GlacioBasis Manual

Revision 2 (14th April 2015)

Compiled by Michele Citterio, GEUS
with contributions by Dirk van As, Søren Nielsen, Martin Veicherts (GEUS)

Table of contents

Preface

This manual provides information required to properly carry out GlacioBasis glaciological fieldwork at the A.P. Olsen ice cap taking advantage of the logistic support provided by the Zackenberg Research Station. The focus is on the field procedures and technical details to be strictly followed to ensure the consistency and continuity of the data collection in the field.

The manual in this first version should be regarded as an evolving document. Furthermore, some information such as deployed sensors or the updated location of the ablation stakes can change slightly over time as a result of additions to the present setup or re-establishment of lost stakes.

Revisions history

3rd November 2009, version 1
14th April 2015, version 2: provided more context and background for the Programme; updated and extended the GPS and GPR sections; updated AWS maintenance checklist; updated contacts information; fixed

GlacioBasis

Greenland's glaciers and Ice Caps are losing mass at a rate more than double of the Ice Sheet when adjusted for size, accounting for 14-20 % of the total sea level rise contribution from Greenland. As the glaciological component of a core WMO GCW CryoNet site and a contributor to the GTN-G through WGMS, GlacioBasis timeseries complement the ongoing monitoring based at Zackenberg, enable process-based understanding of physical processes driven by glacier melt water run-off in the downstream terrestrial and marine ecosystems, and can inform regional and global assessments under AMAP and IPCC.

The primary aim of the GlacioBasis monitoring programme at Zackenberg is to produce a record of high quality glaciological observations from the A.P. Olsen Ice Cap and its outlet glacier in the Zackenberg river basin. The A.P. Olsen Ice Cap is located at 74° 39' N and 21° 42' W. The summit reaches an elevation of 1425 m and the terminus of the outlet glacier contributing to the Zackenberg river basin is at 525 m (figure 1). Zackenberg Research Station is located SE of the site, approximately 35 km downstream. The most direct access to the glacier terminus is through Store Sødal. The need to measure winter accumulation requires fieldwork to be carried out during springtime, immediately before the onset of significant snow melt. This timing is also necessary for snow-mobile use, which greatly simplify access to the glacier and transport of equipment and instrumentation. Fieldwork must be carried out every year in order to maintain the stakes network operational and to service the automatic weather stations (AWS) on the glacier.

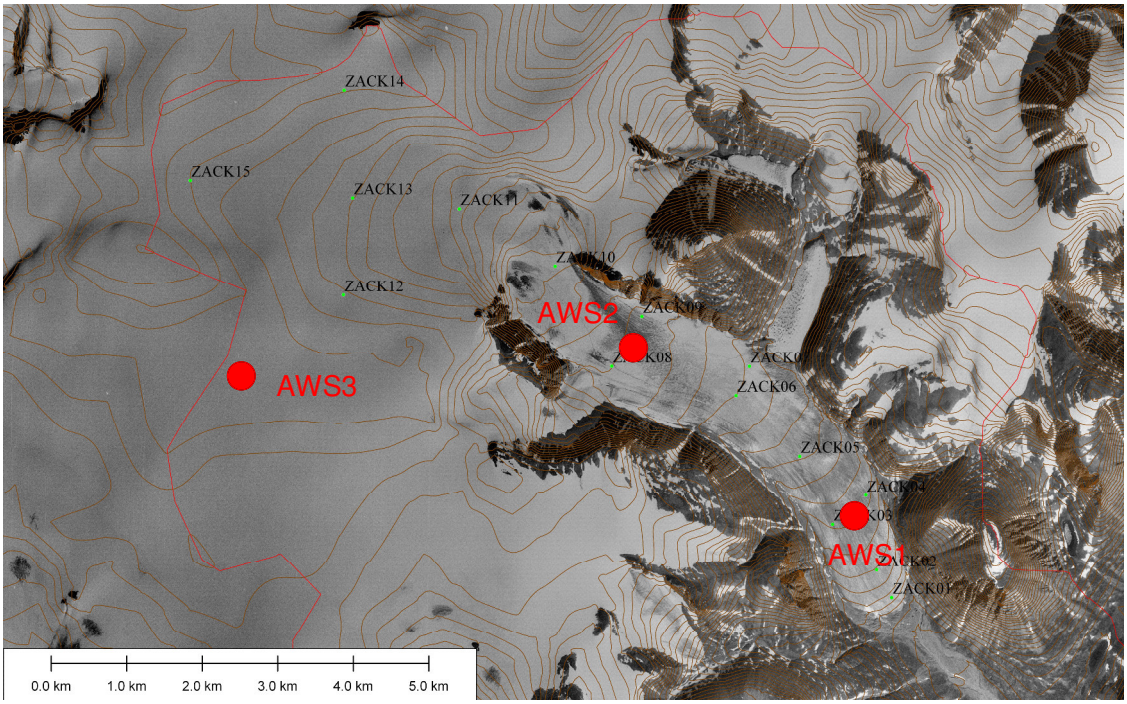


Fig. 1 – Map of the investigated outlet glacier with the position of the stakes and AWS.

The severe scarceness of glacier mass balance measurements from glaciers and local ice caps in East Greenland, the strong impact that local glaciers and ice caps outside the Ice Sheet are expected to exert on sea level rise in the present century (Meier et al. 2007), and the warming expected to occur in the Arctic (IPCC 2007) highlight the scientific importance of GlacioBasis monitoring tasks. The anticipated use of the monitoring data is to model the surface energy balance and the glacier mass budget with physically based models that, once calibrated and validated with in situ data, will allow to model the response of the glacier to future climate change scenarios.

Figure 1 provides an overview and map of the investigated outlet glacier of the A.P. Olsen ice cap. The position of the 14 ablation and displacement stakes is shown, together with the position of the three AWS.

GlacioBasis is conceptually linked to the other monitoring programmes in the Zackenberg river basin through the contribution of glacier melt water discharge to the river, its regime and seasonality, and its impact on solutes and sediment fluxes to the sea. Furthermore, the study site offers opportunities to extend investigations to the glacier dammed lake on the eastern side of the studied outlet glacier (which is regarded as the source of several floods recorded downstream in the past years), and to the formation of superimposed ice, which is expected to be significant at this site.

‘

General instructions

All GlacioBasis personnel arriving at Zackenberg station must have read the ZERO Site Manual (available from http://www.zackenberg.dk/NR/rdonlyres/8BA15D73-8D58-4C17-ABDA-99C0067FD571/0/ZERO20Site20Manual2C20ver2027102008.pdf) and must commit to follow its instructions. This is particularly important with regard to the restricted mobility in the protected areas around Zackenberg Station.

Safety

Safety is the first priority for all GlacioBasis personnel in the field. All instructions from the Zackenberg Station manager must be followed strictly. Due care and sound judgment must be applied at all times, especially upon leaving the immediate surroundings of the station, when driving snowmobiles, on the glacier and when crossing the river.

All personnel will need to be confident in the use of firearms for defending themselves and their colleagues against polar bears and musk oxen, and is requireed to carry a rifle and ammunition when leaving the station's surroundings. Firearms, ammunitions and flare guns are provided by Zackenberg Station. All GEUS personnel is required to have attended to the GEUS shooting course, the first aid course, the glacier safety course and the snow safety course. These courses are intended to provide enable each participant to make an informed assessment of the safety conditions at all times. Since VHF radio communication is not possible from the glacier to Zackenberg Station due to the terrain morphology, an Iridium phone must be carried during every trip to the glacier. VHF radios may be useful to locally coordinate work on the glacier.

It is the participant's responsibility to plan ahead before leaving Denmark and make sure all the required personal field gear is available and in good working conditions. Personal safety equipment for GEUS personnel can be procured through GEUS.

Data collection, safeguarding and documentation

GlacioBasis aims at producing a consistent dataset of high quality observation data, and this requires that documented field procedures are followed strictly and that metadata be always attached to all data produced in the field. All maintenance work on the automatic weather stations (AWS) must be documented using the field maintenance checklist in Appendix A, with special care to recording the serial numbers of the instruments and the initial readings. Pictures should always be numbered and referenced in the maintenance checklist. Further details relevant to the specific monitoring tasks are provided in the following sections.

It is important to safeguard against accidental loss of data by duplicating field notes on a daily basis and ensuring that one copy remains at the Station while the other (if necessary

for reference) is carried in the field. Use of special field notebooks, A4 size sheets and printed forms made with water-resistant paper is recommended. Electronic data will need to be duplicated on at least two separate storage supports to safeguard against data corruption.

Since fieldwork is being carried out at a remote location posing significant operative challenges, a trade-off may be required at times to ensure that the most important measurements are carried out even when the prescribed field plans and procedures can not be implemented fully due to unexpected weather or field conditions. The field leader is responsible for prioritizing the pending tasks in consideration of the available resources and without compromising safety and the quality of the measurements. In the exceptional case that a measurement can not be made by strictly following the prescribed protocol, this must be fully documented in the metadata.

Relevant GlacioBasis data will need to be fed into the Zackenberg Basis database.

Field procedures

Ablation and velocity stakes

A network of 14 ablation and surface velocity stakes distributed along the central flow line has been established in spring 2008 on the outlet glacier of the A.P. Olsen ice cap and along three transects at elevations of approximately 675, 900 and 1300 m (figure 1 and table. 1) respectively. Each 6 m long stake was assembled from 2 m lengths of aluminium tube. A Kovacs drill was used with success, allowing very fast drilling operations.

ID	LAT	LONG
01	N7437.1293	W02121.8890
02	N7437.4536	W02123.0806
05	N7438.3637	W02125.6941
04	N7437.9356	W02123.9705
03	N7437.6633	W02122.1934
06	N7438.5687	W02125.3384
08	N7438.9168	W02128.2411
07	N7438.5616	W02129.0397
10	N7439.6703	W02133.1732
09	N7439.2688	W02130.5855
12	N7439.7398	W02136.0492
11	N7439.0492	W02136.2686
13	N7440.5023	W02136.3062
14	N7439.8511	W02140.4234

Tab. 1 – Position of the ablation and displacement stakes.

The stakes must be measured and re-drilled every year following this procedure (assuming the current planning of only one visit per year in springtime):

1. upon approaching an existing stake, take a picture documenting the undisturbed site
2. measure the length of the stake above the snow surface
3. place the GPS antenna on top of the existing stake and start a measurement with the most accurate survey mode available (see GPS section)
4. take a note of stake number, date, time, picture number, air temperature, GPS survey mode being used for the precision survey and GPS fix from the handheld
5. probe with an avalanche probe the snow depth in the immediate surroundings of the stake and record all measurements, not just the average

6. if time or weather constraints prevent digging a snow pit at every stake, pay particular care when probing to "feel" the snowpack stratigraphy and assess how similar it appears to be to other stake sites where a snow pit has been dug. A snow pit is required in any case if the depth of the altitude of the stake site or the snowpack depth are significantly different from the closest snow pit measured.
7. dig a snow pit in the surroundings of the stake site without disturbing the stake, and survey it following the procedure in the "snow pits" section.
8. recover the emerging lengths of aluminum tube from the old stake
9. mark the top of the remaining old stake by over-expanding one of the old joints in such a way that it will permanently remain attached to the stake end.
10. drill the new stake close to the existing one using the Kovacs drill (pay special care when using the Milwakee electric drill to avoid being hardly hit by the drill in case the drill suddenly gets stuck in the hole. The two strokes engine has lower torque.
11. never leave the drill sitting still in the hole to avoid it getting frozen to the hole, or otherwise it will be lost.
12. measure the length of the new stake above the snow surface
13. place the GPS antenna on top of the stake and start a measurement with the most accurate survey mode available (see GPS section)
14. probe with an avalanche probe the snow depth in the immediate surroundings of the new stake and record all measurements, not just the average
15. upon leaving, take a picture documenting the site conditions

Required equipment:

- map and updated coordinates of the stakes
- handheld GPS with waypoints in memory marking the position of stakes
- measuring tape
- camera and field notebook
- avalanche probe
- spare aluminium tubing and connection fittings
- snow pit kit (see below)

Snow pits

The most important data obtained from a snow pit are the water equivalent depth and the density profile of the snow pack. Additional observations should be carried out whenever possible following this priority: temperature profile, snow crystallography, dusts layers, penetrability profile. The snow pit will be surveyed following this procedure:

1. mark in the snow surface the intended outline of the snow pit orienting it so that the side walls where observations will be made is facing away from the sun
2. take a picture documenting the unexcavated site
3. take a note of snow pit position, date, time, picture number and air temperature
4. dig the snow pit paying attention not to disturb the snow surface along the side where observations will be made
5. deepen the snow pit until glacier ice or the previous summer surface are found and prepare the vertical surface to be measured
6. measure the temperature profile allowing sufficient time for he thermometers to settle to a stable reading
7. identify the snow stratigraphy and mark the surfaces of the various levels
8. measure and record the total thickness of the snowpack and the thickness of the individual snow layers, recording any surface enriched with dust
9. using a sampler of known volume take an horizontal core at 1/3 from the bottom of each snow layer, weigh the sample with a precision spring scale
10. describe the crystallography of the snow and proceed to the following layer
11. take pictures documenting the snow pit
12. fill back the snow pit before leaving

Required equipment:

- map and handheld GPS
- camera and field notebook
- avalanche probe
- shovel (it is recommended to have both a broad light shovel and a smaller one with sharpened edge to negotiate high strength wind drifted or refrozen snow)
- snow pit kit, containing:
- known volume sampler
- rubber head hammer
- plastic bags
- precision scale
- snow stratigraphy survey forms
- reference scale and crystal type card
- insertion thermometers
- lens
- foldable measuring bar or measuring tape

AWS establishment and maintenance

The establishment and setup of an AWS need to be carried out following the checklist in Appendix A, filling in all details required in the form and taking pictures documenting the conditions of the AWS both upon arrival (before any disturbance to the instruments and the snow cover) and upon leaving the site. Particular care must be paid at the proper leveling and orientation of the sensors. All technical details of the AWS desing, internal

and plugs wiring, and datalogger programming are contained in Appendix B, C and D respectively. Fig. 3 provides an overview of the station marked as AWS 1 in Fig. 1

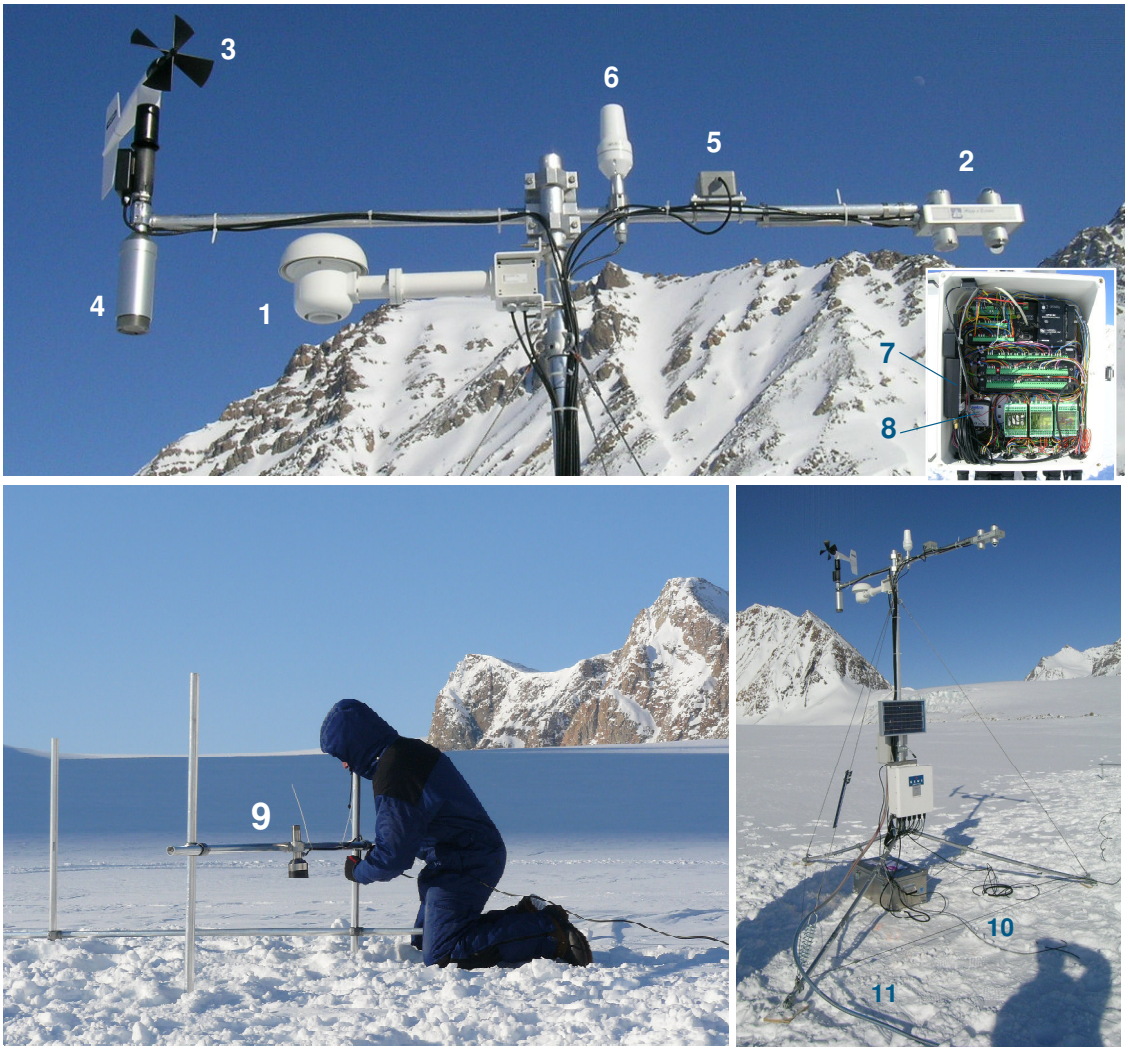


Fig. 3 – Overview of the AWS and the sensors. 1. Air temperature and humidity (aspirated); 2. SW↓, LW↓, SW↑, LW↑ radiation; 3. Wind speed and direction; 4. Snow surface level; 5. two-axes tilt meter; 6. Iridium satellite antenna; 7. GPS receiver and Iridium modem; 8. Air pressure; 9. Surface level (sonic ranger); 10. Thermistors string (GEUS); 11. Hydraulic ablation meter (GEUS)

Telemetry data transmitted through the Iridium satellite system is received at GEUS as email messages and the data can be retrieved from within GEUS intranet by a user with appropriate permissions to access the dedicated mailbox running the Python program attached in Appendix E.

