DELPHI Collaboration DELPHI 97-127 PROG 226
5 August, 1997

DELPHI analysis code administration and distribution

Olof Bärring,
University of Lund,
Physics Departement,
BOX 118,
S – 221 00 LUND, Sweden

Jean-Damien Durand
CERN, CH-1211 Geneva 23, Switzerland
or
Université Claude Bernard de Lyon,
IPNL, IN2P3-CNRS, F-69622,
Villeurbanne Cedex, France

Abstract

This note describes the technical details of the current implementation of the DELPHI offline analysis software administration and distribution system. It is primary intended as a reference manual for the librarian but some of its content also concerns the people providing the software.

1 Philosophy, terminology and recommendations to administrator

The current administration and distribution of DELPHI offline analysis software is based on the following principles:

(a) The files released with a specific *version* (see definition below) should never change. Any bug-fix or new feature will be added in the subsequent *version*.

(b) The files within the *version* should be unique within that *version*.

(c) The files within the *version* should be grouped according to their type and not their contents. Four different types are defined:

  (c1) Source code files (cradles and pams) reside in a src directory (to admit the addition of other types of source code files a sub-level *car* was added so the cradles and pams actually reside in src/car).

  (c2) ASCII data files used for driving the analysis routines (e.g. alignment, calibration, luminosity and run quality files) reside in a dat directory.

  (c3) Libraries reside in a lib directory.

  (c4) Scripts and compiled programs reside in a bin directory.

(d) The file naming conventions:

  (d1) Source code pams are named in lower case letters and have the extension .car

  (d2) Source code cradles are named in lower case letters and have the extension .cra

  (d3) ASCII data files are named in upper case letters.

  (d4) Archive libraries follow the standard UNIX naming with the prefix lib and the extension .a (e.g. libdstanaxx.a).

  (d5) Scripts and programs are named in lower case letters.

  (d6) Cradles and libraries have version-independent names.

(e) The complete *version* is local on each DELSHIFT machine (no NFS or AFS mounts).

(f) All soft links within and between *versions* are relative. The major argument for this is transportability.

Exceptions to the rule (a) are a few ASCII data files, e.g. FAT_GROU.NAMES, which may be asynchronously updated at any time. However, source code files and libraries *never* change within a version.

The following terminology will be used throughout this note:

- *version*: a *version* is a directory tree, structured according to the rules (c1-4) above, which contains the complete set of files for DELPHI offline analysis. The root directory of a *version* is normally named with a date-stamp, YYMMDD (e.g. 960826).

- *production version*: the *production version* is the currently used version to which the DELPHI group environment variables (e.g. DELPHI_PAM) are pointing.

- *release*: a *release* is a ready *version* which is or has once been the *production version*. The primary criteria which determines whether a version is released or not is the AFS ACL of the root share directory of the version in the DELPHI AFS project space. If the AFS ACL group xx:user is within the AFS ACLs the version is considered as released.

- *prerelease*: a *prerelease* is a version built from development files for debugging purpose. The *prerelease* version is rooted under the prerelease directory rather than the usual date-stamped YYMMDD directory. The access to the prerelease-directory is restricted to the AFS ACL group xx:swtest.

- *central repository*: the *central repository* is the primary storage for all DELPHI offline analysis files and libraries. It is defined to be in the DELPHI AFS project space with the absolute paths:
  /afs/cern.ch/delphi/share (share files)
  /afs/cern.ch/delphi/@sys (specific files).

- *access path*: the *access path* is the path which should be used to access the DELPHI offline analysis files and libraries in the central repository. The use of the *access path* is facilitated by the group defined environment variables (e.g. $DELPHI, $DELPHI_PAM, $DELPHI_LIB etc.). On the DELSHIFT nodes where the DELPHI offline analysis files and libraries are local the *access path* is the same as the actual path. On AFS the access path is rooted under the DELPHI AFS group space, for instance
  /afs/cern.ch/group/xx/dstana
  for the dstana product. Use of actual paths under AFS (e.g. the paths to the central repository in the project space) is not recommended because the central repository may contain incomplete and buggy versions.

- *package*: a *package* is defined as a collection of files exported by a *package responsible* (e.g. the UX pam and cradle for the ux *package*). Note that a *package* does not necessary need to be associated to a library. For instance, the *package* btagging contains a collection of btagging calibration files (auxiliary data files) but no source code.

- *export area*: the *export area* of a package is a predefined directory-tree containing all the files which the package responsible wish to appear in the next release. An example of an *export area* is the path:
  /afs/cern.ch/user/d/dstana/public/export
  which is the *export area* of the "dstana" package. A package responsible may choose to create an additional export area for development files which will be used to build the prerelease version. For instance:
  /afs/cern.ch/user/d/dstana/public/new
  is the *export area* for the dstana development version.

- *product*: a version always belongs to a *product*. There are four defined *products* in DELPHI: dstana, database, delana and delsim. Only the dstana and database

products are controlled by the software administration system described in this note. Although the system will be described in the context of dstana (DELPHI offline analysis), everything what is said also applies to the database product.

- *auxiliary data file*: the ASCII data files which are used for the driving of the analysis routines are called *auxiliary data files*.

The DELPHI offline analysis software administration and distribution system may run from any central CERN node which is a true AFS client. The central repository for all DELPHI offline analysis files is found in the DELPHI project area in the AFS cell cern.ch . The full path to the DELPHI project area is: /afs/cern.ch/delphi .

The following rules help to prevent and keep track of mistakes:

(a) Any action which may change the content of an existing (and released) *version* in the central repository is strongly discouraged but if needed it *should* be done by the appointed librarian using the AFS account *dellib*.

