NOTES TECHNIQUES

SCIENCES DE LA TERRE

GÉOLOGIE-GÉOPHYSIQUE

N° 25

2002

Station sismologique et géodésique de Santo

Manuel opérateur

Pierre LEBELLEGARD
Jean-Michel DEVAUX
Jean-Louis LAURENT
Robert PILLET
Marc RÉGNIER

Institut de recherche
pour le développement

/Lebellegard, P.
/Devaux, J.-M.
/Laurent, J.-L.
/Pillet, R.
/Régnier, M.

Station sismologique et géodésique de Santo. Manuel opérateur.

Nouméa : IRD. Novembre 2002. 52 p.
Notes Tech. : Sci. Terre ; Géol.-Géophys. ; 25

SISMOLOGIE ; ACQUISITION DE DONNEES ; GPS. SYSTEME DE POSITIONNEMENT GLOBAL / VANUATU ; SANTO

Imprimé par la Service de Reprographie
Centre IRD de Noumea
Novembre 2002

-I – Introduction

Au mois de septembre 2002, l'IRD a installé sur le site de l'IRHO[4], sur l'île de Santo (Vanuatu), une station permanente d'acquisition sismologique et géodésique. La partie sismologique de l'acquisition est basée sur le système Par4ch de Symmetric Research ; la partie géodésique est basée sur un récepteur GPS Ashtech Z-XII couplé à une antenne Dorne Margolin-T. Les deux parties de l'acquisition sont pilotées par un PC. Même si l'ensemble de l'acquisition est entièrement automatisée, il nous a paru important de permettre à un opérateur d'intervenir, en cas de problème de fonctionnement. C'est le but de la présente note.

II – Description du système

II-1 : Acquisition sismologique

Le système d'acquisition est le système d'acquisition de données 4 canaux PAR4CH de chez Symmetric Research (http://www.symres.com) :

SR PAR4CH SPEC SHEET

The PAR4CH is a 4 channel, 24 bit data acquisition system with an individual A/D per channel architecture. Highest resolution occurs at 10-1000 Hz, with maximum sampling rates of 5 kHz sustained. All PAR4CH inputs are differential with amplitude ranges of +/- 10 volts. ***The on board 2 Mb DRAM allows continuous no loss data acquisition even on heavily interrupted and multitasking PCs****. The PAR4CH sits outside the PC for improved noise performance and communicates its acquired data using the standard PC parallel port.*

Fig . 1 : La carte d'acquisition et le GPS[5] intégrés dans le même boîtier

[4] Institut des Recherche des Huiles Oléagineuses

[5] **IMPORTANT** : il s'agit du boiter GPS qui permet de dater les données sismologiques. Le récepteur GPS de l'acquisition géodésique est un boîtier séparé, connecté par un câble coaxial à un antenne, située sur un mât à l'extérieur du bâtiment. Ce récepteur situé lui à l'intérieur est connecté au PC par un port série (COM2, à 38400 bauds).

Les cadences d'échantillonnage vont de 0 Hz à plusieurs KHz. Connecté au port parallèle d'un PC, le système est basé sur le convertisseur A/D Burr Brown ADS1210 24 bits; il y a un convertisseur pour chaque canal. De plus le système comprend une mémoire de 2 mégaoctets, ce qui permet une acquisition continue même en cas d'indisponibilité temporaire du PC.

Les spécifications complètes de l'acquisition sont les suivantes :

Channels:	4
Resolution:	Max: 24 bits Typical: 23 bits at 10 Hz, 21 at 100 Hz, 19 at 1 kHz
Sample Rate:	Min: 0.00001 Hz, Max: 5 kHz sustained Recommended: 10-3000 Hz
Gain:	1 to 16, program selectable
Crosstalk:	Down more than 140 dB (24 bits) at 100 Hz
Analog input type:	Differential
Common Mode Rejection:	102 dB
Analog input voltage:	+/-10v
Analog input impedance:	Approximately 16k ohms
Analog input header:	15 pin D-Shell or plain wire with TRM15E
Digital I/O:	4 input and 4 output bits on 15 pin D-Shell
Digital I/O levels:	TTL/CMOS 5 volt
PC Connection:	EPP/BPP/ECP PC parallel port
Power Requirements:	9 to 24 vdc or vac, 9 volts DC preferred, 100ma @+9v
A/D Converter:	Burr Brown ADS1210, 1 per channel
A/D Type:	Sigma Delta
Data Buffer:	2 Mb DRAM
Board Layers:	4 including separate power and ground planes
Board Dimension:	5 1/4" x 6"
Enclosure Dimension:	(5 1/2" W x 1 5/8" H x 6 1/2" D)
Supported OS:	WinNT 4.0, Win95/98, DOS, Linux, LabVIEW

Le logiciel d'acquisition qui pilote l'acquisition est le logiciel VDacq de PC Systems Design (http://www.pcsys-design.com/vdaq.htm). L'ensemble d'acquisition est couplé à une antenne GPS, pilotée par le logiciel GpsTime, qui permet de dater à la seconde près les données sismiques acquises. Il y a un fichier généré toutes les 6 minutes environ.

II-2 : Acquisition géodésique

L'acquisition géodésique consiste en la gestion d'une station GPS permanente : il s'agit d'un récepteur Ashtech Z-XII acquérant des données GPS, à la cadence d'une acquisition toutes les 5 secondes. Le récepteur est programmé pour fournir les données GPS sous forme d'un fichier par période de 24 heures (de 0 heure à 24 heures GMT), et gérer l'acquisition géodésique consiste à « vider » le récepteur, c'est-à-dire transférer le fichier de données du récepteur au PC toutes les 24 heures. Le constructeur fournit pour cela le logiciel Cgremote.

II-3 : Gestion des données

L'acquisition sismique génère environ 72 mégaoctets/jour, et l'acquisition géodésique environ 5 mégaoctets, ce qui fait une quantité annuelle de l'ordre de 30 gigaoctets. Bien que le système d'acquisition sismique dispose d'une mémoire tampon de 2 mégaoctets , il a été décidé de perturber au minimum l'acquisition en limitant le nombre des accès disque. Pour cela, on limite la taille du répertoire dans lequel sont stockées les données à la sortie du PAR4CH (C:\DATA, non paramétrable) : il contient tout au plus une heure de données. Les données sont recopiées dans un disque dur monté dans un rack extractible (« disque extractible », de taille suffisante pour assurer l'autonomie de la station entre deux passages de l'opérateur. En cas de perte du disque extractible en cours de transport, le disque dur est géré comme un tampon circulaire qui contient les données de moins de une année, les données plus anciennes sont automatiquement effacées. Il a par conséquent été écrit un programme de synchronisation (cf. Annexe 3) qui :

- Examine le répertoire C:\DATA, et recopie les données présentes sur le disque dur (C:\SANTO) et sur le disque extractible (D:\), si elle n'y sont pas déjà présentes. Ensuite, il élimine de ce répertoire les données plus anciennes que une heure. Le fait de limiter à une heure la quantité de données présentes dans ce répertoire limite le nombre de fichiers présents, donc le nombre de comparaisons, et par conséquent le nombre d'accès disque.

- Examine le disque dur (C:\SANTO), et élimine d'un coup les répertoires qui correspondent à des données antérieures à un an. Sur le disque dur et sur le disque extractible, les données géodésiques (GPS) sont rangées dans le répertoire GPS (un fichier par période de 24 heures), les données sismologiques sont rangées par année/mois/jour :

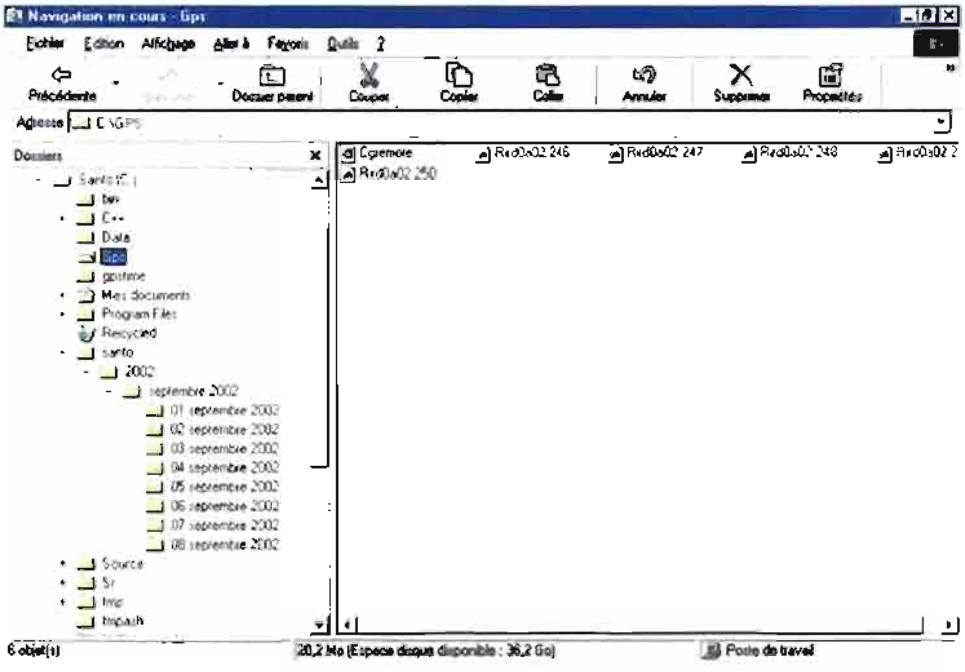


Fig . 2 : Arborescence du disque dur

II-3-1 : Emplacement des utilitaires

L'ensemble des exécutables développés et/ou utilisés ont été regroupés dans le répertoire C:\BIN :

