes BUILDALL.COM and writes the output to the log file [RELEASE]<TCS-version>_BUILD.LOG . This will build the modified subsystems only, ie. those subsystems with a new CMS class.

   Once the build has completed, issue the command:

   ```
   @CHECK_BUILD.COM <TCS-version>
   ```

   This procedure checks both the build log file and each individual subsystem's log files for reported errors with the following severity codes: E(rror), F(atal) and W(arning). If the build was successful, CHECK_BUILD.COM will output six identical lines as follows:

   ```
   %SEARCH-I-NOMATCHES, no strings matched
   ```

   If the build was unsuccessful, then one or more of the six lines will contain an error message. In this case, examine the build log file <TCS-version>_BUILD.LOG to determine at which subsystem the build process failed. The error messages in this file, and in the log file for the failed subsystem, give the reason for the failure. Once the fault has been fixed, go to the subsystem's directory and run BUILDSUB as follows:

   ```
   BUILDSUB <subsys> <class>
   ```

   This will continue building the subsystem, repeat if necessary until the build is successful.
4. An alternative method, useful if only one or two subsystems are modified, is to run BUILDSUB for each subsystem. Issue the command:

   ```
   SETLOGS <TCS-version>
   ```

   This will define the logical names and symbols for the new version.

   For each subsystem that has been modified, issue the command:

   ```
   BUILDSUB <subsystem> <class> <subsystem_dir>
   ```

   This will build the subsystem in the directory pointed to by the logical subsystem_dir. If the build failed, correct the error and run BUILDSUB again.
5. If any files were corrected in steps 3 or 4, replace the corrected files into CMS using the command:

   ```
   CMS INSERT GEN/SUPER <filename> <class> "Reason for replacement"
   ```
6. Add a description of the version to [<tel>_LOGIN.VERSION]VERSIONS.TXT.
7. Update the TCS version number in the file [<tel>_LOGIN.VERSION]DEFAULTS.COM

3.5.2 Testing and Releasing

The new TCS should be tested in simulation mode on the development machine in the same way as for a full release. Once the tests have been completed satisfactorily, the modified subsystems can be copied to the relevant telescope control computer. Go to [RELEASE.BACK] and make a backup saveset using the command similar to:

```
BACKUP <subsystem-1>_dir,<subsystem-2>_dir TCS_<tel>_<base-id>_PART.BCK/SAVE
```

Copy this saveset to [RELEASE.BACK] on the target computer using ftp or DECnet COPY. Log in to the target computer, go to [RELEASE.BACK]and unpack the saveset with the command:

```
BACKUP TCS_<tel>_<base-id>_PART.BCK/SAVE/SEL=[<tel>...] [<tel>...]/BY=PARENT
```

This will restore all the files in the saveset, copying them into the correct locations.

Copy the new version file and VERSIONS.TXT to the target machine.

The new TCS version should be tested in the same way as for a full release. When it is ready to be released for general use, update the files [<tel>_LOGIN.VERSION]DEFAULTS.COM and <TEL>:[etc]tcs_versions.<tel> so that they define the new TCS version.

3.5.3 Updating Help files

Help is implemented using VMS LIBRARY command to build a help library.
LIB/CREATE/HELP. See the OpenVMS Command Definition, Librarian, and Message Utilities Manual for more details.

Files (one per command) have extension .HLP

Files for commands for both telescopes are held in USER
Files for telescope specific commands are held in OWN (INT or WHT)
Mount specific help is found in MNTLIB (ALTAZLIB or EQUATLIB)

Building of the help library USER_HLP.HLB is performed by the build of USER

MAKEHELP.FOR is source for MAKEHLP.EXE, which is run to combine help files into USER_HLP.HLP.

