argo data management

Argo data management
DOI: http://dx.doi.org/10.13155/41152

Argo Trajectory Cookbook

Version 4.1

December 2015

ARGO

part of the integrated global observation strategy

Argo data management

Argo DAC trajectory cookbook

Authors: Megan Scanderbeg / Scripps Institution of Oceanography, Jean-Philippe Rannou / ALTRAN, Justin Buck / BODC, Claudia Schmid / AOML, John Gilson / Scripps Institution of Oceanography, Dana Swift / University of Washington, Kanako Sato/ JAMSTEC

How to cite this document

Megan Scanderbeg / Scripps Institution of Oceanography, Jean-Philippe Rannou / ALTRAN, Justin Buck / BODC, Claudia Schmid / AOML, John Gilson / Scripps Institution of Oceanography. , Dana Swift / University of Washington, **Argo DAC trajectory cookbook**. http://dx.doi.org/10.13155/41152

Table of contents

History of the document

Version	Date	Comment
1.0	June 2012	Original version sent around for comment
1.1	August 2012	Comments from John Gilson, Jean-Philippe Rannou, Justin Buck, Kanako Sato, Mizuho Hoshimoto, Bernie Petolas
1.2	November 2012	Updated measurement code table to three places to allow for many more codes. Added satellite name, error ellipse variables, condensed final questions in preparation for ADMT meeting
1.3	February 2013	Updated with feedback from ADMT-13 meeting
1.4	April 2013	Updated format in anticipation of publication
1.5	April , 2014	Updated code for TET estimation for APEX floats with Argos transmission, condensed float tables in Annex I. Updated cycle definition, added new paragraph describing core-Argo and B-Argo trajectory files. Updated MC codes for TST and AST for APEX floats.
2.0	June 2014	Officially given a DOI and named as Trajectory Cookbook
3.0	September 2014	Large changes to APEX float section after numerous discussions with Dana Swift, author of most of the APEX float software
3.2	February 2015	Move estimation procedures for APEX floats to Annex J. Removed all Annex I tables except PROVOR, ARVOR, NINA
3.3	May 2015	Added instructions on PUMPED and UNPUMPED CTD data for NKE floats Updated SOLO-II tables Updated NOVA tables Updated NINJA tables Added Deep NINJA tables
4.0	June 2015	Removed all sections related to profile files for the new DAC Profile cookbook
4.1	November 2015	Standard Reference ID now optional

Preface

This document is still in progress. As such, there are highlighted sections of text throughout that need to be addressed. Yellow highlighting means this is a topic open to discussion - some things are known about this topic, but agreement needs to be reached. Green highlighting signifies a question that needs to be answered by a float expert or float manufacturer. Red highlighting means the issue needs a solution and nothing has been suggested yet.

One proposed entry point into this cookbook is through Annex I which consists of tables for each float type where the rows are different float models and the columns are variables that need to be filled. Each cell will be a link to that point in the document. The table would then allow a DAC to look up their float model and read across that row to find out how to fill each variable. Alternately, a DAC could go to the section of the cookbook for float type and consult tables there to find out how to fill the variables without referring to Annex I.

1 Introduction

This DAC Trajectory Cookbook is to include instructions for the DACs on how to calculate different variables for the Argo trajectory files. It is separate from other data manuals because users do not need to understand all these details, but that it is important that all DACs to be calculating the variables in the same manner.

Concerning Argo trajectory data specifically, correct data processing requires a good knowledge of the float capabilities. Each float type has its own behavior and within a given type, each float version provides specific data useful to trajectory determination. A large part of the document is based on the work done since 2007 on Argo trajectory data in the framework of the ANDRO project by Jean-Philippe Rannou and Michel Ollitrault. Dana Swift has also had a chance to give input on APEX float timing. He has been responsible for writing much of the firmware on APEX floats and was able to provide detailed information on APEX floats.

The algorithms proposed by Jean-Philippe Rannou and Michel Ollitrault are included in the cookbook, but are not mandatory. While they have been designed to be efficient and robust enough to be deployed in a real time data flow without a visual check, the timing information they provide is an estimate.

There are a couple ways to approach this cookbook. If interested in PROVOR and ARVOR floats, the best entry point is proposed in ANNEX I: Cookbook entry point. In the presented tables, one can find, for a given float version, all useful information that can be decoded, computed or estimated from transmitted float data and a link to the concerned paragraph(s) in the document. For all other float types, it is suggested to look at the appropriate float type under section 3.2.2. Each float type has a section there with tables helping to explain which measurement codes should be used for which float type and version.