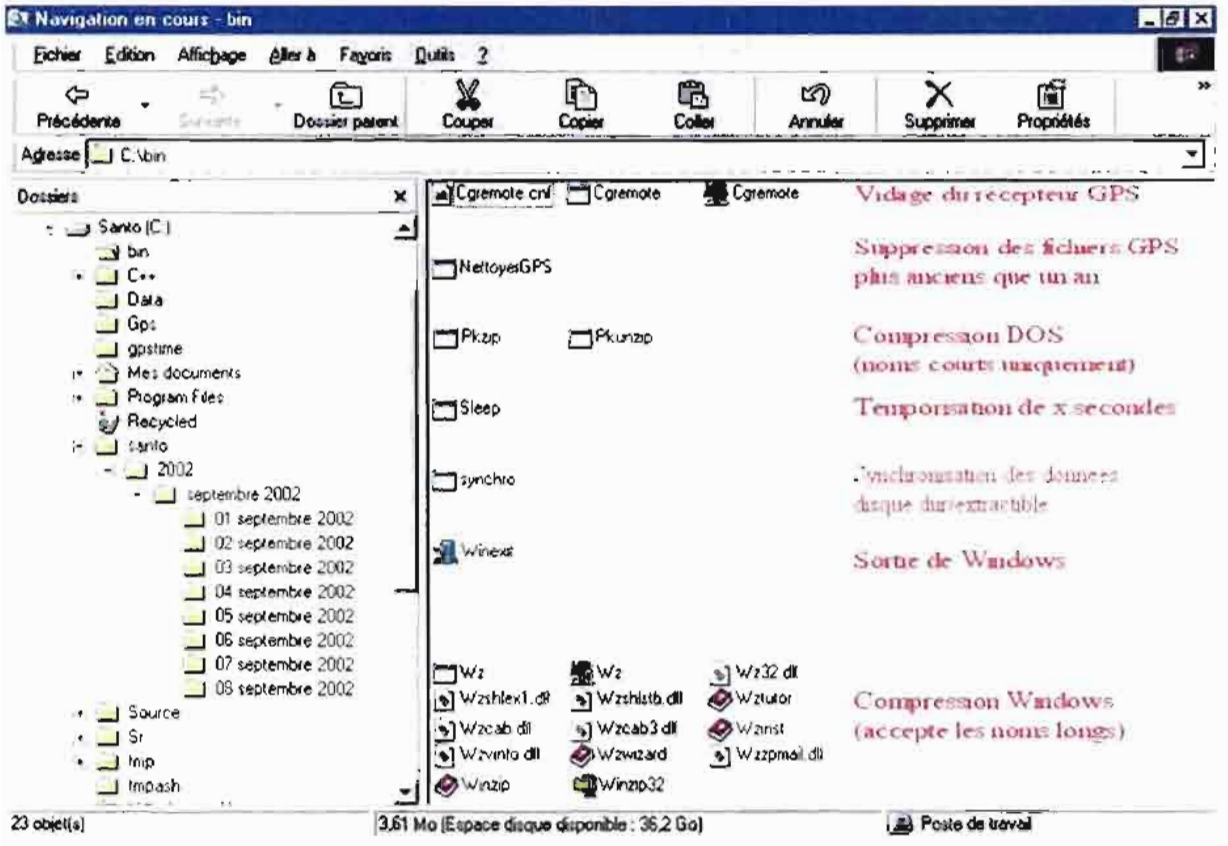
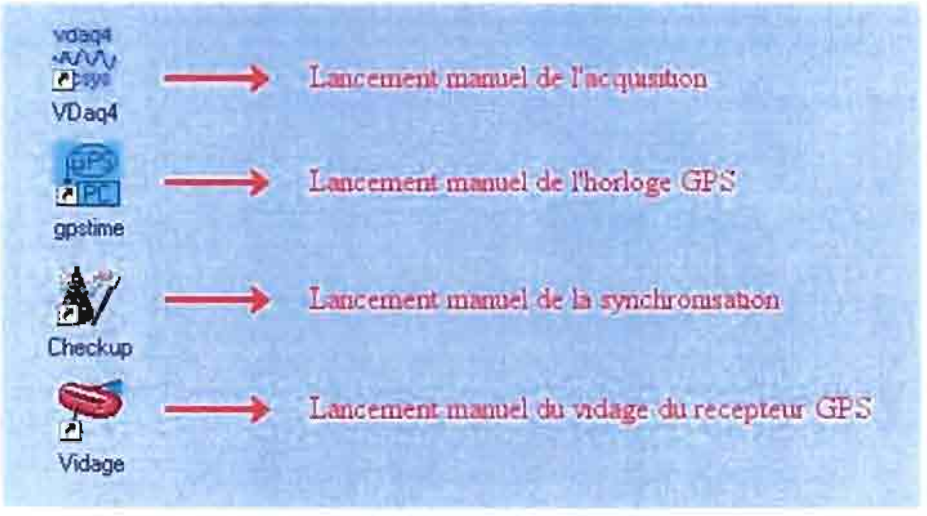


Fig . 3 : Répertoire des exécutables

II-3-2 : Fichiers de commandes et planification

Fig . 4 : Les fichiers de commande

L'appel des exécutables de synchronisation est paramétrable, et l'ensemble est regroupé dans des fichiers de commandes (« .bat »), situés sous C:\WINDOWS\IRD. La signification des principaux fichiers est la suivante :



L'utilitaire de vidage du récepteur GPS (vidage.bat) est lancé toutes les 24 heures :

```
C:
CD \
CD TMPASH
CGREMOTE -C 0 -D 00.0
COPY *.* C:\GPS
MOVE *.* D:\GPS
```

L'existence du répertoire temporaire C:\TMPASH est donc ABSOLUMENT INDISPENSABLE.

Si l'on veut effectuer un vidage manuel du récepteur GPS, on doit lancer vidage.bat, et non pas Cgremote directement.

L'utilitaire de synchronisation des données (checkup.bat) est lancé chaque demi-heure :

```
REM
REM Compression des fichiers OUT et synchronisation avec
REM le disque extractible. On garde dans C:\DATA les fichiers
REM qui ont moins de une heure, et dans C:\santo les fichiers
REM qui ont moins de un an.
REM
synchro c:\data c:\santo d:
REM
```

```
REM Suppression (sur le disque dur) des fichiers GPS de
REM plus de un an.
REM
NettoyerGPS
```

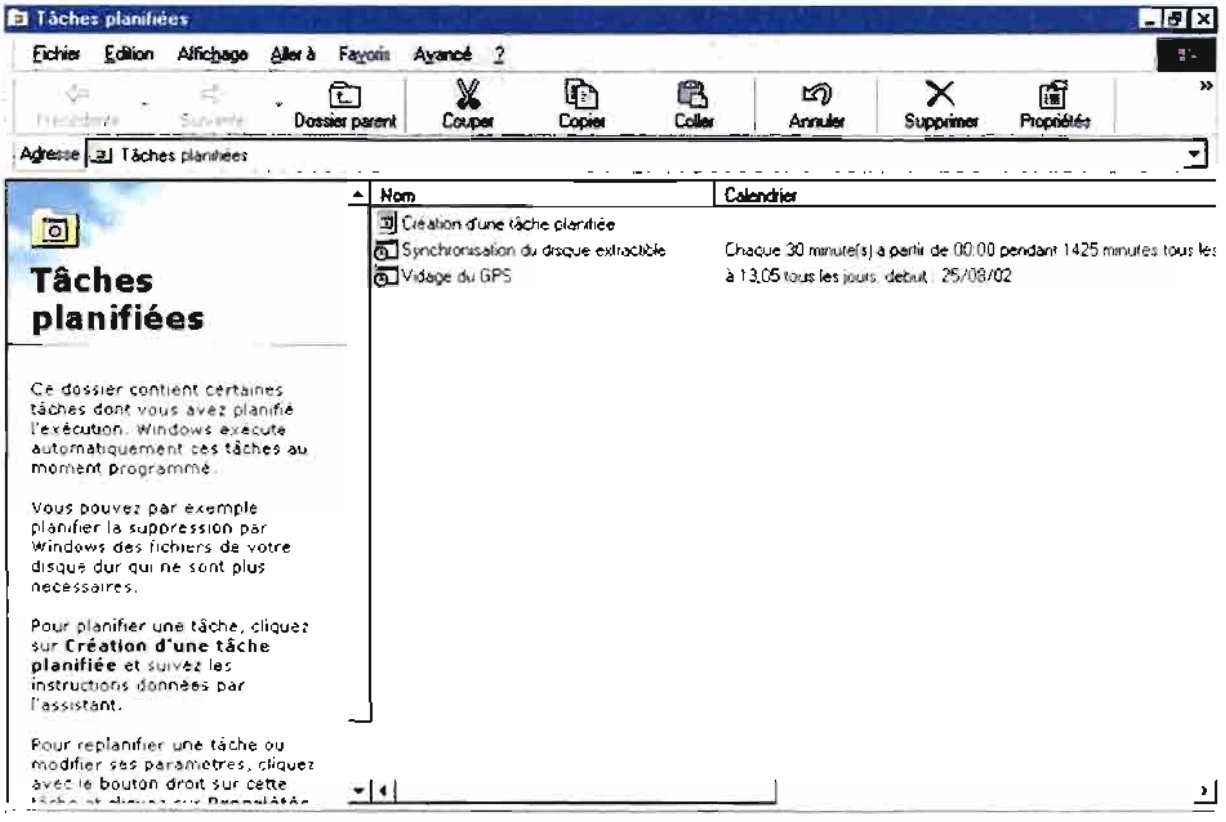


Fig . 5 : Les fichiers de commande gérés par le planificateur de tâches

II-4 : Schéma fonctionnel

L'ensemble des acquisitions sismologiques et géodésiques peut être décrit par le schéma fonctionnel suivant :

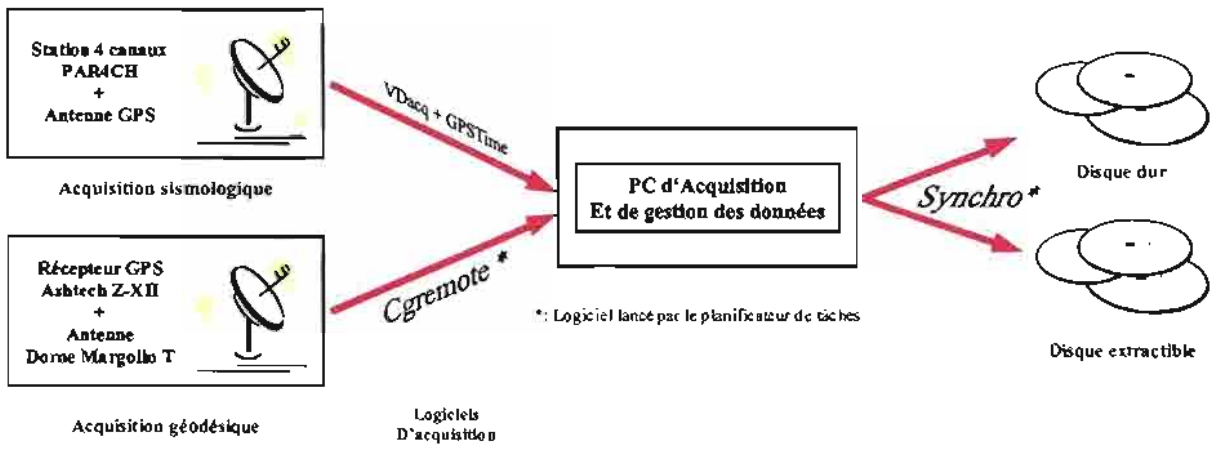


Fig . 6 : Schéma fonctionnel

II-5 : Interface opérateur

L'écran du PC se présente à l'opérateur comme suit :

