Guide de l'administrateur

et

du développeur

T4–T5

Luc Weber

23 juillet 2004

Table des matières

Chapitre 1

INTRODUCTION

Ce document contient des informations pour le développement et la maintenance du logiciel d'observation pour les projets T4 (Euler) ou T5 (Mercator).

Toutes les personnes ayant un implication liée dans le développement ou dans la maintenance du logiciel d'observation sont concernées par ce document.

Les observateurs y trouveront des réponses à leurs interrogations techniques.

Pour permettre à ce document d'être le plus fidèle à la réalité. Peu de fichiers sont listés en exemple. C'est au lecteur de lister les fichiers en usage dans le projet télescope.

Chapitre 2

LES VERSIONS DU LOGICIEL D'OBSERVATION

2.1 Définition des versions alpha, beta, ok

On distingue d'une manière globale et pour la plupart des utilitaires trois versions de logiciel :

alpha	la version de développement
beta	la version d'intégration
ok	la version de sauvegarde

- La version de développement (**alpha**) est la version du développeur.
- La version d'intégration (**beta**) permet l'assemblage de modules provenant de plusieurs développeurs. C'est la version de travail, utilisée par les observateurs, donc la version par défaut.
- La version de sauvegarde (**ok**) est la version validée de la version d'intégration. C'est donc une version de sauvetage que l'on peut utiliser si la version **beta** ne donne pas satisfaction.

Le choix d'une version est fait au moyen fichiers d'initialisations spécifiques ou d'assignation de variables d'environnement.

2.2 Sélection de la version de travail

Le choix d'une version est fait lors du login de l'observateur. Le choix de travailler en **beta** est fait automatiquement lors de la toute première initialisation d'un compte observateur. Changer de version se fait avec les commandes suivantes :

À La Silla et à Genève :

```
Lors du login, l'utilitaire "ugtool" fait la sélection
des "packages" nécessaires à l'observateur. Ainsi:

Édition de ~/.ugtoolrc, qui doit contenir une
seule des deux lignes suivantes:

pour la version beta:    login: t4_beta
pour la version ok:      login: t4
```

À La Palma :

```
Le fichier  ~/.cshrc      lance  ~setup/cshrc
            ~setup/cshrc  lance  ~setup/t5_init.csh
en présence du fichier ~/USE_T5_OK, ~setup/t5_init.csh initialise
                la version ok
ou sans la présence de ce fichier initialise
                la version beta.
Ainsi:
pour la version ok:    touch ~/USE_T5_OK
pour la version beta:  rm    ~/USE_T5_OK
```

2.3 Sélection partielle de versions (usage de $INITENV)

Principalement utilisée par les développeurs, cette sélection permet d'utiliser la version **alpha** d'une partie des utilitaires (notament ceux du dévellopeur) et la version **beta** des autres utilitaires.
Cette sélection se fait dans le fichier d'initialisation de login avec par exemple :

À La Silla et à Genève :

```
~/.login.local qui lance ~/.login.t4
```

À La palma :

```
~/.cshrc_local qui lance ~/.login.t5
```

Voir comme exemple :

```
~weber/.login.t4
```

Une des principale caractéristique de cette initialisation est de définir la variable `$INITENV`.
Cette variable donne le nom d'un fichier de définition qui fixe le choix des versions de chaque utilitaire.
Le format de ce fichier est :

```
# commentaires
<APPLICATION> <version>[+<option>...]
```

Par exemple :

```
# commentaires
INTER          alpha
SPECTRO        alpha
SPECTRO_SRV    alpha+simulateur
```

2.4 Sélection de la version par défaut

Si on a fait une sélection partielle de version, on peut à tout moment, retrouver la version par défaut dans une fenêtre donnée.

Ce passage à la version par défaut est à employer par les développeurs lors des mise à jour des versions **beta** et **ok**.

À La Silla et à Genève, avec l'utilitaire *ugvt* :

```
beta:  ugvt t4_beta
ok:    ugvt t4
```

À La Palma avec les alias :

```
beta:  t5_beta
ok:    t5_ok
```

2.5 Sélection de version pour un utilitaire

Cette sélection est utilisée principalement lors du test d'un utilitaire. Par exemple lorsque que l'observateur travaille en **beta** il demande l'utilisation de la version **alpha** pour un test ou la version **ok** si la version **beta** présente un problème.
Il y a deux manière de sélectionner la version d'un utilitaire (ici un exemple avec edp) :

1. Assignation de la variable d'environnement lié à l'utilitaire :

   ```
   setenv EDP alpha
   ```

2. Assignation dans xrunall :

   ```
   suivre:
      xrunall->Edition de l'environnement->versions->edp->alpha
   ```

Chapitre 3

INSTALLATION

3.1 Variables d'environnement prédéfinie

La variable d'environnement `$OPSYS` donne l'*operating system* et la version de cet OS. Elle est assignée de la manière suivante :

```
setenv  OPSYS  `uname -sr | tr ' ' '_'`

Exemple:
               SunOS_5.8
```

La variable d'environnement `$THOME` donne le nom du directory où s'installe le logiciel d'observation. Elle est assignée de la manière suivante :

```
à La Silla     (dans /unige/obs/util/env/t4*.*sh)
        setenv TROOT   /ccd2/t4
à La Palma     (dans /~setup/t5*.csh)
        setenv TROOT   /disk/02/T5

Pour beta:
        setenv THOME   $TROOT/beta
Pour ok
        setenv THOME   $TROOT/ok
```

Pour des raisons historiques, on peut rencontrer la variable `$T4HOME`. Elle vaut `$THOME` et peut être remplacée par `$THOME`.

3.2 Fichiers de configuration d'installation

La liste exhaustive des applications à installer se trouve dans les fichiers de configuration dépendant du site d'installation et de l'*operating system* :

```
Emplacement:
        $THOME/config/install/
Nom:
        LISTE_APP_$OPSYS_<domainname>

Exemple:
        LISTE_APP_SunOS_5.8_obs.unige.ch
        LISTE_APP_Linux_2.4_mercator.iac.es
```

Chaque ligne donne le path de l'application de la version **alpha**. Exemple :

```
...
#
~blecha/scripts/
#
~maire/src/scripts/
~maire/src/t120/
...
```

L'installation utilise perl, il faut que l'exécutable `perl` existe dans `/usr/local/bin`. Il faut faire le lien si nécessaire.