To decode the Format ID in Annex I, refer to the float version list established by the Argo Technical Coordinator, Mathieu BELBEOCH (mbelbeoch@jcommops.org), to uniquely name each float type and version. The first version of this list, used in this document, can be accessed at https://docs.google.com/spreadsheet/ccc?key=0AitL8e3zpeffdENUQmszRlY3djYweGZhbnBZSU1fTFE&usp=sharing. If the link does not work when clicking on it, please copy and paste it into your browser. This list was created in 2012 and accurately describes the float types available at that time. Since then, some float types have been updated while others have not. If a float type is missing from the table, please contact M. Belbeoch to let him know. The useful information list has been established by Megan SCANDERBEG (mscanderbeg@ucsd.edu) in the framework of the project of providing delayed mode trajectory data to Argo users.

Many of the concerned float versions are obsolete for real time processing. However, even if these floats are no longer active, it is important to collect and transmit how to decode the float data for historical purposes.

The float types presented in this document include:

- The PROVOR float including PROVOR, PROVOR-MT and ARVOR floats in their Argos and Iridium versions
- APEX
- NINJA, Deep NINJA
- SOLO
- SOLO-II
- NEMO
- NAVIS
- NOVA

1.1 Cook book usage and update

The ANNEX I entry point of this cookbook is based on the AIC float version list and measurement codes list (from trajectory file format 3.1).

Each new float version must be fully studied, decoded and the results analyzed (in a "trajectory" point of view) before being added in the ANNEX I: Cookbook entry point tables.

These detailed entry points provide each DAC with the ability to homogeneously process NetCDF TRAJ file contents no matter whether or not the DAC has prior knowledge about a float's trajectory data.

For many float types, the ANNEX I tables are not up to date and should be used with caution. The PROVOR and ARVOR tables are updated through 2014 and are the most reliable.

For float types other than PROVOR and ARVOR, it is suggested to search for the float type under section 3.2.2 which details how to fill in the cycle timing variables by float type. PROVOR and ARVOR floats are included here as well and provide accurate information.

1.2 Real time TRAJ file expected contents

Three main types of data are expected to be stored in the real time NetCDF TRAJ files:

- Surface fixes: Argos or GPS locations of the float surface trajectory,
- Cycle timings: The dated main cycle events associated (when available) with their CTD measurements,
- Other CTD measurements; they can be:
    - Drift phase CTD measurements: possibly dated CTD measurements sampled during the drift phase at parking depth (used to determine the deep displacement immersion),
    - Miscellaneous CTD measurements: depending on float versions, many useful CTD measurements can be provided by the instruments; in particular measurement that help to define the vertical movements of the float and the associated rates.

There will now be up to two types of real time TRAJ files – a core-Argo trajectory file and a B-Argo trajectory file. The core-Argo trajectory file contains the CTD sensor parameters (pressure, temperature, salinity, conductivity) that are measured outside the vertical profiles. Additional parameters from other sensors are stored in a B-Argo trajectory file. The B-Argo trajectory file is very

similar to the core-Argo trajectory file and will include all the same cycle timings and surface fixes. Just the parameters will be different.

1.2.1 Duplicated times

The cycle timing dates must be duplicated in the TRAJ files. They should be stored in the N_CYCLE arrays **and** in the N_MEASUREMENT arrays with the associated MEASUREMENT_CODE value. All Primary and Secondary Measurement Code (MC) events (see Annex F) that **are experienced by the float** are required to be present in the N_MEASUREMENT array and redundantly in the N_CYCLE variables. All other codes are voluntary.

If the float experiences an event **but the time is not able to be determined**, then a *_STATUS = '9' is used. This indicates that it might be possible to estimate in the future and acts as a placeholder.

In the N_CYCLE variables, if the float does not experience an event then *_STATUS = 'FillValue' is used. Only events that are experienced by a float are recorded in the N_MEASUREMENT array so status='FillValue' is not used in those variables.

1.2.2 Data resolution

The decoded data can sometimes have unusual resolutions depending on the measurement code. Store the nominal resolution in the <PARAM>:resolution attribute. If the resolution differs by measurement code, provide this information in the "COMMENT_ON_RESOLUTION" attribute for the concerned variable and as a global attribute.

See section 2.3.5.1 of the Argo User's Manual for a more detailed example.

Examples of differing resolutions are as follows:

- Dates: some times can have a 1 minute or a 6 minutes resolution,
- Pressures: the PROVOR technical and spy pressures are given in bars, this is also the case for APEX descending pressure marks.