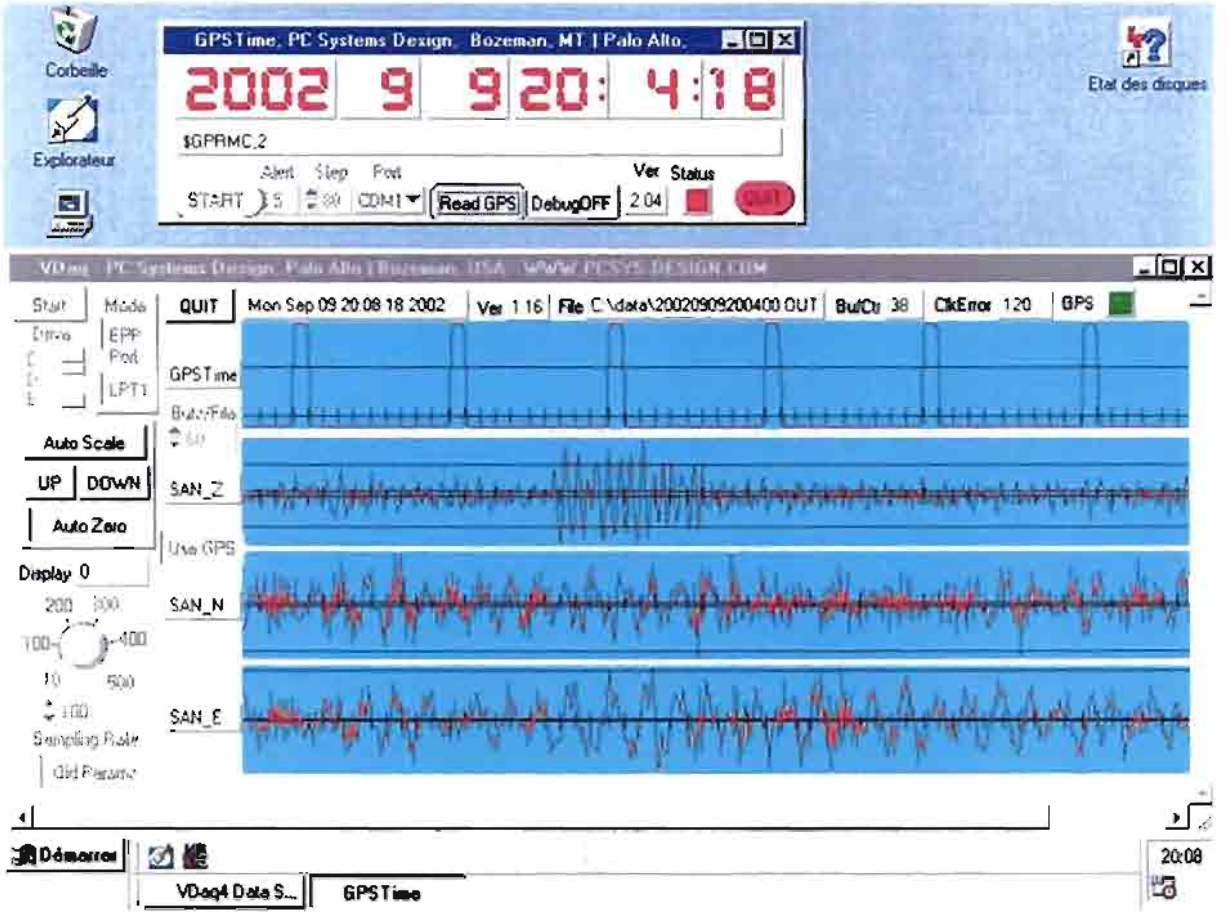
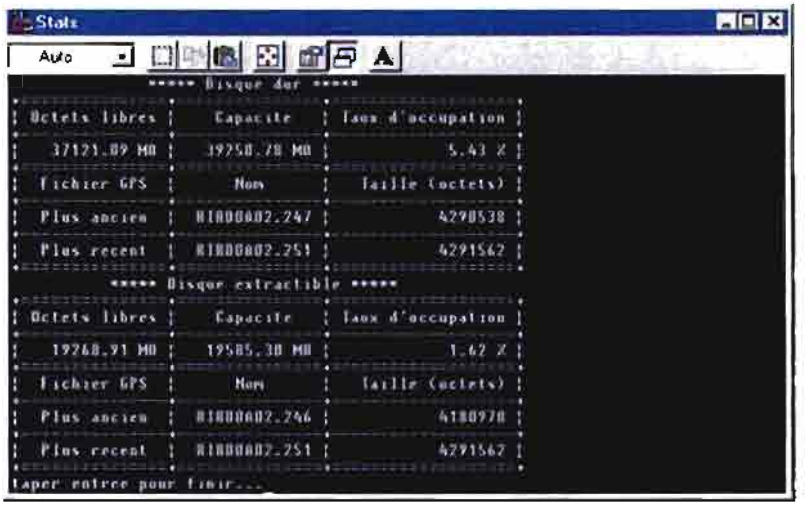


Fig . 7 : Aspect standard du PC de contrôle

Afin de limiter au maximum les fausses manœuvres de la part des opérateurs, le nombre des icônes présentes à l'écran a été réduit au maximum. Seule l'icône « Etat des disques » déclenche un petit utilitaires (« stats »), qui permet de connaître le taux de remplissage du disque dur et du disque extractible :



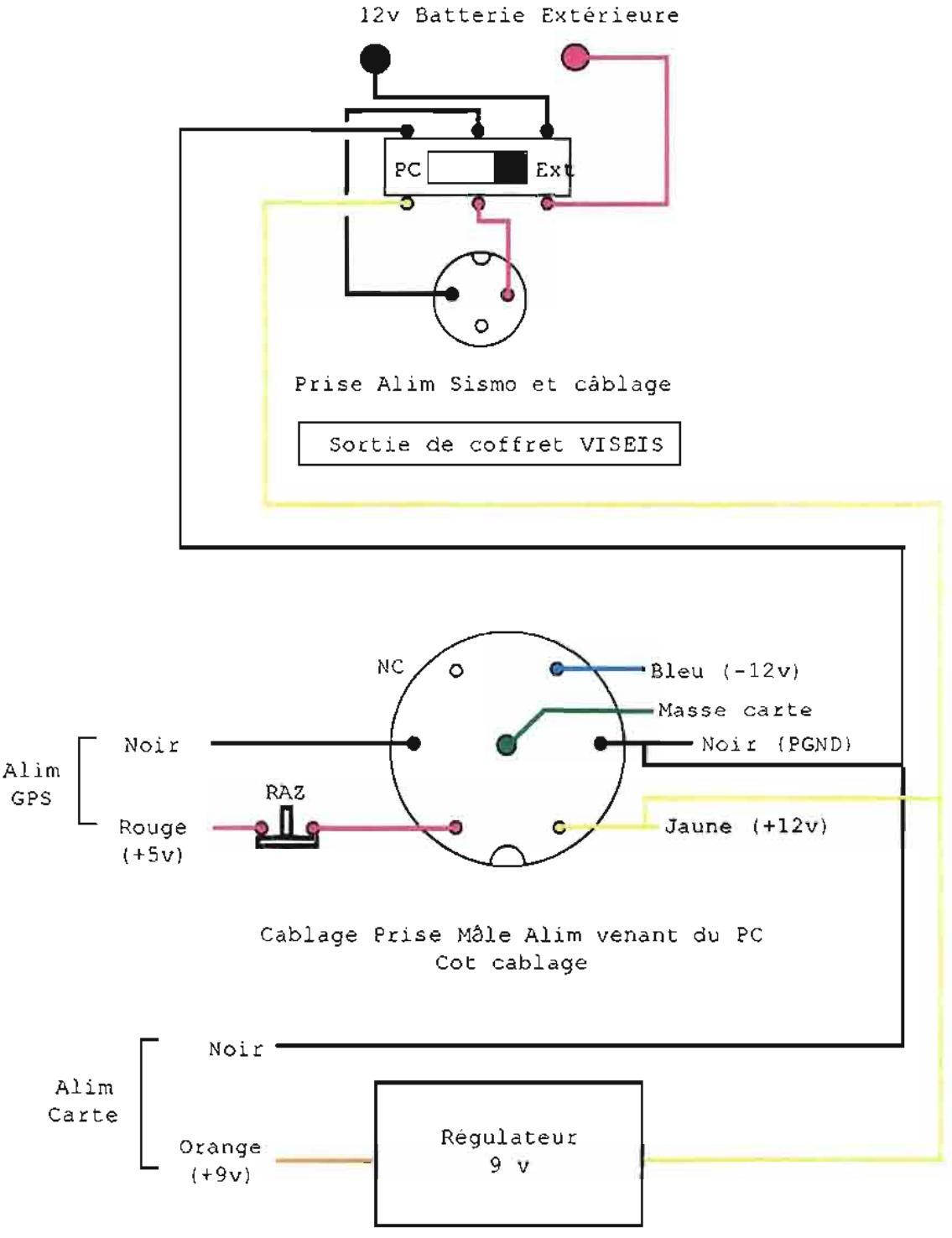
Ainsi lorsque le taux de remplissage du disque extractible devient trop élevé, l'opérateur peut effectuer le remplacement de celui-ci, et expédier les données dans un emballage conçu pour le transport du disque extractible.

ANNEXE 1

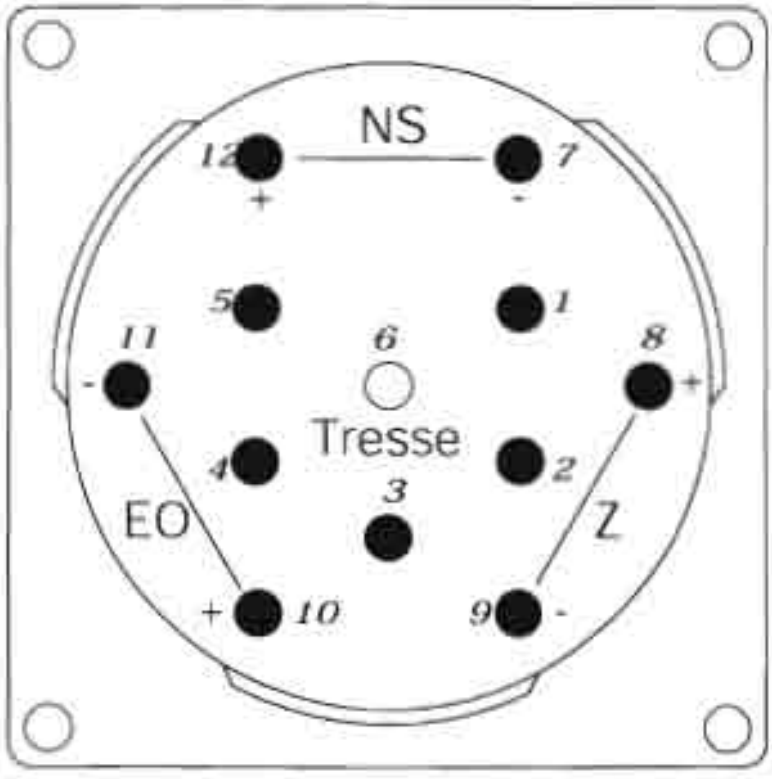
Schémas de câblage

Alimentation coffret 24 bits Santo
(Modifié alimentation sismo et
Carte alimentée en 9 volts)