HELP_DEFINE.COM sets up the logical names needed

```
COMMANDS.LIST ----------------------\
      (USER)                        |
                                    V
USER_DIR:*.HLP    ------->
MNT_DIR:*HLP      ------->     MAKEHELP.EXE     ----->   USER_HELP.HLP
TEL_DIR:*.HLP     ------->

                                                              |
                                                              |
                                                              V

                                                        LIB/CREATE/HELP


                                                              |
                                                              |
                                                              V

                                                        USER_HELP.HLB
```

It can be valid for a help topic to have a help files in more than subsystem. AGSELECT has a top level entry in USER AGSELECT.HLP and details of the command parameters in WHT AGSELECT.HLP

MAKEHELP does report if a help topic is not found, but it is not necessarily an error. For instance, the ALTITUDE command is not available at the INT, so there is no ALTITUDE help. However ALTITUDE is a command listed in COMMANDS.LIST, so when INT USER subsystem is built, there is a message output from MAKHELP.EXE saying "No HELP text for ALTITUDE"

See USER_BUILD.LOG for these build messages.

3.5.3.1 Adding help for a new command

- Create a new command.HLP file in appropriate subsystem library (e.g. USER,WHT, ALTAZLIB…)
- Add the new command to COMMANDS.LIST
- Check that when the subsystem is built, that the command.HLP is extracted (i.e. to USER_DIR) and if necessary update the subsystem MMS file.
- Delete USER_DIR:USER_HELP.HLB and USER_HELP.HLP
- Rebuild the USER subsystem.
- Check USER_DIR:USER_BUILD.LOG for "No HELP text for" messages for this command or other build errors.

3.6 Structure of TCS Directories

The top level directory of the released system is named for the telescope, eg [WHT].

The subdirectories of the top level directory are the root directory for the data directories, [.DATA], and the root directories for the TCS base versions, [.S*n*].

3.6.1 Data directories

The subdirectories of [.DATA] are

• AUTOGUIDER	this holds the autoguider log files. The files are named AG*yymmdd*.DAT.
• CALIBRATE	this holds the files generated by CALIBRATE. Each time CALIBRATE is run it creates a log file for each encoder in use, named <enc-name>*yymmdd*.DAT; a log file for the demanded position, DEMAND*yymmdd*.DAT; a log file for the tracking position, TRACK*yymmdd*.DAT, a file containing the pointing model, PROCS.DAT; and a file containing the graphic output from TPOINT, PGPLOT.PS. The TPOINT log is written to the file TPLOG.LIS, this file is overwritten each time. If the calibrate solution is accepted, the file TRACK*yymmdd*.DAT is renamed to TRACK*yymmdd*.SAV, and the solution is written to the file <focal-station>.RED
• CATALOGUE	this holds the catalogues made by telescope users.
• ENCODER	this holds the encoder log files. The files are named ENC*yymmdd*.DAT.
• POINTING	this holds the files generated by POINT during a pointing test. POINT creates a log file for each encoder in use, named <enc-name>*yymmdd*.DAT; a log file for the demanded position, DEMAND*yymmdd*.DAT; and a log file for the tracking position, TRACK*yymmdd*.DAT.
• PROCESS	this holds the log files created by the TCS processes. There is a log file for each process, DISPLAY.LOG, MONITOR.LOG, OUTLOG.LOG, POINT.LOG, SYNC.LOG (INT only), COMMS.LOG (WHT only), SYSCOMP.LOG, TV.LOG and USER.LOG. CLONE.LOG is created by the CLONE command and TRANSFER.LOG is created by the TRANSFER command.
• SNAPSHOT	this holds files generated by the SNAPSHOT command.
• TV	this holds the TV log files. The files are named TV*yymmdd*.DAT.

At the INT, catalogue files can also be held on the Unix disk. The file specification as seen from the TCS computer is <TEL>:[cat]<catfile>.CAT, on the Unix file system <TEL>:[cat] equates to /<tel>/cat.

3.6.2 TCS version directories

Each TCS base version has a subdirectory for each TCS subsystem, under these subdirectories is a subdirectory for the subsystem version, named for the CMS class. For example, the directory containing the built subsystem RUN, class v0-6, for the WHT base version S9 is [WHT.S9.RUN.V0-6] and is pointed to by the logical name RUN_DIR.

4. OTHER RELATED SOFTWARE

4.1 DRAMA (INT only)

If a new DRAMA version has been obtained from AAO, create a subdirectory of [DRAMA] called <version> and unpack the new version into [DRAMA.<version>...]