Required equipment:

The following table 2 (prepared by Søren Nielesn, GEUS) describes various recommended set of tools to setup or carry out maintenance on a GlacioBasis AWS. Some of the items may only be required when testing the instruments at Zackenberg Station and may be left behind when leaving for the glacier.

MINIMUM		MEDIUM		COMPLETE
Svensknøgler	8" el. 6" med brede kæber			
Svensknøgle	4"			
Løse skraldefastnøgler	10 - 13 - 17 mm	Fastnøglesæt 8-10-13-15-17-19 mm		
Skruetrækker & bit sæt				
Urmagerskruetrækkersæt				
Bacho PH1x75	For iridium antenna	Bacho 1x5,5x100	BE-8150	
Vandpumpetang				
Stor skævbider				
Mora Kniv				
Stanleykniv				
Skalpel				
Umbraco, mm	1 sæt løse, 1 - 10 mm	Umbraco, mm, with handle	4 - 6 - 8 mm	Nødvendige umbrako:
Umbraco, "	1 sæt løse, 1/16 - 3/8 "	Umbraco, ", with handle	1/4 - 5/16	1/4" - 5/16" KeeKlamp
				4 mm CNR1
		Umbraco, mm, "Knife model"		8 mm Letrona
Elektronik skruertækker	Campbell	Umbraco, ", "Knife model"		
Lille elektronik skruertækker	CK 2,5x75 el. Bacho 0,4x75			
		Elektronik bidetang		
Multimeter		Elektronik spidstang		
Digitalt waterpas	S-Digit Mini	Loddekolbe, gas eller opladelig		
Magnetisk kompas	Silva pejlekompas	Loddetin		
186 mm Kabelbindere		Batteries:	AAA - AA - 9V	
360 mm Kabelbindere				
		Wiresaks		
Låseolie		1" Rørskærer		
		Juniorbue		
Silicone	DowCorning 732 (Cheap)	Topnøgler 5,5 - 7 - 8 mm		5,5: 3 mm nuts on plugs
				7 - mm: Forskellige
do.	DowCorning 3140 (Flowable)	Lille borsæt		spændbånd
		Crimptang		

Bolts 8x20 SS	5mm battery nuts	1mm² wire	Crimp conn. (Red)	Washers
Nuts 8 mm	Nuts 5 mm	Small cable tiers	Tube clamps small	6mm Bolts
Bolts 8x16 SS	3mm Bolts & Nuts	M10 Bolts & Nuts	Tube clamps medium	6 mm Nuts
Nuts 8 mm	Selftapping screws	Terminal strips	CF Cards	KeeKlamp screws

Spare plugs:	CA6LS	For sensors (Solder)	2
	CA3LD	For batterycable, solar panel	2
	CA6GD	Panel connectors, sensors (Solder)	2
	CA3GS	Panel connectors, power	2
	9pole plugs	For t:string, CNR1 (Crimp)	2
	9pole panel sockets	Panel conn., 9 pole (Crimp)	2

Div. Spares:	Silicon	2 tubes, 732 (Cheap) - 3140 (Fluid)
	Hose clamps	Large (ex. For CNR1)

Tab. 2 – Various options for the recommended tools for AWS setup and maintenance.

GPS surveys

GlacioBasis has two main uses for GPS measurements: tracking the displacement of the stakes and positioning the GPR radar traces. Both require accurate carrier phase differential, and all efforts should be expended in obtaining them. Standalone GPS measurements should be avoided since they are much less accurate, and should only be taken when the available equipment or other technical circumstances prevent operations

in differential mode. Postprocessing baselines start from the master station which is setup at the terminus of the glacier (Fig. 2). No permanent marker nor surveyed point is available, so the accurate position of the master station is determined every year by the Precise Point Positioning (PPP) technique.

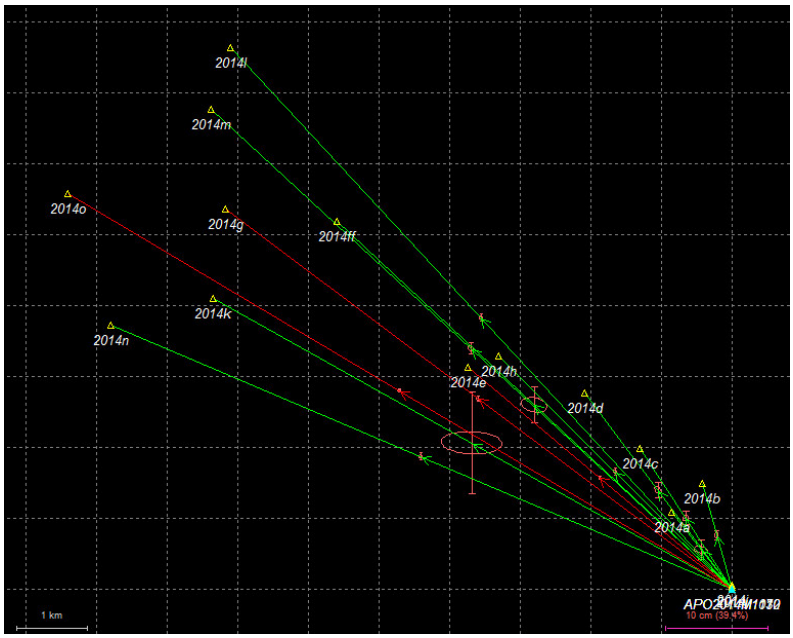


Fig. 2 – A.P. Olsen baselines from the master station at the terminus to the ablation stakes.

The detailed procedure for taking DGPS measurements vary with the specific receiver model used, see the instrument documentation for details.

GlacioBasis has two Trimble R7 GNSS receivers which are programmed as required and are ready to start recording by simply powering them on.

Whatever the instrument used, the model, configuration and setup must be recorded and special care must be paid in properly configuring the antenna parameters and in positioning the antenna on the stake. Unobstructed view of the sky must be ensured.

GPR surveys

Snow depth profiles are surveyed with an 800 MHz shielded antenna and either constant time (0.25 sec) or constant distance (0.5 m) settings by setting up the antenna on a fiberglass sled towed by a snowmobile. The monitor should be setup so that the snowmobile driver can see the data being acquired and adjust the speed consequently. Every few hundred meters, and also every time unclear features are seen on the display,

the snowmobile must stop and a few measurements of the snowpack density must be taken with the avalanche probe, recording all the values and not just the average.

Data files will be frequently downloaded from the GPR instrument to a USB stick and at daily backed up to a second storage support to be kept at Zackenberg Station.

The tracks to be followed during the GPR survey should cover the glacier as evenly as feasible, but avoiding dangerous crevassed areas unless the snow cover is abundant and safe. As an example, Fig. 3 shows the tracks followed in 2008. GIS files with the tracks surveyed during previous years are available from within GEUS intranet in the folder:

\\Geusnt1\glaciologi\GlacioBasis\GlacioBasis Manual\GIS files\

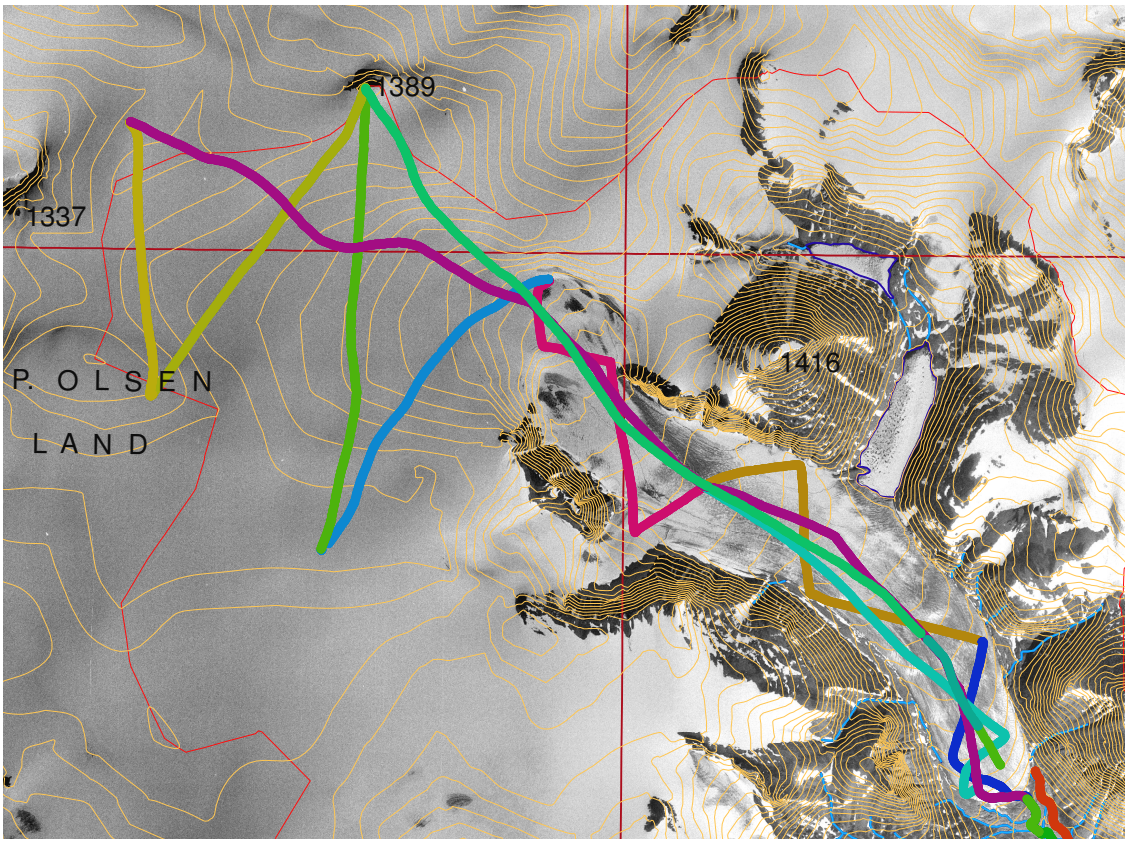


Fig. 3 – GPR tracks followed during the 2008 snow depth survey.

Reasonably accurate positioning of the GPS traces is important, even though D-GPS accuracy is not a strict requirement for snow depth monitoring.

Contacts

This is the updated list of contact details including phone numbers and addresses for use during the fieldwork season fieldwork. It is mainly for the personnel in the field, but also contains some contacts useful during the trip to Greenland and for arranging unanticipated shipments.

name	email	phone	mobile (*)	
Michele Citterio	mcit@geus.dk	+45 91333832		scientific and technical issues, fieldwork planning, project management
Andreas Ahlstrøm	apa@geus.dk	+45 91333810		scientific issues
Martin Veicherts	mav@geus.dk	+45 91333831		technical issues with AWS, support with shipments
Marianne Vestergaard	mve@geus.dk	+45 31103038		reporting of accidents and injuries during fieldwork (for GEUS personnel)
GEUS switchboard	geus@geus.dk	+45 91333500		General GEUS contact
Malå Geoscience		+46 95334550		technical support for GPR-related issues
Geoteam	support@geoteam.dk	+45 77332233		technical support for GPS-related issues
Zackenberg Station	logistics@zackenberg.com			communications to people in the field (must specify name)
Jørgen Skafte	jska@bios.au.dk	+45 87158674	+45 23227110	Logistics
Henrik Spanggård	hsp@bios.au.dk	+45 87158675		Logistics

* several phone numbers are not published here and must be obtained before leaving Denmark. The complete page is available from within the GEUS intranet in this folder: \\Geusnt1\glaciologi\GlacioBasis\GlacioBasis Manual\contacts.doc

Boxes and all cargo to be delivered to GEUS shall always be addressed to:

GEUS varemodtagelsen, Riegensgade 13, DK-1316 Copenhagen K

Alternatively and only upon agreement with the logistic staff at Zackenberg Station, the delivery can be routed through AU in Roskilde. This option may in general be cheaper but slower, and either Michele and/or Martin at GEUS will need to be notified of the shipment.

The address for regular mail correspondence is:

Att. Michele Citterio, GEUS, Øster Voldgade 10, 1350 Copenhagen K

Appendix A - AWS establishment and maintenance checklist

(Dirk van As and Søren Nielsen, GEUS)

AWS checklist and maintenance guide version 7

Station name	
People present	instructed by:
Purpose	standard maintenance / installation / take-down / quick visit / emergency
Date	
Transport	helicopter / fixed wing / snow mobile / dog sleds / on foot from:
Weather	warm / cold / very cold / no wind / some wind / rain / cloud cover %:

Notes:

Metadata <u>before</u> maintenance (if establishing a new AWS: skip this page)

Time difference between logger clock and UTC	+ / -
Adjust logger time to UTC (be sure PC runs on UTC time)	Y / N
Download data to PC and/or change CF Card (wait for green light)	Y / N
Name of logger program	
Photo or screen dump of values in fast scan mode	Y / N
Latitude (dd mm.mmm)	
Longitude (dd mm.mmm)	
Photos of tripod, sensors, logger box wiring etc. (more is better)	Y / N
Mast tilt in boom direction (+ if radiometer tilting down)	°
Mast tilt across boom looking from radiometer (- if clockwise)	°
Boom direction relative to north	° true/magn
Box on wind sensor exactly towards mast? If not, measure direction.	Y / N
Radiometer aligned with mast and inclinometer? If not, measure.	Y / N
Temperature / humidity height (bottom of casing – surface)	cm
Sonic ranger on AWS height (membrane – surface)	cm
Wind sensor height (center of propeller – surface)	cm
Sonic ranger on stakes height (membrane – surface)	cm
Free length of stake – outer, holding boom with sonic ranger	cm
Free length of stake – middle	cm
Free length of stake – outer, holding boom without sonic ranger	cm
Free length of ablation hose on/above the ice surface	cm
Alt: depth of ablation hose using markings on hose	cm
Height of ablation hose fluid level above the ice surface	cm
Vertical difference in surface height (at station – at hose) (approx)	cm
Length of thermistor string on surface from surface marking	cm
Height of surface irregularities	cm
Stake assembly still standing?	Y / N / more or less
Apparent damage to sensors? If yes, which?	Y / N

Notes:**Part**	**Old part number**	**New part number, if replaced**	**Potential maintenance tasks**
Radiometer			- Aligned with mast: Y/N - Clean: Y/N
Inclinometer			- Aligned with radiometer within 0.5°: Y/N
Satellite antenna			
Wind sensor			- Box on sensor exactly towards mast (south): Y/N
Temperature / humidity assembly			
T/hum assembly casing			- Fan spins and sounds OK: Y/N
T/hum assembly bracket			
Temperature probe in T/hum assembly			
HygroClip in T/hum assembly			- 40 °C offset in HygroClip temperature: Y/N
Sonic ranger on AWS			- Old sensor with new membrane: Y/N - Clicking 5 times as it should: Y/N
Solar panel			- Output OK: Y/N - Clean: Y/N
GPS antenna			- Stuck to top of logger enclosure: Y/N
Data logger			- New internal battery in old logger: Y/N - New operating system: Y/N
Card reader			- LED green of flashing orange (active): Y/N
Multiplexer			- Clicking as it should: Y/N
Iridium modem IMEI number			- Transmissions tested: Y/N - Problems: Y/N
Barometer			
Sonic ranger on stakes			- Old sensor with new membrane: Y/N - Clicking 5 times as it should: Y/N - In danger of being buried next winter: Y/N
Thermistor string			- Old string left in ice: Y/N/cut
Ablation hose assembly			- Old hose left in ice: Y/N/cut - Air in hose: Y/N/removed - Bladder 50% full: Y/N / after emptying/filling
Logger enclosure	- Replaced (including everything inside): Y/N - New vent filter in logger enclosure: Y/N - Moisture inside: Y/N - New desiccant bags (2): Y/N		
Battery box	- New box with new batteries: Y/N - Old box with new batteries: Y/N - Moisture inside: Y/N - New desiccant bags (2): Y/N		
Stakes	- New ones drilled in: Y/N - Length: 6 / 8 / 10 m - Caps in bottom end to reduce melt-in: Y/N		
Tripod	- New parts: Y/N - Boom direction: true north-south: Y/N		

Metadata <u>after</u> maintenance / establishment

Time logger clock set (to UTC)?	Y / N
Name of logger program	
Photo or screen dump of values in fast scan mode	Y / N
Latitude (dd mm.mmm)	
Longitude (dd mm.mmm)	
Photos of tripod, sensors, logger box wiring etc. (more is better)	Y / N
Mast tilt in boom direction (+ if radiometer tilting down)	°
Mast tilt across boom looking from radiometer (- if clockwise)	°
Boom direction relative to north (radiometer should point to true south)	° true/magn
Box on wind sensor exactly towards mast? If not, change.	Y / N
Radiometer aligned with mast and inclinometer? If not, align.	Y / N
Temperature / humidity height (bottom of casing – surface)	cm
Sonic ranger on AWS height (membrane – surface)	cm
Wind sensor height (center of propeller – surface)	cm
Sonic ranger on stakes height (membrane – surface)	cm
Free length of stake – outer, holding boom with sonic ranger	cm
Free length of stake – middle	cm
Free length of stake – outer, holding boom without sonic ranger	cm
Free length of ablation hose on/above the ice surface	cm
Alt: depth of ablation hose using markings on hose	cm
Height of ablation hose fluid level above the ice surface	cm
Vertical difference in surface height (at station – at hose) (approx)	cm
Length of thermistor string on surface from surface marking	cm
Height of surface irregularities	cm

Notes:

Appendix B – AWS station design

(Michele Citterio, GEUS)

Glaciobasis Main AWS

Rev. MCIT 17 Mar 2008

NOTE: This is the working setup with program revision 1.3
Design changes or additions from the previous revision (9 Feb 2008) are written ***like this***. See also the list of changes at the end of the document.