(b) Never remove a released version in the central repository without having first completely removed all dependence on it (inter-version dependence will be explained later in section 3).

(c) Before releasing a version make sure that it has been successfully distributed to all subscribed nodes.

Unreleased versions in the central repository may be removed without causing any damage. On DELSHIFT there is no inter-version dependence so it is less dangerous to remove a released version which is not in production. However, it should then be made sure that no job depends on this version. Normally a version should be left on disk at least a few days after it has been put out of production.

2 The dellib account

All administration of the DELPHI offline analysis software is done from the AFS account dellib. The entire system is controlled by a set of scripts which are found in the public/etc directory-tree. The different scripts will be described in section 4.1.

Any system status message will be mailed to dellib@afsmail.cern.ch and it is the responsibility of the librarian to arrange with an appropriate .forward file.

When running a software administration script the *product* (dstana or database) context is determined by the base-name of the configuration directory from which the script runs. For instance when running in public/etc/dstana or public/etc/devlibs/dstana the context is *dstana*. The configuration directory contains configuration files and some driver scripts for the software administration system. The configuration and driver files which may need to be modified by the librarian will be described in the following subsections.

2.1 Configuration file for the library building: build_defs.pl

build_defs.pl contains the settings for all parameters related to the building of the libraries. Currently the following (perl) parameters are set in build_defs.pl :

- name= $SHARE, type=scalar, value: path to the root directory for all ASCII files (source code and auxiliary data files) in the DELPHI AFS project space.
- name= @OSTYPE, type=array, value(s): supported OS types. The format is normally in the form of the predefined AFS symbol @sys (e.g. hp700_ux90 of HP-UX 9.0x systems).
- name= @BUILD_MACHINES, type=array, value(s): node names of the reference machines
- name= @BUILD_DIRS, type=array, value(s): root path for the building of the libraries on the corresponding reference machine (there should be an one-to-one correspondence between each entry in @BUILD_MACHINES and @BUILD_DIRS).
- name= %LIB_RESPONSIBLES, type=associative array, key(s): full library name of each library which is built. Value(s): full mail-address to the responsible of that library.

Here follows an example of the file build_defs.pl

```
#!/usr/local/bin/perl
#
# ## Definition needed by update_specific.pl for building libs.
#
$SHARE="/afs/cern.ch/delphi/share/$PRODUCT";
@OSTYPES=('hp700_ux90',
          'alpha_osf1',
          'rs_aix32',
          'hp700_ux100');
@BUILD_MACHINES=('shift10.cern.ch',
                 'shift27.cern.ch',
                 'cernsp.cern.ch',
                 'shift34.cern.ch');
@BUILD_DIRS=("/scratch/dellib/build/$PRODUCT",
             "/usr/dellib/build/$PRODUCT",
             "/afs/cern.ch/delphi/rs_aix32/$PRODUCT/build",
             "/var/users/dellib/build/$PRODUCT");
%LIB_RESPONSIBLES = ( 'libdstanaxx.a'  , 'Tzanko.Spassoff@cern.ch'
                    , 'libskelanaxx.a' , 'Tzanko.Spassoff@cern.ch'
                    , 'libvdclapxx.a'  , 'Tzanko.Spassoff@cern.ch'
                    , 'libuxxx.a'      , 'grodid@frcpn11.in2p3.fr'
                    , 'libpxdstxx.a'   , 'Yves.Sacquin@cern.ch'
                    , 'libpxtag2xx.a'  , 'John.Wickens@cern.ch'
                    , 'libufieldxx.a'  , 'John.Wickens@cern.ch'
                    , 'libuhlibxx.a'   , 'John.Wickens@cern.ch'
                    , 'libtanagraxx.a' , 'John.Wickens@cern.ch'
                    , 'libtriggerxx.a' , 'Carlos.Lacasta@cern.ch'
                    , 'libherlibxx.a'  , 'Pierpaolo.Rebecchi@cern.ch'
                    , 'libkalxx.a'     , 'anders.borgland@fi.uib.no'
                    );
```

2.2 Package configuration files: ⟨ package⟩ _config.pl

Collecting the (share) files from the export area of each package is probably the most important but also the most complex and vulnerable part of the software administration

system. In 1993 it had to be forseen that the responsible for a package could choose to work on CERNVM, VXCERN or a UNIX node. In addition, there were no imposed standards on file-naming and directory structure for the files contained in the package. The package configuration files were originally designed to resolve any file naming conflicts with the rules listed in section 1 and to provide the system operations of copying files from the export area to the central repository. There is one configuration file for each package (e.g. phdst_config.pl for the phdst-package).

The variable names in the package configuration files have the prefix server when referring to the package export area and *client* when referring to the share area in the DELPHI AFS project space (given by the $SHARE parameter defined in build_defs.pl). For instance, the perl variable $server_file refers to a file in the package area whereas $client_file refers to a file in the share area of the DELPHI AFS project space.

Each ⟨ package⟩ _config.pl file contains the following perl routines:

- **INIT** : initialisation of the package specific parameters like the package name and server root directory (the export area). INIT also performs the action of providing a list (normally just a " ls -R" of the export area if it is under AFS) of the package files to the software administration system.
- **FILTER_FILE_NAME** : translates (if needed) the server file name into an appropriate client file name according to the rules in section 1.
- **INIT_GET_FILES** : CERNVM related function which was needed because ftp had to be used rather than rcp to retreive files from CERNVM.
- **END_GET_FILES** : CERNVM related function which is dual to INIT_GET_FILES.
- **NEW** : decision function to determine what to do if there are new files on the server side. Normally this function simply calls UPDATE since this is the normal action when a new file appears in the server area.
- **UPDATE** : does the actual update of a file from the server to the client area. In the case the server area is under AFS the action in UPDATE is a simple " cp". If the server area is on VXCERN the UPDATE does a " rcp". UPDATE normally also checks for AFS cache corruption.