[DRAMA.<version>] should contain the files DRAMASTART.COM, DRAMA_MAKE.COM and the directory DRAMA_SOURCE.DIR

To build the new version, go to [DRAMA.<version>] and login with logging enabled:

```
SET HOST/LOG=DRAMA_BUILD.LOG LPAS3
```

Once logged in, execute the following commands:

```
DEFINE/JOB/TRANS=CONCEAL DRAMADISK SYS$SYSDEVICE:[DRAMA.<version>]
```

```
SET DEF DRAMADISK:[000000]
```

```
@DRAMA_MAKE
```

Once the build has finished, either successfully or not, log out and examine DRAMA_BUILD.LOG

Fix any obvious errors, but if it looks as if a file is missing or it is not obvious how to fix the problem, email tjf@aaoepp.aao.gov.au with details of the problem and ask for help.

The built version will be in [DRAMA.<version>.RELEASE...]

4.2 SYSLOGD

The source code is kept in `[SYSLOGD.SOURCE]`. It is not possible to build a working version of SYSLOGD with the current version of VMS, but the existing built version can still be used.

To release SYSLOGD on a target machine, create the directory [SYS0.UCX_SYSLOGD] if it does not already exist on the target machine and copy to it the following files from [SYSLOGD.SOURCE] :

LOGIN.COM, LOGGER.EXE, SYSLOGD.EXE, SYSLOGD.CFG, UCX_SYSLOGD_STARTUP.COM

Amend SYSLOGD.CFG so that the second line refers to the observing system computer, for the WHT the second line would read:

```
*.*         @whtics.roque.ing.iac.es
```

Check that the account UCX_SYSLOGD has been created. If it has not, create the account by copying the account on LPAS3. If it has already been created, make sure that the DISUSER flag is not set, it can be unset by the SYSUAF command

```
UAF> MOD/FLAGS=NODISUSER UCX_SYSLOGD
```

Check that the UCX service SYSLOGD has been defined. If it has not, define the service by copying the file DEFINE_SYSLOG_SERVICE.COM from LPAS3 to the target machine and executing it. This will define the service and enable it. To enable it by hand, type:

```
$ TCPIP ENABLE SERVICE SYSLOG
```

Add the above line into SYS$MANAGER:SYSTARTUP_VMS.COM, immediately after the line:

```
$ @SYS$STARTUP:TCPIP$STARTUP
```

This will enable the SYSLOG service at boot time.

SYSLOG is part of the Talker system which is documented in OBS-TALK-4.

4.3 CAMTEST (INT only)

CAMTEST source files are held in CMS, in the CMS library tree. Once a new class has been created and the files modified for a new version, CAMTEST can be built using BUILDSUB. First create a directory for the new built version, called [CAMTEST.<class>]. Go to this directory, then execute the command

```
SETLOGS <version>
```

where `<version>` is any recent version. Then execute the command

```
BUILDSUB CAMTEST <class> []
```

to build the new version. Any errors in the build will be reported on the screen, and also in the log file, CAMTEST_BUILD.LOG .

Once the new version has been tested successfully, the system logical name CAMTEST_DIR should be repointed to SYSDEV:[CAMTEST.<class>]. The definition of CAMTEST_DIR should also be updated in the file SYS$MANAGER:TCS_STARTUP.COM, which is run at boot time.

4.4 PLOT

PLOT source files are held in CMS, in the CMS library tree. Once a new class has been created and the files modified for a new version, PLOT can be built using BUILDSUB. First create a directory for the new built version, called [PLOT.<class>]. Go to this directory, then execute the command

```
SETLOGS <version>
```

where `<version>` is any recent version. Then execute the command

```
BUILDSUB PLOT <class> []
```

to build the new version. Any errors in the build will be reported on the screen, and also in the log file, PLOT_BUILD.LOG .

Once the new version has been tested successfully, the system logical name PLOT_DIR should be repointed to SYSDEV:[PLOT.<class>]. The definition of PLOT_DIR should also be updated in the file SYS$MANAGER:TCS_STARTUP.COM, which is run at boot time.