3.3 Installation de la version beta

Une installation **beta** consiste à :

1. rapatrier les sources de chaque developpeur dans la structure $THOME/src
2. linker les fichiers includes dans le directory $THOME/include
3. fabriquer les libraries
4. linker les libraries dans le directory $THOME/lib/$OPSYS
5. fabriquer les exécutables

6. linker les exécutables dans le directory $THOME/bin/$OPSYS

L'ordre chronologique est essentiel dans cette instalation.

Les utilitaires permettant d'effectuer ces opérations travaillent tous de la même manière et indiquent ce qu'il vont faire avant de le faire (prompt). Pour des installations en batch, l'option "`-y`" permet de ne pas répondre aux questions.

3.3.1 Rapatrier les sources de chaque developpeur

Avec la commande :

```
        Tcopy.pl
ou
        ~weber/src/perl/Tcopy.pl
```

3.3.2 Fabriquer l'ensemble des liens (links)

Avec la commande :

```
        Do_all_links_beta.csh
ou
        ~weber/src/Do_all_links_beta.csh
```

3.3.3 Fabriquer les libraries

Avec la commande :

```
        Compile_lib.pl
ou
        ~weber/src/perl/Compile_lib.pl
```

3.3.4 Fabriquer les exécutables

Avec la commande :

```
        Compile_exe.pl
ou
        ~weber/src/perl/Compile_exe.pl
```

3.3.5 Séquence d'installation beta

La séquence des instructions pour une installation depuis scratch est :

```
<s'initialiser en ok>, voir section 2.4:
             "Sélection de la version par défaut"
cd $THOME/
rm -rf src/*
~weber/src/perl/Tcopy.pl
cd $THOME/src
~weber/src/Do_all_links_beta.csh
~weber/src/perl/Compile_lib.pl
~weber/src/Do_all_links_beta.csh
~weber/src/perl/Compile_exe.pl
~weber/src/Do_all_links_beta.csh
```

Les quatres dernières opérations (fabrication des libraries, des exécutables et création des liens) doivent se faire pour tous les *operating systems* à disposition (autant que de fichier `LISTE_APP_*`).

3.4 Installation de la version ok

La version **ok** est une copie (tar) de la version **beta**. Pour que cette copie fonctionne réellement en ok, il faut refabriquer les liens.

3.4.1 Séquence d'installation ok

```
<s'initialiser en ok>, voir section 2.4:
             "Sélection de la version par défaut"
cd $THOME/
~weber/src/Install_ok.csh  (effectue le tar et crée les liens)
```

3.5 Différences entre deux versions

Avec la commande :

```
        Tdiff.pl
ou
        ~weber/src/perl/Tdiff.pl
```

Permet de comparer les fichiers des applications listés dans les fichiers `LISTE_APP_*` avec les même fichiers d'une autre structure de directory. Cet utilitaire utilise les fichiers `Copies` pour determiner les fichiers à comparer.

Chapitre 4

LE FILE SYSTEM

L'arborescence des fichiers pour le projet T4–T5 est ancré sous le directory nommé dans la variable `$TROOT`. On y trouve les points de départ des versions **ok** et **beta**. C'est à dire :

```
$TROOT/ok/      == $THOME (en ok)
$TROOT/beta/    == $THOME (en beta)
```

Certains directories ne contiennent que des *links* pointant sur les fichiers des développeurs sous `$THOME/src` :

```
-$THOME-|-astromed---     distribution astromed--CCD
        |-bin--------     exécutables
        |-btl--------     exécutables transputer
        |-config-----     fichiers de configuration
        |-driver-----     drivers CCD
        |-includes---     fichiers includes C
        |-lib--------     librairies
        |-perl-------     modules perl
        |-prc--------     procédure inter
        |-scripts----     scripts
        |-smmacro----     procédure SuperMongo
        |-src--------     exécutables
        |-tpoint-----     distribution tpoint
```

4.1 Sources

Les sources des modules sont sous le directory `$THOME/src`. Leurs installations en **beta** est sous la responsabilité des développeurs.
La structure sous `$THOME/src` démarre avec les noms des utilisateurs puis avec les modules. C'est à dire (exemple) :

```
$TROOT-|-beta-|-src-|-maire-|-t120
                    |       |-t120_srv
                    |
                    |-weber-|-spectro
                            |-spectro_srv
```

4.2 Variables d'environnements

Les principales variables d'environnements suivantes sont prédéfinies pour les versions **beta** et **ok**. Par exemple pour beta :

```
$TROOT                  Home directory
$TDATA                  Data directory (nuit)
$CDATA                  Catalogues
$PHOME                  Base de Données Photomètre
$THOME                  $TROOT/beta
$INC_DIR                $THOME/include
$LIB_DIR                $THOME/lib/$OPSYS
$EXE_DIR                $THOME/bin/$OPSYS
$SCR_DIR                $THOME/scripts
$BTL_DIR                $THOME/btl
$CON_DIR                $THOME/config
$PRC_DIR                $THOME/prc
$SMO_DIR                $THOME/sm
$DEBUGGER               debugger
```

Chapitre 5

LES INSTRUMENTS, usage de $INSTRUMENT

Les instruments ont des noms réservés :

coralie	Spectromètre La Silla
camera2	Caméra CCD La Silla
camera3	Caméra CCD La Palma
p7	Photomètre La Silla
grb_guid_coralie	Caméra de guidage pour observation GRB
grb_imag_camera2	Caméra image C2 pour observation GRB
grb_imag_camera3	Caméra image C3 pour observation GRB

L'instrument est définis dans xrunall ou doit être déclaré interactivement pour tout lancement d'Inter en mode labo. Exemple :

```
setenv INSTRUMENT coralie
```