These routines are required although only INIT and UPDATE are really important. Obviously some parts related to collecting package files form CERNVM are obsolete in the package configuration files after the run-down of CERNVM. However, all the routines listed above are called at least once and must therefore be provided.

If a new package is to be added by the librarian it is normally sufficient to copy one of the old ⟨ package⟩ _config.pl files (e.g. dstana_config.pl) and modify the INIT function according to the new package (change $PACKAGE and $SERVER_ROOT).

2.3 The library building steering script mkspecific.csh

Except for the PHDST library all DELPHI offline analysis libraries are built using a sophisticated script called makedlib. For any new library it is highly recommended that

the cradle is adapted to run with makedlib. The requirements for building a library with makedlib will be explained in section 4.1.18.

The script mkspecific.csh constitutes the steering of the building of the libraries on the reference machine. It performs the following tasks:

- Install the source files associated with the version to be built.
- Call makedlib for each library (except for PHDST where phlib.csh is called instead).
- tar the libraries into a file ⟨ OS-type⟩ _specific.tar
- Sends status mails to the dellib account. If a compilation fails the corresponding log file together with the failing FORTRAN file(s) are sent to dellib@afsmail.cern.ch .

If a new library is to be added it is normally sufficient to add a line:

```
../bin/makedlib lib<newlib>.a <newlib>.cra >>&makedlib.log
```

where ⟨ newlib⟩ should be substituted with the name of the new library (e.g. dstanaxx).

For the support of a VMS version of the libraries there is a corresponding DCL com-file mkspecific.com *(which calls* makedlib.com*) to run on VXCERN.*

2.4 Adding a new library

In summary: the adding of a new library normally implies the adding of an entry to the %LIB_RESPONSIBLES associative array in build_defs.pl, creation of a ⟨ package⟩ _config.pl file and adding an entry to the mkspecific.csh script (if a VMS version is supported an entry must also be added in mkspecific.com).

To add a new package which is not associated to the building of a library, the only action needed is to create and configurate a new package configuration file.

3 Layout of the share and specific areas in the DELPHI AFS project space

Several released versions of the dstana (and database) software are stored in the DELPHI AFS project area. Up to about four months old versions can be found if the updates have not been to frequent (in average once per two weeks or so). The only restriction for storing older versions is the limited disk space.

The absolute paths of the share and specific areas under the DELPHI AFS project area are:

- /afs/cern.ch/delphi/share/⟨ product⟩
- /afs/cern.ch/delphi/@sys/⟨ product⟩

where ⟨ product⟩ **is either** database, delana, delsim **or** dstana**.**

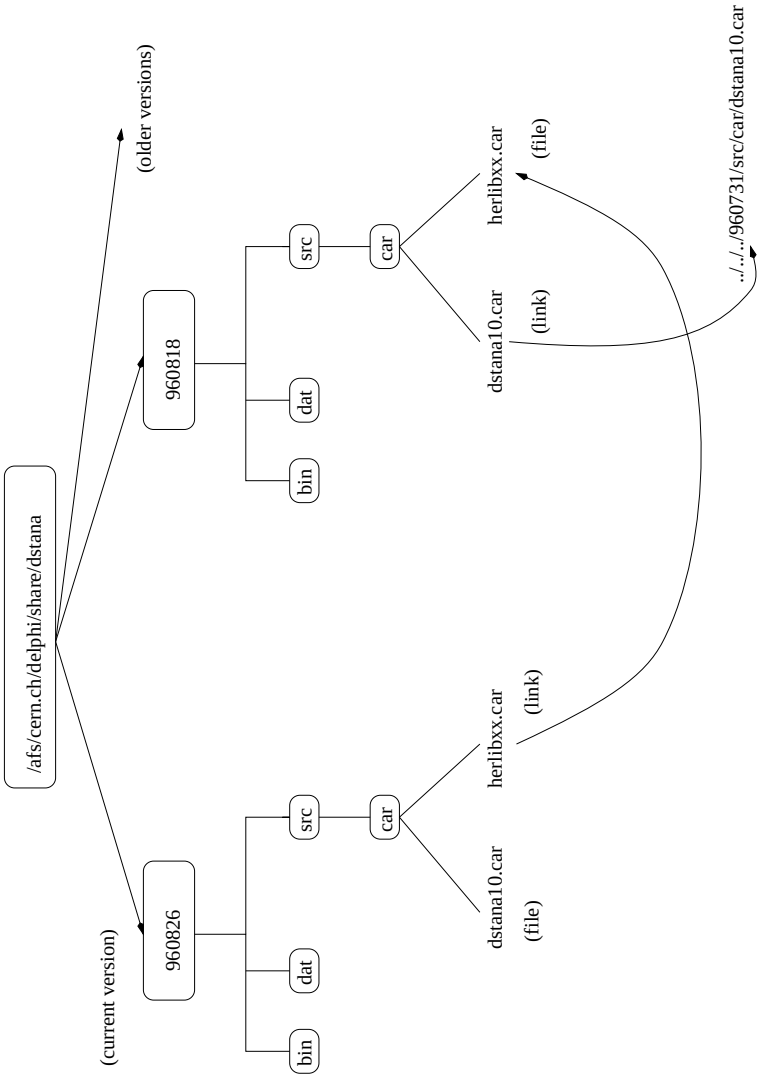

Figure 1: Example of soft link connections between two versions.