4.5 PGPLOT

This graphics package is available from Caltech, free to academic and non-commercial organisations. The built system is in [PGPLOT], the source is in [PGPLOT.SOURCE...] and instructions for building and installing PGPLOT are in the file [PGPLOT.SOURCE]INSTALL.TXT

Documentation for PGPLOT is in Postscript files in [DOCS.PGPLOT] on LPAS3.

4.6 SLALIB

4.6.1 C version

The source code is in [STARLINK.LIB.SLALIB_C]. To rebuild the object library, use the MMS file SLALIB.MMS that is provided.

The C version of Slalib is a Starlink product released for Unix. This version was copied from the Unix Starlink distribution by Pete Bunclark.

4.6.2 Fortran version

The source code is in [STARLINK.LIB.SLALIB_F]. To rebuild the text and object libraries, execute the command

```
@[STARLINK.LIB.SLALIB_F]BUILDLIBS
```

The Fortran version of Slalib is a Starlink product and is documented in Starlink User Note 67.

4.7 TPOINT

The source code is in [TPOINT.CODE]. To rebuild TPOINT, rename [TPOINT.CODE]RELEASE.DIR to something suitable, eg. [TPOINT.CODE]OLD_RELEASE.DIR as the first thing the build procedure does is to delete everything in [TPOINT.CODE.RELEASE] if that directory exists. The build procedure copies the data files from [TPOINT.CODE] to [TPOINT.CODE.RELEASE], including PROCS.DAT which defines the pointing model for each telescope. If this file is changed, make sure that it is changed in both [TPOINT.CODE] and [TPOINT.CODE.RELEASE].

TPOINT is a Starlink product and is documented in Starlink User Note 100.

4.8 COCO

Both the source code and the executable image are in [STARLINK.UTILITY.COCO]. If you need to rebuild it, execute the command:

```
@[STARLINK.UTILITY.COCO]CREATE
```

COCO is a Starlink product and is documented in the Starlink User Notes.

4.9 CAMAC Interface Routines, device driver and USSAST

The CAMAC package was provided by Hytec. The Interface routines are provided as an object library ESONE_SUBS.OLB; the device driver is the executable SYS$PCI_1386_DRIVER.EXE. These files are stored in [HYTEC.PCI_V*n*] together with a command file PCI_1386_DRIVER.LOAD which loads the driver.

Hytec also provided the utility USSAST which sends an AST to another process. The Macro source file is USSAST.MAR, and the build procedure is BUILD_USSAST.COM. The TCS build procedure copies the executable into SHARE_DIR:USSAST_SHARE.EXE and installs it as a shareable image. All USSAST files are stored in [HYTEC.USSAST].

5. TCS ALPHA MACHINE ASSIGMENTS (AT MAY 2024)

5.1 WHT

5.1.1 WHT Operational Alpha LPAS5

The Alphastation lpas5.ing.iac.es is used to run the operational VMS TCS at the WHT. In normal operation the TCS is started from the Observing Support Assistant workstation osadisplay1.ing.iac.es. An icon (labelled lpas5) on the desktop can be used to start the session. Alternatively use the command

telnet lpas5

It's also possible to start the TCS session from other workstations, such as that used for the observing system: whticsdisplay1 (and it's spare: whticdisplay2).

Lpas5 resides in the WHT Computer room.

5.1.2 INT Operational Alpha LPAS2

The Alphastation lpas3.ing.iac.es is used to run the operational VMS TCS at the INT. In normal operation the TCS is started from the observer's workstation inticsdisplay.ing.iac.es. An icon (labelled TCS) on the desktop can be used to start the session. Alternatively use the command

telnet lpas2

Lpas2 resides in the INT CLIP centre.

5.1.3 TCS Development Alpha LPAS3

The Alphastation lpas3.ing.iac.es is used for software development. Connect using command

telnet lpas3

lpas3 can run the INT TCS in a simulation mode without connection to CAMAC.

Lpas3 can also run the WHT TCS in a simulation mode without communication to a bridge computer. As noted in 3.4.2

However, in many cases, it's more useful to run the TCS configured to communicate with a development bridge computer and a PLC running in simulation mode. This is described in

Running+the+TCS+Simulation+System+with+the+Spare+Beckhoff+PLC