1.2.3 Cycle number management in RT TRAJ

Cycle numbers

A cycle is defined as a series of actions made by a float and includes either a descending profile or an ascending profile (or, rarely, both); it may also include immersion drift or surface drift. An Argo cycle starts with a descent toward deep water, usually from the surface. It ends with the next ascent to shallow water and data transmission (in some situations or for some floats, data transmission may not always occur). Each cycle of a float has a unique cycle number, increased by one after each ascent to shallow water. For most floats, this will be the cycle number transmitted by the float. In some cases, this number will need to be calculated by the operator. Simple checks on cycle number can be performed in real time.

For floats that provide cycle number, DACs should compare the provided cycle number with the expected cycle number. If they agree, the provided cycle number will be stored in CYCLE_NUMBER and CYCLE_NUMBER_INDEX variables. If they disagree, cycle number should be computed to be coherent with time versus cycle duration. Care should be taken not to overwrite a current cycle.

For the other floats, cycle number should be computed to be coherent with time versus cycle duration. These cycle numbers should be stored in CYCLE_NUMBER in real time.

Both CYCLE_NUMBER and CYCLE_NUMBER_INDEX need to be filled in real time. The cycle number in CYCLE_NUMBER must match the profile cycle number, which is the number recorded in the CYCLE_NUMBER variable in the profile file. If a mismatch is detected between a trajectory cycle number and a profile cycle number, the trajectory cycle number must be changed to match the profile file cycle number and replaced on the GDAC.

CYCLE_NUMBER_INDEX indicates which cycle number information is contained in that index of the N_CYCLE array. For example, CYCLE_NUMBER_INDEX(4)=3 means the 4th element of all N_CYCLE variables is associated with the WMO_003.nc profile file. Additionally, all the elements of the N_MEASUREMENT variables for which CYCLE_NUMBER = 3 are likewise associated with the 4th N_CYCLE elements and with the WMO_003.nc profile file. This stops confusion over which index in the N_CYCLE array corresponds to which cycle number in the N_MEASUREMENT array.

The CYCLE_NUMBER_ADJUSTED and the CYCLE_NUMBER_INDEX_ADJUSTED variables will contain a cycle numbering which has been assessed and may be adjusted to be correct, especially for the purpose of trajectory calculations.

If a cycle is recovered during delayed mode, DACs must choose to either (a) create a new profile file and renumber all profile files accordingly and then rewrite the trajectory file with the changes in CYCLE_NUMBER & CYCLE_NUMBER_INDEX to match the profile files OR (b) not create a new profile file and add the new cycle into the CYCLE_NUMBER_ADJUSTED and CYCLE_NUMBER_ADJUSTED_INDEX variables. Two examples of case (b) are below.

The first example is where cycle number 5 is recovered either in delayed mode. The cycle number variables must be rewritten as follows:

CYCLE_NUMBER	1, 2, 3, 4, _, 6, 7, 8, 9, 10, 11,… ,
CYCLE_NUMBER_INDEX	1, 2, 3, 4, _, 6, 7, 8, 9, 10, 11,…,
CYCLE_NUMBER_ADJUSTED	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, _
CYCLE_NUMBER_ADJUSTED_INDEX	1, 2, 3, 4, 5, 6, 7, 8 9, 10, _

Here, FillValue is added to CYCLE_NUMBER and CYCLE_NUMBER_INDEX to indicate that no profile files exist with cycle number 5. The trajectory file must be rewritten to add in the new cycle number information and any other information recovered for that profile.

A second example of errors that might be discovered in cycle number in delayed mode involves floats that do not send cycle number and for which cycle number must be calculated. In this situation, there are times when cycle numbers are incorrectly skipped. Here, cycle number 5 was incorrectly skipped in real time and added back in delayed mode:

CYCLE_NUMBER	1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12,…
CYCLE_NUMBER_INDEX	1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12,…
CYCLE_NUMBER_ADJUSTED	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, _, _
CYCLE_NUMBER_ADJUSTED_INDEX	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, _, _

Missing cycles

A cycle is defined as a series of actions, including collection of data, made by a float that ends with transmission of data, or the attempt to transmit data. If the float fails to collect or transmit data, a cycle has not occurred and can be defined as missing.

Missing cycles should NOT be stored in the TRAJ file. No place holders are necessary and will not work with the new TRAJ file.