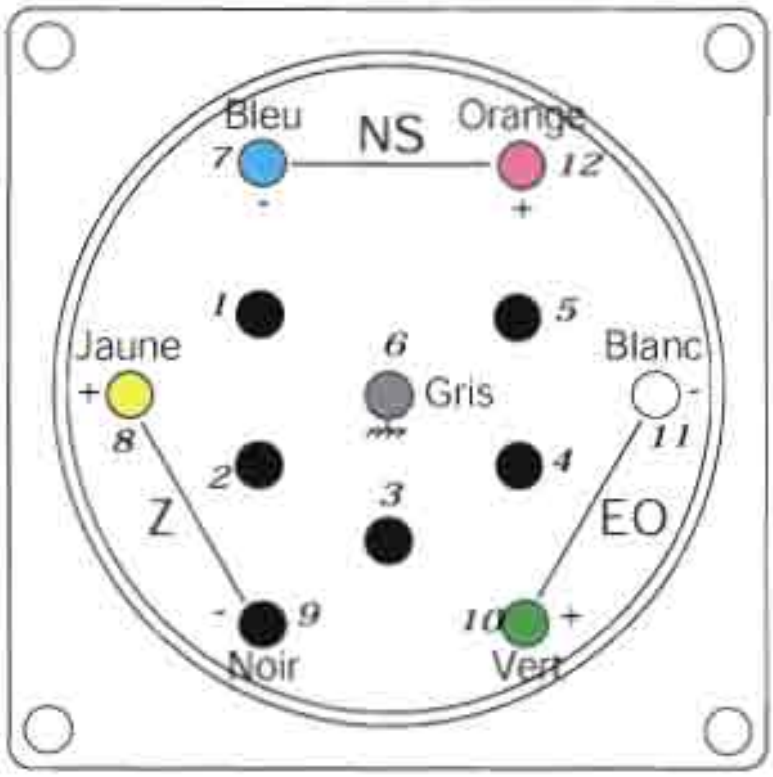
ALIM.PC → Entrée sur coffret VISEIS



BRANCHEMENT DU SISMO
Prise JAEGER

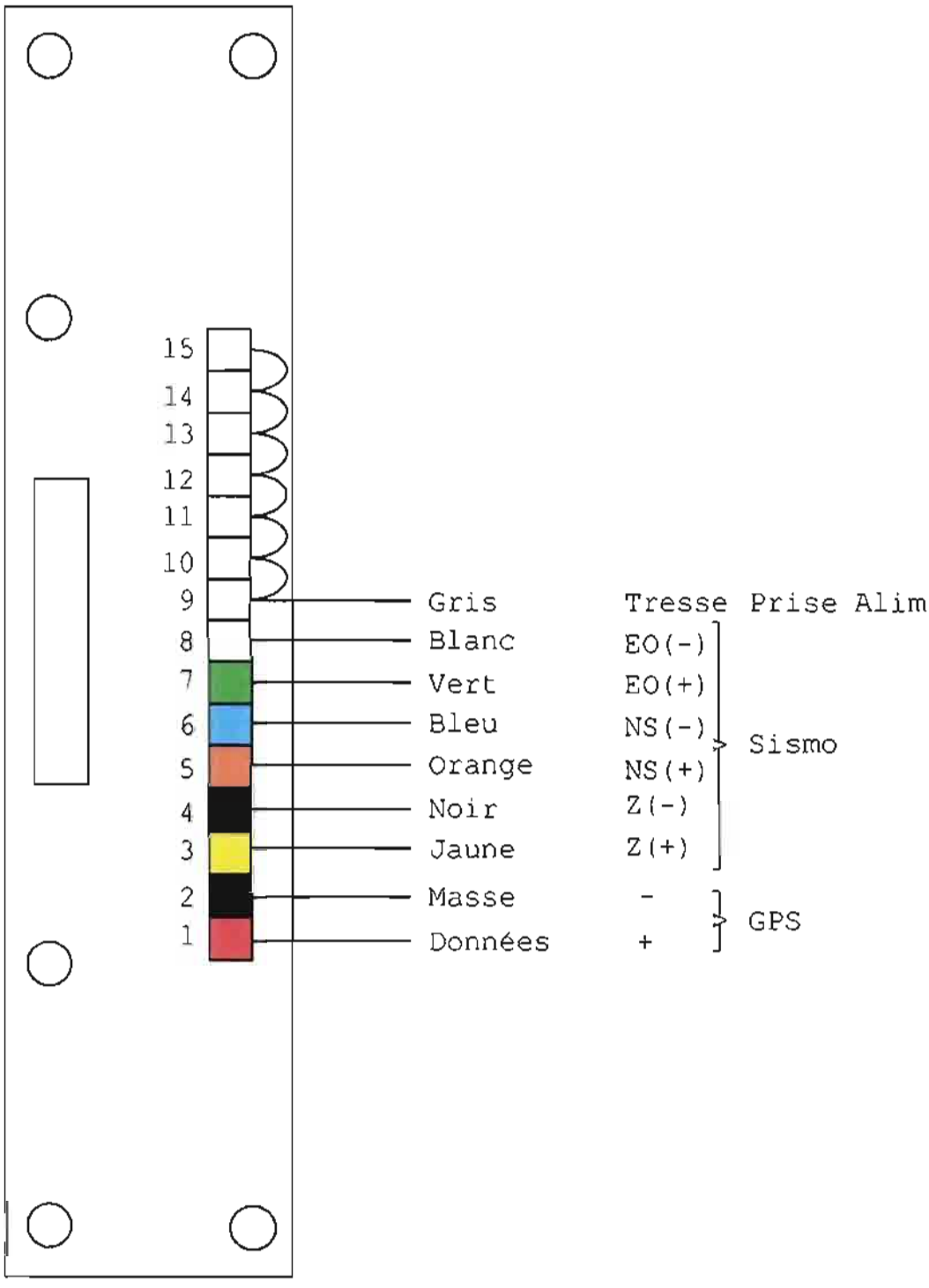


AVANT

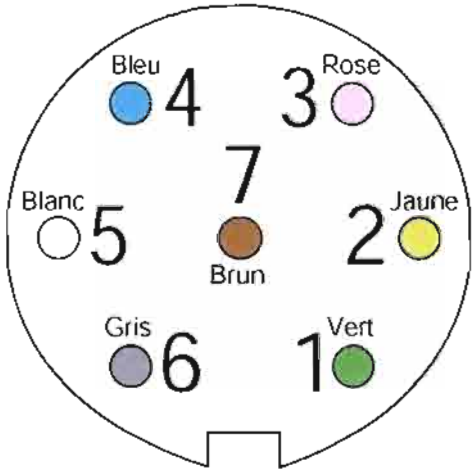


ARRIERE

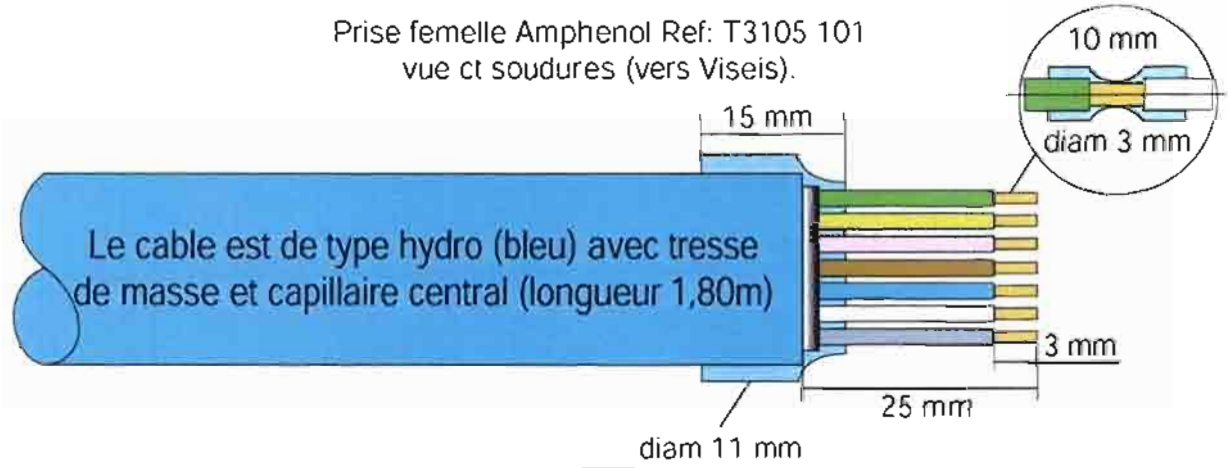
BRANCHEMENT
DU SISMO ET GPS
Carte 4 voies, 24 bits
(Santo)

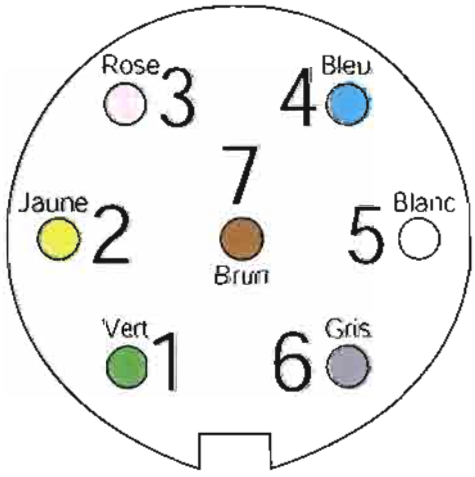


CABLE D'ALIMENTATION PC -> VISEIS



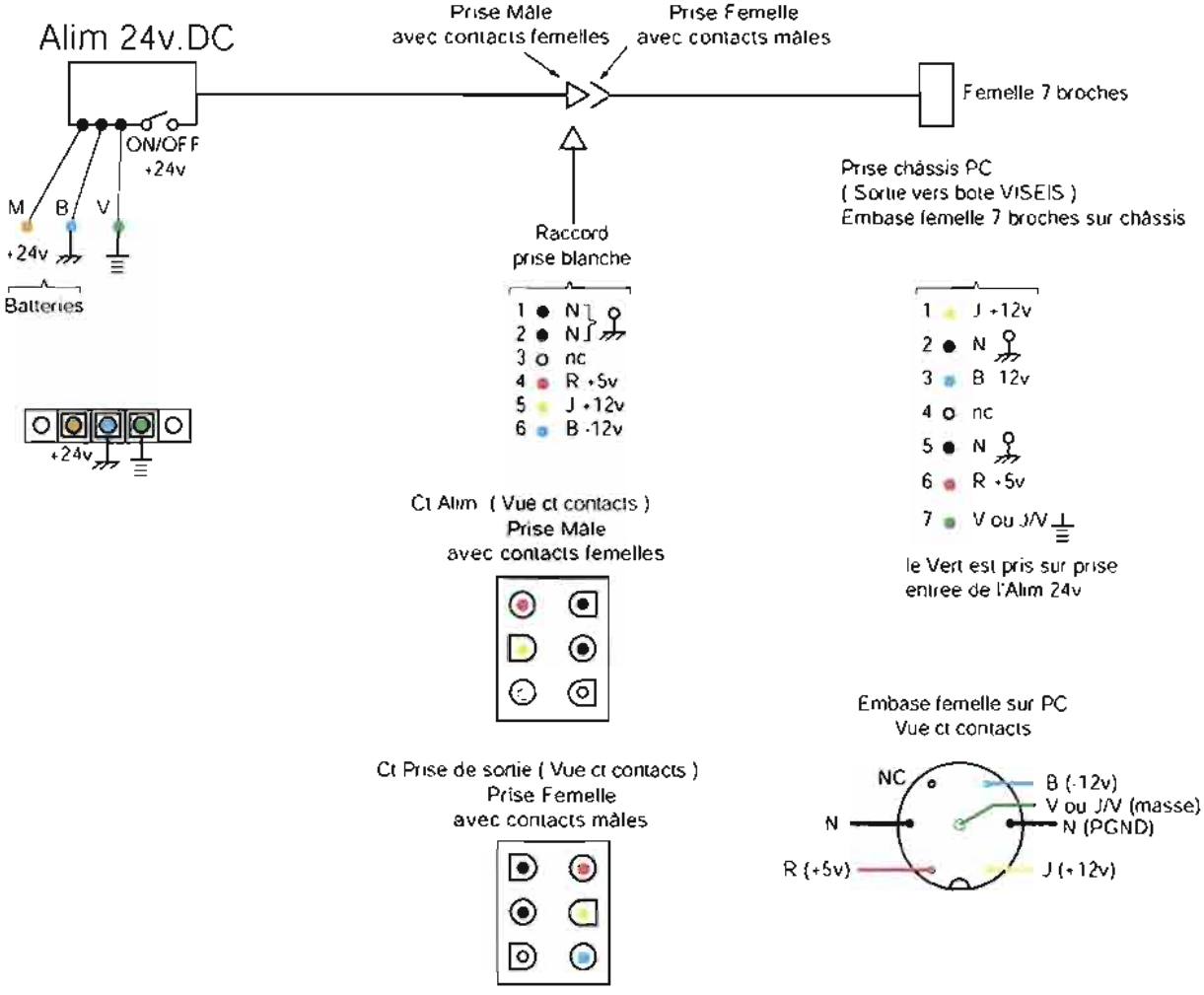
Prise femelle Amphenol Ref: T3105 101
vue ct soudures (vers Viseis).