Chapitre 6

LES PROCÉDURES

```
Emplacement:
        $THOME/prc/<application>/

Nom:
        <nom>.prc
```

Ce sont les fichiers de commandes interprétés par les Inters :

- Inter (Synchro)
- CCD (Imager, Guidage)
- T120 (Euler, Mercator)
- Spectro (Coralie)

```
|-prc-|-DOC--------      (description des erreur en HTML)
      |-ERRCODE----      (texte et format des message d'erreur Inter)
      |-ccd--------      (ccd)
      |-elodie-----      (réduction Elodie OHP)
      |-p7---------      (photomètre P7)
      |-reduction--      (reduction Coralie)
      |-rfg--------      (roue de filtre guidage)
      |-rfi--------      (roue de filtre image)
      |-simccd-----      (simulation CCD)
      |-simrfg-----      (simulation rfg)
      |-simrfi-----      (simulation rfi)
      |-simspectro-      (simulation spectro)
      |-simt120----      (simulation t120)
      |-spectro----      (spectromètre Coralie)
      |-synchro----      (synchro)
      |-t120-------      (télescope)
      |-tacos------      (réduction Coralie)
      |-testccd----      (ccd pour CCD)
      |-util-------      (utilitaires)
      |-wfa--------      (wave front analysis)
      |
      |-may_be_not_in_use-|-ccd-----
                          |-p7------
                          |-rfg-----
                          |-rfi-----
                          |-spectro-
                          |-synchro-
                          |-t120----
                          |-testccd-
                          |-util----
                          |-wfa-----
```

6.1 Chemin d'acces aux procédures

Le chemin d'acces aux procédures est défini dans la variable Inter **DIRPRC**.
La variable **DIRPRC** est initialisée lors du login de chaque Inter. Voir comme exemple :

```
$THOME/prc/util/synchro_login.prc
```

Le directory courant est pris prioritairement dans la recherche d'une procédure.

6.2 Gestion des version avec RCS

Les procédures sont gérées sous RCS pour empêcher les modifications simultanées de plusieurs développeurs (par verouillage des fichiers) et garder un historique des modifications.
Un rappel de quelques commandes :

```
Initialiser RCS:            mkdir RCS ; chmod g+w RCS

Initialiser une
procédure dans RCS:         rcs -i -A<procédure_ref> <procédure> ;
                            ci -u <procédure>

Avant de modifier une
procédure:                  co -l <procédure>

Pour figer une
procédure:                  ci -u <procédure>

Récupérer une révision
particulière:               co -l<rev> <procédure>

Différence entre
deux révision:              rcsdiff [-r<rev>] [-r<rev>] <procédure>

Supprimer une
procédure:                  rm <procédure> RCS/<procédure>,v
```

6.3 Test de procédure

Il est préférable de tester (ou faire tester) une procédure avant de la mettre en commun.
La technique habituelle est la suivante :

1. vérouillage de la procédure (`co -l <procedure>`)

2. copie dans le directory courant
3. édition - test
4. copie dans le directory d'origine
5. dévérouillage de la procédure (`ci -u <procedure>`)

6.4 Procédures inutilisées

Pour l'installation de C3 (de janvier à juin 2004), un tri des procédures a été effectué dans le but de réduire le nombre de procédures à celles réellement utilisées.
Les procédures n'ayant pas été utilisées durant plusieurs mois ont été déplacées dans une structure identique à la structure de base sous le directory :

```
$THOME/prc/may_be_not_in_use/
```

Le chemin d'acces aux tient en compte ce directory lors des logins des Inters. Ainsi, par exemple l'Inter–Imager qui recherche une partie de ses procédures dans
`$THOME/prc/ccd`
va également les rechercher dans
`$THOME/prc/may_be_not_in_use/ccd`
Cette technique permet d'accéder d'une manière transparente les procédures qui ont été déplacées de manière "involontaire".

6.5 Notification des procédures inutilisées

Chaque fois qu'une procédure est accédée dans le directory `$THOME/prc/may_be_not_in_use/` un message d'information est inscrit dans le fichier `.prc_found_in_may_be_not_in_use` dans le directory de travail de l'observateur. C'est donc à l'administrateur de surveiller l'émergence de tels fichiers et résinsérer les procédures dans la structure standard (voir section suivante).

6.6 Réinsertion d'une procédure inutilisée

La commande `reuse` déplace une procédure de la structure `$THOME/prc/may_be_not_in_use/` à la structure `$THOME/prc/`. Exemple :

```
reuse  util  <procedure sans extension>
```

Chapitre 7

LES FICHIERS DE CONFIGURATION

Il en existe de plusieurs types et donc de plusieurs syntaxes différentes :
- fichiers de configuration pour Inter (lus par `read_config.prc`)
- fichiers de configuration CCD (lus par les controlleurs astromed)
- fichiers de configuration perl (lus par les utilitaires perl)
- squelettes FITS (modèles pour l'enregistrement des descripteurs FITS)

Ils sont sous `$THOME/config` dans la structure suivante :

```
$THOME-|-config-|-ccd---------|-applic-----
                |             |-chip-------
                |             |-controller-
                |             |-fits-------
                |-coralie-----
                |-environment-
                |-errcode-----
                |-general-----
                |-grb---------
                |-guidage-----
                |-icon--------
                |-imager------|-fits-------
                |-install-----
                |-rdbselect---
                |             |-fits-------
                |-reduction---|-llcal------
                |             |-masks------
                |             |-prc--------
                |-sm----------
                |-sounds------
                |-spectro-----|-fits-------
                |-synchro-----|-fits-------
                |-system------
                |-t120--------|-applic-----
                |             |-fits-------
                |-tacos-------|-fits-------
```

7.1 Fichiers de configuration CCD

```
Emplacement:
   $THOME-|-config-|-ccd---------|-applic-----
                                 |-chip-------
                                 |-controller-
                                 |-fits-------
```

Les fichiers de configuration sont lus par `read_config.prc` lors du boot d'un Inter-CCD (Inter-Imager et Inter-Guidage) :

```
Emplacement:
        $THOME/config/ccd/applic/<instrument>/
Nom:
        server.cfg
```

Ces fichiers donnent les caratéristiques physiques du chip (taille, pre-over scans, centre de champ, ...) ainsi que le nom du controleur et du chip.