It is obviously a waste of disk space to store the same file twice if it has not been changed in between two versions. To avoid this a complex system of soft links between the different versions is maintained. An example is show in figure 1. Between the two versions 960826 and 960818 the file herlibxx.car was not updated. Rather than storing the file twice the 960826 version of the file is actually a link pointing back to the 960818 version of the file (the real file). Similarly the file dstana10.car was not updated between the versions 960818 and 960731 (not visible in the figure). Therefore the 960818 version of the file is actually a link pointing back to the 960731 version of the file (the real file). All links are relative (which follows the rule (f) listed in section 1) and point always to real files (there are never more than one level of links). This means that the 960826 version of herlibxx.car points to ../../../960818/src/car/herlibxx.car. Relative links facilitates the transportability of a version to another repository.

A similar structure to the *share* area is maintained for the *specific* areas (/afs/cern.ch/delphi/@sys/dstana). Currently (960901) the following specific areas are

supported: alpha_osf1, alpha_osf20, alpha_osf32, alpha_osf32c, rs_aix32, rs_aix41 and hp700_ux90 (however all alpha_... paths actually point to alpha_osf1 and similarly rs_aix41 → rs_aix32).

Since the share and specific files are spread out over different directories, a special access path has been setup for each version in the DELPHI AFS *group* directory /afs/cern.ch/group/xx/dstana. Thus, when accessing the 960826 version of the pam-file herlibxx.car the access path is
/afs/cern.ch/group/xx/dstana/960826/src/car/herlibxx.car
and when accessing the 960826 version of the library libherlibxx.a the access path is:
/afs/cern.ch/group/xx/dstana/960826/lib/libherlibxx.a
Any file withing a version should be accessed using these paths or, even more convenient, using the group defined environment variables $DELPHI_... .

There is one serious disadvantage with the net of soft links maintained between the released versions in the central repository, namely: the removing of a version becomes quite cumbersome. Simply removing the directory tree with rm -r 960818 implies that the soft link 960826/src/car/herlibxx.car points to nowhere. A special script, rmversion.csh, (described in section 4.1.14) must therefore be used to disconnect (remove all link dependencies) and remove a version.

4 The software administration and distribution system

The DELPHI offline analysis software administration and distribution consists of the following steps:

(a) Create a new version tree in the share area in the central repository and copy to it all new files from the package export areas. These actions are performed by the script update_share.pl.

(b) Spawn the library building on the reference machines. This action is performed by the script update_specific.pl.

(c) When all compilations finished on the reference machines the compiled libraries (and programs if any) must be copied back to the new version tree in the specific area of the central repository. This action is performed by the script update_specific.pl.

(d) Distribute all files belonging to the new version to the local disks of each node in the DELSHIFT cluster. This action is performed by the scripts shift/mkall.csh.

(e) Release the version for production. This action is performed by the script release.csh.

In addition to the listed basic tasks of the DELPHI offline analysis software librarian there is also the following duty:

(f) Check disk space in the central repository and remove old versions if necessary. Actions are performed by the scripts check_space.csh and rmversion.csh.

In addition to the action scripts listed above there are a few utilities needed for checking the compilation status and AFS cache corruptions, cleanup non-released versions and distribute a given file to all DELSHIFT nodes.

Two dual scripts start_libupdate.csh and end_libupdate.csh runs as acron-jobs on CERNSP every night. These two scripts performs essentially all the tasks listed above including checking the disk space and removing old versions. The output from these the two acron-jobs are mailed to the dellib account from which it should be fowarded to the DELPHI offline analysis software librarian. The only task *not* performed by the two scripts is the actual release of the version. This *must* be done manually by the appointed DELPHI offline analysis software librarian after having carefully checked that all files have been properly installed and distributed to all the subscribed (DELSHIFT) nodes.

The rest of this section contains a detail description of the individual scripts in the software administration and distribution system. The scripts are listed in alphabetical order.

4.1 Scripts

Although all utility scripts which are parts of the software administration and distribution system will be described in this section, the librarian should normally only use the following: cleanup.csh, shift/mkall.csh, release.csh, rmversion.csh, update_share.pl and update_specific.pl.

4.1.1 check_space.csh

check_space.csh checks the current space used in the central repository and compares it with the AFS volume quota. Also the partition quota is checked since over-allocation of AFS volumes may cause out-of-space on the partition. The alarm limits are set by the two variables volumelimit (currently 80%) and partitionlimit (currently 95%). check_space.csh returns 0 if the alarm limits are *not* exceeded and 1 in all other cases. check_space.csh takes no arguments. The product context is given by the basename of the directory from where it runs, e.g. it is dstana if the working directory is ~/public/etc/dstana (see section 2).

4.1.2 check_status.pl [*version*]

check_status.pl checks the current status of the library compilations (if any) running on the reference machines. The status information (written by makedlib) is taken from the file upderr.log which is local on each reference machine. check_status.pl reads the library building configuration file build_defs.pl and retrieves the upderr.log files using " rcp" from each nodes/directories given by the arrays @BUILD_MACHINES and @BUILD_DIRS . The following checks are done:

(a) Problems reported by makedlib for any of the libraries.

(b) Checks that there is one record for each library given by associative array %LIB_RESPONSIBLES .