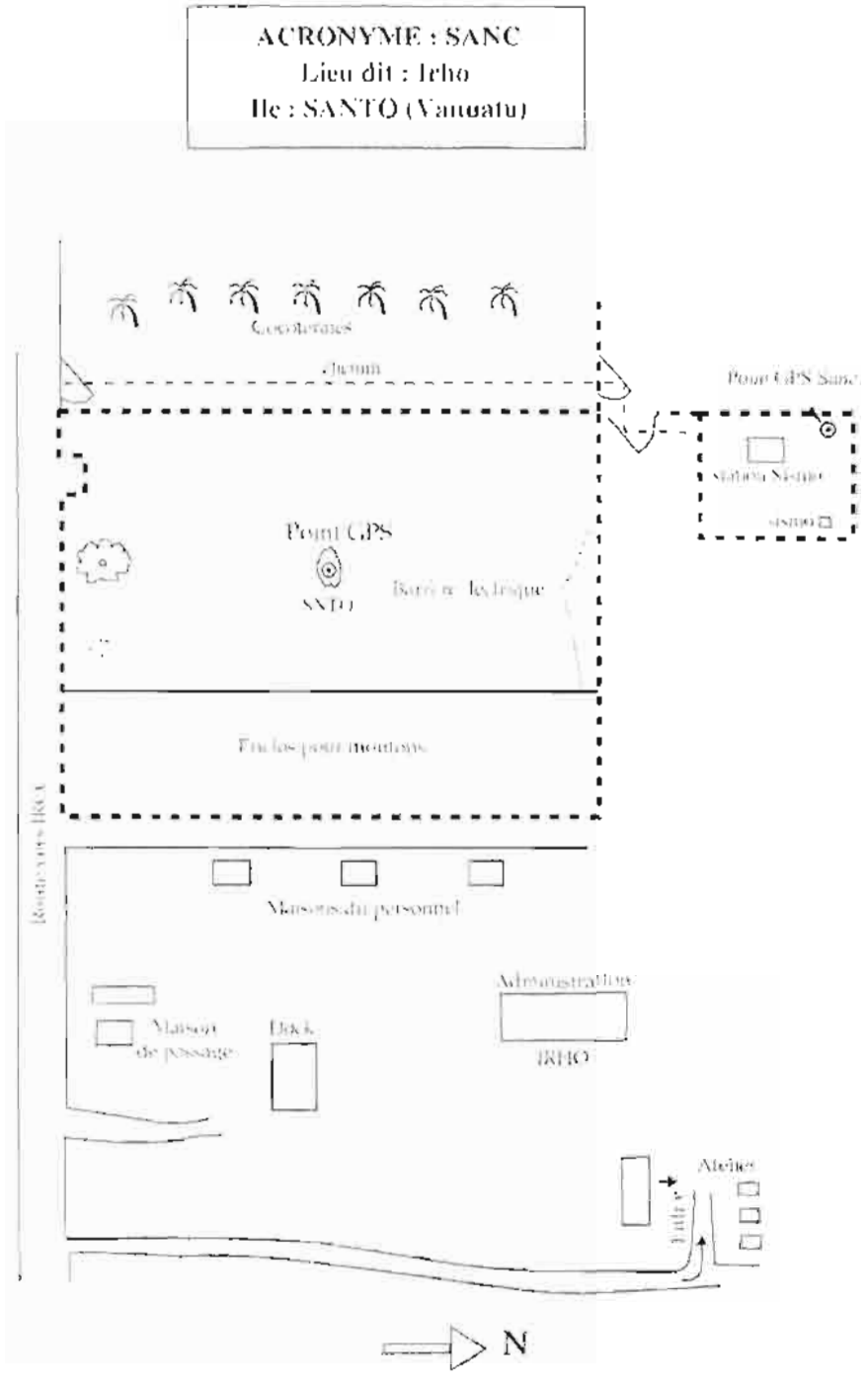
Prise mâle Amphenol Ref: T3104 101
vue ct soudures (vers PC).

CABLAGE DES ALIMENTATIONS



ANNEXE 2

Fiches descriptives du site GPS


ACRONYME : SANC
Lieu dit : Irho
Ile : SANTO (Vanuatu)
Cocoteraies
Chemin
Point GPS Sanc
Station Météo
Point GPS
SNTO
Barrière électrique
Enclos pour moutons
Route vers IRHO
Maisons du personnel
Administration
IRHO
Maison de passage
Dock
Ateliers
N

STATION PERMANENTE GPS
IRHO SANTO VANUATU

Latitude : 15° 26' S
Longitude : 167° 12' E
Hauteur : 110 m

Située à l'IRHO, Station Expérimentale des Oléagineux, à une distance de 10 km de Luganville. Prendre la route vers Port Olry.

Contact : M. David Nakedau, tél. : (678) 36830, ou l'IRHO, tél. : 36320, télécopie : 36560.

Avant d'aller sur le site de la station, prévenir l'administration IRHO de votre passage. La station GPS est à l'intérieur de la plantation derrière les logements, sur un plateau bien dégagé, dans un enclos à moutons. Bien refermer les portails à chaque passage. On peut s'y rendre en voiture « normale » par temps sec ; on peut loger à la maison de passage, demander à la secrétaire. Acheter sa nourriture à Luganville, pas de problème pour l'eau et l'électricité. Au campement du personnel, il y a un petit magasin.