Ces deux dernières valeurs determinent les fichiers de configuration envoyés au controlleur CCD.
Ceux-ci sont :

```
Emplacement:
        $THOME/config/ccd/chip/<nom_du_chip>/
Nom:
        ccd.cfg
        ccd.def
```

```
Emplacement:
        $THOME/config/ccd/controller/<nom_du_controlleur>/
Nom:
        4200hw.def
        atlinit.btm
        camera.r2h
        hard.cfg
        install.btl
        procinit.r2h
```

7.2 Fichiers de configuration T120 (modèle de pointage)

```
Emplacement:
        $THOME/config/t120/applic/<application>/
Nom:
        server.cfg
```

Les fichiers de configuration sont lus par `read_config.prc` lors du boot d'un Inter-T120 (Inter-Euler et Inter-mercator).
Ces fichiers donnent les caratéristiques physiques du télescope et les paramètres du modèle de pointage.

7.3 Fichiers de configuration de l'environnement

```
Emplacement:
        $THOME/config/environment/
Nom:
        <application>_<site>_<instrument>.cfg
        general_<site>_<instrument>.cfg
```

Ces fichiers décrivent pour chaque application :

la géometrie : (∗_GEO) position du coin haut droit et pour certains le taille

le display : (∗_DIS) identificateur de l'écran

le workspace : (∗_WSP) le nom du plan de travail

autre : (∗) variables d'environnement associées

Ces fichiers sont lus par *xrunall* qui se configure en fonction de leur contenu.

ATTENTION, la configuration de *xrunall* se fait une seule fois lors de son lancement. Ainsi *xrunall* se configure avec les termes contenus dans le fichier de configuration de l'instrument par défaut. Il est donc essentiel que sur un site tous les fichiers de configuration definissent les mêmes variables, quelque soit l'instrument, sinon les variables des autres instruments ne seront pas visibles dans *xrunall*.

Les instruments GRB ayant les mêmes configurations que les instruments d'observation, leurs fichiers de configurations sont donc des links sur les fichiers configurations des instruments d'observation.

Nous avons donc (exemple pour La palma) :

```
general_lapalma_camera3.cfg
general_lapalma_p7.cfg
guidage_lapalma_camera3.cfg
guidage_lapalma_p7.cfg
imager_lapalma_.cfg                    -> imager_lapalma_camera3.cfg
imager_lapalma_camera3.cfg
imager_lapalma_grb_imag_camera3.cfg    -> imager_lapalma_camera3.cfg
rfg_lapalma_camera3.cfg
rfi_lapalma_camera3.cfg
rfi_lapalma_grb_imag_camera3.cfg       -> rfi_lapalma_camera3.cfg
t120_lapalma_camera3.cfg
t120_lapalma_grb_imag_camera3.cfg      -> t120_lapalma_camera3.cfg
t120_lapalma_p7.cfg
```

Remarque : le fichier *imager_lapalma_.cfg* peut être utile en mode labo si l'utilisateur n'a pas défini $INSTRUMENT.

7.4 Fichiers de configuration perl

```
Emplacement:
        $THOME/config/general/

Nom:
        <utilitaire>.cfg
```

Ce sont des links vers les directories des sources perl des développeurs. Ils contiennent les paramètres liés à l'aparence (jeux de caratères, couleur) et aux paramètres de configuration (définition des champ (edp), menus (uif), ...).

7.5 Squelettes FITS

```
Emplacement:
        $THOME/config/<application>/fits/

        avec <application>:

                - synchro     - imager
                - ccd         - tacos
                - t120        - spectro

Nom:
        descripteurs.ske
        <instrument>_descripteurs.ske
```

Ils ont un format similaire à un header FITS mais possèdent des caractères de fin de ligne. Ils peuvent donc être édités.
La syntaxe est la suivante :

```
Format numérique simple précision:
UNSEQ   =              9999.999 / u.u_nseq   / No de sequence unique

Format numérique double précision:
JD      = '9999.999'            / s.jd       /DBL/ Jour Julien

Format caractère:
OBSERVER= '9999.999'            / g.observer / Liste des observateurs
```

A noter :

- Le contenu d'un descripteur est remplacé par le contenu de la variable Inter nommée entre les deux premier slash ("/") du commentaire
- S'il n'y a pas de variable Inter, le contenu reste celui donné dans le squelette
- La présence de /DBL/ dans le commentaire force le formatage en double précision de la variable Inter de type caratère qui contient un chiffre
- Dans le squelettes apparaissent des lignes contenant des descripteurs nommés `DUMMYDxx` précédant des blocs de commentaires. Ces descripteurs sont placés ici car un bug dans la librarie libcfitsio enlève systématiquement chaque descripteur placé avant un bloc de commentaires.

7.5.1 Procédures utilisant les squelettes FITS

Les squelettes sont utilisés par les Inters Synchro, Imager, T120 et Spectro principalement par les procedures :

camera_read_and_archive.prc pour Imager C2 et C3

ima_read_and_archive.prc : pour Imager coralie

w.prc pour flatfields des Imager C2 et C3 (@flats)

archive_tmp_descripteurs.prc (Synchro, T120 et Spectro)

7.5.2 Sélection des squelettes FITS

Si pour une application donnée, le fichier `<application>_descripteur.ske` n'existe pas alors le fichier `descripteur.ske` est pris en remplacement (c'est le cas pour Coralie).