(c) Checks for version-consistency between the reference machines so that there are not some machines compiling e.g. 960826 while others are compiling 960818.

If check_status.pl is called with the *version* argument, the version consistency is checked against the specified version.

If at least one of these checks does not return success status (or in case of internal problems) check_status.pl sends a mail to dellib@afsmail.cern.ch and exits with error (1) status. If makedlib reports ypatchy or compilation problems with a library check_status.pl also sends a mail to the appropriate library responsible (given by %LIB_RESPONSIBLES).

Since check_status.pl reads the build_defs.pl files it must run in a product configuration directory. The product context is given by the basename of the directory from where it runs, e.g. it is dstana if the working directory is ~/public/etc/dstana (see section 2).

The status records in the upderr.log have the format:

```
#<version>:<status>:<library>:<cradle>
```

For instance:

```
#961004:success:libdstanaxx.a:dstanaxx.cra
#961004:fortran:libskelanaxx.a:skelana.cra
#961004:ypatchy:libvdclapxx.a:vdclapxx.cra
#961004:success:libuxxx.a:uxxx.cra
#961004:success:libufieldxx.a:ufield.cra
#961004:success:libuhlibxx.a:uhlib.cra
#961004:success:libpxdstxx.a:pxdstxx.cra
#961004:success:libtanagraxx.a:tanagraxx.cra
#961004:success:libtriggerxx.a:triggerxx.cra
#961004:success:libpxtag2xx.a:pxtag2xx.cra
#961004:success:libherlibxx.a:herlibxx.cra
#961004:success:libkalxx.a:kalxx.cra
```

4.1.3 cleanup.csh

cleanup.csh cleans up non-released versions in the DELPHI AFS project space. Only versions which are newer than the current production version are removed. If there are non-released versions which are older than the current production version cleanup.csh reports a warning but does not remove the version. This must be done by calling rmversion.csh (see section 4.1.14) directly for that version.

cleanup.csh uses the criteria that neither the AFS ACL group xx:user nor xx:swtest (for the prerelease) are in the AFS ACLs of the root share directory of the version in the DELPHI AFS project area.

cleanup.csh does not take any argument.

4.1.4 cmp_dirs.pl *dir1 dir2*

cmp_dirs.pl compares the files in *dir1* with the files in *dir2* and returns non-zero status in either of the two cases:

(a) one or more files in *dir1* are missing in *dir2*

(b) one (or several) files in *dir2* whose content is not identical with the the content of the corresponding file in *dir1*.

Note that it is the *dir1* which determines which files cmp_dirs.pl looks for in *dir2*. Thus, *dir2* may contain files which are not present in *dir1*. cmp_dirs.pl should be executed in the directory at the level next above *dir2*. cmp_dirs.pl can be used to check whether a package export area (under AFS) is different from the current share area in the central repository. For instance:

```
cd $DELPHI_PAM/..
~/public/etc/cmp_dirs.pl ~dstana/public/export/car car
```

compares the current $DELPHI_PAM with the pams in the dstana export area.

4.1.5 daily_tasks.csh

daily_tasks.csh is called by start_libupdate.csh. It simply calls other scripts to perform daily tasks like copying the latest delana from the *FARM* or distributing new luminosity or btagging calibration files to the DELSHIFT nodes.

4.1.6 distribute_fat_grou.csh

distribute_fat_grou.csh checks whether the FAT_GROU.NAMES file has changed in the export area and if so the new version is copied to the current production version in the central repository and distributed to all DELSHIFT nodes. FAT_GROU.NAMES is a particularly critical auxiliary data file which contains all FATMEN nicknames for the different DELPHI datasets. distribute_fat_grou.csh is executed as a separated acron-job once per hour. distribute_fat_grou.csh does not take any arguments.

4.1.7 distribute_files.pl *export-dir*

distribute_files.pl checks the export directory *export-dir* for new auxiliary data files which are not present in the current production version in the central repository. If it finds any *new* files they are copied to the current production version in the central repository as well as to all DELSHIFT nodes. Note that existing files will never be *updated* in the current production version, only new files can be added.

Only a restricted set of auxiliary data files are qualified to be distributed this way. They are normally related to new datasets which cannot be analysed without the presence of these files. Source code files should *never* be distributed this way (distribute_files.pl is protected against this). Source code should always be synchronously released with a new version (as was stated in the rules listed in section 1).

distribute_files.pl is called by daily_tasks.csh (section 4.1.5).

4.1.8 end_libupdate.csh

end_libupdate.csh runs as a daily acron-job dual to start_libupdate.csh. It performs the following tasks:

- It checks whether there is a pending new version or prerelease (created by start_libupdate.csh) which needs to be installed.
- If there is a new version (or prerelease) it checks calls check_status.pl to see if the compilation has finished and that everything is OK.

- If check_status.pl returns OK, end_libupdate.csh calls the update_specific.pl script with the -assemble option to install the new libraries in the specific area in the central repository.
- After the libraries have been successfully installed end_libupdate.csh calls shift/mkall.csh to distribute the new version to all subscribed nodes (the DELSHIFT cluster).
- If check_status.pl returns not OK, end_libupdate.csh calls cleanup.csh to remove the unsuccessful version.

If a new official version (not a prerelease) is successfully installed and distributed by end_libupdate.csh it can be released by the librarian using the release.csh utility. If the version was not successful or if the version was a prerelease the librarian does not have to do anything (a prerelease is never *released* it is simply installed and opened to the AFS ACL group xx:swtest). In case of system or compilation problems the librarian might need to follow up the cause for the problems.