Datalogger and multiplexer box

CR1000 wirings

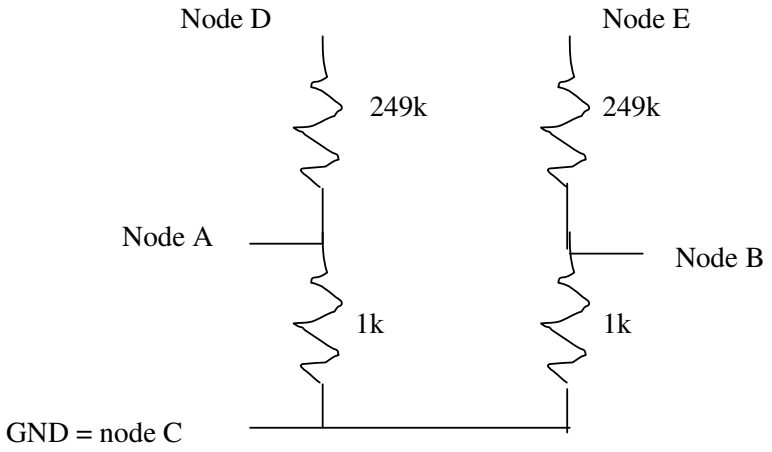
Top wiring strip			
CR1000 side		***Sensor side***	***Device name***
DIFF1	SE1	H → EX2	
	SE2	L – RED	HygroClip n. 1 (through a Campbell 4WPB100)
GND		G -- VIO	HygroClip n. 1 (through a Campbell 4WPB100)
DIFF2	SE3	ORA	HygroClip n. 1
	SE4	BLK	HygroClip n. 1
GND		***shield***	***HygroClip n. 1***
DIFF3	SE5	H → EX2	
	SE6	L – RED	CNR1 – Pt100 cable (through a Campbell 4WPB100)
GND		G -- BLU	CNR1 – Pt100 cable (through a Campbell 4WPB100)
DIFF4	SE7	YEL	CNR1 – Pt100 cable
	SE8	GRE	CNR1 – Pt100 cable
GND		SHIELD	CNR1 – Pt100 cable
EX1		BLU	Young wind
GND		open	
P1		RED	Young wind
GND		***BLK***	***Young wind***
P2		open	
GND		open	
Middle wiring strip			
DIFF5	SE9	H → EX2	
	SE10	L – RED	HygroClip n. 2 (through a Campbell 4WPB100)
GND		G -- VIO	HygroClip n. 2 (through a Campbell 4WPB100)
DIFF6	SE11	ORA	HygroClip n. 2
	SE12	BLK	HygroClip n. 2
GND		***shield***	***HygroClip n. 2***
DIFF7	SE13	→ MUX ODD H	Multiplexer COMMON, ODD DIFF. CHANNEL
	SE14	→ MUX ODD L	Multiplexer COMMON, ODD DIFF. CHANNEL
GND		→ MUX GND ***+ YEL + YEL***	Multiplexer COMMON ***HygroClip n. 1 and n. 2***
DIFF8	SE15	→ RES. PANEL n. A	Resistor bridges panel for the thermistor string
	SE16	→ RES. PANEL n. B	Resistor bridges panel for the thermistor string
GND		→ RES. PANEL n. C	Resistor bridges panel for the thermistor string
EX2		→ SE1+SE5+SE9	The black wires of the Campbell 4WPB100 bridges
GND		shield + shield	SR50 n.1 e n. 2
EX3		WHI (thermistor sring) ***+ RED (NT1400)***	Termistor string ***and NT1400 bridge pressure transducer***

Bottom wiring strip			
G		open	
5V		open	
G		BLK	Setra barometer
SW-12		→ SW-12V DISTRIB.	
G		→ MUX GND	Multiplexer CONTROL & POWER
12V		open	
12V		open	
G		→ LED node A	Low current LED with resistor
COM1	C1	→ MUX RES	Multiplexer CONTROL & POWER
	C2	→ MUX CLK	Multiplexer CONTROL & POWER
COM2	***C3***	***open***	***From CR1000 manual, this shouldn't be used since we already use COM2 as a serial port***
	C4	WHI	Garmin GPS
G		BLK+YEL+shield	Garmin GPS
COM3	***C5***	***→LED node B***	***Low current LED with resistor***
	C6		TX module relais control, active high
COM4	C7	GRE + GRE	SR50 n.1 e n. 2
	C8		Fan relais control, active high
G		WHI+BLK+WHI+BLK	SR50 n.1 e n. 2
other			
RS232		To NAL Iridium and GPS module. Needs a male-to-male 9 poles D adapter	
Ground lug		***Current sensing shunts wiring box terminal post 15***	

RESISTOR BRIDGES PANEL FOR THE THERMISTOR STRING

(see drawing for the bridge circuit and the meaning of the node letters below)

Circuit side	*Logger and string side*	*Device name*
Node A	→ SE15	CR1000
Node B	→ SE16	CR1000
Node C	→ GND	CR1000
Node D	→ MUX EVEN H	Multiplexer COMMON, EVEN DIFF. CHANNEL
Node E	→ MUX EVEN L	Multiplexer COMMON, EVEN DIFF. CHANNEL



MULTIPLEXER

(it must be configured as a 4x16 MUX)

Multiplexer side			***Device side***	***Device name***
CONTROL & POWER		RES	→ C1	CR1000
		CLK	→ C2	CR1000
		GND	→ G	CR1000
		12V	→ SW-12V DISTRIB. PANEL	SW-12V DISTRIB. PANEL
COMMON	ODD	H	SE13	CR1000
		L	SE14	CR1000
	GND		→ GND (between SE14 - SE15)	CR1000
	EVEN	H	SE15	CR1000
		L	SE16	CR1000
GND			open	
1	1	H	GRE	Setra Barometer
		L	GRE	Young wind
	GND		open	
	2	H	open	
		L	open	
GND			***WHI, shield***	***Young wind***
2	***3***	***H***	***→current sense shunts, post 13***	***Battery low side current sensing R hi***
		L	***→current sense shunts, post 14***	***Battery low side current sensing R lo***
	GND		open	
	4	H	ORA	Termistor string level 1
		L	BLK	Termistor string level 2
GND			***open***	
3	***5***	***H***	***→current sense shunts, post 9***	***Solar panel low side current sensing R hi***
		L	***→current sense shunts, post 10***	***Solar panel low side current sensing R lo***
	GND		open	
	6	***H***	***BLU***	***Termistor string level 3***
		L	***YEL***	***Termistor string level 4***
GND			open	
4	***7***	***H***	***→current sense shunts, post 6***	***Fan low side current sensing R hi***
		L	***→current sense shunts, post 5***	***Fan low side current sensing R lo***
	GND		open	
	8	***H***	***GRE***	***Termistor string level 5***
		L	***BRO***	***Termistor string level 6***
GND			open	
5	***9***	***H***	***→current sense shunts, post 2***	***Iridium low side current sensing R hi***
		L	***→current sense shunts, post 1***	***Iridium low side current sensing R lo***
	GND		open	
	10	***H***	***PNK***	***Termistor string level 7***
		L	***VIO***	***Termistor string level 8***
GND			open	
6	11	H	YEL	Tilt sensor
		L	GRE	Tilt sensor
	GND		GRY	Tilt sensor
	12	H	open	

		L	open	
GND			open	
7	13	H	RED	CNR1 – Radiometers cable
		L	BLU	CNR1 – Radiometers cable
	GND		open	
	14	H	open	
		L	open	
GND			open	
8	15	H	GRY	CNR1 – Radiometers cable
		L	YEL	CNR1 – Radiometers cable
	GND		open	
	16	H	open	
		L	open	
GND			shield	
9	17	H	WHI	CNR1 – Radiometers cable
		L	BLK	CNR1 – Radiometers cable
	GND		open	
	18	H	open	
		L	open	
GND			open	
10	19	H	BRO	CNR1 – Radiometers cable
		L	GRE	CNR1 – Radiometers cable
	GND		open	
	20	H	open	
		L	open	
GND			open	
11	21	H	WHI	HygroClip n. 1
		L	BRO	HygroClip n. 1
	GND		***GRY***	***HygroClip n. 1***
	22	H	open	
		L	open	
GND			open	
12	23	H	WHI	HygroClip n. 2
		L	BRO	HygroClip n. 2
	GND		***GRY***	***HygroClip n. 2***
	24	H	open	
		L	open	
GND			open	
13	***25***	***H***	***YEL***	***NT1400***
		L	***BLU***	***NT1400***
	GND		***WHI***	***NT1400***
	26	H	open	
		L	open	
GND			open	
14	27	H	open	
		L	open	
	GND		open	
	28	H	open	
		L	open	

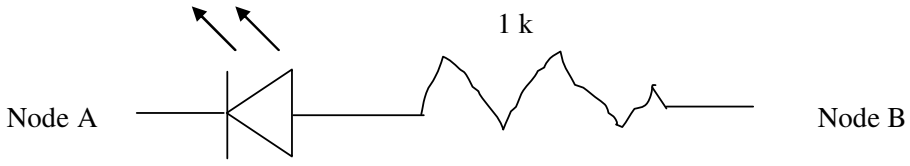
GND			open	
15	29	H	open	
		L	open	
	GND		open	
	30	H	open	
		L	open	
GND			open	
16	31	H	open	
		L	open	
	GND		open	
	32	H	open	
		L	open	
GND			open	

SW-12V DISTRIBUTION PANEL

Circuit side	*Sensor side*	*Device name*
Just wire everything together	→ SW-12	CR1000
	GRE	HigroClip n. 1
	GRE	HigroClip n. 2
	RED	Setra barometer
	→ MUX 12V	Multiplexer CONTROL & POWER
	RED	SR50 n.1
	RED	SR50 n.2
	BRO	Tilt sensor

LED

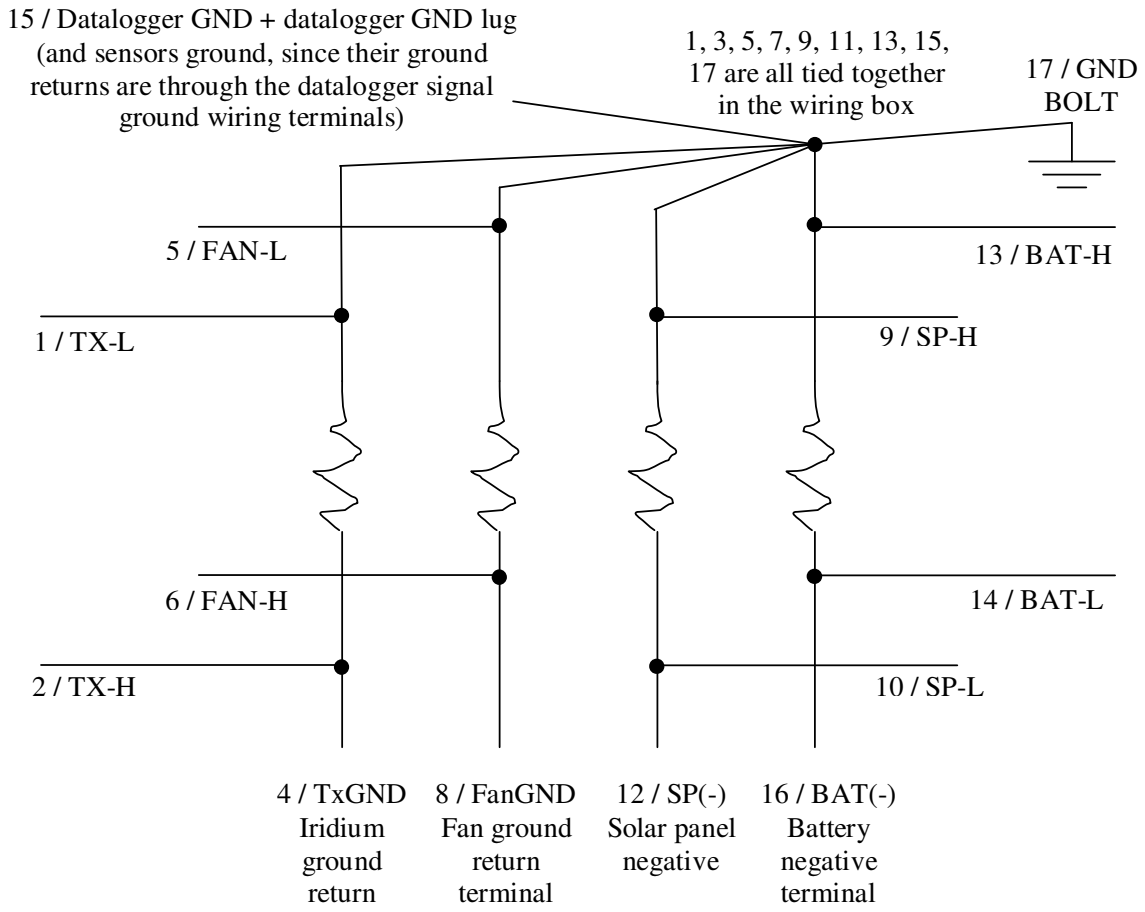
The LED ishould be a low current type not to overload the CR1000 output which are specified to max. 2 mA. Buy RS code n. 826-521 or 826-802 (buy both, RS must have some mistake on their website so it isn't clear which one is transparent, but they are still cheap enough…). Otherwise use type recommended by Peer Jorgensen.



CURRENT SENSING SHUNTS

(see drawing for the bridge circuit and the meaning of the node letters below)

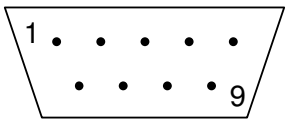
Circuit side	*other side*	*Device name*
15 Datalogger GND	POWER IN G GND LUG	CR1000
12 SP(-)	Solar panel negative	Solar panels
16 BAT(-)	Batteries negative	Batteries (the negative terminals are all tied together)
4 TxGND	Tx ground return	Iridium module
8 FanGND	Fan ground return	Rotronic Fan
9 SP-H	MUX SE5H	MUX
10 SP-L	MUX SE5L	MUX
13 BAT-H	MUX SE3H	MUX
14 BAT-L	MUX SE3L	MUX
6 FAN-H	MUX SE7H	MUX
5 FAN-L	MUX SE7L	MUX
2 TX-H	MUX SE9H	MUX
1 TX-L	MUX SE9L	MUX
17 GND BOLT	GND BOLT	CR1000 (connect to the GND screw on the enclosure)



RS232 GENDER CHANGER AND TX-RX CROSSOVER ADAPTER

(numbers refer to a standard 9 pins D-type connector, see drawing)

SYN-DC-936	*CR1000 RS232*
1	open
2	3
3	2
4	6
5	5
6	4
7	8
8	7
9	open



viewed from the pins side

List of design changes

1. The LED is moved from C3 to C5 in order to leave C3 unused. This is required according to the CR1000 documentation because we are using COM2 as a serial port (for the Garmin)
2. The previous connection from C5 to the manual FastScan switch is therefore removed, and the switch is not available anymore since no spare control ports are available.
3. The EX3 excitation channel is now also exciting the NT1400 pressure transducer
4. In the MUX, the connections to the current sensing shuts have been added (at SE channels 3, 5, 7, 9)
5. In the MUX, the missing connection to the other thermistors in the thermistor string have been added (at SE channels 6, 8, 10)
6. in the MUX the connection for the NT1400 has been added (at SE channel 25)
7. added the wirings for the current sensing shunts
8. added RS232 adapter wiring
9. changed ground routing for the return of the pulse signal of the
10. added RS232 adapter wiring
11. changed ground routing for the return of the pulse signal of the Young sensor
12. the GREY wires of the HygroClips have moved from the CR1000 to the MUX

Appendix C – AWS plugs internal wirings

(Søren Nielsen, GEUS)

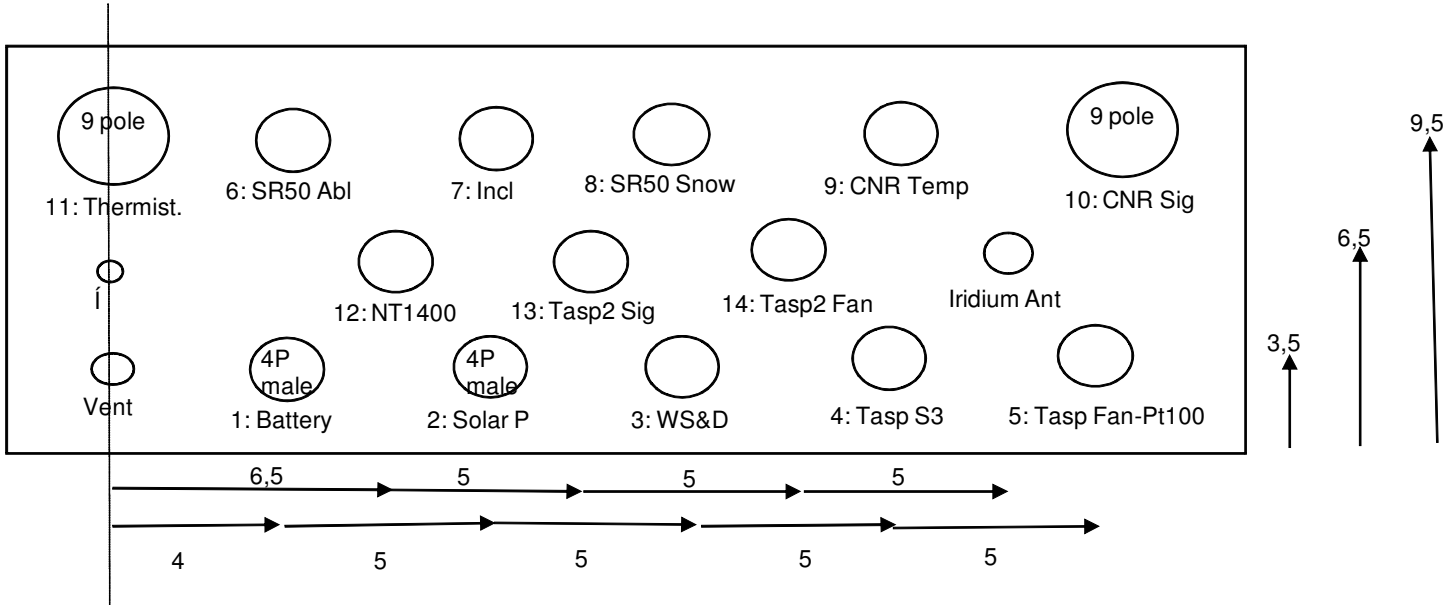
Plug no.	Plug type	Sensor	Remarks:
ZAK_M and ZAK_S1 wiring:			
Plug 1	4 pole	Battery box	Female cable, male logger box
Plug 2	4 pole	Solar Panel	Female cable, male logger box
Plug 3	7 pole	WD&S	
Plug 4	7 pole	Tasp Signal	
Plug 5	7 pole	Tasp Fan	
Plug 6	7 pole	SR50 Abl - Adress 0	
ZAK_M only wiring:			
Plug 7	7 pole	Inclinometer	
Plug 8	7 pole	SR50 Snow - Adress 1	
Plug 9	7 pole	CNR Temp	
Plug 10	9 pole	CNR-1 signal	
Plug 11	9 pole	Thermistor string	
Plug 12	7 pole	NT1400	
Plug 13	7 pole	Tasp2 Signal	
Plug 14	7 pole	Tasp2 Fan	
Gland	PG11	Iridium antenna	
Internal		Barometer	
Internal		Iridium modem	
Internal		GPS antenna to modem	
ZAK_S1 only wiring:			
Internal		Garmin GPS	

	Zak_M	ZAK_S
4 pole	2 male	2 male
7 pole	9 female	4 female
9 pole	2 female	0
Total	14 + PG11	6
On battery box:		
4 pole	1 female	1 female

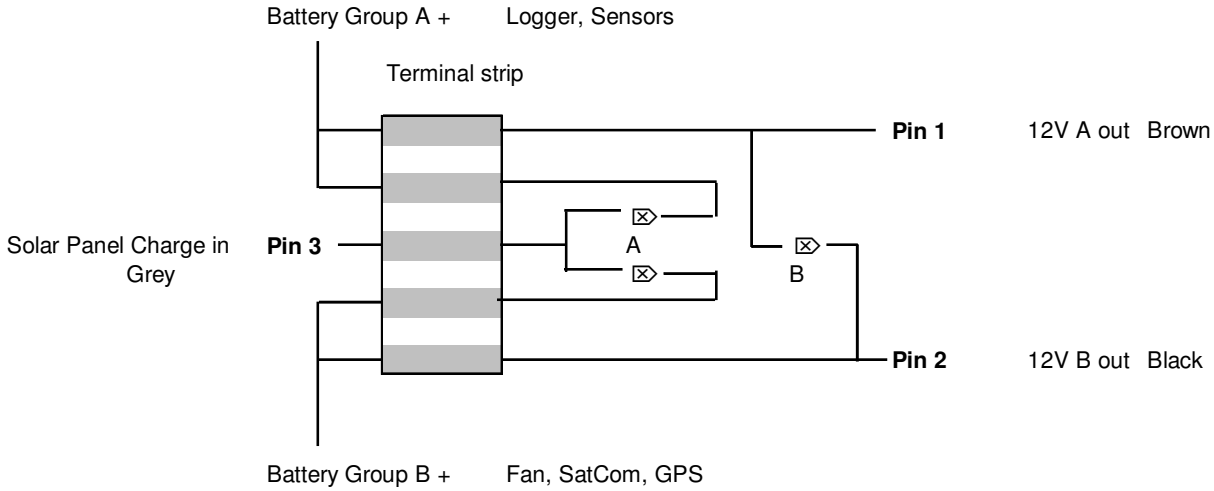
SN for the ZAK_M amd ZAK_S1 stations:

ZAK_M:	CR1000	E1350
	CFM100	3355
	AS16/32	E3903
	CS100	3440 409
ZAK_S1:	CR1000	E1349
	CFM100	2489
	Garmin 16-HVS	3333 4963

Plug numbers and positions for ZAK MAIN AWS
ZAK SECONDARY AWS only uses plug 1 - 6



Unmarked plugs are femal 7 pole.