7.6 Fichiers de configuration des poses GRB

```
Emplacement:
        $THOME/config/grb/
Nom:
        <instrument>.rdb
```

Ce sont des fichiers rdb contenant les informations de base pour l'observation **GRB**. La position de l'objet est rajoutée à ces fichiers lors d'une détection avant d'être envoyée à l'éditeur de pose (EDP).
Ils contiennent les colonnes :

- sequence
- centrage
- defoc

En usage actuellement :

```
coralie.rdb
grb_imag_camera2.rdb
grb_imag_camera3.rdb
```

Chapitre 8

LES FICHIERS DE MAINTENANCE

8.1 Fichiers de log des utilitaires

```
Emplacement:
        $TDATA/services/MAINTENANCE/log/<nuit>/
Nom:
        <utilitaire>.<tunix>
       (<utilitaire>.<tunix>.gz)
```

Remarque : Ces fichiers sont compressé automatiquement par `xrunall`.

Ces fichiers réunissent l'ensemble des messages envoyés sur la sortie standard de chaque utilitaire (Inters, scripts perl, ...). La valeur `<tunix>` est l'instant du lancement de l'utilitaire.

8.2 Fichiers de log de Xrunall

```
Emplacement:
        $TDATA/services/MAINTENANCE/xrunall/
Nom:
        <date>          (log du contenu de la fenêtre de xrunall)
        <date>.log      (log de xrunall)
```

Remarque : C'est l'administrateur qui compresse ou supprime les fichiers.

8.3 Fichiers de Crash Log

```
Emplacement:
        $TDATA/services/MAINTENANCE/crash_log/<nuit>/<tunix>_<tcl>/
Nom:
        last_15_min_full.log
        <utilitaire>.<tunix>
```

Remarque : C'est l'administrateur qui compresse ou supprime les fichiers.

Chaque déclenchement d'un "Crash Log" fabrique un directory dédié à un évênement dans un directory dédié à la nuit.
Dans ce directory sont stockés dans les fichiers `<utilitaire>.<tunix>` les 1000 dernières lignes de la sortie standard de chaque `<utilitaire>`. Le fichier `last_15_min_full.log` contient les messages datés (avec un tunix) de moins de 15 minutes avant déclenchement du "Crash Log", ordrés selon le temps, des sorties standards des utilitaires et du logbook.

8.4 Fichiers logbook

```
Emplacement:
        $TDATA/services/MAINTENANCE/logbooks/<nuit>/
Nom:
        full_logbook
        poses.edp
```

Remarque : C'est l'administrateur qui compresse ou supprime les fichiers.

Ces fichiers sont fabriqués ou copiés lors de la fin de nuit par l'Inter-Synchro. Il sont stocké dans ce directory car il font partie du directory de la nuit et donc peuvent être enlevés par l'observateur.

full_logbook : contient les messages datés (avec un tunix), ordrés selon le temps, des sorties standards des utilitaires et du logbook.

poses.edp : est une copie du fichier de pose de la nuit.

8.5 Fichiers Spectro Coralie

```
Emplacement:
        $TDATA/services/MAINTENANCE/spectro
Nom:
        <fonction>.log
        spectro_srv.log
```

Remarque : Ces fichiers ne doivent ni être compressés ni être supprimés.

Message de log des configurations du spectro. Le fichier `spectro_srv.log` contient les temps de configuration des électromécanismes. Il peut être utilisé pour la maintenance préventive et comparant les temps de transition de certains éléments.

8.6 Fichiers serveur t120

```
Emplacement:
        $TDATA/services/MAINTENANCE/t120_srv
Nom:
        t120_srv.<tunix>
```

Remarque : C'est l'administrateur qui compresse ou supprime les fichiers.

En usage sur demande, `t120_srv` fourni une quantité d'information beaucoup plus importante. Voir avec Charles Maire pour la mise en fonction de ce type de log

8.7 Fichiers t120 SDB

```
Emplacement:
        $TDATA/services/MAINTENANCE/t120sdb
Nom:
        t120sdb.<date>
```

Remarque : C'est l'administrateur qui compresse ou supprime les fichiers.
Ce fichier contient l'enregistrement des status dans une table rdb. Cette table contient 413 colonnes.

Chapitre 9

LES OUTILS DE MAINTENANCE ET DE DEBUG

Sont regroupés dans ce chapître les utilitaires et techniques en usage pour solutionner les problèmes informatiques.

9.1 Moniteur de sémaphores

Ipcmonit visualise en "temps réel" l'état des semaphores et du bloc de communication.

On l'enclenche avec un click sur la touche **start**.

Ce moniteur permet de libérer un Inter bloqué suite à un problème informatique. Ce cas n'arrive pas durant une nuit d'observation, mais peut arriver lors de test de procédure.

Par exemple : si un Inter est réservé et que le client qui l'a réservé crash avant de libérer le sémaphore, le client est bloqué. Une action sur le moniteur de sémaphore (`reset semaphore #1`) permet de revenir à une situation normale en libérant le client.

Plus d'information concernant les sémaphores dans le document :

"libipc.a"
Le tutorial
Synchronisation Client-Server

9.2 Suivi des opérations à distance

La commande Unix "tail -f" affiche les dernières lignes d'un fichier et celles qui s'y rajoutent.
C'est bon emploi de les utiliser sur les fichiers de maintenance cités plus haut.

9.3 Prompter

Chaque Inter en mode serveur peut être connecté par un ou plusieurs prompter. Il est ainsi possible d'envoyer des commandes *remotes*. Utilisé conjointement avec "tail -f", l'administrateur peut effectuer un semblant de commande à distance (attention, le flush sur stdout n'est pas vraiment performant dans ce cas là et il peut être utile de taper des commandes *dummies* pour permettre le flush (exemple : "`show /c`").

On lance un prompter en donnant l'identificateur du bloc de communication de l'Inter :

```
Commande:
                 prompter -k <key>

Avec <key>:

1000 pour CCD-Imager
1100 pour CCD-Guidage
2000 pour SPECTRO
3000 pour INTER-Synchro
3100 pour INTER-Reduc
4000 pour INTER-T120
5000 pour INTER-RFI
6000 pour INTER-RFG
```

9.4 Clone

Les utilitaires suivants peuvent travailler en mode clone :

- t120sdb
- spesdb
- ipcmonit

Ce mode permet lancer ces utilitaires sur une (ou plusieurs) machine distante, les connecter ensemble et transférer les informations de l'original au clone d'une manière manuel (conseillée pour la Silla) ou automatique. Ce mode de transfert est dépendant de la bande passante du réseau.
On les lance avec les options :

–lsclone : en remote sur La Silla

–lpclone : en remote sur La Palma

–clone : sur le même site

Attention le mode clone depuis Genève ne peut se faire que depuis obssf1 car les ports de connection n'ont été libéré par l'ESO que depuis cette machine.