1.2.4 Clock offset

Some Argo float versions provide times for dated events or dated measurements. Over time, the float's clock may drift. Clock drift can be defined as the drift of the clock in hours/ minutes/ seconds per year. To correct for this, we must apply a clock offset where clock offset is defined as a measurement, done at a given time, of the offset of the clock due to clock drift. Thus a clock offset should be estimated for each of these float times.

Note that clock offset can also embrace a clock that has not been correctly set or a clock that has been set in local time. Of course, in these cases, clock offset is not only revealing a drift of the float clock...

Float clock offset is defined as: Float clock offset = Float time - UTC time.

A good estimate of the clock offset can be obtained when the float transmits its Real Time Clock (RTC) time in the technical data. It can then be compared to the time from Argos of the corresponding message to compute a clock offset for all the float times of the concerned cycle.

Unfortunately this is not always the case, some floats do not transmit their RTC time and even if they do, this RTC time is not always received.

Here are some remarks on RTC time transmitted by Argo float versions:

- For APEX Argos floats: the float times come from float versions which send the RTC time only once in the test message (thus around the launch time),
- For APEX Iridium floats: the float times are given with the RTC time for every cycle, thus they can always be corrected,
- For PROVOR and ARVOR Argos floats: the RTC time is in the technical message. If we do not receive the technical message, the float times cannot be computed for this cycle. Thus, when we cannot compute the clock offset, there are no float times to correct from clock offset,
- For PROVOR Iridium floats: the RTC is set each cycle (using GPS time). Thus clock offset is considered to be equal to zero,
- For NINJA Argos floats: the RTC time is provided each cycle but the corresponding message can be missing (not received),
- For SIO and WHOI SOLO floats: no float time is transmitted, but the SIO float clock is reset each time, making clock offset essentially non-existent.
- For SOLO-II: float time is transmitted. In addition the float clock is reset each surfacing, making clock offset essentially non-existent.
- For NEMO Argos floats: the decoding has been done by Optimare and we do not know how they manage clock offsets.
- For NEMO Iridium floats: the RTC time is in the technical message. If we receive the technical message, we can correct RTC time by using GPS time.

1.2.4.1 How to put clock offset into trajectory file in real time

If a float can be corrected for clock offset in real time, DACs should determine the drift and adjust the time (inclusive of adjustment of zero). The corrected time should go in the JULD_ADJUSTED (N_MEASUREMENT) variable.

The JULD_ADJUSTED_STATUS should be set to "3" if the clock offset is computed from the RTC time.

The JULD_ADJUSTED_QC should also be filled.

Simultaneously, the DATA_MODE should be marked as "A" indicating an adjusted float, and the CLOCK_OFFSET (N_CYCLE) variable should be appropriately filled.

If a float cannot be corrected for clock offset in real time, the JULD_ADJUSTED* variables and the CLOCK_OFFSET variable should all be fill value.

1.2.4.2 How to put clock offset in trajectory file in delayed mode

If the float is corrected for clock offset in delayed mode, the corrected time should go in the JULD_ADJUSTED (N_MEASUREMENT) variable.

The JULD_ADJUSTED_STATUS should be set to "3" if the clock offset is computed from the RTC time or to "1" if it is estimated using information sent by the float or if it is estimated using procedures that rely on typical float behavior.

The JULD_ADJUSTED_QC should also be filled.

The clock offset itself goes in the CLOCK_OFFSET(N_CYCLE) variable and the DATA_MODE should be marked as "D" indicating a delayed mode correction.

If a float cannot be corrected for clock offset in delayed mode, the CLOCK_OFFSET variable should be fill value. The JULD_ADJUSTED* variables may be filled if other estimates are done on the timing information not related to clock offset.

2 Trajectory files

2.1 Surface fixes

2.1.1 Launch position and time

The launch position and time values should be duplicated from the META file to the TRAJ file.

They should be stored as the first LATITUDE, LONGITUDE and JULD of the N_MEASUREMENT array with:

- CYCLE_NUMBER = -1,
- POSITION_QC = 0,
- POSITION_ACCURACY = _FILLValue,
- MEASUREMENT_CODE = 0
- JULD_STATUS = 4 - determined by satellite.

The launch time should be as reliable as possible (because it is used in Argos surface location selection, see §3.1.3). Therefore, methods, based on transmitted information, can be used to check the launch time. Once the launch position has been checked, its QC should be set to 1.

2.1.2 For Argos APEX floats

Argos Apex floats send the information "Time from startup" in the test message. This information can be used to compute the STARTUP_DATE (in the metafile) of the float (using the time of the Argos message used for "Time from startup" information decoding).