4.1.9 mkxxlinks.pl

mkxxlinks.pl is the script which links the access paths in the DELPHI AFS group area /afs/cern.ch/group/xx/dstana to the central repository in the DELPHI AFS project area /afs/cern.ch/delphi/share/dstana and /afs/cern.ch/delphi/@sys/dstana (similarly for the database). mkxxlinks.pl does not take any arguments. It should be started from the root directory for the access path (e.g. /afs/cern.ch/group/xx/dstana for the dstana product). Normally the librarian does not need to call mkxxlinks.pl directly, it is called from release.csh whenever a new release is put in production.

4.1.10 purge.pl *version*

purge.pl disconnects *version* (removes all dependence on it from other versions) in the central repository. It does *not* remove the version. It has to be called for individually for the share directory and each of the specific directories of the version. purge.pl takes one argument, the version to be removed. purge.pl should run from the directory at the level next above the root directory of the version to be removed. For instance:

```
   prompt > cd /afs/cern.ch/delphi/share/dstana
   prompt > ls
960617      960723      960814      960826
960715      960731      960818      prerelease
   prompt > ~/public/etc/purge.pl 960723
```

purge.pl does not disconnect released version, i.e. a version which is opened to the AFS ACL group xx:user. purge.pl functions as follows:

- All real files in *version* are moved upstream to next higher version and a soft link is maintained from the previous path of the file to the new path (this prevents that the files "disappear" from the users while purge.pl is running). A file is removed if it cannot be moved upstream, i.e. there is already a real file in the next higher version.

- When all files have been moved or removed from the version all links pointing to it from other versions are updated to point to the corresponding files in the next higher version.

The librarian should normally *not* call purge.pl directly since a wrong usage can corrupt other versions in the central repository. The script rmversion.csh should be used to remove a version. Once purge.pl is running it should not be killed.

4.1.11 purge_oldest.csh

purge_oldest.csh disconnects and removes the oldest version in the central repository. It does not take any arguments and it is called from the daily acron-job start_libupdate.csh if there is not enough space in the central repository.

4.1.12 rcp_all.csh *file remote-path*

rcp_all.csh distributes the file *file* to the path *remote-path* on all DELSHIFT nodes. It is useful if there is an urgent need for distributing a new auxiliary data file.

4.1.13 release.csh *product version*

release.csh should be used to release and put in production the version *version* of the product *product* (dstana or database). release.csh does the following:

- Checks that the version is properly installed in the central repository.
- Opens the version to the AFS ACL group xx:user.
- Calls the script mkxxlinks.pl (see section 4.1.9) to create the access path to the version in the DELPHI AFS group directory.
- Does a rsh to pubxx account on shift10 and shift27 to execute the script new_libs which changes the group defined environment variables ($DELPHI, $DELPHI_PAM, ... etc.) to point to the new version and broadcasts to all interactive DELPHI users on shift10 and shift27 that the libraries have been updated.

Calling release.csh is the final step in the software administration and distribution system. It should always be executed manually by the librarian after he has checked that the version is properly installed everywhere (on AFS and DELSHIFT nodes).

release.csh can also be used to go back to the previous version if something went wrong in the current production version. For example, if a disastrous bug was found in the new version which cause all analysis programs to crash, there will be an urgent need to go back to the previous production version. On DELSHIFT it must then be made sure that the previous version is still installed (which is normally normally the case).

Using release.csh to go back one version works fine on DELSHIFT (which is the most urgent) but it has no effect on the current production version under AFS. This is because on AFS the group environment variables are automatically set to point to the access path for the latest version. To go back one version on AFS one simply remove the access path to the version in the DELPHI AFS group directory (/afs/cern.ch/group/xx/dstana for the dstana product). Note that this will not remove the version itself from the central

repository, only the access path to it. To entirely remove the version the librarian should use the rmversion.csh script.

4.1.14 rmversion.csh *version*

rmversion.csh is the librarian interface to remove a (released) version from the central repository. Before calling rmversion.csh the librarian must remove the AFS ACL group xx:user from the AFS ACL of the root share directory of the version. For example, assume that the dstana version 960617 shall be removed:

```
prompt > cd /afs/cern.ch/delphi/share/dstana
prompt > fs sa -dir 960617 -acl xx:user none
prompt > ~/public/etc/rmversion.csh 960617
```

rmversion.csh uses the DELPHI group environment variables to determine from which product (dstana or database) it shall remove the *version*. To switch between the dstana and database products the command dellevel should be used (do dellevel -h to find out the usage).

4.1.15 start_libupdate.csh

start_libupdate.csh runs as a daily acron-job dual to end_libupdate.csh. start_libupdate.csh follows different branches depending on the weekday. For instance, in the beginning of the week (Sunday or Monday) it checks if a new version should be created and proceed with the creation of one if it finds new code in the package export areas. If there is no need for a new version (i.e. the files in the package export areas have not changed) or if the weekday is not Sunday or Monday it tries to do a prerelease update. The motivation for this logic is that a new version should only be released in the beginning of the week when people are around. If a bug is found in the new release it is normally immediately reported by angry users and the librarian will switch back to the previous version. Releasing a new version on a Friday may stall all physics analysis jobs running on DELSHIFT during the whole Weekend.

A prerelease on the other hand can be done at anytime since it is only used by the people developing the code. A bug in the prerelease will not cause thousands of analysis jobs to crash.