ANNEXE 3

Code source C des utilitaires

```
#define     SECONDE     (1)
#define     MINUTE      (60*SECONDE)
#define     HEURE       (60*MINUTE)
#define     JOUR        (24*HEURE)
#define     SEMAINE     (7*JOUR)
#define     MOIS        (30*JOUR)
#define     ANNEE       (365*JOUR)

#define     DELAICOURT  (1*HEURE)
#define     DELAILONG   (1*ANNEE)

#define     MAXFICS     10000
#define     MAXLEN      1024

/*
 *    Répertoire d'origine par défaut
 */
#define     SOURCE_DIR  "C:\\DATA"
/*
 *    Répertoire destination par défaut sur le disque dur
 */
#define     DISQUEDUR   "C:\\SANTO"
/*
 *    Répertoire destination par défaut sur le disque extractible
 */
#define     EXTRACTIBLE "D:"
/*
 *    Répertoire temporaire
 */
#define     TEMP_DIR    "C:\\TMP"

#define     COMMANDE    "COMMANDE.BAT"
```

```
/*
 * Synchro.cpp
 *
 * Date: 17 juillet 2002
 *
 * Auteur: Pierre Lebellegard
 *
 * Ce programme vise à recopier le contenu d'un répertoire origine
vers un répertoire destination.
 * Ces répertoires sont censés contenir des fichiers de la forme
aaaammjjhhmmss.out, et on ne
 * tente de recopier que les fichiers plus récents qu'une certaine
date, afin de ne pas monopoliser
 * le processeur, pour ne pas perturber les programmes d'acquisition.
 */
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <direct.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <io.h>
#include "Constantes.h"

void comprime(char *fich)
{
	int		Status;
	FILE	*batch;
	char	radical[MAXLEN];
	char	ch[MAXLEN];
	char	orig[MAXLEN];
	char	*pt;

	printf("\t***** Compression de %s\n", fich); fflush(stdout);

	_getcwd(orig, MAXLEN);

	strcpy(radical, fich);
	pt = strchr(radical, '.');
	*pt = '\0';

	Status = chdir(TEMP_DIR);
	if (Status != 0)
	{
		fprintf(stderr, "Can't cd to temp dir: %s.\n",
TEMP_DIR);
		exit(1);
	}

	batch = fopen(COMMANDE, "w");
	if (batch == NULL)
	{
		fprintf(stderr, "Can't create batch file: %s.\n",
COMMANDE);
		exit(1);
	}

	fprintf(batch, "@ECHO OFF\n");
	fprintf(batch, "CD %s\n", TEMP_DIR);
	fprintf(batch, "DEL %s.ZIP\n", radical+8);
	sprintf(ch, "%s.TXT", radical+8);
	fprintf(batch, "DIR > %s\n", ch);
	fprintf(batch, "PKZIP -a %s %s\n", radical+8, ch);
	fprintf(batch, "PKZIP %s -d %s\n", radical+8, ch);
	fprintf(batch, "DEL %s\n", ch);
	fprintf(batch, "MOVE %s\\%s.ZIP %s\\%s.ZIP\n", TEMP_DIR,
radical+8, SOURCE_DIR, radical);
	fprintf(batch, "SLEEP 10\n");
	fprintf(batch, "WINZIP32 -a %s\\%s %s\\%s\n", SOURCE_DIR,
radical, SOURCE_DIR, fich);
	fprintf(batch, "SLEEP 10\n");
	fclose(batch);

	_flushall();

	if (system(COMMANDE) == -1)
	{
		fprintf(stderr, "Impossible d'executer: %s.\n",
COMMANDE);
		exit(1);
	}

	Status = chdir(orig);
}

/*
 * Comprimer les fichiers .OUT vieux de plus de 1/4 heure
 */
void init_compressions(void)
{
	struct _finddata_t	rep;
	long	h_rep;
	int		Status, i;
	int		NbFichiers = 0;
	char	ListeFich[MAXFICS][50];
	char	PlusRecent[50];
	char	ch[MAXLEN];
	char	orig[MAXLEN];
```

```
	time_t today, temp;

	_getcwd(orig, MAXLEN);

	Status = chdir(SOURCE_DIR);
	if (Status != 0)
	{
		fprintf(stderr, "Impossible d'acceder au repertoire du
disque dur: %s.\n", SOURCE_DIR);
	}

	/*
	 * Récupérer la date du jour
	 */
	time(&today);

	if ( (h_rep=_findfirst("*.OUT", &rep)) == -1L )
	{
		fprintf(stderr, "Il n'y a pas de fichier .out dans le
repertoire %s.\n", SOURCE_DIR);
		exit(1);
	}
	else
	{
		do
		{
			if (strlen(rep.name) < 4)
				continue;
			/*
			 * Ne pas comprimer les fichiers de moins de
1/4 heure
			 */
			temp = today-rep.time_create;
			if (temp > (15*MINUTE) )
			{
				strcpy(ListeFich[NbFichiers],
rep.name);
				if (++NbFichiers > MAXFICS)
				{
					fprintf(stderr,	"Noumbre	de
fichiers .OUT superieur au maximum (%d).\n", MAXFICS);
					exit(1);
				}
			}
		} while (_findnext(h_rep, &rep) == 0);
		_findclose(h_rep);
	}

	strcpy(PlusRecent, ListeFich[0]);
	for (i=1; i<NbFichiers; i++)
	{
		if (strcmp(PlusRecent, ListeFich[i]) < 0)
			strcpy(PlusRecent, ListeFich[i]);
	}

	for (i=0; i< NbFichiers; i++)
	{
		if (!strcmp(ListeFich[i], PlusRecent))
			continue;
		comprime(ListeFich[i]);

		_flushall();
		_chmod(ListeFich[i], _S_IWRITE );

		sprintf(ch,	"ATTRIB	-A	%s\\%s\n",	SOURCE_DIR,
ListeFich[i]);
		system(ch);

		_flushall();
		Status = _unlink(ListeFich[i]);
		if( Status == -1 )
		{
			fprintf(stderr, "Impossible de supprimer %s\n.",
ListeFich[i]);
			fflush(stderr);
		}
	}
	Status = chdir(orig);
}

int main(int argc, char* argv[])
{
	char	orig[MAXLEN];
	char	disquedur[MAXLEN];
	char	extractible[MAXLEN];
	char	ch[MAXLEN];
	int		Status;
	struct _finddata_t	rep;
	long	h_rep;
	int		annee, mois, jour;
	char	annees[20], moiss[20], jours[20];
	char	heures[20], minutes[20], secondes[20];

	char *MoisASCII[13]=
	{
		"",
		"janvier",
		"fevrier",
		"mars",
		"avril",
```

```
                "mai",
                "juin",
                "juillet",
                "aout",
                "septembre",
                "octobre",
                "novembre",
                "decembre"
        };

        if (argc != 4)
        {
                fprintf(stderr, "%s: Usage: %s <repertoire origine>
repertoire destination>\n",
                        argv[0], argv[0]);

                strcpy(orig, SOURCE_DIR);
                strcpy(disquedur, DISQUEDUR);
                strcpy(extractible, EXTRACTIBLE);

                fprintf(stderr, "Les valeurs par defaut suivantes
seront prises:\n\n");
                fprintf(stderr,
"Origine ..... : %s\n", orig);
                fprintf(stderr,         "Destination          (disque
dur) ..... : %s\n", disquedur);
                fprintf(stderr,         "Destination          (disque
extractible) ... : %s\n\n", extractible);
        }
        else
        {
                strcpy(orig, argv[1]);
                strcpy(disquedur, argv[2]);
                strcpy(extractible, argv[3]);

                fprintf(stderr,
"Origine ..... : %s\n", orig);
                fprintf(stderr,         "Destination          (disque
dur) ..... : %s\n", disquedur);
                fprintf(stderr,         "Destination          (disque
extractible) ... : %s\n\n", extractible);
        }

        init_compressions();

        Status = chdir(disquedur);
        if (Status != 0)
        {
                Status = _mkdir(disquedur);
                if (Status == 0)
                {
                        Status = chdir(disquedur);
                        if (Status != 0)
                        {
                                fprintf(stderr,  "Impossible  d'acceder
au repertoire du disque dur: %s.\n", disquedur);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr,  "Impossible  de  creer  le
repertoire du disque dur: %s.\n", disquedur);
                        exit(1);
                }
        }

        Status = chdir(extractible);
        if (Status != 0)
        {
                Status = _mkdir(extractible);
                if (Status == 0)
                {
                        Status = chdir(extractible);
                        if (Status != 0)
                        {
                                fprintf(stderr,  "Impossible  d'acceder
au repertoire du disque dur: %s.\n", extractible);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr,  "Impossible  de  creer  le
repertoire du disque dur: %s.\n", extractible);
                        exit(1);
                }
        }

        struct tm               *ptoday;
        time_t                  today;
        struct tm               phier;
        time_t                  hier;
        int                             aaa, msa, jja, hha, mma, ssa;
        struct _stat    buf_orig, buf_dest;
        int                             aac, msc, jjc, hhc, mmc, ssc;
        /*
         * Récupérer la date du jour
         */
        time(&today);
```

```
	ptoday =	gmtime(&today);
	aaa =		ptoday->tm_year + 1900;
	msa =		ptoday->tm_mon + 1;
	jja =		ptoday->tm_mday;
	hha =		ptoday->tm_hour;
	mma =		ptoday->tm_min;
	ssa =		ptoday->tm_sec;

	Status = chdir(orig);
	if (Status != 0)
	{
		fprintf(stderr, "Impossible d'acceder au repertoire
origine: %s.\n", orig);
		exit(1);
	}

	if ( (h_rep=_findfirst("*.ZIP", &rep)) == -1L )
	{
		fprintf(stderr, "Il n'y a pas de fichier .zip dans le
repertoire %s.\n", orig);
		exit(1);
	}
	else
	{
		/*
		 * Recopier les fichiers présents à la fois sur le
disque dur, et sur le disque extractible,
		 * s'ils ne s'y trouvent pas déjà.
		 *
		 * Durée de vie des fichiers:
		 *
		 *  - Dans le répertoire origine: une semaine
(DELAICOURT);
		 *  - Dans le répertoire disque dur: une année
(DELAILONG);
		 * - Sur le disque extractible: les fichiers sont
conservés quelle que soit leur date de création.
		 *
		 * Dans les deux destinations, il y a un répertoire
par année, et un sous-répertoire par mois.
		 */
		do
		{
			strncpy(ch, rep.name, 4); ch[4] = '\0'; annee =
atoi(ch);
			sprintf(annees, "%04d", annee);

			strncpy(ch, rep.name+4, 2); ch[2] = '\0'; mois
= atoi(ch);
```



```
			sprintf(moiss,	"%s	%s",	MoisASCII[mois],
annees);

			strncpy(ch, rep.name+6, 2); ch[2] = '\0'; jour
= atoi(ch);
			sprintf(jours, "%02d %s", jour, moiss);

			/*
			 * Copie sur le disque dur
			 */
			Status = chdir(disquedur);
			if (Status != 0)
			{
				Status = _mkdir(disquedur);
				if (Status == 0)
				{
					Status = chdir(disquedur);
					if (Status != 0)
					{
						fprintf(stderr,
"Impossible	d'acceder	au	repertoire	du	disque	dur:	%s.\n",
disquedur);
						exit(1);
					}
				}
				else
				{
					fprintf(stderr,	"Impossible	de
creer le repertoire du disque dur: %s.\n", disquedur);
					exit(1);
				}
			}

			Status = chdir(annees);
			if (Status != 0)
			{
				Status = _mkdir(annees);
				if (Status == 0)
				{
					Status = chdir(annees);
					if (Status != 0)
					{
						fprintf(stderr,
"Impossible d'acceder au repertoire du disque dur: %s.\n", annees);
						exit(1);
					}
				}
				else
				{
```

```
                                fprintf(stderr,  "Impossible  de
creer le repertoire du disque dur: %s.\n", annees);
                                exit(1);
                        }
                }

                Status = chdir(moiss);
                if (Status != 0)
                {
                        Status = _mkdir(moiss);
                        if (Status == 0)
                        {
                                Status = chdir(moiss);
                                if (Status != 0)
                                {
                                        fprintf(stderr,
"Impossible d'acceder au repertoire du disque dur: %s.\n", moiss);
                                        exit(1);
                                }
                        }
                        else
                        {
                                fprintf(stderr,  "Impossible  de
creer le repertoire du disque dur: %s.\n", moiss);
                                exit(1);
                        }
                }

                Status = chdir(jours);
                if (Status != 0)
                {
                        Status = _mkdir(jours);
                        if (Status == 0)
                        {
                                Status = chdir(jours);
                                if (Status != 0)
                                {
                                        fprintf(stderr,
"Impossible d'acceder au repertoire du disque dur: %s.\n", jours);
                                        exit(1);
                                }
                        }
                        else
                        {
                                fprintf(stderr,  "Impossible  de
creer le repertoire du disque dur: %s.\n", jours);
                                exit(1);
                        }
                }
```



```
                        printf("%s\r", rep.name); fflush(stdout);
                        /*
                         * Si le fichier en cours n'est pas présent, il
faut le recopier dans le répertoire destination
                         */
                        sprintf(ch, "%s", rep.name);
                        Status = _access(ch, 0 );
                        if( Status == -1 )
                        {
                                sprintf(ch, "copy %s\\%s /Y> %s\\NULL",
                                        orig, rep.name,       TEMP_DIR);
                               printf("copy %s\\%s /Y> %s\\NULL", orig,
rep.name,      TEMP_DIR);
                                system(ch);
                        }
                        /*