For APF9a/t floats, all firmware revisions are capable of self-activation via the pressure-activation mechanism. For floats that self-activate, the “time from startup” is AFTER launch time. For floats that were manually started while still on-board the ship, the start-time is before launch time.

2.1.3 For APF8 floats, some firmware revisions had the pressure-activation mechanism while others did not. This means the "time from startup" can either be before launch time, if manually started, or after launch time if self-activated.Argos surface locations

All Argos surface locations provided by CLS and **occurring after the launch time** should be stored in the TRAJ file.

The concerned data:

- Location time,
- Location latitude and longitude,
- Location class (POSITION_ACCURACY),
- Satellite name,
- Error ellipse parameters (when/if available).

should be stored with their **full resolution** (*some DACs don't store the seconds of the location time*).

However, the DACs should filter the received locations so that only one surface location is preserved for a given time and a given satellite pass. Criteria to be used are:

1. The one with the better location class,
2. The one with the better error ellipses characteristics (*to be defined*),
3. The one computed from the longest satellite pass,
4. The one from the more recent download from CLS,
5. The one that succeeds to the test #20 "Questionable Argos position test" described in the Argo quality control manual (http://www.argodatamgt.org/content/download/341/2650/file/argo-quality-control-manual-V2.7.pdf) and detailed in ANNEX G: Implementation of the JAMSTEC trajectory quality control method.

NOTE: Some Argos locations can be computed twice by CLS in (near) real time and the improved results sent again. The Argos location data set of a given cycle should be updated from all CLS incoming data at least 2 days after the theoretical end of the Argos transmission (*cf. ADMT12 action#53*).

The real time quality control test #20 (Questionable Argos Position test, also in ANNEX G: Implementation of the JAMSTEC trajectory quality control method) should be used on surface Argos locations to define the position QCs.

2.1.4 Iridium/GPS surface locations

For Iridium floats, it is best to use the GPS position and time as the position fixes (LATITUDE and LONGITUDE) and JULD_LOCATION when available. When a GPS fix is not available, no position or time should be included in the trajectory file.

All the GPS positions should be stored in the TRAJ file with a measurement code of 703

For APEX floats

GPS locations provided in log file **and** message files should be merged. One suggested way to find the fixes for APEX 001087 floats, is to parse the log file using:

GPS_FIX = ['GpsServices() Profile ' sprintf('%d', a_cycleNum) ' GPS fix obtained in'];

For PROVOR floats

The time of the GPS position provided in the technical message is the float's time and date (also provided in this technical message).

2.2 How to calculate cycle timing variables

Each Argo float cycle is composed of programmed events. Depending on float type, some of these events can be dated and associated CTD measurements can be provided. The following figure shows an example cycle, with the times ordered for Argos satellite communications. For Iridium floats, the order of surface events may be different.

The sixteen following timed events can be highlighted.