A:	2 Diodes 1N5450 (RS Comp 348-5460)
B:	Diode 80SQ045 (RS Comp 254-0730)

Pin G: All negative battery terminals Blue

Plug: 4 pole female

PLUGS 1-6 ZAK_M + ZAK_S1

Plug no.	Pin no.	Sensor cable	Function	Inside cable	Connected to IN MAIN	Connected to IN SEC
Plug 1	⌀	Blue	Ground	Black 1mm²	Interface 1 no 16 (Power meas.)	Interface 1 no 12
Batterycable	1	Brown	12 V A in	Red 1mm²	Cr1000 +12V	Cr1000 +12V
Black cable	2	Black	12 V B in	Red 1mm²	Interface 3 no 4	Interface 2 no 4
1,5 mm²	3	Grey	12 V charge out	Grey 1,5mm²	Plug 2 pin 1	Plug 2 pin 1
Plug 2	⌀	Blue	Ground	Black 1mm²	Interface 1 no 12 (Power meas.)	Interface no 8
Solar panel	1	Brown	Power +	Grey 1,5mm²	Plug 1 pin 3	Plug 1 pin 3
Blue cable	2		nc			
0,75 mm²	3		nc			
Plug 3	⌀	Shield	Shield	White	MUX1G	G (EX1)
WS&D	1	Black	WS reference	Orange	G (P1)	G (P1)
Sensor cable	2	White	AZ reference	Black	MUX2G	G8
	3	Green	AZ signal	Blue	MUX1L	SE8
	4	Blue	AZ exitation	Yellow	EX1	EX1
	5	Red	WS signal	Green	P1	P1
	6		nc	Brown	nc	nc
Plug 4	⌀	Drain	Shield	White	nc	nc
Tasp A	1	Red	Probe Supply	Orange	Interface 2 no 10	Interface 1 no 16
Hygroclip +	2	Black	GND	Black	G6	G6
Cable comp.	3	Blue	nc	Blue	nc	nc
	4	Yellow	Cable comp.	Yellow	G14	G14
9721 cable	5	Green	RH signal	Green	MUX21H	SE5
	6	White	T signal	Brown	MUX21L	SE6
Plug 5	⌀	Drain	Shield	White	G4	G4
Tasp B	1	Red	Fan Supply	Orange	Interface 3 no 1	Interface 2 no 1
Fan + Pt100	2	Black	Fan GND	Black	Interface 1 no 8	Interface 1 no 4
	3	Blue	Pt100 Ex+	Blue	SE2L (4WPB100 L)	SE2L (4WPB100 L)
	4	Yellow	Pt100 Ex-	Yellow	G1 (4WPB100 G)	G1 (4WPB100 G)
9721 cable	5	Green	Pt100 Signal+	Green	SE3	SE3
	6	White	Pt100 Signal-	Brown	SE4	SE4
Plug 6	⌀	Clear	Shield	White	G (EX2)	G (EX2)
SR50 Abl	1	Black	Power ground	Orange	G (C8)	G (C8)
Sensor cable	2	Red	12 V	Black	Interface 2 no 13	Interface 1 no 18
Adress 1	3	Green	SDI-12 databus	Blue	C7	C7
	4	White	Not used	Yellow	G (C8)	G (SW12V)
	5		nc	Green	nc	nc
	6		nc	Brown	nc	nc
Extension cable for SR50 Abl:			15 m 9721	**3 Campbell 4-wire Terminal Input Modules**		
	⌀	Drain		4WPB100	DIFF1 - Black EX2 (Pt100 Tasp 1)	
Male plug	1	Red	Female plug	ZAK_M only:		
	2	Black		4WPB100	DIFF3 - Black EX2 (Pt100 CRN-1)	
	3	Blue		4WPB100	DIFF5 - Black EX2 (Pt100 Tasp 2)	
	4	Yellow				
	5	Green				
	6	White				

Rotronic Tasp and RH wiring:

Function	MP100H Col	Terminal box	Cable Colour	Plug	Pin	Internal Colour
Probe Supply	Green	1	Red	4	1	Orange
RH Signal	White	2	Green	4	5	Green
T Signal	Brown	3	White	4	6	Brown
Cable Comp.	Yellow	4	Yellow	4	4	Yellow
GND	Grey	5	Black	4	2	Black
Digital I/O	nc	6	Blue (nc)	4	3	Blue (nc)
			Drain	4	⌀	White
Pt100 Ex+	Red	7	Blue	5	3	Blue
Pt100 Signal+	Pink	8	Green	5	5	Green
Pt100 Signal-	Black	9	White	5	6	Brown
Pt100 Ex-	Violet	10	Yellow	5	4	Yellow
Fan Supply	Fan Brown	11	Red	5	1	Orange
Fan GND	Fan Blue	12	Black	5	2	Black
Shield	nc		Drain	5	⌀	White
	Sensor cable		9721 cables			In enclosure

PLUGS 7-14: ZAK_M only

Plug no.	Pin no.	Sensor cable	Function	Inside cable	Connected to
Plug 7	⌀	Black	Ground	White	MUX11G
Inclinometer	1	Red	+7…24 V	Orange	Interface 2 no 16
	2		nc	Black	nc
9720/1 cable	3	White	Xout	Blue	MUX11H
	4	Green	Yout	Yellow	MUX11L
	5		nc	Green	nc
	6		nc	Brown	nc
Plug 8	⌀	Clear	Shield	White	G (EX2)
SR50 Snow	1	Black	Power ground	Orange	G (C8)
	2	Red	12 V	Black	Interface 2 no 15
Adress 0	3	Green	SDI-12 databus	Blue	C7
	4	White	nc	Yellow	nc
	5		nc	Green	nc
	6		nc	Brown	nc
Plug 9	⌀	Black	Shield	White	G8
CNR-1 temp.	1		nc	Orange	nc
	2	Blue	Pt100 exitation -	Black	G6 (4WPB100 G)
Sensor cable	3	Red	Pt100 exitation +	Blue	SE6 (4WPB100 L)
	4	Yellow	Pt100 signal +	Yellow	DIFF4H
	5	Green	Pr100 signal -	Green	DIFF4L
	6		nc	Brown	nc
Plug 10	1	Red	CM3 up signal	Orange	MUX13H
CNR-1 signal	2	Blue	CM3 up ref	Black	MUX13L
	3	White	CM3 down signal	Blue	MUX17H
Sensor cable	4	Black	CM3 down ref	Yellow	MUX17L
	5	Grey	CG3 up signal	Green	MUX15H
	6	Yellow	CG3 up ref	Brown	MUX15L
	7	Brown	CG3 down signal	Pink	MUX19H
	8	Green	CG3 down ref	Violet	MUX19L
	9	Shield	Shield	White	MUX16G
Plug 11	1	Orange	Signal level 1	Orange	MUX4H
Thermistor	2	Black	Signal level 2	Black	MUX4L
string	3	Blue	Signal level 3	Blue	MUX6H
	4	Yellow	Signal level 4	Yellow	MUX6L
Cable :	5	Green	Signal level 5	Green	MUX8H
Farnell	6	Brown	Signal level 6	Brown	MUX8L
123-5597	7	Pink	Signal level 7	Pink	MUX10H
	8	Violet	Signal level 8	Violet	MUX10L
	9	White	Exitation	White	EX3
Plug 12	⌀	Shield	Shield	White	
NT1400	1	White	VS -	Orange	MUX25G
	2	Red	VS +	Black	EX3
Sensor cable	3		nc	Blue	nc
	4		nc	Yellow	nc
	5	Blue	VO -	Green	MUX25L
	6	Yellow	VO +	Brown	MUX25H
Plug 13	⌀	Drain	Shield	White	nc
Tasp A	1	Red	Probe Supply	Orange	Interface 2 no 12
Hygroclip +	2	Black	GND	Black	MUX23G
Cable comp.	3	Blue	nc	Blue	nc
	4	Yellow	Cable comp.	Yellow	G14
9721 cable	5	Green	RH signal	Green	MUX23H
	6	White	T signal	Brown	MUX23L
Plug 14	⌀	Drain	Shield	White	G12
Tasp B	1	Red	Fan Supply	Orange	nc
Fan + Pt100	2	Black	Fan GND	Black	nc
(Fan nc)	3	Blue	Pt100 Ex+	Blue	SE10L (4WPB100 L)
	4	Yellow	Pt100 Ex-	Yellow	G10 (4WPB100 G)
9721 cable	5	Green	Pt100 Signal+	Green	SE11
	6	White	Pt100 Signal-	Brown	SE12

Internal ZAK_M

		Function	Inside cable	Connected to:
Internal	Pin 1	Ext. Trigger	Green	Interface 2 no 18
Barometer	Pin 2	nc		
	Pin 3	GND	Black	G (5V)
	Pin 4	12V	Orange	Interface 2 no 14
	Pin 5	VOUT	Yellow	MUX1H
Internal	RS232	1	Brown	nc
Iridium modem		2	Red	CR1000 RS232 Pin 3
		3	Orange	CR1000 RS232 Pin 2
	NAL female	4	Yellow	CR1000 RS232 Pin 6
	changed to	5	Green	CR1000 RS232 Pin 5
	RS232 male	6	Blue	CR1000 RS232 Pin 4
		7	Violet	CR1000 RS232 Pin 8
		8	Grey	CR1000 RS232 Pin 7
		9	Black	nc
	Red	+		Interface 3 no 7
	Black	GND		Interface 1 no 12
	SMA Gold	Iridium antenna		Antenna cable
	SMA Silver	GPS antenna		GPS in enclosure
Iridium anten.	Cable	RG58: RS Comp 638-9472		
	Gland	PG11: RS Comp 444-2622		
	TNC male	RS Comp 295-8226		Antenna
	SMA male	RS Comp 5120115		Modem SMA Gold
Interface 1	Power measurement			
MUX9H	2 Black		1 Orange	MUX9L
Modem G Black	4 Black	5 mohm	3	nc
MUX7H	6 Yellow		5 Black	MUX7L
Fan G Plug 5 pin 2	8 Black	5 mohm	7	nc
MUX5L	10 Green		9 Brown	MUX5H
Solar Panel G Plug 2 pin ß	12 Black	5 mohm	11	nc
MUX3L	14 White		13 Yellow	MUX3H
Battery G Plug 1 pin ß	16 Black	5 mohm	15	nc
nc	18 Green/Yellow		17 Black	POWER IN G
Interface 2	Resistors for thermistorstring measurement, SW12 V			Twisted pairs ? Next time !
CR1000 SE15	1 Blue	249 kohm	2 Orange	MUX EVEN H
nc	3	1 kohm	4	nc
CR1000 SE16	5 Yellow	1 kohm	6 Black	G16
nc	7	249 kohm	8 Orange	MUX EVEN L
SW12V (from CR1000)	9 Red		10 Orange	Tasp1 12V Plug 4 pin 1
MUX 12V	11 Orange		12 Orange	Tasp2 12V Plug 13 pin 1
SR50 Abl Plug 6 pin 2	13 Black		14 Orange	Barometer Pin 4 (12V)
SR50 Snow Plug 8 pin 2	15 Black		16 Orange	Incl. Plug 7 pin 1
nc	17		18 Green	Barometer Pin 1 (Ext. Trigger)
Interface 3	Fan Supply, Modem supply			
GND (12V) (Black)	2	IP521 Fan Supply	1	12V B OUT Fan Plug 5 pin 1 (Orange)
12V B IN Plug 1 pin 2 (Red)	4		3	NC
C8 (Yellow)	6		5	NC
GND To 2 (Black)	8	IP521 Modem supply	7	12V B OUT Modem SYN-DC + (Red)
12V B IN To 4 (Orange)	10		9	NC
C6 (Blue)	12		11	NC
NC	14		13	NC
NC	16		15	NC
NC	18		17	NC
Internal connections:	**CR1000**	**Colour**	**MUX**	**Function**
	SE13	Orange	ODD H	Signals to CR1000
	SE14	Yellow	ODD L	do.
	G14	Black	G COMMON	do.
	G (SW12V)	Black	GND	
	C1	Blue	MUX RES	Reset
	C2	Brown	MUX CLK	Clock
	POWER IN G	Black 1 mm²	Interface 1 no 17	
	POWER IN 12V	Red 1mm²	Plug 1 pin 1	
Cathode (Short leg)	G (C5)	Diode RS Comp 564009	C5	Anode (Most metal)
Main Ground:	Enclosure Ground Lug - CR1000 Ground Lug - AM16/32 Ground Lug			
	All connections 2,5 mm² green/yellow			

Internal ZAK_S1

		Function	Connected to:	
Internal	Red	Power 12 V	Relay 2 no 14	
Garmin GPS	Black	Power ground	G (12V)	
	Yellow	Remote On/off	G (12V)	
	Blue	Port 1 data in	nc	
	White	Port 1 data out	C4	
	Grey	PPS	nc	
	Green	Port 2 data in	nc	
	Violet	Port 2 data out	nc	
	Drain	Shield	G (12V)	
Interface 1	Power measurement			
SE13 Yellow	2	5 mohm	1	SE14 White
Fan G Plug 5 pin 2 (Black)	4		3	nc
SE12 Green	6	5 mohm	5	SE11 Blue
Solar Panel G Plug 2 pin	8		7	nc
SE10 White	10		9	SE9 Yellow
Battery G Plug 1 pin	12	5 mohm	11	CR1000 GROUND LUG
nc	14		13	G BOLT Enclosure
H.Clip Plug 4 pin 1 Orange	16		15	nc
SR50 Abl Plug 6 pin 2 Black	18		17	SW12v from CR1000 (Red 1 mm²)
Interface 2: Fan, GPS supply:				
GND (12V) (Black)	2	IP521 Fan Supply	1	12V B OUT Fan Plug 5 pin 1 (Orange)
12V B IN Plug 1 pin 2 (Red)	4		3	NC
C8 (Yellow)	6		5	NC
GND To 2 (Black)	8	IP521 Modem supply	7	12V B OUT GARMIN (Red)
12V B IN To 4 (Orange)	10		9	NC
C6 (Blue)	12		11	NC
NC	14		13	NC
NC	16		15	NC
NC	18		17	NC
	POWER IN G		nc	
	POWER IN	Red 1mm²	Plug 1 pin 1	
Cathode (Short leg)	G (C5)	Diode RS Comp 564009	C5	Anode (Most metal)