                         * S'il est présent, il faut qu'il ait la même
taille. On compare donc les tailles du fichier
                         * origine et du fichier destination.
                         */
                        else
                        {
                                sprintf(ch, "%s\\%s", orig, rep.name);
                                Status = _stat(ch, &buf_orig);
                                if (Status != 0)
                                {
                                        fprintf(stderr,  "Impossible  de
lire les infos du fichier origine %s.\n", rep.name);
                                        exit(1);
                                }
                                sprintf(ch,        "%s\\%s\\%s\\%s\\%s",
disquedur, annees, moiss, jours, rep.name);
                                Status = _stat(ch, &buf_dest);
                                if (Status != 0)
                                {
                                        fprintf(stderr,  "Impossible  de
lire les infos du fichier destination %s\\%s\\%s\\%s\\%s.\n",
                                                disquedur, annees, moiss,
jours, rep.name);
                                        exit(1);
                                }
                                if          (buf_dest.st_size          <
buf_orig.st_size)
                                {
                                        sprintf(ch,     "copy     %s\\%s
/Y> %s\\NULL",
                                                orig,           rep.name,
        TEMP_DIR);
                                        printf("copy              %s\\%s
/Y> %s\\NULL", orig, rep.name,       TEMP_DIR);
```

```
                        system(ch);
                }
        }

        /*
         * Copie sur le disque extractible
         */
        Status = chdir(extractible);
        if (Status != 0)
        {
                Status = _mkdir(extractible);
                if (Status == 0)
                {
                        Status = chdir(extractible);
                        if (Status != 0)
                        {
                                fprintf(stderr,
"Impossible d'acceder au repertoire du disque extractible: %s.\n",
extractible);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr, "Impossible de
creer le repertoire du disque extractible: %s.\n", extractible);
                        exit(1);
                }
        }

        chdir("\\");

        Status = chdir(annees);
        if (Status != 0)
        {
                Status = _mkdir(annees);
                if (Status == 0)
                {
                        Status = chdir(annees);
                        if (Status != 0)
                        {
                                fprintf(stderr,
"Impossible d'acceder au repertoire du disque extractible: %s.\n",
annees);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr, "Impossible de
creer le repertoire du disque extractible: %s.\n", annees);
                        exit(1);
                }
        }

        Status = chdir(moiss);
        if (Status != 0)
        {
                Status = _mkdir(moiss);
                if (Status == 0)
                {
                        Status = chdir(moiss);
                        if (Status != 0)
                        {
                                fprintf(stderr,
"Impossible d'acceder au repertoire du disque extractible: %s.\n",
moiss);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr, "Impossible de
creer le repertoire du disque extractible: %s.\n", moiss);
                        exit(1);
                }
        }

        Status = chdir(jours);
        if (Status != 0)
        {
                Status = _mkdir(jours);
                if (Status == 0)
                {
                        Status = chdir(jours);
                        if (Status != 0)
                        {
                                fprintf(stderr,
"Impossible d'acceder au repertoire du disque extractible: %s.\n",
jours);
                                exit(1);
                        }
                }
                else
                {
                        fprintf(stderr, "Impossible de
creer le repertoire du disque extractible: %s.\n", jours);
                        exit(1);
                }
```

```
                        }

                        printf("%s\r", rep.name); fflush(stdout);
                        /*
                         * Si le fichier en cours n'est pas présent, il
faut le recopier dans le répertoire destination
                         */
                        sprintf(ch, "%s", rep.name);
                        Status = _access(ch, 0 );
                        if( Status == -1 )
                        {
                                sprintf(ch, "copy %s\\%s /Y> %s\\NULL",
                                        orig, rep.name,         TEMP_DIR);
                                system(ch);
                        }
                        /*
                         * S'il est présent, il faut qu'il ait la même
taille. On compare donc les tailles du fichier
                         * origine et du fichier destination.
                         */
                        else
                        {
                                sprintf(ch, "%s\\%s", orig, rep.name);
                                Status = _stat(ch, &buf_orig);
                                if (Status != 0)
                                {
                                        fprintf(stderr,  "Impossible  de
lire les infos du fichier origine %s.\n", rep.name);
                                        exit(1);
                                }
                                sprintf(ch,         "%s\\%s\\%s\\%s\\%s",
extractible, annees, moiss, jours, rep.name);
                                Status = _stat(ch, &buf_dest);
                                if (Status != 0)
                                {
                                        fprintf(stderr,  "Impossible  de
lire les infos du fichier destination %s\\%s\\%s\\%s\\%s.\n",
                                                extractible,      annees,
moiss, jours, rep.name);
                                        exit(1);
                                }
                                if          (buf_dest.st_size          <
buf_orig.st_size)
                                {
                                        sprintf(ch,      "copy      %s\\%s
/Y> %s\\NULL",
                                                orig,           rep.name,
        TEMP_DIR);
                                        system(ch);
                                }
                        }
```



```
                }

                /*
                 * Si le fichier en cours d'examen est plus
ancien que le "délai court" (douze heures),
                 * alors le supprimer. Pour faire le test, on
transforme la date du fichier en nombre
                 * de secondes depuis le 01/01/1970
                 */
                strncpy(annees,                 rep.name, 4);
        annees[4] = '\0';       aac = atoi(annees);
                strncpy(moiss,          rep.name+4, 2);
        moiss[2] = '\0';        msc = atoi(moiss);
                strncpy(jours,          rep.name+6, 2);
        jours[2] = '\0';        jjc = atoi(jours);
                strncpy(heures,                 rep.name+8,     2);
                heures[2] = '\0';       hhc = atoi(heures);
                strncpy(minutes,        rep.name+10,            2);
        minutes[2] = '\0';      mmc = atoi(minutes);
                strncpy(secondes,       rep.name+12,            2);
        secondes[2] = '\0';     ssc = atoi(secondes);

                phier.tm_year =         aac - 1900;
                phier.tm_mon = msc - 1;
                phier.tm_mday =         jjc;
                phier.tm_hour =         hhc;
                phier.tm_min = mmc;
                phier.tm_sec = ssc;

                hier = mktime(&phier);
                if (hier<today-DELAICOURT)
                {
                        /*
                         * On supprime
                         */
                        Status = chdir(orig);
                        if (Status != 0)
                        {
                                fprintf(stderr,       "Impossible
d'acceder au repertoire origine: %s.\n", orig);
                                exit(1);
                        }
                        if (_unlink(rep.name))
                                fprintf(stderr,  "Impossible  de
supprimer %s.\n", rep.name);
                }
        } while (_findnext(h_rep, &rep) == 0);
}

/*
```

```
	* On va maintenant examiner le répertoires sur le disque dur, et virer (del récursif) ceux plus
	* anciens que "delailong" (un an en principe). La suppression est faite sur le nom du fichier,
	* et non pas en fonction du contenu de ces répertoires. Ceci pour ne pas avoir à balayer une
	* quantité invraisemblable de fichiers (même s'il y a des chances pour que ce soit fait de toute
	* façon par ce del récursif. Exemple: Date du jour = 14 juillet 2002. Date limite = 14 juillet 2001.
	*
	* On va donc supprimer:
	*
	* - tous les répertoires précédant 2001, par ex. del /q /s 2000, s'il existe;
	*
	* - dans le répertoire 2001: suppression des répertoires janvier 2001, etc. jusqu'à juin 2001;
	*
	* - enfin dans le répertoire 2001\juillet 2001, tous les répertoires quotidiens jusqu'au
	*   13 juillet.
	*
	*/

	struct _finddata_t    year;
	long   h_year;
	struct tm             *plimite;
	int                           aah, msh, jjh, hhh, mmh, ssh;
	/*
	 * Récupérer la date du jour
	 */
	time(&today);
	ptoday =       gmtime(&today);
	aaa =          ptoday->tm_year + 1900;
	msa =          ptoday->tm_mon + 1;
	jja =          ptoday->tm_mday;
	hha =          ptoday->tm_hour;
	mma =          ptoday->tm_min;
	ssa =          ptoday->tm_sec;

	/*
	 * Calculer la date limite
	 */
	hier =         today - DELAILONG;
	plimite =      gmtime(&hier);
	aah =          plimite->tm_year + 1900;
	msh =          plimite->tm_mon + 1;
	jjh =          plimite->tm_mday;
	hhh =          plimite->tm_hour;
	mmh =          plimite->tm_min;
	ssh =          plimite->tm_sec;

	Status = chdir(disquedur);
	if (Status != 0)
	{
		fprintf(stderr, "Impossible d'acceder au repertoire disque dur: %s.\n", disquedur);
	}
	if ( (h_year=_findfirst("????", &year)) == -1L )
	{
		exit(1);
	}
	do
	{
		if (strlen(year.name)!=4)
			continue;

		aac = atoi(year.name);
		if (aac == 0)
			continue;

		/*
		 * L'année est supérieure à l'année de la date limite: ne rien faire
		 */
		if (aac > aah)
			continue;

		/*
		 * L'année est inférieure à l'année de la date limite: la supprimer dans son intégralité
		 */
		if (aac < aah)
		{
			/*
			sprintf(ch, "RMDIR \"%04d\" /S /Q", aac);
			*/
			sprintf(ch, "DELTREE /Y \"%04d\"", aac);
			system(ch);
			continue;
		}

		/*
		 * L'année est égale à l'année de la date limite: il faut examiner les sous-répertoires
		 */
		struct _finddata_t    month;
		long   h_month;
```

```
		Status = chdir(year.name);
		if (Status != 0)
		{
			fprintf(stderr,	"Impossible	d'acceder	au
repertoire: %s.\n", year.name);
		}

		if ( (h_month=_findfirst("*", &month)) == -1L )
			continue;
		do
		{
			if (!strcmp(month.name, "."))
				continue;
			if (!strcmp(month.name, ".."))
				continue;
			/*
			* Comme les répertoires sont nommés "en clair",
on compare un par un
			*/
			sprintf(ch, "%s", month.name);
			char *pt = strchr(ch, ' ');
			*pt = '\0';
			/*
			* A cet endroit, ch ne contient plus que le
mois
			*/
			for (msc=0; msc<13; msc++)
			{
				if (!strcmp(MoisASCII[msc], ch))
					break;
			}
			if (msc==0)
				continue;

			if (msc==13)
				continue;

			/*
			* msc contient l'indice du mois trouvé (7 pour
juillet, p. ex.)
			*/
			/*
			* Le mois est supérieur au mois de la date
limite: ne rien faire
			*/
			if (msc > msh)
				continue;

			/*
			* Le mois est inférieur au mois de la date
limite: le supprimer dans son intégralité
			*/
			if (msc < msh)
			{
				fprintf(stderr, "Suppression de %s\n",
month.name);
				/*
				sprintf(ch,	"RMDIR	\"%s\"	/S	/Q",
month.name);
				*/
				sprintf(ch,	"DELTREE	/Y	\"%s\"",
month.name);
				system(ch);
				continue;
			}

			/*
			* Le mois est égal au mois de la date limite:
il faut examiner les sous-répertoires
			*/
			struct _finddata_t	day;
			long	h_day;

			Status = chdir(month.name);
			if (Status != 0)
			{