9.5 Arborescence des procédures : prctree

Affiche l'arborescence depuis la procédure donnée en argument. Indique les appels par @ou par `call` :

```
prctree

Exemple:
           prctree acquisition_camera
```

9.6 Références croisées des procédures : prcpxref

Affiche les procédures appelée par et appelant la procédure donnée en argument :

- déclarée `global`
- déclarée `local`
- assignée
- utilisée

```
prcpxref

Exemple:
           prcpxref cimier
```

9.7 Références croisées des variables Inter : prcvxref

Affiche les procédures dans lesquelles la variable donnée est :

- déclarée `global`
- déclarée `local`
- assignée
- utilisée

```
prcvxref

Exemple:
          prcvxref e.nseq
```

Chapitre 10

LE DÉVELOPPEMENT D'UN MODULE

Ce chapître informe sur les méthodes et standards utilisés dans le cadre du projet T4 et T5 pour le développement des modules du logiciel d'observation.

10.1 Résumé

```
Pour développer un module if faut:

Standardiser le nom des modules avec le suffixe $OPSYS.

Placer les exécutables et les libraries dans les directories
adapté et avoir un $PATH correct.

Fabriquer un Makefile avec des targets standard T4-T5.

Fabriquer un Rulesfile dépendant de l'operating system.
```

10.2 Différenciation des *Operating Systems* (OS)

Lorsque l'on travaille sous différentes architectures ou sous différentes *release* d'un même OS, les fichiers binaires (exécutables, objets ou librairies) peuvent être incompatibles.
Pour distinguer les fichiers dépendant d'un OS, on les suffixe par le nom et la version de l'OS.
Par exemple, sous Solaris, SunOS version 5.8, le suffixe est :

```
SunOS_5.8
```

Les noms des fichiers deviennent par exemple :

```
inter    -> inter_SunOS_5.8
inter.o  -> inter_SunOS_5.8.o
libgop.a -> libgop_SunOS_5.8.a
```

Ce suffixe est donné par la variable d'environnement `$OPSYS`.

10.3 Exécutables

Les exécutables sont placés dans les directories :

```
alpha:  ~/bin/SunOS_5.8
beta :   $TROOT/beta/bin/SunOS_5.8
ok   :   $TROOT/ok/bin/SunOS_5.8
```

Ce directory est ajouté dans le *path* dans les fichier d'initialisation :

```
alpha:  setenv PATH   ... $HOME/bin/$OPSYS ...
beta:   setenv PATH   ... $THOME/bin/$OPSYS ...
```

Ce directory contient des *links* (sans suffixe) sur les exécutables situés sous `$THOME/src/....` Ainsi, quelque soit l'OS sous lequel on travaille, le nom d'un exécutable est toujours le même. Ex :

```
alpha:  ~/bin/SunOS_5.8/inter -> ~/src/inter/inter_SunOS_5.8
beta:   $THOME/bin/SunOS_5.8/inter ->
                   $THOME/src/weber/inter/inter_SunOS_5.8
```

10.4 Scripts interfaces aux exécutable

La plupart des utilitaires (exécutables ou scripts perl) ont un script interface qui permet de choisir la version (**alpha**, **beta** ou **ok**) et de donner un défaut aux variables dont l'utilitaire à besoin.
Ce script a le même nom que l'exécutable et a l'extension du shell utilisé. Exemple :

```
inter.csh
```

On ne lance donc pas directement l'exécutable, mais son script d'interface.

Ces scripts sont dans les directories :

```
alpha:  ~/scripts
beta :   $TROOT/beta/scripts
ok   :   $TROOT/ok/scripts
```

Ce directory est ajouté dans le *path* dans les fichier d'initialisation **avant** le directory des exécutables :

```
alpha:  setenv PATH   ... $HOME/scripts:$HOME/bin/$OPSYS ...
beta :  setenv PATH   ... $THOME/scripts:$THOME/bin/$OPSYS ...
ok   :  setenv PATH   ... $THOME/scripts:$THOME/bin/$OPSYS ...
```

il contient les *links* des interfaces aux exécutables (sans extension). Ex :

```
alpha:  ~/scripts/inter -> ~/src/inter/inter.csh
beta:   $THOME/scripts/inter -> $THOME/src/weber/inter/inter.csh
```

10.5 Lancement d'un exécutable

On lance un exécutable par l'intermédiaire de son script interface. Comme le script est dans un directory défini avant celui de l'exécutable dans le `path` c'est lui qui prime. Si le script n'existait pas, alors l'exécutable serait accédé directement.
Le script détermine son comportement à l'aide des variables d'environnement.
Le script utilise principalement une variable d'environnement ayant le même nom (mais en majuscule) que l'exécutable et définissant la version à utiliser et optionellement le mode de fonctionnement de l'exécutable.
Par exemple, la gestion du spectro Coralie est faite par le programme nommé `spectro_srv`. Pour le faire fonctionner en mode simulation, on donne simplement :

```
setenv SPECTRO_SRV beta+simulation
```

si en plus de la simulation on veut le *debugger*, on donne :

```
setenv SPECTRO_SRV beta+simulation+debug
```

Si un exécutable a un besoin spécifique de variables, elles doivent toutes commencer par le nom de la variable associée. Ex :

```
setenv SPECTRO_SRV_VERBOSE 3
```

Remarque : le choix du *debugger* se fait au travers de la variable d'environnement `$DEBUGGER`. Par exemple :

```
setenv DEBUGGER ddd
```

10.6 Help

Les options standards du help sont :

`"-h"`	pour les exécutables
`"-H"`	pour les scripts

Le *help* de l'exécutable renseigne sur les options de l'exécutable, tandis que celui du script renseigne plutôt sur l'environnement et l'usage des variables associées.
Remarque : dans tous les cas ces deux options doivent être interceptées, c'est à dire que si l'exécutable ne gère pas le `"-h"`, c'est le script qui doit le faire.