Appendix D – Datalogger program

(Michele Citterio, GEUS)

```
'CR1000 Program for the GlacioBasis GEUS AWSs rev. 1.4M (23/03/2008).
'Written by Michele Citterio, GEUS, Copenhagen.


SequentialMode
'*************************************************************************
'******** WHEN CONFIGURING A NEW ZACK-STYLE AWS, YOU DON'T NEED TO EDIT ANYTHING
'*************************************************************************
Const AppendInstantaneous = True
Const HoursInstantaneous = 3' in hours 3
Const RemoteReconfig = False'unfinished, don't set True
Const BeginOfSummer = 90' in days
Const BeginOfWinter = 305' in days
Const ScanRate = 600'600 ' in sec (can't be shorter than 15 + 2*TxTimeout + 60)
Const MemSaveRate = 1'2 'in numebr of scans. This is both the averaging and the
Const SummerTxRate = 6'1 'in numebr of scans. This is both the averaging and the
Const WinterTxRate = 18'4 'in numebr of scans (=1440). This is both the averagin
Const SummerGPSrate = 3'1 'in number of scans (in this revision it must be an in
Const WinterGPSrate = 18'1 'in number of scans (in this revision it must be an i
Const SummerTableBuffer = 72
Const WinterTableBuffer = 24
Const DiagnosticsRate = 24'(=36) in hours
Const TermStringSz = 8' no. of thermistors in the thermistor sting
Const LoBattThre = 11.5' the threshold in V for Low_Batt to be set (entering pow
Const LoBattResHyst = 0.5 'the hysteresis in V above LoBattThre for resetting Lo
Const FastScanReps = 10 'how many times to run Measure routine in fastscan mode
Const FastScanRate = 20 'how often to run the Measure routine when in fastscan m
Const GPSType = "N" 'use 'G' for Garmin, 'N' for the one integrated in the NAL I
Const AdvanceFanStart = 2 'in number of FastScans. How much in advance of the sc
Const AdvanceGPSstart = 120'20'in sec. Increase!!
Const TxStringMaxLen = 420 'may be increased with future releases of the Iridium
Const TxMaxInitAttempts = 3' max 3, more doesn't make any sense and the wasted t
Const TxTimeout = 30 ' in sec, before the Tx sub times out. If service becomes a
Const SnowheightSR50address = 1' SDI12 address
Const AblationSR50address = 0' SDI12 address
Const GarminWarmup = 3' in seconds (doesn't mean it will then output a fix, but
Const SR50Warmup = 3' in seconds
Const HygroClipWarmup = 4' warm-up time needed by the HygroClip module after pow
Const NT1400Warmup = 5' time required for the pressure transmitter before it is
Const GarminBaudRate = 1200' Damn Garmin!!!!! Some units come preset at 1200 bau
'*************************************************************************
'******** WHEN CONFIGURING A NEW ZACK-STYLE AWS, YOU DON'T NEED TO EDIT ANYTHING
'*************************************************************************

Public WinterDataTerminator As String *3
Public SummerDataTerminator As String *3
Public InstantDataTerminator As String *3
Public DiagnDataTermninator As String *3
Public TxAllowed As Boolean
Public FanPowerDueForSwOff As Boolean
Public TableInstantaneousString As String * 100
Public TxPowerDueForSwOff As Boolean
Public GarminInitOk As Boolean
Public ConfigFile_H As Long
Public ConfigArrayIT
'Public DiagInfieldWarning As Boolean'check for it not being set before field de
Public NGPSFLAG As Boolean
Public GGPSFLAG As Boolean
Public LoggerOSrev
Public TimersUnit
Public TimersUnitMult
Public Diag
```

```
Public SkippedScans
Public SkippedSlowScans1, SkippedSlowScans2
Public GPSinUse As String * 1
Public TimeSince12Von
Public TimeSinceVx3on
Public LastSuccessfulMOMSN
Public SendRecordBacklog
Public SendRecordIT
Public TxTimeTaken
Public TxSendAttempts
Public TxTimeoutTimer
Public TxInitilizeAttempt
Public SBDIoffset
Public SBDIsentence As String * 40
Public SBDIparsed(6) As String *10
Public SBDsession As Boolean
Public TxInitializeOk As Boolean
Public StrIndex
Public TxSendSuccess As Boolean
Public SentWithSuccess
Public TxMObuffCleared As Boolean
Public TxMOuploaded As Boolean
Public TxRSSI' As String
Public TxSvcAvail As Boolean
Public TxNotifyON As Boolean
Public RS232EchoOk As Boolean
Public TxTypeNAL960x As Boolean
Public TxModuleATok As Boolean
Public TxModuleRFon As Boolean
Public TxModuleReply As String * 41
'Public TxSerialBuffer As String * 401'set as needed
'Public TxPowerOn As Boolean
Public TxModuleGPSPowerOn As Boolean
Public TxBufferString As String * TxStringMaxLen + 1 +80'to hold the null termin
Public TxStringUnstripped As String * TxStringMaxLen + 1 +80'sorry, there's some
Public TxString As String * TxStringMaxLen + 1'to hold the null termination
Public StringStripperIT
Public StripStringStart
Public StringChar As String * 1
Public MeasCycleDone As Boolean
Public C5status' As Boolean
Public C5statusPrev' As Boolean
Public LastFastScan
Public temp
Public Currents(4)
Alias Currents(1) = BatteryCurrent
Alias Currents(2) = SolarPanelCurrent
Alias Currents(3) = FanCurrent
Alias Currents(4) = TxGPSCurrent
Public MPFLAG As Boolean
Public FastScanSwitch As Boolean
Public FastScanCount
Public FastScanMode As Boolean
Public RTime(9) 'WARNING! this is used as a temporary variable in many different
Alias rTime(1) = Year
Alias rTime(2) = Month
Alias rTime(3) = DayOfMonth
Alias rTime(4) = Hour
Alias rTime(5) = Minute
Alias rTime(6) = Second
Alias rTime(7) = uSecond
Alias rTime(8) = WeekDay
Alias rTime(9) = DayOfYear
```

```
Public IsSummer As Boolean
Public Batt_VoltI
Public Batt_VoltF
Public Batt_V_Drop
Public Low_Batt As Boolean
Public Asp_fan_on As Boolean
Public Aspirated_meas As Boolean
Public PTemp_C
Public SR50_SnowHeight(2)
Alias SR50_SnowHeight(1)=SnowHeight  ''sr50
Alias SR50_SnowHeight(2)=SnowHeightQuality  ''sr50
Public SR50_Ablation(2)
Alias SR50_Ablation(1)=Ablation ''sr50
Alias SR50_Ablation(2)=AblationQuality
Public Ablation_meter  ''sr50

Public temptemp,temptemptemp
'Public WaterColumnPressure
Public WS_ms      ''Young
Public WindDir    ''Young
Public BP_mmHg
Public AS_T    'Air temperature, ventilated shield
Public AS_RH   'Relative humidity, ventilated shield
Public AS_Pt100'PT100 air temperature, ventilated shield
Public US_T    'Air temperature, ventilated shield
Public US_RH   'Relative humidity, ventilated shield
Public US_Pt100'PT100 air temperature, ventilated shield
Public temp1 ''remove then
Public Xtilt
Public Ytilt
Public TermStringT(8)
Public CNR1_SWin
Units CNR1_SWin = mV
Public CNR1_SWout
Public CNR1_LWin
Public CNR1_LWout
Public CNR1_Pt100
Dim SW12Von As Boolean
Public _MUXRES
Dim MUXCLK As Boolean
Dim it
'GPS stuff
'Public LatL As Long
'Public SaveGPSFix
Public BestHDPseen
Public sBestHDPseen
'Public GPGGAoffset
'Public GPSData As String * 301
Public GPGGAsentence As String * 72
Public PreParseStr(18) As String * 15
Alias PreParseStr(7) = PreQual
Alias PreParseStr(9) = PreHDP
Public PreQual_FP
Public PreHDP_FP
Public ParseStrIT
Public ParseStr(18) As String * 15
Alias ParseStr(1) = GPGGA
Alias ParseStr(2) = TIME
Alias ParseStr(3) = LAT
Alias ParseStr(4) = HEMINS
Alias ParseStr(5) = LONGI
Alias ParseStr(6) = HEMIEW
Alias ParseStr(7) = QUAL
```

```
Alias ParseStr(8) = NUMSATS
Alias ParseStr(9) = HDP
Alias ParseStr(10) = ALTDE
Alias ParseStr(11) = ALTUNIT
Alias ParseStr(12) = GIODAL
Alias ParseStr(13) = GEOUNIT
Alias ParseStr(14) = AGE
Alias ParseStr(15) = DIFFREF
Alias ParseStr(16) = ASTERISK
Alias ParseStr(17) = CHCKSUM
Alias ParseStr(18) = CRLF

Units Batt_VoltI=Volts
Units Batt_VoltF=Volts
Units Batt_V_Drop=Volts
Units PTemp_C=Deg C
Units WS_ms=meters/second
Units WindDir=Degrees
Units BP_mmHg=mV (mmHg)
'Units WaterColumnPressure=mV/V
Units AS_T=mV
Units AS_RH=mV
Units AS_Pt100=mV?
Units US_T=mV
Units US_RH=mV
Units US_Pt100=mV?
Units Xtilt=mV
Units Ytilt=mV
Units TermStringT(8)=Deg C
Units CNR1_SWin=mV
Units CNR1_SWin =mV
Units CNR1_SWout=mV
Units CNR1_LWin=mV
Units CNR1_LWout=mV
Units CNR1_Pt100=mV


'Define Data Tables
DataTable (InstantaneousTable,True,1)
Sample(1,BP_mmHg,FP2)
Sample(1,AS_Pt100,FP2)
Sample(1,AS_T,FP2)
Sample(1,AS_RH,FP2)
WindVector (1,WS_ms,WindDir,FP2,False,0,0,0)
Sample(1,InstantDataTerminator,String)
EndTable

DataTable(TxBufferTable,True,SummerTableBuffer)
Sample(1,TxBufferString, String)
EndTable

DataTable(TableSummerTx,True,1,)'SummerTableBuffer)'then, size = 1
DataInterval(0,SummerTxRate*ScanRate,Sec,1)
Average(1,BP_mmHg,FP2,False)
' StdDev(1,BP_mmHg,FP2,False)
Average(1,AS_Pt100,FP2,False)
' StdDev(1,AS_Pt100,FP2,False)
Average(1,AS_T,FP2,False)'AS_ = Rotronics Aspirated Shield
' StdDev(1,AS_T,FP2,False)
Average(1,AS_RH,FP2,False)' This is conceptually wrong
' StdDev(1,AS_RH,FP2,false)' This is conceptually wrong
WindVector (1,WS_ms,WindDir,FP2,False,0,0,0)
FieldNames("WS_ms_S_WVT,WindDir_D1_WVT,WindDir_SD1_WVT")'From manual:"Mean hori
```

```
Average(1,CNR1_SWin,FP2,False)
' StdDev(1,CNR1_SWin,FP2,False)
Average(1,CNR1_SWout,FP2,False)
' StdDev(1,CNR1_SWout,FP2,False)
Average(1,CNR1_LWin,FP2,False)
' StdDev(1,CNR1_LWin,FP2,False)
Average(1,CNR1_LWout,FP2,False)
' StdDev(1,CNR1_LWout,FP2,False)
Average(1,CNR1_Pt100,FP2,False)
' StdDev(1,CNR1_Pt100,FP2,False)
Sample(1,SnowHeight,FP2)
Sample(1,SnowHeightQuality,FP2)
Sample(1,Ablation,FP2)
Sample(1,AblationQuality,FP2)
Average(1,Ablation_meter,FP2,False)
' StdDev(1,Ablation_meter,FP2,False)
Average(8,TermStringT(),FP2,False)''check if ok this way of passing arrays
FieldNames("Thermistor_1,Thermistor_2,Thermistor_3,Thermistor_4,Thermistor_5,Th
Average(1,Xtilt,FP2,False)
' StdDev(1,Xtilt,FP2,False)
Average(1,Ytilt,FP2,False)
' StdDev(1,Ytilt,FP2,False)
Sample(1,TIME,String)
Sample(1,LAT,String)
Sample(1,LONGI,String)
Sample(1,ALTDE,String)
Sample(1,NUMSATS,String)
Sample(1,HDP,String)
Average(1,currents(3),FP2,False)
Average(1,US_Pt100,FP2,False)
' StdDev(1,US_Pt100,FP2,False)
Average(1,US_T,FP2,False)'US_ = Rotronics Un-Aspirated Shield
' StdDev(1,US_T,FP2,False)
Average(1,US_RH,FP2,False)
' StdDev(1,US_RH,FP2,false)
Maximum(1,Batt_VoltI,FP2,False,False)
Minimum(1,Batt_VoltF,FP2,False,False)
Average(1,Batt_V_Drop,FP2,False)
Sample(1,SummerDataTerminator,String)
EndTable

DataTable(TableWinterTx,True,1)'WinterTableBuffer)'then, size = 1
DataInterval(0,WinterTxRate*ScanRate,Sec,1)
Average(1,BP_mmHg,FP2,False)
' StdDev(1,BP_mmHg,FP2,False)
Average(1,AS_Pt100,FP2,False)
' StdDev(1,AS_Pt100,FP2,False)
Average(1,AS_T,FP2,False)'AS_ = Rotronics Aspirated Shield
' StdDev(1,AS_T,FP2,False)
Average(1,AS_RH,FP2,False)' This is conceptually wrong
' StdDev(1,AS_RH,FP2,false)' This is conceptually wrong
WindVector (1,WS_ms,WindDir,FP2,False,0,0,0)
FieldNames("WS_ms_S_WVT,WindDir_D1_WVT,WindDir_SD1_WVT")'From manual:"Mean hori
Average(1,CNR1_SWin,FP2,False)
' StdDev(1,CNR1_SWin,FP2,False)
Average(1,CNR1_SWout,FP2,False)
' StdDev(1,CNR1_SWout,FP2,False)
Average(1,CNR1_LWin,FP2,False)
' StdDev(1,CNR1_LWin,FP2,False)
Average(1,CNR1_LWout,FP2,False)
' StdDev(1,CNR1_LWout,FP2,False)
Average(1,CNR1_Pt100,FP2,False)
' StdDev(1,CNR1_Pt100,FP2,False)
```

```
Sample(1,SnowHeight,FP2)
Sample(1,SnowHeightQuality,FP2)
Sample(1,Ablation,FP2)
Sample(1,AblationQuality,FP2)
Average(1,Ablation_meter,FP2,False)
' StdDev(1,Ablation_meter,FP2,False)
Average(8,TermStringT(),FP2,False)''check if ok this way of passing arrays
FieldNames("Thermistor_1,Thermistor_2,Thermistor_3,Thermistor_4,Thermistor_5,Th
Average(1,Xtilt,FP2,False)
' StdDev(1,Xtilt,FP2,False)
Average(1,Ytilt,FP2,False)
' StdDev(1,Ytilt,FP2,False)
Sample(1,TIME,String)
Sample(1,LAT,String)
Sample(1,LONGI,String)
Sample(1,ALTDE,String)
Sample(1,NUMSATS,String)
Sample(1,HDP,String)
Average(1,currents(3),FP2,False)
Average(1,US_Pt100,FP2,False)
' StdDev(1,US_Pt100,FP2,False)
Average(1,US_T,FP2,False)'US_ = Rotronics UN-Aspirated Shield
' StdDev(1,US_T,FP2,False)
Average(1,US_RH,FP2,False)
' StdDev(1,US_RH,FP2,false)
Maximum(1,Batt_VoltI,FP2,False,False)
Minimum(1,Batt_VoltF,FP2,False,False)
Average(1,Batt_V_Drop,FP2,False)
Sample(1,WinterDataTerminator,String)
EndTable

DataTable(TableMem,True,-1)
DataInterval(0,MemSaveRate*ScanRate,Sec,100)'are those many lapses needed?
CardOut(0,-1)