The execution flow of start_libupdate.csh is as follows:

- Define the environment and check that the necessary AFS commands are present. Check if a version corresponding to todays date already exists and if so exit without further action. This could be the case if there was a manual updating done by the librarian during the day.
- Execute daily_tasks.csh see section 4.1.5.
- Cleanup unreleased versions (call cleanup.csh).
- Check if there is enough space in the DELPHI AFS project space. If not try first to remove tar-files. If this does not help call purge_oldest.csh until the space is sufficient.

- Determine whether the new version should be a prerelease or an official one. If it is Sunday start_libupdate.csh follows the official version branch. If it is Monday and there was no official Sunday release start_libupdate.csh also follows the official version branch. If there is no need for an official release or if it is not Sunday or Monday, start_libupdate.csh follows the prerelease branch.

- Call update_share.pl to update the share area with the new export files.

- If update_share.pl returned success status (i.e. library updating is needed) call update_specific.pl -distribute to distribute the compilation of the libraries to the reference machines.

4.1.16 update_rhosts.csh

update_rhosts.csh simply distributes an appropriate .rhosts file to all DELSHIFT machines. It is normally called from daily_tasks.csh .

4.1.17 update_share.pl

The two (perl) scripts update_share.pl and update_specific.pl constitute the kernel of the DELPHI software administration and distribution system.

update_share.pl takes no arguments. It is driven by the configuration files (perl-scripts) ⟨ package⟩ _config.pl. The PRODUCT context is given by the base-name of the current directory, e.g. PRODUCT=dstana if update_share.pl is started from /afs/cern.ch/user/d/dellib/public/etc/dstana .

update_share.pl is a rather complex script with several hard-coded patches to take care of dirty package export directories [1].

The log-output from update_share.pl is written to a file update_share.log in the current working directory. The log-output does only contain messages from update_share.pl. Any stdout or stderr output from system calls inside update_share.pl is not written to update_share.log.

In rough outline update_share.pl performs the following tasks:

- Setup a new *version* directory in the share area of the central repository. The new version directory contains no files, only links which are pointing back to files in previous versions. It can be considered as a clone of the last previous released version of the share files in the central repository.

- Require the ⟨ package⟩ _config.pl files one by one. The layout of the ⟨ package⟩ _config.pl scripts is described in section 2.2.

  - Call INIT to get a list of files in the package export area.
  - The format of the file-list depends on the operating system where the export area reside (UNIX or VMS). It is therefore decoded into a OS independent file list which is stored in an internal associative array. The decoding procedure calls FILTER_FILE_NAME for each file it finds in the export directory.

[1] Some package responsible prefer to use the export directory for the code development etc. which has as a consequence their export directory may contain emacs backup or recovery files, CVS administration directories etc.. All this "crap" should not be copied to the central repository.

  - From the decoded file-list a (perl) script is written to get the files. If the file already exist in on the client side (i.e. in the central repository) the script calls UPDATE. If the file does not exist the script calls NEW. For historical reasons the "get-file"-scripts are named: ⟨ package⟩ .cntx (get files in the current package context).
  - Call the ⟨ package⟩ .cntx script to get the files. The UPDATE routine should normally copy the file file-name to a temporary file file-name#package which is the file name update_share.pl searches for when it resolves file-clashes between the packages.

- Check for file-clashes between the packages. This procedure checks for file file-name#package. If there are more than one package which exported file-name the contents of the files are compared. If they are not different the file is silently installed as file-name. If the files are different they are left as they are and a clash warning is signalled in the log-file.

- If there were no file-clashes and if at least one file was updated, update_share.pl calls /bin/tar to produce two tar-files: src.tar and dat.tar . The tar'ing may be suppressed for either or both tar-files by setting (in the build_defs.pl file) the parameters $TAR_SRC and $TAR_DAT to 0 (default is 1).

- update_share.pl returns 0 if the updating was successful (no clashes) and if at least one file was updated. In all other cases it returns 1.

The decision to update a file from a package export area to the central repository is taken on the basis of the contents of the files. This is obviously a quite heavy task since all files must be read. Originally the file updating strategy was based on file modification times. However, this turned out to be unsafe because a package responsible may choose to go back to an older version of the file and a simple UNIX " mv" (/bin/mv) preserves the last modification time of the file.

In addition to the above tasks update_share.pl also make sure that there are only one level of soft links between versions in the central repository. Assume for instance that the last released version is 960916 and that it contains a link:

```
aabtagxx.car -> ../../../960617/src/car/aabtagxx.car
```

where the 960617 version contains the real file aabtagxx.car. When update_share.pl creates a new version it makes sure that aabtagxx.car points to the real file, i.e.

```
aabtagxx.car -> ../../../960617/src/car/aabtagxx.car
```

rather than the link

```
aabtagxx.car -> ../../../960916/src/car/aabtagxx.car
```

4.1.18 update_specific.pl

The two (perl) scripts update_share.pl and update_specific.pl constitute the kernel of the DELPHI software administration and distribution system.

update_specific.pl has two operation modes: distribution and assembling. The operation mode is determined by the by the flags -distribute and -assemble of which exactly one must be given.

As usual the PRODUCT context is give by the base-name of the current directory, e.g. PRODUCT=dstana if update_specific.pl is started from /afs/cern.ch/user/d/dellib/public/etc/dstana . update_specific.pl requires the configuration file build_defs.pl described in section 2.1. update_specific.pl calls several scripts which all should reside in the current working directory. Of the scripts listed below only mkVMSspecific.csh, mkspecific.com and makedlib.com are optional. All others must be provided.

The following scripts must be provided in the current working directory:

- distribute_share.csh : called by update_specific.pl in distribute mode. It copies to each reference machine the file src.tar which contains the entire src directory of the version to be built. distribute_share.csh also copies the two scripts mkspecific.csh and makedlib and starts the mkspecific.csh script on each reference machine.