10.7 Les librairies

Les librairies sont des fichiers dépendant de l'OS, elles se nomment donc selon la manière définie pour les exécutables, par exemple :

```
libgop_SunOS_5.8.a
```

Elles sont dans :

```
alpha:  ~/lib/SunOS_5.8
beta:   $THOME/lib/SunOS_5.8
ok:     $THOME/lib/SunOS_5.8
```

Ce directory contient des *links* (sans suffixe) sur les librairies situés sous `$THOME/src/...` :

```
alpha:  ~/lib/SunOS_5.8/libgop -> ~/src/gop/libgop_SunOS_5.8
beta:   $THOME/lib/SunOS_5.8/libgop ->
                       $THOME/src/weber/gop/libgop_SunOS_5.8
```

Le `path` des libraries fournit dans le `Makefile` ou `Rulesfile` provient de la variable d'environnement $LIB_DIR. Définie par exemple comme :

```
setenv LIB_DIR ~/lib/SunOS_5.8
```

10.8 Les fichiers *includes*

Ces fichiers sont indépendant de l'OS car leur code est écrit en fonction de l'OS (*preprocessing*).
Ils sont dans le directory :

```
alpha:  ~/src/incl
beta:   $THOME/include
```

Ce directory contient des *links* sur les fichiers *includes*. Ex :

```
alpha:  ~/src/incl/gop.h -> ~/src/gop/gop.h
beta:   $THOME/include/gop.h -> $THOME/src/weber/gop/gop.h
```

Le `path` des fichiers include fournit dans le `Makefile` ou `Rulesfile` provient de la variable d'environnement $INC_DIR. Définie par exemple comme :

```
alpha: setenv INC_DIR ~/src/include
beta : setenv INC_DIR $THOME/include
ok   : setenv INC_DIR $THOME/include
```

10.9 Makefiles, Rulesfiles

Un *Makefile* contient principalement deux types de déclarations :
– les dépendances entre fichiers
– les ordres de constructions (compilation, *link*...)
Par nature, les ordres de constructions sont dépendant de l'OS alors que les dépendances ne le sont pas.
De ce fait est né l'utilisation de fichiers que l'on nomme *Rulesfiles* (standard T4–T5). Ce sont des fichiers suffixés selon `$OPSYS` et contenant les options de compilation, de *link*... Par exemple :

```
Rules_SunOS_5.8.mke
```

Ce fichier est inclus dans le *Makefile* en fonction de la variable `$OPSYS` de la manière suivante :

```
include Rules_${OPSYS}.mke
```

10.10 Makefiles : *Targets* obligatoires

Pour permettre les mises à jour globales. Chaque *Makefile* doit posséder les trois *targets* suivantes :

```
all: all_lib all_exe
all_lib: <liste des librairies construites
          dans ce directory>
all_exe: <liste des exécutables construits
          dans ce directory>
```

Avec ce système il est possible de fabriquer en deux temps l'ensemble des librairies puis l'ensemble des exécutables, en lancant :

```
make all_lib
make all_exe
```

Remarque : voir chapître : "INSTALLATION"

10.11 Makefiles : Exemple

```
include Rules_${OPSYS}.mke
LIBAFF     = libaff_${OPSYS}.a
AFFTORTURE = afftorture_${OPSYS}

all: all_lib all_exe
all_lib: $(LIBAFF)
all_exe: $(AFFTORTURE)

.KEEP_STATE:
.PRECIOUS: $(LIBAFF)

$(LIBAFF): $(LIBAFF)(libaff.o) aff.h $(INC_DIR)/gop.h
	$(RANLIB) $(LIBAFF)

$(AFFTORTURE): afftorture.f $(LIBAFF)
	$(F77) $(FFLAGS) afftorture.f -o $(AFFTORTURE)\
		$(LDFLAGS) $(LDLIBS)
```

10.12 Rulesfiles : Exemple

```
#
# definitions specifiques a l'operating system
#
TINC     = $(THOME)/include
TLIB     = $(THOME)/lib/$(OPSYS)

INCLUDE_PATH =  -I .                        \
                -I $(INC_DIR)               \
                -I $(TINC)                  \
                -I /usr/local/include

LIBRARY_PATH =  -L .                        \
                -L $(LIB_DIR)               \
                -R $(LIB_DIR)               \
                -L $(TLIB)                  \
                -R $(TLIB)                  \
                -L /usr/local/lib           \
                -R /usr/local/lib           \

CFLAGS  += -g -DSYSV $(INCLUDE_PATH)
FFLAGS  += -g

LDFLAGS  = $(LIBRARY_PATH)
LDLIBS   = -laff -lgop -ltpudummy -lsocket -lnsl

RANLIB  = echo
```

Chapitre 11

L'INTÉGRATION D'UN MODULE

11.1 Définition d'un module

On définit un module comme étant l'ensemble des fichiers nécessaires :

- à la construction d'une application
- à la documentation une application
- au fonctionnement d'une application
- au test d'une application

Une application peut être et/ou regrouper :

- un utilitaire principal
- un ensemble d'utilitaires annexes
- une librairie

D'une manière générale, quand deux applications peuvent avoir un développement indépendant, elles donnent naissance à deux modules distincts.