Average(1,BP_mmHg,IEEE4,False)
Average(1,AS_Pt100,IEEE4,False)
Average(1,AS_T,IEEE4,False)'AS_ = Rotronics Aspirated Shield
Average(1,AS_RH,IEEE4,False)
WindVector(1,WS_ms,WindDir,FP2,False,0,0,0)
FieldNames("WS_ms_S_WVT,WindDir_D1_WVT,WindDir_SD1_WVT")'From manual:"Mean hori
Average(1,CNR1_SWin,IEEE4,False)
Average(1,CNR1_SWout,IEEE4,False)
Average(1,CNR1_LWin,IEEE4,False)
Average(1,CNR1_LWout,IEEE4,False)
Average(1,CNR1_Pt100,IEEE4,False)
Sample(1,SnowHeight,FP2)
Sample(1,SnowHeightQuality,FP2)
Sample(1,Ablation,FP2)
Sample(1,AblationQuality,FP2)
Average(1,Ablation_meter,FP2,False)
Average(8,TermStringT(),IEEE4,False)''check if ok this way of passing arrays
FieldNames("Thermistor_1,Thermistor_2,Thermistor_3,Thermistor_4,Thermistor_5,Th
Average(1,Xtilt,FP2,False)
Average(1,Ytilt,FP2,False)
Sample(1,TIME,String)
Sample(1,LAT,String)
Sample(1,HEMINS,String)
Sample(1,LONGI,String)
Sample(1,HEMIEW,String)
Sample(1,ALTDE,String)
Sample(1,ALTUNIT,String)
Sample(1,GIODAL,String)
Sample(1,GEOUNIT,String)
```

```
Sample(1,QUAL,String)
Sample(1,NUMSATS,String)
Sample(1,HDP,String)
Average(1,US_Pt100,IEEE4,False)
Average(1,US_T,IEEE4,False)'US_ = Rotronics UN-Aspirated Shield
Average(1,US_RH,IEEE4,False)
Average(1,PTemp_C,IEEE4,False)
Maximum(1,Batt_VoltI,IEEE4,False,False)
Minimum(1,Batt_VoltF,IEEE4,False,False)
Average(4,Currents(),FP2,False)
FieldNames("Battery_current_avg,Solar_Panel_current_avg,Fan_current_avg,Tx+GPS_
EndTable

DataTable(TableDiagnostics, True, -1)
DataInterval(0,DiagnosticsRate,hr,100)
CardOut(0,-1)
Sample(1,status.StationName,String)
Sample(1,status.StartTime,NSEC)
Sample(1,status.RunSignature,uint2)
Sample(1,LoggerOSrev,String)
Sample(1,status.Low12VCount,Uint2)
Average(1,PTemp_C,FP2,False)
StdDev(1,PTemp_C,FP2,False)
Average(1,Batt_VoltI,FP2,False)
Average(1,Batt_VoltF,FP2,False)
Average(1,Batt_V_Drop,FP2,False)
Maximum(1,Batt_VoltI,FP2,False,False)
Minimum(1,Batt_VoltF,FP2,False,False)
Average(4,Currents(),FP2,False)
FieldNames("Battery_current_avg,Solar_Panel_current_avg,Fan_current_avg,Tx+GPS_
StdDev(4,Currents(),FP2,False)
FieldNames("Battery_current_std,Solar_Panel_current_std,Fan_current_std,Tx+GPS_
Totalize(1,TxInitializeOk,FP2,False)
Totalize(1,GarminInitOk,FP2,False)
Sample(1,SentWithSuccess,FP2)
Sample(1,SendRecordBacklog,FP2)
Sample(1,DiagnDataTermninator,String)
EndTable


'Declaration of subroutines

Sub TxModuleInit
If NOT (SummerTxRate OR WinterTxRate OR (GPSType="N")) Then ExitSub
TxInitializeOk = False
If NOT TxInitializeOk Then
SerialOpen(ComRS232,19200,0,0,400)
SerialOut(ComRS232,"A","",25,20)' to allow 5 secs for the Iridium Module to
For TxInitilizeAttempt = 1 To TxMaxInitAttempts
'        TxInitializeOk = False
'        ExitFor
'      EndIf'
SerialOut(ComRS232,"TE1"+CHR(13)+CHR(10),"",1,10) 'should be default but d
SerialOut(ComRS232,"ATV1"+CHR(13)+CHR(10),"",1,10)' FIXME!! Then RS232Echo
If SerialOut(ComRS232,"AT"+CHR(13)+CHR(10),"OK"+CHR(13)+CHR(10),1,10) Then
If SerialOut(ComRS232,"ATI4"+CHR(13)+CHR(10),"IRIDIUM 9600 Family"+CHR(13)
'Delay(0,1,sec)'\RREEMMOOVVEE!!!!
'SerialFlush(ComRS232)'needed to clean up the rest of the unused answer to
If TxModuleATok AND True AND TxTypeNAL960x Then 'FIXME!! RS232EchoOk
TxInitializeOk = True
ExitFor
Else
TxInitializeOk = False
```

```
EndIf
Next
SerialClose(ComRS232)
EndIf
EndSub


Sub IridiumTx
If NOT (SummerTxRate OR WinterTxRate OR (GPSType="N")) Then ExitSub
Call TxModuleInit
If NOT TxInitializeOk Then
SerialClose(ComRS232)
ExitSub
EndIf

SerialOpen(ComRS232,19200,0,0,400)
If SerialOut(ComRS232,"AT*S=1"+CHR(13)+CHR(10),"OK"+CHR(13)+CHR(10),1,10) Then
If SerialOut(ComRS232,"AT+SBDD0"+CHR(13)+CHR(10),"0",1,10) Then TxMObuffCleare
If SerialOut(ComRS232,"AT+SBDWT"+CHR(13)+CHR(10),"READY" + CHR(13) + CHR(10),
SerialOutBlock (ComRS232,TxString+CHR(13),Len(TxString)+10)
If SerialOut(ComRS232,"","0"+CHR(13)+CHR(10),1,100) Then TxMOuploaded = True
EndIf

If NOT (TxMObuffCleared AND TxMOuploaded AND TxModuleRFon) Then
SerialClose(ComRS232)
ExitSub
EndIf

Timer(1,sec,2)
Do
'\GPc1 = GPc1+1
TxTimeoutTimer = Timer(1,sec,4)
If TxTimeoutTimer >= TxTimeout Then 'the = is needed because the timer has a
Timer(1,sec,3)
SerialClose(ComRS232)
ExitSub
EndIf
SerialFlush(ComRS232)
If SerialOut(ComRS232,"AT+CIER=1,1,1,0"+CHR(13)+CHR(10),"OK"+CHR(13)+CHR(10)
Do
TxTimeoutTimer = Timer(1,sec,4)
If TxTimeoutTimer >= TxTimeout Then 'the = is needed because the timer has
Timer(1,sec,3)
SerialClose(ComRS232)
ExitSub
EndIf
SerialIn(TxModuleReply,ComRS232, TxTimeout*100, CHR(13)+CHR(10),15)' put h
If InStr(1,TxModuleReply,"+CIEV:0,",4) Then TxRSSI = Mid(TxModuleReply, In
If InStr(1,TxModuleReply,"+CIEV:1,",4) Then TxSvcAvail = Mid(TxModuleReply
Loop Until TxSvcAvail AND TxRSSI > 0
If SerialOut(ComRS232,"AT+CIER=0,1,1,0"+CHR(13)+CHR(10),"OK"+CHR(13)+CHR(10)
SerialFlush(ComRS232)
If NOT TxAllowed Then ExitDo
TxSendAttempts=TxSendAttempts+1
SBDsession = True'\
SerialOut(ComRS232,"AT+SBDI"+CHR(13),"",1,10)
SerialOut(ComRS232,"","+SBDI:",1,6000)' WAIT up to 1 minute for the outcome
SerialIn(TxModuleReply,ComRS232, 0, 13, 40)
SplitStr(SBDIparsed(1),TxModuleReply,CHR(44),6,0)
If SBDIparsed(1)="1" AND SBDIparsed(2) <> LastSuccessfulMOMSN Then
TxSendSuccess = True
SentWithSuccess = SentWithSuccess + 1
LastSuccessfulMOMSN = SBDIparsed(2)
```

```
Else
TxSendSuccess = False
TxSvcAvail = false
TxRSSI = 0
EndIf
SBDsession = False'\
Loop Until TxSendSuccess' actually, it keeps trying until TxTimeout
TxTimeTaken = TxTimeoutTimer
Timer(1,sec,3)
SerialClose(ComRS232)
EndSub


'*****************************************************************************
' The main program. On datalogger power-on and after a few seconds needed for th
' lights the LED for 3 seconds to notify that the station is up and running. The
' the only SlowSequence Scan.
'*****************************************************************************

BeginProg

'light the LED for 2 seconds ("main program started")
PortSet(5,true)
Delay(0,500,msec)
PortSet(5,false)

'SetStatus ("USRDriveSize",8192)' for use with OpenFile when implementing bina

Low_Batt = True'will be cleared after the first scan if battery is ok. Prevent
FastScanMode = False' True
FastScanCount = FastScanReps
IsSummer = False
TxAllowed = True

WinterDataTerminator="!W"
SummerDataTerminator="!S"
InstantDataTerminator="!I"
DiagnDataTermninator="!D"

LoggerOSrev = Right(Status.OSVersion,2)' it assumes that Cambell will keep num
If LoggerOSrev < 14 Then TimersUnitMult = 1000000 Else TimersUnitMult = 1'Here

Battery(Batt_VoltI)
' If Batt_VoltI > LoBattThre + LoBattResHyst Then Low_Batt = False

GPSinUse = UpperCase(GPSType)
BestHDPseen = 100
sBestHDPseen = 100

TxInitializeOk=False
PortGet(C5status,8)
SerialOpen(com2,GarminBaudRate,0,0,1000)'due to a possible bug in the CR1000 O

'*****************************************************************************
' The main Scan calls the Measure subroutine to do the actual measurement job, t
' Iridium module. Depending on which GPS receiver is being used (the Trimble or
' it can wait for the GPS of the Iridium module to return a valid position and t
' proceeds issuing the CallTable commands for the TableMem AND for the TableTxXx
' season. Then if needed it acesses the relevant TableTxXxxxxx, assembles the me
' module, uploads it, initiates the SBD transfer by sending the AT+SBDRB or AT+S
' until either it succeeds or the Max_SBD_Attempts is reached. Finally it calls
'*****************************************************************************
Scan(FastScanRate,Sec,1,0)
```

```
' If Batt_VoltI > LoBattThre + LoBattResHyst Then Low_Batt = False

'Initial Datalogger Battery Voltage measurement Batt_Volt (do this before turn
' If Batt_VoltI = -999 Then Battery(Batt_VoltI)

PulsePort(5,10000)

'Excites Vx3 to allow the NT1400 to thermally stabilize
If (TimeIntoInterval(0,ScanRate,sec) OR FastScanMode) Then
ExciteV(Vx3,2500,0)
Timer(3,sec,2)
TimeSinceVx3on = Timer(3,sec,4)
Else
Timer(3,sec,3)'we won't need to also switch Vx3 off explicitly: it is done
EndIf

'deals with the fastscan state
''    C5statusPrev = C5status'to deal with the hardware FastScan switch in a saf
''    'PortsConfig(&B1000,&B0000)
''    PortGet(C5status,5)'ReadIO(C5status, &B10000)'
''    If C5status AND NOT C5statusPrev Then FastScanMode = True
''    If C5statusPrev AND NOT C5status Then
''      FastScanMode = False
''      LastFastScan = True
''    EndIf
If FastScanMode AND FastScanCount = 0 Then FastScanCount = FastScanReps
If NOT FastScanMode AND FastScanCount > 0 Then FastScanCount = 0
If FastScanMode AND FastScanCount > 0 Then FastScanCount = FastScanCount - 1
If FastScanMode AND FastScanCount = 0 Then LastFastScan = True
If FastScanMode AND FastScanCount = 0 Then FastScanMode = False


'Starts the Rotronics aspirated shield fan
If (TimeIntoInterval(ScanRate-AdvanceFanStart*FastScanRate,ScanRate,sec) OR
If Batt_VoltI = -999 Then Battery(Batt_VoltI)
  If NOT Low_Batt Then
PortSet(8,true)
Asp_fan_on = True
EndIf
EndIf


'if needed, it powers up the Tx module and initializes its integrated GPS re
If (GPSinUse = "G" OR GPSinUse = "N") Then
If Batt_VoltI = -999 Then Battery(Batt_VoltI)
  If NOT Low_Batt Then
If IsSummer Then
If (TimeIntoInterval(SummerGPSrate*ScanRate-AdvanceGPSstart,SummerGPSr
If (GPSinUse = "G") Then
PortSet(6,True)
TxModuleGPSPowerOn = True
'\'SerialOpen(com2,4800,0,0,150)'due to a possible bug in the CR10
SerialFlush(Com2)
GPGGAsentence=""
SerialIn (GPGGAsentence,COM2,GarminWarmup * 100,-1,10)
'\'SerialClose(com2)
If Len(GPGGAsentence)>0 Then
GarminInitOk = True
Else
GarminInitOk = False
EndIf
EndIf
If (GPSinUse = "N") Then
```

```
temptemp=temptemp+1
PortSet(6,True)
TxModuleGPSPowerOn = True
SerialOpen(ComRS232,19200,0,0,100)
SerialFlush(ComRS232)
Call TxModuleInit
If TxInitializeOk Then
SerialOut(ComRS232,"AT*S=0"+CHR(13)+CHR(10),"",1,10)
SerialOut(ComRS232,"AT+PP=1"+CHR(13)+CHR(10),"",1,10)'should be
SerialOut(ComRS232,"AT+PNAV=1"+CHR(13)+CHR(10),"",1,10)'should b
SerialClose(ComRS232)
EndIf
EndIf
If NOT(GarminInitOk OR TxInitializeOk) Then
TxPowerDueForSwOff = True
EndIf
EndIf
Else'else it is winter
If (TimeIntoInterval(WinterGPSrate*ScanRate-AdvanceGPSstart,WinterGPSr
If (GPSinUse = "G") Then
PortSet(6,True)
TxModuleGPSPowerOn = True
'\'SerialOpen(com2,4800,0,0,150)'due to a possible bug in the CR10
SerialFlush(Com2)
GPGGAsentence=""
SerialIn (GPGGAsentence,COM2,GarminWarmup * 100,-1,10)
'\'SerialClose(com2)
If Len(GPGGAsentence)>0 Then
GarminInitOk = True
Else
GarminInitOk = False
EndIf
EndIf
If (GPSinUse = "N") Then
temptemp=temptemp+1
PortSet(6,True)
TxModuleGPSPowerOn = True
SerialOpen(ComRS232,19200,0,0,100)
SerialFlush(ComRS232)
Call TxModuleInit
If TxInitializeOk Then
SerialOut(ComRS232,"AT*S=0"+CHR(13)+CHR(10),"",1,10)
SerialOut(ComRS232,"AT+PP=1"+CHR(13)+CHR(10),"",1,10)'should be
SerialOut(ComRS232,"AT+PNAV=1"+CHR(13)+CHR(10),"",1,10)'should b
SerialClose(ComRS232)
EndIf
EndIf
If NOT(GarminInitOk OR TxInitializeOk) Then
TxPowerDueForSwOff = True
EndIf
EndIf
EndIf
EndIf
EndIf


'Executes the measurements ************************************************
'*****************************************************************************
If Batt_VoltI = -999 Then Battery(Batt_VoltI)

If (FastScanMode AND FastScanCount > 0) OR TimeIntoInterval(0,ScanRate,sec)
SW12(True)'Enabling switched 12V supply
SW12Von = True
```

```
Timer(2,sec,2)
TimeSince12Von = Timer(2,sec,4)

'Do the measurements
'\'SerialOpen(com2,4800,0,0,150)'due to a possible bug in the CR1000 OS, i



'Wiring Panel Temperature measurement PTemp_C:
PanelTemp(PTemp_C,_50Hz)
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pa

'get a feel of the season...
RealTime(RTime)
If DayOfYear >= BeginOfSummer AND DayOfYear < BeginOfWinter Then IsSummer = Tr

'open COM2 for the Garmin GPS (com2 gets closed when SW12 goes off, so thet
'    SerialOpen(com2,4800,0,0,2000)

'do now whatever needs the 12V switched supply to be ON but doesn't use the