- get_specific.csh : called by update_specific.pl in assemble mode. It copies the TAR files containing the compiled libraries (created by mkspecific.csh) and unTARs them in the appropriate specific area in the central repository.

- makedlib : is a sophisticated script which performs the actual compilation of a library. It takes two arguments: the full library name and the full cradle name. makedlib redefines the DELPHI environment variables $DELPHI_PAM and $DELPHI_LIB to point to ../src/car and ../lib respectively. Thus, makedlib is intended for creation of a local *version* of the target library and assumes that it is running within that *version*. In the current setup mkspecific.csh calls makedlib from a tmp directory within the local *version*. By redefining the DELPHI environment makedlib makes sure that it uses the pams within the new version and *not* in the production version when creating the libraries. For each operative step the return status must be checked. If something goes wrong the building of the target library should be immediately interrupted and a status record should be appended to the file ../upderr.log. In case of successful creation of the library a success status record should be written to ../upderr.log.

- mkspecific.csh : is started by distribute_share.csh on each reference machine. In the current setup mkspecific.csh is called with two arguments: the absolute path to the build directory (given by $BUILD_DIRS) and the version to be built (e.g. 960924). Called with these two arguments mkspecific.csh redirects its output to mkspecific.log and puts itself in background. mkspecific.csh cleans up the build directory and unTARs the src.tar file and calls makedlib (or phlib.csh) to build the libraries. mkspecific.csh is described in more detail in section 2.3.

- mkVMSspecific.csh : copies the DCL COM files mkspecific.com and makedlib.com and the two TAR-files src.tar *and* dat.tar to VXCERN and submits two batch jobs MKSPECIFIC.COM two ALPHA$LONG and VSCRNA_SYS$LONG for creation of ALPHA/VMS and VAX/VMS libraries respectively. mkVMSspecific.csh is optional (update_specific.pl calls it if present in current working directory).

- mkspecific.com : (optional) VMS DCL version of the mkspecific.csh script.

- makedlib.com : (optional) VMS DCL version of the makedlib script.

update_specific.pl assumes the version to be the most recent it finds in the *share* area of the central repository and creates, if necessary, the corresponding version directories in the *specific* area of the central repository.

4.1.19 shift/mkall.csh *version* [*working-directory*]

The mkall.csh script in the ~/public/etc/shift directory distributes and installs a ready version to all DELSHIFT machines. The *working-directory* argument is optional and if specified mkall.csh makes a cd to the working directory and puts itself in background. mkall.csh calls the script install script mkshift.csh for each node in the DELSHIFT cluster.

5 Validation of a library release

Before any validation of an **official** release, the librarian is testing its stability with a testjob. This section do not apply to prerelease mechanism, in which libraries contents is left to their authors for testing and (or) debugging purposes.

5.1 The librarian testjob

The `dellib` account have to check that a testjob, with which the mostly used routines of DELPHI official codes are called, thus tested, is running correctly with this hypothetical new release. The location of the `dellib` testjob area is the directory:

/afs/cern.ch/user/d/dellib/public/etc/testjob/

In this area:

- `skelexa.cra` is the cradle
- `skelexa.car` is the pam file
- `test_libraries.csh` is the script submitted to batch queues

All of these three files have been taken from the official DELPHI template area: `/afs/cern.ch/user/p/pubxx/public/testjob/`, with very small modifications to fit the librarian need.

The hypothetical new release libraries, source code and data files, after being copied to any DELSHIFT node (see the section on `mkall.csh`), are tested once for every DELSHIFT Operating System (e.g. HP-UX9, HP-UX10, OSF1 as of date of this note), resulting to the following files:

- [library_version]_$OS.o$QSUB_REQID

where:

- [library_version] is the name of the library release, like: 970720, for example.
- $OS is the Operating System, like HP-UX10.
- $QSUB_REQID is the NQS identifier of the batch job

For example resulting files for the test of the 970720 release were:

- 970720_HP-UX9.o52745 for the HP-UX9 Operating System,
- 970720_HP-UX10.o52744 for the HP-UX10 Operating System,
- 970720_OSF1.o52743 for the OSF1 Operating System

If the status of all the scripts running on all the possible DELSHIFT Operating Systems is good, e.g. :

- no crash during patchy-ing, compilation, linking and execution
- the desired output files are created (currently: one ntuple and selected events on another file)
- global status code is 0

The `release.csh` script is executed and the tested version become the new official one.

5.2 Putting Information on the World Wide Web

The next step is to inform all DELPHI people of this new release: the librarian have to:

- send a mail to the experts of delphicore mailing-list (delphi-core@delphi-lb.cern.ch) and to the other librarians (currently: the one for the in2p3.fr cell, and the one for the Linux Operating System). This mail will be archived automatically, accessible through the delphicore WWW page: http://delphiwww.cern.ch:8001/archive/delphicore/Welcome.html
- Once this mail is appearing into the archive (maximum 10 minutes after the mail send to delphicore mailing-list is received), he have to update the file `/afs/cern.ch/user/d/dellib/www/release_history.html` to reflect this change into its own WWW page, creating a link to the mail archived in the delphicore server
- send a news to the newsgroup called DELPHI.OFFLINE, telling people that a release has been achieved, and a link to its WWW page, namely: http://wwwcn.cern.ch/~dellib/release_history.html This news will be visible from the DELPHI NNTP server at: http://delnews.cern.ch:8065/VAXNEWS