11.2 Résumé

```
Pour integrer un module if faut:

Le fichier Copies qui permet son tranfert avec Docopies

Le fichier Links qui crée les links avec Dolinks
```

11.3 Emplacement des modules

Il est important que les modules soient situés sous des *directories* donnés sans rapport de hiérarchie entre–eux. Par exemple :

```
alpha:  ~/src/inter
             /spectro
             /spectro_srv
             /gop

beta:   $THOME/src/weber/inter
                        /spectro
                        /spectro_srv
                        /gop
```

11.4 Copie de modules

La copie d'un module est plus plus évoluée qu'une simple copie (`/bin/cp`) car on doit pouvoir créer certains directories ou exécuter certaines commandes d'installation. De plus, dans notre cas, copier signifie surtout :

- copier uniquement les fichiers nécessaires à l'application
- copier que les fichiers modifiés, en cas de copies successives

Pour réaliser cela, chaque module posséde un fichier nommé `"Copies"` qui contient la liste des fichiers à copier ainsi que les ordres d'installation.
Les fichiers `"Copies"` sont interprétés par l'utilitaire `Docopies`.

Dans la pratique, `Docopies` est un script (perl) qui fabrique et exécute un *Makefile* effectuant des copies conditionnelles. Ce script permet également de copier intégralement un module sans tenir comptes des dates des fichiers avec :

```
setenv COPY_FORCE
Docopies <directory>/<application>
```

11.5 Format d'un fichier `Copies`

Un fichier `"Copies"` contient une liste de fichiers destinés à la copie, des ordres exécutables et des commentaires, selon le format suivant :

```
# commentaires
@<command> [<argument> ...]
<file_name>
```

Les commandes à disposition sont les suivantes :

```
mkdir $<$directory$>$ ...
                        crée <directory> s'il n'existe pas
cp_if_not_exist <file> ...
                        copie un fichier s'il n'existe pas dans
                        le directory de destination
cp_makefile
                        copie de Makefile ou makefile si aucun
                        makefiles n'existe
cp_if_defined <env_var> <file> ...
                        copie conditionnelle si la variable
                        <env_var> existe
execute_if_exist <file> <Unix_command>
                        exécute <Unix_command> si le fichier existe
execute_if_not_exist <file> <Unix_command>
                        exécute <Unix_command> si le fichier
                        n'existe pas
execute <Unix_command>
                        exécute <Unix_command>
make_copy
                        effectue la copie avant la fin du fichier
```

Remarques :

1. la variable `$ORG` donne le directory source
2. les lignes de commentaires sont affichées à l'écran

11.6 Utilisation de Docopies

L'utilisation de `Docopies` est la suivante :

```
cd <destination_dir>
Docopies <origine_dir>
```

De plus, si la variable `COPY_FORCE` est définie, la copie des fichiers ne tient pas compte des dates. Par exemple :

```
setenv  COPY_FORCE
Docopies <origine_dir>
```

11.7 Exemple d'un fichier `Copies`

```
# Copies file for inter (generic)
@cp_makefile
@cp_if_defined COPYCSH_RULE Rules_$OPSYS.mke
@cp_if_not_exist Rules_$OPSYS.mke
@mkdir for hlp exe tex tmp prc
@cp_if_not_exist buffer.par
Links
inter.f
for/int_intermidas.inc
copy_def.csh
@make_copy
@execute copy_def.csh $ORG
@execute_if_not_exist block.ref grep AAAAAA ...
                       ... inter.blk.def > block.ref
@execute inicommandref.sh
```

11.8 Création de link

Comme décrit dans ce document, l'accès aux fichiers se fait principalement au travers de *links*. Ces links sont fabriqués automatiquement.
Chaque module contient un fichier nommé `Links` qui contient sur chaque ligne : le nom du directory où doit se trouver le *Link*, le nom du fichier à *linker* et le nom du *link* s'il est différent du nom du fichier. Son format est le suivant :

```
# commentaires
<directory_du_link> <nom_fichier> <nom_link>
```

Par exemple :

```
$LIB_DIR libgop_$OPSYS.a    libgop.a
$LIB_DIR libsrvgop_$OPSYS.a libsrvgop.a
$INC_DIR gop.h
```

On effectue les *links* avec la commande :

```
Dolinks
```

11.9 Validation des versions

ATTENTION, aucun développement ne doit se faire dans les version beta et ok. En effet, partant du principe qu'il y a un espace de développement (version **alpha**), il ne peut pas y avoir de développement en aval de cet espace.
Lorsqu'une version est validée elle passe au niveau supérieur. Par exemple :

```
alpha    --->   beta
beta     --->   ok
ok       --->   sauvegarde sur CD
```

C'est le développeur qui valide sa version **alpha**, c'est à dire qu'il doit posséder un minimum de moyen de test à exécuter avant de propager ses applications.
La version **beta** est validée par son utilisation. C'est pour cela que les observateurs travaillent avec cette version.

11.10 Indépendance des Rulesfiles par rapport les versions

Les *Rulesfiles* doivent être portables entre les versions. C'est à dire qu'un développeur accède ses libraries en **alpha** lorsqu'il *link* en alpha, mais que ce même *Rulesfiles* accède uniquement les librarie en **beta** lors d'une compilation en **beta**. Pour réaliser cela, le *path* des fichiers *includes* et celui des librairies doivent être définis de la sorte :

```
TINC    = $(THOME)/include
TLIB    = $(THOME)/lib/$(OPSYS)

INCLUDE_PATH =  -I .                    \
                -I $(INC_DIR)           \
                -I $(TINC)              \
                ...

LIBRARY_PATH =  -L .                    \
                -L $(LIB_DIR)           \
                -L $(TLIB)              \
                ...
```

Ainsi en mode alpha le développeur qui a défini `$LIB_DIR` comme `~/lib/$OPSYS` utilise en priorité ses propres libraries, alors que ce même développeur travaillant en mode **beta** aura `$LIB_DIR` prédéfini comme `$THOME/lib/$OPSYS` et il utilisera la version **beta** pour toutes les libraries. Même remarque pour les fichiers includes.

11.11 Utilisation d'une librairie d'un autre développeur

L'utilisation d'une librairie d'un autre développeur (version **alpha**) est normalement momentanée (validation) et se fait manuellement en rajoutant un *link* dans son propre directory de librairies.
Par exemple, un développeur voulant utiliser la version **alpha** de `libgop.a` doit faire :

```
cd ~/lib/$OPSYS
ln -s ~weber/lib/$OPSYS/libgop.a .
```

Ce link doit naturellement être détruit après validation.