'Enables the MUX by bringing its _RES input high and then explicitly resets t
PortSet(1,1)  'brings the _RES input on the MUX high (means the MUX is in ope
Delay(0,1,sec)'waits for the MUX to be fully awake after SW12V came up and _R
PulsePort(1,20000)  'pulses low the _RES input on the MUX for 20 ms

'advances to MUX differential channel 1 and measures Young Wind Direction Sens
'(optimized solution to to share the same MUX position). So, first measure Win
PulsePort(2,5000)
Delay(0,20,msec)' allows for settling time
PulseCount(WS_ms,1,1,1,1,0.098,0)
BrHalf(WindDir,1,mV2500,14,1,1,2500,True,0,_50Hz,355,0)''consider using fast
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pa
If WindDir>=360 Then WindDir=0

'...then measure BP_mmHg from the CS100. This must never occur less than 1 se
VoltSe(BP_mmHg,1,mV2500,13,1,0,_50Hz,0.2,600.0)
BP_mmHg=BP_mmHg*0.75006

'cycles through and measures the 8 termistors in the termistor string and the
For it = 1 To TermStringSz Step 2
  'first measure the thermistors
PulsePort(2,5000)' COULD THIS REPLACE THE FOLLOWING DELAY???
Delay(0,20,msec)' allows for settling time -> don't remove or measures gets
Therm107 (TermStringT(it),1,15,Vx3,20000,_50Hz,1.0,0)
Therm107 (TermStringT(it+1),1,16,Vx3,20000,_50Hz,1.0,0)
'      Delay(0,300,msec)
'then measure the current shunts located at the same MUX position
VoltDiff (Currents((it+1)/2),1,mV7_5,7,True,0,_50Hz,200,0)'200 is for a sh
Next
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pa

'advances to MUX differential channel 11 and measures both the Xtilt and Ytilt
PulsePort(2,5000)
Delay(0,20,msec)
VoltSe(Xtilt,1,1,13,1,0,_50Hz,1,0)
VoltSe(Ytilt,1,1,14,1,0,_50Hz,1,0)

'advance the MUX to measure the 4 radiation signals from the CNR1 (can too be
PulsePort(2,5000)
```

```
Delay(0,20,msec)' allows for settling time ... needed??
VoltDiff (CNR1_SWin,1,mV25,7,True,0,_50Hz,1,0)' may go out of range with ver
PulsePort(2,5000)
Delay(0,20,msec)' allows for settling time ... needed??
VoltDiff (CNR1_LWin,1,mV7_5,7,True,0,_50Hz,1,0)' may go out of range with ve
PulsePort(2,5000)
Delay(0,20,msec)' allows for settling time ... needed??
VoltDiff (CNR1_SWout,1,mV25,7,True,0,_50Hz,1,0)
PulsePort(2,5000)
Delay(0,20,msec)' allows for settling time ... needed??
VoltDiff (CNR1_LWout,1,mV7_5,7,True,0,_50Hz,1,0)
BrHalf4W (temp,1,mV25,mV25,3,Vx2,1,2035,True ,True ,0,_50Hz,1.0,0)'CNR1 Pt100
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pa
PRT (CNR1_Pt100,1,temp,1.0,0)

'SR50 Sonic Ranging Sensor (SDI-12 Output) measurements DT, TCDT, and DBTCDT:
While TimeSince12Von < SR50Warmup
TimeSince12Von = Timer(2,sec,4)
Wend
If FastScanMode OR LastFastScan OR TimeIntoInterval(0,ScanRate,sec) Then
SDI12Recorder(SR50_SnowHeight(),7,SnowHeightSR50address,"M1!",1.0,0)'Snowh
SDI12Recorder(SR50_Ablation(),7,AblationSR50address,"M1!",1.0,0)
EndIf

'Rotronics aspirated shield sensor suite measurement AS_T and AS_RH (the fan i
PulsePort(2,5000)
Delay(0,20,msec)
While TimeSince12Von < HygroClipWarmup
TimeSince12Von = Timer(2,sec,4)
Wend
VoltSe(AS_T,1,6,14,0,0,_50Hz,1,0)'autorange, no gnd offset correction
VoltSe(AS_RH,1,1,13,1,0,_50Hz,1,0)
BrHalf4W (temp,1,mV25,mV25,1,Vx2,1,2035,True ,True ,0,_50Hz,1.0,0)'Rotronic Pt
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pa
If Asp_fan_on Then Aspirated_Meas = True Else Aspirated_Meas = False' so tha
FanPowerDueForSwOff = True'If NOT (FastScanMode) Then FanPowerDueForSwOff =
''      PortSet(8,false)
''      Asp_fan_on = False
''    EndIf
AS_T=AS_T*.1
AS_RH=AS_RH*.1
PRT (AS_Pt100,1,temp,1.0,0)

'Rotronics UN-aspirated shield sensor suite measurement US_T and US_RH
PulsePort(2,5000)
Delay(0,20,msec)
VoltSe(US_T,1,6,14,0,0,_50Hz,1,0)'autorange, no gnd offset correction
VoltSe(US_RH,1,1,13,1,0,_50Hz,1,0)
BrHalf4W (temp,1,mV25,mV25,5,Vx2,1,2035,True ,True ,0,_50Hz,1.0,0)'Rotronic Pt
ExciteV(Vx3,2500,0)'This MUST be right after any bridge, P107 and Pane
US_T=US_T*.1
US_RH=US_RH*.1
PRT (US_Pt100,1,temp,1.0,0)

'Now the NT1400 water column pressure transducer
PulsePort(2,5000)
Delay(0,20,msec)
ExciteV(Vx3,2500,0)
While TimeSinceVx3on < NT1400Warmup
TimeSinceVx3on = Timer(3,sec,4)
Wend
BrFull (ablation_meter,1,mV250,7,Vx3,1,2500,True ,True ,10000,_50Hz,1.0,0)
Timer(3,sec,3)
```

```
'resets the MUX (so that even if 12V switched is powered, the MUX itself shoul
PortSet(1,false)

'do now whatever needs the 12V switched supply to be ON and requires warm-up, bu


'GPS work for Garmin (G) or NAL (N) units, or both!

PreParseStr(7) = 0
PreParseStr(9) = 0

'The Garmin part finds its com2 already open and the prog closes it afterwar
'around the CR1000 bug causing higher power draw if we open/close com2 here
If GarminInitOk AND TxModuleGPSpowerOn AND (GPSinUse = "G") Then
GGPSFLAG = true
GPGGAsentence = ""
SerialFlush(com2)'added yesterday
If SerialOut(com2,"","$GPGGA",1,250) Then
SplitStr(PreParseStr(1),GPGGAsentence,CHR(44),15,5)
PreQual_FP = PreParseStr(7)'we need to do this explicit conversion to
PreHDP_FP = PreParseStr(9)'we need to do this explicit conversion to f
If PreQual_FP > 0 AND PreHDP_FP > 0 AND PreHDP_FP <= BestHDPseen Then
SplitStr(ParseStr(1),GPGGAsentence,CHR(44),15,5)
BestHDPseen = HDP
'          ElseIf NOT (BestHDPseen < 100) Then'so to avoid deleting a previously
'            For ParseStrIT=1 To 15
'              ParseStr(ParseStrIT)=""
'            Next
EndIf
If NOT (BestHDPseen < 100) Then'so to avoid deleting a previously obta
For ParseStrIT=1 To 15
ParseStr(ParseStrIT)=""
Next
EndIf
EndIf
GGPSFLAG = false
TxPowerDueForSwOff = True
EndIf


'The NAL part (it opens and closes his ComRS232 port itself)
If TxInitializeOk AND TxModuleGPSpowerOn AND (GPSinUse = "N") Then
NGPSFLAG = true
GPGGAsentence = ""
SerialOpen(ComRS232,19200,0,0,150)
SerialOut(ComRS232,"AT"+CHR(13)+CHR(10),"",1,10)'this is to let the CR1000
SerialOut(ComRS232,"AT+PA=1"+CHR(13)+CHR(10),"",1,10)
If SerialOut(ComRS232,"","$GPGGA",1,150) Then
SerialIn(GPGGAsentence,ComRS232,150,"*",150)'was 13
SplitStr(PreParseStr(1),GPGGAsentence,CHR(44),15,5)
PreQual_FP = PreParseStr(7)'we need to do this explicit conversion to
PreHDP_FP = PreParseStr(9)'we need to do this explicit conversion to f
If PreQual_FP > 0 AND PreHDP_FP > 0 AND PreHDP_FP <= BestHDPseen Then
SplitStr(ParseStr(1),GPGGAsentence,CHR(44),15,5)
BestHDPseen = HDP
EndIf
'''          If NOT (sBestHDPseen < 100) Then'so to avoid deleting a previously
'''            For sParseStrIT=1 To 15
'''              sParseStr(sParseStrIT)=""
'''            Next
'''          EndIf
Else
```

```
TxInitializeOk = False
EndIf
If NOT (FastScanMode OR LastFastScan) Then
SerialOut(ComRS232,"AT+PP=0"+CHR(13)+CHR(10),"",1,10)
EndIf
SerialClose(ComRS232)
NGPSFLAG = false
TxPowerDueForSwOff = True
ElseIf (FastScanMode OR LastFastScan)
Call TxModuleInit
If TxInitializeOk Then
SerialOut(ComRS232,"AT*S=0"+CHR(13)+CHR(10),"",1,10)
SerialOut(ComRS232,"AT+PP=1"+CHR(13)+CHR(10),"",1,10)'should be sent as
SerialOut(ComRS232,"AT+PNAV=1"+CHR(13)+CHR(10),"",1,10)'should be sent a
SerialClose(ComRS232)
EndIf
If NOT (BestHDPseen < 100) Then'so to avoid deleting a previously obtained
For ParseStrIT=1 To 15
ParseStr(ParseStrIT)=""
Next
EndIf
EndIf


'Final Battery Voltage measurement Batt_Volt and Low_Batt check (do this befor
Battery(Batt_VoltF)
Batt_V_Drop = Batt_VoltI - Batt_VoltF
If Batt_VoltF < LoBattThre Then Low_Batt = True
If Batt_VoltF >= LoBattThre + LoBattResHyst Then Low_Batt = False

'*************************************************************************
'*************************************************************************
'\'SerialClose(com2)
Timer(2,sec,3)
  SW12(False)'Disables switched 12V supply
SW12Von=False
EndIf

If TimeIntoInterval(0,ScanRate,sec) Then
CallTable(TableDiagnostics)
CallTable(TableMem)
'  If TableMem.Output(1,1) Then
'   BestHDPseen = 100
'      EndIf

If FanPowerDueForSwOff AND NOT FastScanMode
PortSet(8,false)
Asp_fan_on = False
FanPowerDueForSwOff = False
EndIf

If IsSummer Then
  CallTable(TableSummerTx)
  If TableSummerTx.Output(1,1) Then
  BestHDPseen = 100
    TxBufferString=""
    TableInstantaneousString=""
    TxStringUnstripped=""
    TxString=""
    GetRecord(TxBufferString,TableSummerTx,1)
    If AppendInstantaneous AND TimeIntoInterval(0, HoursInstantaneous, hr) T
  CallTable(InstantaneousTable)
      GetRecord(TableInstantaneousString,InstantaneousTable,1)
```

```
      TxBufferString=Left(TxBufferString, Len(TxBufferString)-2)+","+Mid(Tab
    EndIf
CallTable(TxBufferTable)
    If SendRecordBacklog < SummerTableBuffer Then SendRecordBacklog = SendRe

If TimeIntoInterval(0,DiagnosticsRate,hr) AND SendRecordBacklog < Summ
    TxBufferString=""
GetRecord(TxBufferString,TableDiagnostics,1)
CallTable(TxBufferTable)
SendRecordBacklog = SendRecordBacklog + 1
EndIf

    For SendRecordIT = SendRecordBacklog To 1 Step -1
TxString=""
      MpFLAG=true'remove!!
      Delay(0,2,sec)'remove!!        'Call AppendInstantMeas        'Cal
      GetRecord(TxStringUnstripped,TxBufferTable,SendRecordIT)
      StripStringStart = InStr(1,TxStringUnstripped,CHR(34)+CHR(34),4)
      For StringStripperIT=StripStringStart To Len(TxStringUnstripped)-5'5 i
        StringChar=Mid(TxStringUnstripped, StringStripperIT, 1)
        If StringChar<>CHR(34) Then TxString=TxString+StringChar
      Next StringStripperIT
      TxString=TxString+",!M"'**************in summer too
      'TxString=Right(TxString,Len(TxString)-InStr(1,TxString,CHR(44),4)+0)
      Call IridiumTx
      MPFLAG=false'remove!!
      If NOT TxSendSuccess Then ExitFor
    Next
    If SendRecordIT = 0 Then ResetTable(TxBufferTable)
    SendRecordBacklog = SendRecordIT
TxPowerDueForSwOff = True
  EndIf
EndIf
If NOT IsSummer Then
  CallTable(TableWinterTx)
  If TableWinterTx.Output(1,1) Then
  BestHDPseen = 100
    TxBufferString=""
    TableInstantaneousString=""
    TxStringUnstripped=""
    TxString=""
    GetRecord(TxBufferString,TableWinterTx,1)
    If AppendInstantaneous AND TimeIntoInterval(0, HoursInstantaneous, hr) T
  CallTable(InstantaneousTable)
      GetRecord(TableInstantaneousString,InstantaneousTable,1)
      TxBufferString=Left(TxBufferString, Len(TxBufferString)-2)+","+Mid(Tab
    EndIf
CallTable(TxBufferTable)
    If SendRecordBacklog < WinterTableBuffer Then SendRecordBacklog = SendRe

If TimeIntoInterval(0,DiagnosticsRate,hr) AND SendRecordBacklog < Wint
    TxBufferString=""
GetRecord(TxBufferString,TableDiagnostics,1)
CallTable(TxBufferTable)
SendRecordBacklog = SendRecordBacklog + 1
EndIf

    For SendRecordIT = SendRecordBacklog To 1 Step -1
TxString=""
      MpFLAG=true'remove!!
      Delay(0,2,sec)'remove!!        'Call AppendInstantMeas        'Cal
      GetRecord(TxStringUnstripped,TxBufferTable,SendRecordIT)
      StripStringStart = InStr(1,TxStringUnstripped,CHR(34)+CHR(34),4)
```

```
      For StringStripperIT=StripStringStart To Len(TxStringUnstripped)-5'5 i
        StringChar=Mid(TxStringUnstripped, StringStripperIT, 1)
        If StringChar<>CHR(34) Then TxString=TxString+StringChar
      Next StringStripperIT
      TxString=TxString+",!M"'**************in summer too
      'TxString=Right(TxString,Len(TxString)-InStr(1,TxString,CHR(44),4)+0)
      Call IridiumTx
      MPFLAG=false'remove!!
      If NOT TxSendSuccess Then ExitFor
    Next
    If SendRecordIT = 0 Then ResetTable(TxBufferTable)
    SendRecordBacklog = SendRecordIT
TxPowerDueForSwOff = True
  EndIf
EndIf
EndIf

If TxPowerDueForSwOff AND NOT (FastScanMode OR LastFastScan) Then
SerialOut(ComRS232,"AT*F"+CHR(13)+CHR(10),"OK",1,100)' to allow 5 secs f
Delay(0,1,sec)
PortSet(6,false)
TxModuleGPSPowerOn = False
TxInitializeOk = False'turn this off??
TxPowerDueForSwOff = False
EndIf
If LastFastScan Then LastFastScan = False

If FanPowerDueForSwOff AND NOT FastScanMode
PortSet(8,false)
Asp_fan_on = False
FanPowerDueForSwOff = False
EndIf

SerialClose(ComRS232)
Batt_VoltI = -999' Batt_VoltF doesn't need such a NODATA value, so it is lef
PulsePort(5,10000)
NextScan
EndProg
```