				fprintf(stderr,	"Impossible	d'acceder
au repertoire: %s.\n", month.name);
			}

			if ( (h_day=_findfirst("*", &day)) == -1L )
				continue;
			do
			{
				if (!strcmp(day.name, "."))
					continue;
				if (!strcmp(day.name, ".."))
					continue;
				/*
				* Comme les répertoires sont nommés
"en clair", on compare un par un
				*/
				sprintf(ch, "%s", day.name);
				char *pt = strchr(ch, ' ');
				*pt = '\0';
				jjc = atoi(ch);

				/*
```

```
                                * Le jour est supérieur au jour de la
date limite: ne rien faire
                                */
                               if (jjc > jjh)
                                      continue;

                               /*
                                * Le jour est inférieur au jour de la
date limite: le supprimer dans son intégralité
                                */
                               if (jjc < jjh)
                               {
                                      fprintf(stderr,     "Suppression
de %s\r", day.name);
                                      /*
                                     sprintf(ch, "RMDIR \"%s\" /S /Q",
day.name);
                                      */
                                      sprintf(ch, "DELTREE /Y \"%s\"",
day.name);
                                      system(ch);
                                      continue;
                               }
                        } while (_findnext(h_day, &day) == 0);
                        Status = chdir("..");
                        if (Status != 0)
                        {
                               fprintf(stderr,  "Impossible  d'acceder
au repertoire: %s.\n", "..");
                               exit(1);
                        }
                } while (_findnext(h_month, &month) == 0);
                Status = chdir("..");
                if (Status != 0)
                {
                        fprintf(stderr,   "Impossible   d'acceder   au
repertoire: %s.\n", "..");
                        exit(1);
                }
        } while (_findnext(h_year, &year) == 0);
        return 0;
}
```

```
#define SECONDE         (1)
#define MINUTE          (60*SECONDE)
#define HEURE                   (60*MINUTE)
#define JOUR                    (24*HEURE)
#define SEMAINE         (7*JOUR)
#define MOIS                    (30*JOUR)
#define ANNEE                   (365*JOUR)

#define DELAI                   (1*ANNEE)

#define MAXFICS         10000
#define MAXLEN          1024

/*
 *      Répertoire d'origine par défaut
 */
#define SOURCE_DIR              "C:\\GPS"
/*
 *      Répertoire temporaire
 */
#define TEMP_DIR                "C:\\TMP"
```

```
// nettoyerGPS.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <direct.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
#include <time.h>

#include "Constantes.h"

int main(int argc, char* argv[])
{
	struct _finddata_t    rep;
	long   h_rep;
	int            Status;
	char   ch[MAXLEN];

	time_t today, date_modif;

	/*
	 * Récupérer la date du jour
	 */
	time(&today);

	Status = chdir(SOURCE_DIR);
	if (Status != 0)
	{
		fprintf(stderr, "Impossible d'acceder au repertoire du disque dur: %s.\n", SOURCE_DIR);
	}

	if ( (h_rep=_findfirst("*.*", &rep)) == -1L )
	{
		fprintf(stderr, "Il n'y a pas de fichier dans le repertoire %s.\n", SOURCE_DIR);
		exit(1);
	}
	else
	{
		do
		{
			if (strlen(rep.name) < 4)
				continue;

			date_modif = rep.time_create;
			/*
```

```
                        * Le fichier est trop recent pour etre supprime
                        */
                       if ( (today-date_modif) < DELAI)
                              continue;

                       _flushall();
                       _chmod(rep.name, _S_IWRITE );

                       sprintf(ch, "ATTRIB -A %s\\%s\n", SOURCE_DIR, rep.name);
                       system(ch);

                       _flushall();
                       Status = _unlink(rep.name);
                       if( Status == -1 )
                       {
                              fprintf(stderr, "Impossible de supprimer %s\n.", rep.name);
                              fflush(stderr);
                       }

               } while (_findnext(h_rep, &rep) == 0);

               _findclose(h_rep);
       }

       return 0;
}
```

```
#define MAXLEN 1024

#define GPS_DIR "\\GPS"

#define TEMP_DIR        "C:\\TMP"
#define TEMP_FILE       "TAGADA.TXT"
```

```
// Stats.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>
#include <io.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include "Constantes.h"

int main(int argc, char* argv[])
{
	char	buf[MAXLEN];
	char	orig[MAXLEN];
	char	ch[MAXLEN];

	char	*pt, *pt1;

	FILE	*fic;

	int		i;

	int		dd_total, dd_libres;
	double	mod_total, mod_libres;
	int		de_total, de_libres;
	double	moe_total, moe_libres;
	double	taux_dd, taux_de;

	time_t	today;

	/*
	 * Récupérer la date du jour
	 */
	time(&today);

	struct fichier
	{
		char	nom[MAXLEN];
		int		taille;
		int		datemodif;
	};

	struct fichier prdGPS, pvdGPS;
	struct fichier preGPS, pveGPS;

	struct _finddata_t	rep;
	long	h_rep;
```



```
prdGPS.datemodif = 0; pvdGPS.datemodif = today;
preGPS.datemodif = 0; pveGPS.datemodif = today;

if( _getcwd(orig, MAXLEN ) == NULL )
	perror( "_getcwd error" );

/*
 * Stats disque dur.
 */
chdir(TEMP_DIR);
sprintf(ch, "CHKDSK C: > %s", TEMP_FILE);
system(ch);
fic = fopen(TEMP_FILE, "r");
if (fic == NULL)
{
	fprintf(stderr, "Impossible d'obtenir les stats.\n");
	exit(1);
}

for (;;)
{
	if (fgets(buf, MAXLEN, fic) == NULL)
		break;

	pt = strstr(buf, "kilo-octets");
	if ( pt != NULL)
	{
		if (strstr(buf, "total") != NULL)
		{
			--pt;
			*pt = '\0';
			pt = ch; pt1 = buf;
			for (i=0; i < strlen(buf); i++)
			{
				if (isdigit(*pt1))
					*pt++ = *pt1;
				++pt1;
			}
			*pt = '\0';
			dd_total = atoi(ch);
			continue;
		}
		if (strstr(buf, "libres") != NULL)
		{
			--pt;
			*pt = '\0';
			pt = ch; pt1 = buf;
			for (i=0; i < strlen(buf); i++)
			{
```

```
                        if (isdigit(*pt1))
                                *pt++ = *pt1;
                        ++pt1;
                }
                *pt = '\0';
                dd_libres = atoi(ch);
                continue;
        }
    }
}

fclose(fic);

/*
 * Stats disque extractible.
 */
chdir(TEMP_DIR);
sprintf(ch, "CHKDSK D: > %s", TEMP_FILE);
system(ch);
fic = fopen(TEMP_FILE, "r");
if (fic == NULL)
{
        fprintf(stderr, "Impossible d'obtenir les stats.\n");
        exit(1);
}

for (;;)
{
        if (fgets(buf, MAXLEN, fic) == NULL)
                break;

        pt = strstr(buf, "kilo-octets");
        if ( pt != NULL)
        {
                if (strstr(buf, "total") != NULL)
                {
                        --pt;
                        *pt = '\0';
                        pt = ch; pt1 = buf;
                        for (i=0; i < strlen(buf); i++)
                        {
                                if (isdigit(*pt1))
                                        *pt++ = *pt1;
                                ++pt1;
                        }
                        *pt = '\0';
                        de_total = atoi(ch);
                        continue;
                }
                if (strstr(buf, "libres") != NULL)
                {
                        --pt;
                        *pt = '\0';
                        pt = ch; pt1 = buf;
                        for (i=0; i < strlen(buf); i++)
                        {
                                if (isdigit(*pt1))
                                        *pt++ = *pt1;
                                ++pt1;
                        }
                        *pt = '\0';
                        de_libres = atoi(ch);
                        continue;
                }
        }
    }

    fclose(fic);

    /*
     * Stats GPS (disque dur)
     */
    chdir("C:\\");
    chdir(GPS_DIR);

    if ( (h_rep=_findfirst("*.*", &rep)) == -1L )
    {
            fprintf(stderr, "Il n'y a pas de fichier dans le
repertoire %s.\n", GPS_DIR);
            exit(1);
    }
    else
    {
            do
            {
                    if (strlen(rep.name) < 4)
                            continue;

                    if (!strcmp(rep.name, "cgremote.sum") )
                            continue;

                    if (!strcmp(rep.name, "CGREMOTE.SUM") )
                            continue;

                    if (rep.time_write > prdGPS.datemodif)
                    {
                            strcpy(prdGPS.nom, rep.name);
                            prdGPS.taille = rep.size;
                            prdGPS.datemodif = rep.time_write;
                    }
```

```
                if (rep.time_write < pvdGPS.datemodif)
                {
                        strcpy(pvdGPS.nom, rep.name);
                        pvdGPS.taille = rep.size;
                        pvdGPS.datemodif = rep.time_write;
                }
        } while (_findnext(h_rep, &rep) == 0);
        _findclose(h_rep);
}

/*
 * Stats GPS (disque extractible)
 */
chdir("D:\\");
chdir(GPS_DIR);

if ( (h_rep=_findfirst("*.*", &rep)) == -1L )
{
        fprintf(stderr, "Il n'y a pas de fichier dans le
repertoire %s.\n", GPS_DIR);
        exit(1);
}
else
{
        do
        {
                if (strlen(rep.name) < 4)
                        continue;

                if (!strcmp(rep.name, "cgremote.sum") )
                        continue;

                if (!strcmp(rep.name, "CGREMOTE.SUM") )
                        continue;

                if (rep.time_write > preGPS.datemodif)
                {
                        strcpy(preGPS.nom, rep.name);
                        preGPS.taille = rep.size;
                        preGPS.datemodif = rep.time_write;
                }

                if (rep.time_write < pveGPS.datemodif)
                {
                        strcpy(pveGPS.nom, rep.name);
                        pveGPS.taille = rep.size;
                        pveGPS.datemodif = rep.time_write;
                }
        } while (_findnext(h_rep, &rep) == 0);
        _findclose(h_rep);
}

if (dd_total >= 1)
        taux_dd = ( (float) dd_total - (float) dd_libres )/
(float) dd_total;
else
        taux_dd = -1;

if (de_total >= 1)
        taux_de = ( (float) de_total - (float) de_libres )/
(float) de_total;
else
        taux_de = -1;

mod_libres = (double) dd_libres; mod_libres /= 1024.0;
mod_total = (double) dd_total; mod_total /= 1024.0;

moe_libres = (double) de_libres; moe_libres /= 1024.0;
moe_total = (double) de_total; moe_total /= 1024.0;

printf("              ***** Disque dur *****\n");
printf("+===============+==============+==================+
\n");
printf("| Octets libres |     Capacite    | Taux d'occupation
|\n");
printf("+---------------+--------------+------------------
+\n");
printf("| %10.2f MO | %10.2f MO |          %6.2lf %% |\n",
        mod_libres, mod_total, taux_dd*100.0);
printf("+===============+==============+==================+
\n");
printf("|  Fichier GPS  |       Nom       |   Taille (octets)
|\n");
printf("+---------------+--------------+------------------
+\n");
printf("|  Plus ancien  |  %12s |          %8d |\n",
        pvdGPS.nom, pvdGPS.taille);
printf("+---------------+--------------+------------------
+\n");
printf("|  Plus recent  |  %12s |          %8d |\n",
        prdGPS.nom, prdGPS.taille);
printf("+===============+==============+==================+
\n");

printf("          ***** Disque extractible *****\n");
printf("+===============+==============+==================+
\n");
printf("| Octets libres |     Capacite    | Taux d'occupation
|\n");
```

```
        printf("+---------------+--------------+------------------
+\n");
        printf("| %10.2f MO | %10.2f MO |          %6.2lf %% |\n",
               moe_libres, moe_total, taux_de*100.0);
        printf("+===============+==============+==================+
\n");
        printf("|  Fichier GPS  |       Nom       |   Taille (octets)
|\n");
        printf("+---------------+--------------+------------------
+\n");
        printf("|  Plus ancien  |  %12s |          %8d |\n",
               pveGPS.nom, pveGPS.taille);
        printf("+---------------+--------------+------------------
+\n");
        printf("|  Plus recent  |  %12s |          %8d |\n",
               preGPS.nom, preGPS.taille);
        printf("+===============+==============+==================+
\n");
        printf("taper entree pour finir..."); gets(ch);

        return 0;
}
```