Appendix E – Telemetry data retrieval program

(Michele Citterio, GEUS)

```
#getdata 1.60f, written by Michele Citterio (mcit@geus.dk) at GEUS, Copenhagen
#coding=cp850
#
#install and use: install pyhton 2.5.2 and pywin32 212, copy this file into an
#empty folder and run it from there. Four files and a folder can be created
#for each IMEI number found when retrievig the emails from the Exchange server:
#1) IMEI#.txt    readable comma-separated ascii file - AWS observations
#2) IMEI#-D.txt  readable comma-separated ascii file - AWS diagnostics
#3) IMEI#-F.txt  readable comma-separated ascii file - malformed msg (from 1.56)
#4) IMEI#-X.txt  file with whatever else has been received from that IMEI device
#5) IMEI#/       a folder where individual received email attachments are stored
#One further file msgshash.dat is also created on the first program run and
#updated at every run to store the hashes of already processed mesages, so that
#they can be identified and skipped on later program runs.
#
#revision history:
# 1.51, 10-05-2009  + configured PROMICE 2009 binary format
#                   + configured Quadra Mining 2009 Malmbjerg binary format
#                   + added Python version 2.5.2 check and made Py3k-aware
#                   + some code cleanup of try..except blocks
# 1.52, 11-05-2009  + configured Glaciobasis 2009 Zackenberg Top binary format
# 1.53, 09-06-2009  + configured Glaciobasis 2009 Zackenberg Main binary format
#                   + much improved detection of bin format specification errors
#                   + fixed wrong format specifications 5, 7 and 12
#                   + improved the try..except blocks by setting the error types
#                   + tested for compatibility with up to Python 2.5.4
#                   + improved handling of user's infostores or folders choices
#                   + added warning for binary messages longer than specified
#                   + fixed support of Unicode pathnames, so x:\Søren works now!
# 1.54a, 19/06/2009 + attempted workaround to crash in pywin32 on some machines
# 1.55, 23/09/2009  + fixed 1 h timestamp error. Epochs handling more portable
# 1.60f, 06/10/2009 + added prompt to sort the *.txt files based on timestamps
#                   + fixed crash decoding malformed (too long/short) messages
#                   + truncated messages are decoded using ? for missing values
#                   + added option to put malformed messages in a separate file
#                   + added exceptions handling while opening the *.txt files
#                   + logoff from Exchange server even on unhandled exceptions
#                   + added check for re-delivered messages (Iridium or NAL bug)
#                   + MOMSN is also hashed when detecting duplicates
#                   + use the current user's name as default for Exchange logon
#                   + improved the printed output during the run and at the end
#                   + now uses 128 bits hashes: must retrieve all messages again
#       since -f    + do not try to logoff when logon failed due to wrong userid
#                   + pickle is replaced with cPickle for speed
#                   + ctrl-break is also trapped and handled same as ctrl-c
#
```

```
# known issues: the stats about the processed messages and some screen output
#               while running are wrong (but the data files generated are ok).
#               Also, when user aborts a file write error, the line is still counted as added
#               Cause is known and harmless, will be fixed in 1.60 final.
#
#  Updated list of format ID numbers (range of format specifications) in use:
#    1 (5-9), 2 (10-14), 3 (15-19), 4 (20-24)
#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

from __future__ import with_statement
try:
   import sys
   if sys.hexversion >= 0x030000F0:
      print ("ERROR: Python 3000 is not supported. Use Python 2.5.4 instead!\n")
      sys.exit()
   if sys.hexversion < 0x020502F0:
      print "WARNING: untested on earlier versions than Python ver. 2.5.2\n"
   if sys.hexversion > 0x020504F0:
      print "WARNING: untested on later versions than Python ver. 2.5.4\n"
   import os
   import string
   import time
   import glob
   import hashlib
   import cPickle
   import calendar
   import signal
   from bintx import *
   try:
      import win32api
      from win32com.client import Dispatch, pywintypes
   except ImportError:
      print 'Error - PyWin32 is not installed.'
      sys.exit()
except ImportError:
   print 'ERROR: needed modules from the standard library are not available. Reinstall
Python\n'
   sys.exit()

signal.signal(signal.SIGBREAK, signal.default_int_handler)#to trap ctrl-break

localtestmode = False#True
FilterMalformed = True
HashFunc = hashlib.sha1#before 1.60 hashlib.sha512() was used but this was much
overkill: even with 128 bits
```

```
#there's about 10^-18 chances of an error after 10^10 messages received...

FormatBytesNum = {}
FormatBytesNum['f'] = 2# value encoded as 2 bytes base-10 floating point (GFP2)
FormatBytesNum['l'] = 4# value encoded as 4 bytes two's complement integer (GLI4)
FormatBytesNum['t'] = 4# timestamp as seconds since 1990-01-01 00:00:00 +0000
encoded as GLI4
FormatBytesNum['g'] = 4# GPS time encoded as GLI4
FormatBytesNum['n'] = 4# GPS latitude encoded as GLI4
FormatBytesNum['e'] = 4# GPS latitude encoded as GLI4

FormatSpec = {}
#GlacioBasis 2009 Top
FormatSpec[10] = [49, "tfffffffffffffffffffffffffffffffffgneffffffffff", "GlacioBasis 2009
Top 1-h summer message"]
FormatSpec[11] = [56, "tfffffffffffffffffffffffffffffffffgneffffffffffffffff", "GlacioBasis
2009 Top 1-h summer message (+ instant.)"]
FormatSpec[12] = [49, "tfffffffffffffffffffffffffffffffffgneffffffffff", "GlacioBasis 2009
Top 3-h winter message"]
FormatSpec[13] = [56, "tfffffffffffffffffffffffffffffffffgneffffffffffffffff", "GlacioBasis
2009 Top 3-h winter message (+ instant.)"]
FormatSpec[14] = [22, "tfffffffffffffffffff", "GlacioBasis 2009 Top diagnostic
message"]
#GlacioBasis 2009 Main
FormatSpec[20] = [41, "tfffffffffffffffffffffffffgneffffffffff", "GlacioBasis 2009 Main 1-
h summer message"]
FormatSpec[21] = [50, "tfffffffffffffffffffffffffgneffffffffffffffffff", "GlacioBasis 2009
Main 1-h summer message (+ instant.)"]
FormatSpec[22] = [41, "tfffffffffffffffffffffffffgneffffffffff", "GlacioBasis 2009 Main 3-
h winter message"]
FormatSpec[23] = [50, "tfffffffffffffffffffffffffgneffffffffffffffffff", "GlacioBasis 2009
Main 3-h winter message (+ instant.)"]
FormatSpec[24] = [22, "tfffffffffffffffffff", "GlacioBasis 2009 Main diagnostic
message"]


# Win32 epoch is 1st Jan 1601 but MSC epoch is 1st Jan 1970 (MSDN gmtime docs),
same as Unix epoch.
# Neither Python nor ANSI-C explicitly specify any epoch but CPython relies on the
underlying C
# library. CRbasic instead has the SecsSince1990() function.
UnixEpochOffset = calendar.timegm((1970, 1, 1, 0, 0, 0, 0, 1, 0)) #this should always be
0 in CPython on Win32
CRbasicEpochOffset = calendar.timegm((1990, 1, 1, 0, 0, 0, 0, 1, 0))
EpochOffset = UnixEpochOffset + CRbasicEpochOffset
```

```
#what follows should be turned into methods of a future FormatSpecTable object

FormatSpecAreOK = True
for FormatSpecEntry in FormatSpec.iteritems():
   #FormatSpecEntry[1][0]
   if FormatSpecEntry[1][0] != len(FormatSpecEntry[1][1]):
      print 'WARNING: bad format definition for message type %i (%s):\n format
unconsistency (%i values declared but %i specified)' %(FormatSpecEntry[0],
FormatSpecEntry[1][2],FormatSpecEntry[1][0],len(FormatSpecEntry[1][1]))
      FormatSpecAreOK = False#sys.exit()
   Length = 1#all messages have a trailing one-byte format ID
   for ValueFormat in FormatSpecEntry[1][1]:
      try:
         Length += FormatBytesNum[ValueFormat]
      except KeyError:#ValueFormat is not an existing key
         Length = -1
         print 'WARNING: Bad format definition for message type %i (%s):\n unknown
symbol "%s"' %(FormatSpecEntry[0], FormatSpecEntry[1][2],ValueFormat)
         FormatSpecAreOK = False#sys.exit()
   try:#if the byte length was specified in the format configuration, check it
      if FormatSpec[FormatSpecEntry[0]][3] != Length:
         print 'WARNING: Bad format definition for message type %i (%s):\n length
unconsistency (%i bytes declared but %i expected)' %(FormatSpecEntry[0],
FormatSpecEntry[1][2],FormatSpec[FormatSpecEntry[0]][3],Length)
         FormatSpecAreOK = False#sys.exit()
   except IndexError:#the byte length was not specified in the format configuration, so set
it now
      FormatSpec[FormatSpecEntry[0]].append(Length)#it initializes a further element in
the table: the message length in bytes
if not FormatSpecAreOK:
   print 'ERROR: the message format specifications need corrections'
   sys.exit()
  # return Length


UnicodeEncoding = 'cp850'
BaseDir = os.getcwd()
#BaseDir = BaseDir.encode(UnicodeEncoding)
try:
   with open(BaseDir + '\\msgshash.dat', 'rb') as msgshash:
      IsFirstRun = False
      SeenMessageHashes = cPickle.load(msgshash)
except IOError:
   IsFirstRun = True
   print 'WARNING: msgshash.dat not found. Already processed messages (if any) may
be duplicated\n'
```

```
   SeenMessageHashes = []
SeenMessageHashesPreviousRuns = SeenMessageHashes[:]#TODO learn better why this
is actually required!


AppendedSummer = 0
AppendedWinter = 0
AppendedWithInstant = 0
AppendedDiagnostics = 0
ApendedMalformed = 0
ProcessedInfostoreName = ''
ProcessedFolderName = ''

try:
   cdo = Dispatch("MAPI.session")
   DeafultLoginName = win32api.GetUserName()
   Login_Prompt = 'Enter the login name (defaults to "%s"): ' %DeafultLoginName
   LoginName = raw_input(Login_Prompt)
   if not LoginName: LoginName = DeafultLoginName
   try: #this is to catch all otherwise unhandleld exceptions and logoff from the server
      try:
         print 'Logging in as %s...' %LoginName,
         cdo.Logon(LoginName)  # MAPI profile name
         LoggedIn = True
         print ' OK'
      except pywintypes.com_error, errordetails:
         print 'ERROR: COM API error %s (%s) - wrong login' %(errordetails[0],
errordetails[1])
         sys.exit()

      InfoStores=cdo.InfoStores
      InfoStoresCount = cdo.InfoStores.Count
      print '\nThere are %i infostores available:' %InfoStoresCount
      DefaultInfoStore = 'Postkasse - ice'
      DefaultInfoStoreNum = 0
      #InfoStoreNum = 0
      InfoStoresList = []
      for it in range(InfoStoresCount):
         InfoStoreName = InfoStores[it].Name
         InfoStoresList.append(InfoStoreName)
         print '%i) %s' %(it+1, InfoStoreName.encode(UnicodeEncoding))
         if InfoStoreName == DefaultInfoStore: DefaultInfoStoreNum = it#+1
      if DefaultInfoStoreNum:
         InfoStore_Prompt = 'Enter the desired infostore [1, %i] (defaults to "%s"): '
%(InfoStoresCount, DefaultInfoStore.encode(UnicodeEncoding))
      else:
```

```
        InfoStore_Prompt = 'Enter the desired infostore [1, %i]: ' %InfoStoresCount
    while True:
        InfoStore = raw_input(InfoStore_Prompt)
        if not len(InfoStore): break
        try:
            InfoStore = int(InfoStore)
            if InfoStore < 0: raise ValueError
            if InfoStore > InfoStoresCount: raise ValueError
            break
        except ValueError:
            print "WARNING: not in the range [1, %i]" %InfoStoresCount
    if not InfoStore:
        InfoStoreNum = DefaultInfoStoreNum
    else:
        InfoStoreNum = InfoStore - 1#needs to be 0-based to be used as an index
    try:
        ProcessedInfostoreName = cdo.Infostores[InfoStoreNum -
0].Name.encode(UnicodeEncoding)
        print 'Looking for folders in %s...' %ProcessedInfostoreName,
        Folders = cdo.Infostores[InfoStoreNum - 0].RootFolder.Folders
        print ' OK'
    except:#TODO: restrict the error type
        print 'Error - Bad infostore.'
        #cdo.logoff
        sys.exit()


    FoldersCount = Folders.Count
    print '\nThere are %i folders available:' %FoldersCount
    DefaultFolder = 'Indbakke'
    DefaultFolderNum = 0#remove?
    FolderNum = 0
    FoldersList = []
    for it in range(FoldersCount):
        FolderName = Folders[it].Name
        FoldersList.append(FolderName)
        print '%i) %s' %(it+1, FolderName.encode(UnicodeEncoding))
        if FolderName == DefaultFolder: DefaultFolderNum = it#+1
    if DefaultFolderNum:
        Folder_Prompt = 'Enter the desired folder [1, %i] (defaults to "%s"): '
%(FoldersCount, DefaultFolder.encode(UnicodeEncoding))
    else:
        Folder_Prompt = 'Enter the desired folder [1, %i]: '%FoldersCount
    while True:
        Folder = raw_input(Folder_Prompt)
        if not len(Folder): break
```

```
            try:
                Folder = int(Folder)
                if Folder < 0: raise ValueError
                if Folder > FoldersCount: raise ValueError
                break
            except ValueError:
                print "WARNING: not in the range [1, %i]" %FoldersCount
        if not Folder:
            FolderNum = DefaultFolderNum
        else:
            FolderNum = Folder - 1#needs to be 0-based to be used as an index
        try:
            ProcessedFolderName = Folders[FolderNum -
0].Name.encode(UnicodeEncoding)
            print 'Looking for messages in %s...' %ProcessedFolderName,
            Messages = Folders[FolderNum - 0].Messages
            print ' OK'
        except:
            print 'Error - Bad folder.'
            #cdo.logoff
            sys.exit()


        MessagesCount = Messages.Count
        print '\nThere are %i messages available... (wait!)' %MessagesCount
        ProcessedMessages = 0
        ProcessedAttachments = 0
        AppendedLines = 0
        SkippedDuplicated = 0
        SkippedAlreadyParsed = 0
        FoundMalformed = 0
        Message = Messages.GetFirst()# ref. http://mail.python.org/pipermail/python-
list/2004-July/270944.html
        ModifiedFiles = []
        while Message:# ref. http://mail.python.org/pipermail/python-list/2004-
July/270944.html
        #for Message in Messages:# ref. http://mail.python.org/pipermail/python-list/2004-
July/270944.html
            ProcessedMessages += 1#TODO move to lower down
            if 'sbdservice@sbd.iridium.com' in Message.Sender.Name.lower() and
Message.Attachments.Count == 1:
                ProcessedAttachments += 1
                Attachment = Message.Attachments.Item(1)
                AttachmentName = Attachment.Name.lower().encode(UnicodeEncoding)
                IMEI = AttachmentName[0:AttachmentName.index('_')]
```

```
            MOMSN =
AttachmentName[AttachmentName.index('_')+1:AttachmentName.index('.')]
            ThisIMEIDir = BaseDir + '\\' + IMEI + '\\'#TODO make it more portable using
the path module
            if not glob.glob(ThisIMEIDir): #this IMEI has already been seen
               os.mkdir(BaseDir + '\\' + IMEI.encode(UnicodeEncoding))#TODO make it
more portable using the path module
            Attachment.WriteToFile(ThisIMEIDir + AttachmentName)#TODO add
exceptions handling
            with open(ThisIMEIDir + AttachmentName, 'rb') as InFile:#TODO add
exceptions handling
               IsTooLong = False
               IsTooShort = False
               DataLine = InFile.read(420)
               if len(DataLine) == 0: break
               if DataLine[0].isdigit():
                  IsKnownBinaryFormat = False
                  MessageFormatNum = -9999
               else:
                  MessageFormatNum = ord(DataLine[0])
                  try:
                     MessageFormat = FormatSpec[MessageFormatNum]
                     IsKnownBinaryFormat = True
                  except KeyError:
                     IsKnownBinaryFormat = False
                     UnknMsgFormNum = MessageFormatNum
                  if IsKnownBinaryFormat:#TODO bring all this "if" into the try logic
above
                     print '%s-%s (binary)' %(IMEI, MOMSN) , MessageFormat[2]
                     ExpectedMsgLen = FormatSpec[MessageFormatNum][3]
                     if len(DataLine) < ExpectedMsgLen:
                        IsTooShort = True
                     elif len(DataLine) > ExpectedMsgLen:
                        IsTooLong = True
                     BinaryMessage = DataLine[1:]
                     DataLine = '' #this is a bit crap but works... to be fixed when turning the
decoder into a func
                     BytePointer = 0
                     for ValueNum in range(0, MessageFormat[0]):#TODO - use an iterator
instead of indexes
                        ValueBytes = []
                        ValueBytesNum =
FormatBytesNum[MessageFormat[1][ValueNum]]
                        if MessageFormat[1][ValueNum] == 'f':
                           try:
                              for offset in range(0,ValueBytesNum):
```

```
                                ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                            BytePointer = BytePointer + ValueBytesNum
                            Value = GFP2toDEC(ValueBytes)
                            if Value == 8191:
                               DataLine = DataLine + "NAN"
                            elif Value == 8190:
                               DataLine = DataLine + "INF"
                            elif Value == -8190:
                               DataLine = DataLine + "-INF"
                            else:
                               DataLine = DataLine + str(Value)
                            #print ValueBytes, Value
                         except IndexError:
                            DataLine = DataLine + '?'
                      if MessageFormat[1][ValueNum] == 'l':
                         try:
                            for offset in range(0,ValueBytesNum):
                               ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                            BytePointer = BytePointer + ValueBytesNum
                            Value = GLI4toDEC(ValueBytes)
                            DataLine = DataLine + str(Value)
                            if Value == -2147483648:
                               DataLine = DataLine + "NAN"
                            else:
                               DataLine = DataLine + str(Value)
                            #print ValueBytes, Value
                            #print ValueBytes, Value
                         except IndexError:
                            DataLine = DataLine + '?'
                      elif MessageFormat[1][ValueNum] == 't':
                         try:
                            for offset in range(0,ValueBytesNum):
                               ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                            BytePointer = BytePointer + ValueBytesNum
                            Value = GLI4toDEC(ValueBytes)
                            DataLine = DataLine + time.strftime("%Y-%m-%d
%H:%M:%S", time.gmtime(Value + EpochOffset)) + ',' + str(Value)
                            #print ValueBytes, Value, time.asctime(time.gmtime(Value +
631148400))
                         except IndexError:
                            DataLine = DataLine + '?'
                      elif MessageFormat[1][ValueNum] == 'g':
                         try:
```

```
                        for offset in range(0,ValueBytesNum):
                            ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                        BytePointer = BytePointer + ValueBytesNum
                        Value = GLI4toDEC(ValueBytes)/100.0
                        DataLine = DataLine + str(Value)
                        #print ValueBytes, Value
                    except IndexError:
                        DataLine = DataLine + '?'
                elif MessageFormat[1][ValueNum] == 'n':
                    try:
                        for offset in range(0,ValueBytesNum):
                            ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                        BytePointer = BytePointer + ValueBytesNum
                        Value = GLI4toDEC(ValueBytes)/100000.0
                        DataLine = DataLine + str(Value)
                        #print ValueBytes, Value
                    except IndexError:
                        DataLine = DataLine + '?'
                elif MessageFormat[1][ValueNum] == 'e':
                    try:
                        for offset in range(0,ValueBytesNum):
                            ValueBytes.append(ord(BinaryMessage[BytePointer +
offset]))
                        BytePointer = BytePointer + ValueBytesNum
                        Value = GLI4toDEC(ValueBytes)/100000.0
                        DataLine = DataLine + str(Value)
                        #print ValueBytes, Value
                    except IndexError:
                        DataLine = DataLine + '?'
                DataLine = DataLine + ','
            DataLine = DataLine[:-1] # to remove the trailing comma character
        IsDiagnostics = '!D' in DataLine[-5:-3] or MessageFormatNum % 5 ==
4#FIXME: the stats are wrong because we don't always go through here
        IsObservations = '!M' in DataLine[-2:] or IsKnownBinaryFormat
        IsSummer = ('!S' in DataLine and '!M' in DataLine[-2:]) or
MessageFormatNum % 5 in (0, 1)
        IsWinter = ('!W' in DataLine and '!M' in DataLine[-2:]) or
MessageFormatNum % 5 in (2, 3)
        IsWithInstant = '!I' in DataLine[-5:-3] or MessageFormatNum % 5 in (1, 3)
        if not IsKnownBinaryFormat:
            print '%s-%s' %(IMEI, MOMSN),
            if IsDiagnostics: print '(ascii) generic diagnostic message',
            elif IsObservations and IsSummer:print '(ascii) generic summer
observations message',
```

```
                elif IsObservations and not IsSummer: print '(ascii) generic winter
observations message',
                else: print 'unrecognized message format',
                if IsWithInstant:
                   print '(+ instant.)'
                else:
                   print ''
             IsMalformed = IsTooLong or IsTooShort
             if IsMalformed:
                FoundMalformed += 1
                print '  WARNING - Message is malformed: any missing value will be
replaced by '?'"
             DataLineHash = HashFunc(IMEI + MOMSN + DataLine).hexdigest()
             if not DataLineHash in SeenMessageHashes:
                if IsMalformed and FilterMalformed:
                   MalformedFilePath = BaseDir + '\\' + IMEI + '-F.txt'
                   while True:
                      try:
                         with open(MalformedFilePath, 'a') as OutFile:
                            OutFile.writelines(DataLine+'\n')
                         if not MalformedFilePath in ModifiedFiles:
                            ModifiedFiles.append(MalformedFilePath)
                         break
                      except IOError:
                         while True:
                            AbortOrRetry = raw_input('ERROR while opening %s: Abort or
Retry? [A/R]' %MalformedFilePath)
                            if AbortOrRetry.upper() in ('A', 'R'): break
                         if AbortOrRetry.upper() == 'A': break
                elif IsDiagnostics:
                   DiagnosticFilePath = BaseDir + '\\' + IMEI + '-D.txt'
                   while True:
                      try:
                         with open(DiagnosticFilePath, 'a') as OutFile:
                            OutFile.writelines(DataLine+'\n')
                         if not DiagnosticFilePath in ModifiedFiles:
                            ModifiedFiles.append(DiagnosticFilePath)
                         break
                      except IOError:
                         while True:
                            AbortOrRetry = raw_input('ERROR while opening %s: Abort or
Retry? [A/R]' %DiagnosticFilePath)
                            if AbortOrRetry.upper() in ('A', 'R'): break
                         if AbortOrRetry.upper() == 'A': break
                elif IsObservations:
                   ObservationFilePath = BaseDir + '\\' + IMEI + '.txt'
```

```
                    while True:
                        try:
                            with open(ObservationFilePath, 'a') as OutFile:
                                OutFile.writelines(DataLine+'\n')
                            if not ObservationFilePath in ModifiedFiles:
                                ModifiedFiles.append(ObservationFilePath)
                            break
                        except IOError:
                            while True:
                                AbortOrRetry = raw_input('ERROR while opening %s: Abort or
Retry? [A/R]' %ObservationFilePath)
                                if AbortOrRetry.upper() in ('A', 'R'): break
                            if AbortOrRetry.upper() == 'A': break
                else: #if not diagnostics nor a properly terminated message or known
binary format, then it's garbage and gets dumped here
                    GarbageFilePath = BaseDir + '\\' + IMEI + '-X.txt'
                    while True:
                        try:
                            with open(GarbageFilePath, 'a') as OutFile:
                                OutFile.writelines(DataLine+'\n')
                            break
                        except IOError:
                            while True:
                                AbortOrRetry = raw_input('ERROR while opening %s: Abort or
Retry? [A/R]' %GarbageFilePath)
                                if AbortOrRetry.upper() in ('A', 'R'): break
                            if AbortOrRetry.upper() == 'A': break
                    #the -X.txt garbage files are not sorted since we don't know what is in
them
                SeenMessageHashes.append(DataLineHash)
                AppendedLines += 1
                if IsSummer: AppendedSummer += 1
                if IsWinter: AppendedWinter += 1
                if IsWithInstant: AppendedWithInstant += 1
                if IsDiagnostics: AppendedDiagnostics += 1
                if IsMalformed: ApendedMalformed += 1
            else:
                if DataLineHash in SeenMessageHashesPreviousRuns:
                    SkippedAlreadyParsed += 1
                    print '  NOTE - Message has already been parsed: skipping'
                else:#note this is not 100% accurate: on subsequent runs, repeated are just
detected as duplicated
                    SkippedDuplicated += 1
                    print '  NOTE - Message was sent more than once (an Iridium or NAL
bug): skipping'
                    #see emails in March 2008 with NAL engineers recognizing the issue
```

```
            #DataLine = ''
            Message = Messages.GetNext()# ref. http://mail.python.org/pipermail/python-
list/2004-July/270944.html
        try:
            with open(BaseDir + '\\msgshash.dat', 'wb') as msgshash:
                cPickle.dump(SeenMessageHashes, msgshash)
        except IOError:
            print "Warning - Couldn't save the hashes of the new messages"
    except:
        print 'ERROR: an unhandled exception occurred, terminating the program.'
        raise
    finally:
        if 'LoggedIn' in locals():
            try:
                print '\nlogging off from the Exchange server...',
                cdo.logoff
                print 'ok\n'
            except:
                print '\nWARNING - Failed to log off from the Exchange server\n'
except pywintypes.com_error, errordetails:
    print 'ERROR: COM API error %s (%s) - MAPI may not be available on this machine'
%(errordetails[0], errordetails[1])
    sys.exit()

#MissingAttachments = 100 - ProcessedMessages / ProcessedAttachments * 100
if ProcessedMessages:
    print 'messages processed:        %i' %ProcessedMessages, '   (from %s/%s)'
%(ProcessedInfostoreName, ProcessedFolderName)
    print 'data attachments found:    %i' %ProcessedAttachments
if SkippedAlreadyParsed:
    print 'known messages skipped:    %i    (already parsed during previous program runs)'
%SkippedAlreadyParsed
if SkippedDuplicated:
    print 'repeated messages skipped: %i    (due to a bug on the Iridium or NAL side)'
%SkippedDuplicated
    print 'new records appended:      %i' %AppendedLines
if AppendedSummer: print '           at summer rate: %i' %AppendedSummer
if AppendedWinter: print '           at winter rate: %i' %AppendedWinter
if AppendedWithInstant: print '  with instantaneous data: %i' %AppendedWithInstant
if AppendedDiagnostics: print '              diagnostic: %i' %AppendedDiagnostics

if ApendedMalformed:
    print '\nWARNING: %i malformed messages parsed!' %ApendedMalformed,
    if FilterMalformed:
        print '(using setting: append to IMEI#-F.txt)'
    else:
```

```
        print '(using setting: append to IMEI#.txt)'
    print """        Messages shorter than expected have missing values replaced with ?
      * Any massage longer than expected is decoded up to the expected length
      * All original files can be found in the subfolder of the relevant IMEI
      * Set bool FilterMalformed to control where these data get appended"""

DoSort = ''
#print '\n'
while ModifiedFiles:#so that it only asks if any file has actually been modified
    DoSort = raw_input('New lines were appended. Time-sort the %i relevant .txt files?
[Y/N]'%len(ModifiedFiles))
    if DoSort.upper() in ('Y', 'N'): break
if DoSort.upper() == 'Y':#however the -X.txt garbage files are never sorted since we don't
know what is in them
    for FileName in ModifiedFiles:
        print '%s - reading' %FileName,
        while True:
            try:
                with open(FileName, 'r') as InFile:
                    FileContent = InFile.readlines()
                break
            except IOError:
                while True:
                    AbortOrRetry = raw_input('ERROR while opening %s: Abort or Retry?
[A/R]' %FileName)
                    if AbortOrRetry.upper() in ('A', 'R'): break
                if AbortOrRetry.upper() == 'A': break
        print 'ok, sorting' ,
        try:
            FileContent.sort()
        except NameError:
            continue
        #os.rename(FileName, FileName + '.old')#TODO: for debug only
        print 'ok, writing...' ,
        while True:
            try:
                with open(FileName, 'w') as OutFile:
                    OutFile.writelines(FileContent)
                break
            except IOError:
                while True:
                    AbortOrRetry = raw_input('ERROR while opening %s: Abort or Retry?
[A/R]' %FileName)
                    if AbortOrRetry.upper() in ('A', 'R'): break
                if AbortOrRetry.upper() == 'A': break
        print 'done!'
```

pass