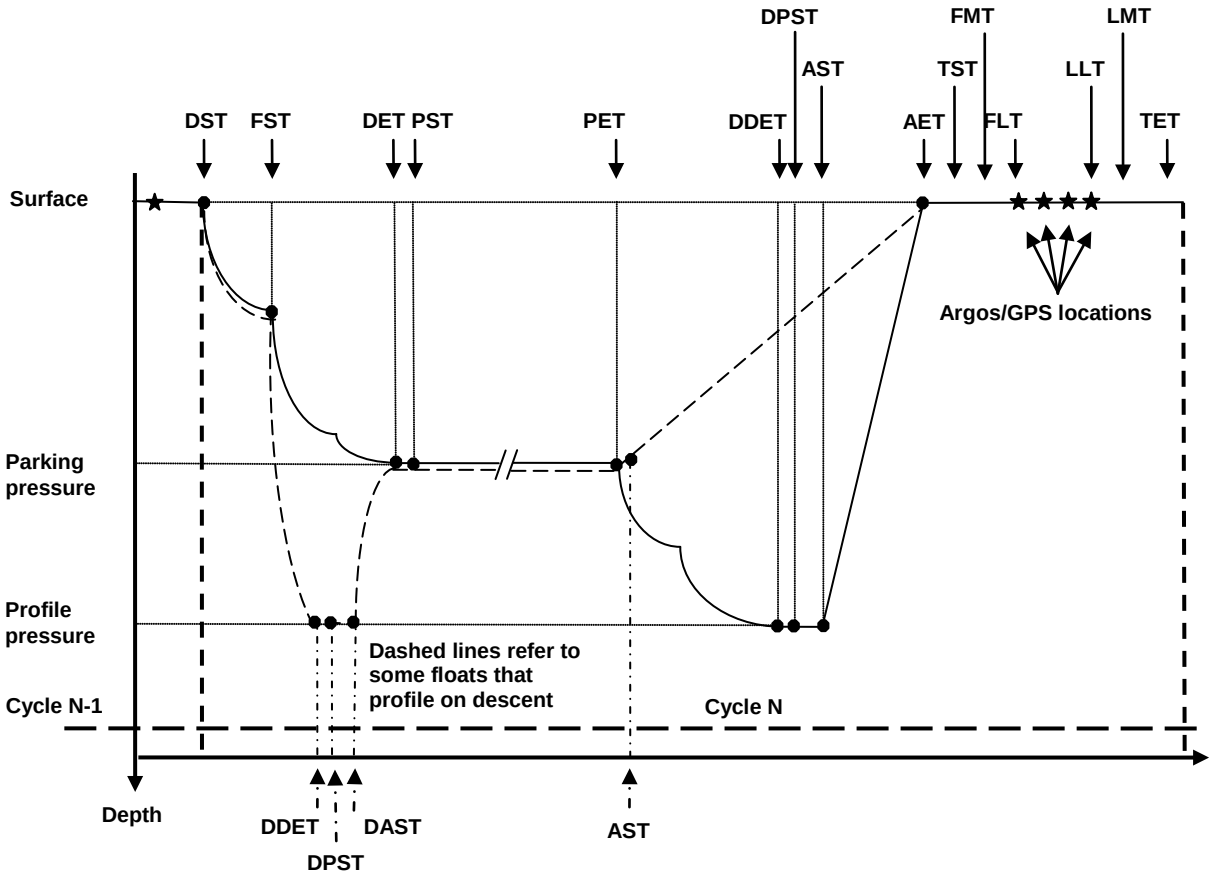


Figure 1: Figure showing float cycle and the cycle timing variables. Floats can profile either on descent or ascent. Most floats profile on ascent. Their path is shown with a solid black line. Some floats profile on descent. One such float, the new SOLO-II Deep float, has a cycle as shown by the dashed line.

Floats that profile on ascent would have the following primary cycle timings:

DST, DET, PET, DDET, AST, AET, TST, all surface times and TET

Floats that profile on descent might have the following cycle timings:

DST, DDET, DAST, DET, PET, AST, AET, TST,all surface times and TET

NOTE: if a float is programmed to experience a primary cycle timing event, but no timing information is sent back and no estimate is possible, fill value should be inserted in the JULD array with the measurement code corresponding to the primary cycle timing event. Examples of this include:

- SOLO and APEX APF8 floats which send back no cycle timing information should have fill value for the primary measurement codes unless an estimated time can be determined

- Ice detection floats that detect ice at the surface and then do not surface should have fill value for measurement codes 600-704 for the affected cycles.

NOTE: the diagram above shows the chronological order of timing events for an Argos float. Iridium floats have a different chronological order of timing events. In either case, times in the JULD variable should be arranged chronologically. The only exception to this if a clock offset has been applied and then the JULD variable may have an inversion, but the JULD_ADJUSTED variable must be arranged chronologically.

Time	MC	Long name	N_CYCLE variable name	Description
DST	100	Descent Start Time	JULD_DESCENT_START JULD_DESCENT_START_STATUS	Time when float leaves the surface, beginning descent.
FST	150	First Stabilization Time	JULD_FIRST_STABILIZATION JULD_FIRST_STABILIZATION_STATUS	Time when a float first becomes water-neutral.
DET	200	Descent End Time	JULD_DESCENT_END JULD_DESCENT_END_STATUS Note: Float may approach drift pressure from above or below.	Time when float first approaches within 3% of the eventual drift pressure. Float may be transitioning from the surface or from a deep profile. This variable is based on pressure only and can be measured or estimated by fall-rate. In the case of a float that overshoots the drift pressure on descent, DET is the time of the overshoot.
PST	250	Park Start Time	JULD_PARK_START JULD_PARK_START_STATUS	Time when float transitions to its Park or Drift mission. This variable is based on float logic based on a descent timer (i.e. SOLO), or be based on measurements of pressure (i.e. Provor).
Note on DET and PST: DET and PST might be near in time or hours apart depending on float model and cycle-to-cycle variability. PI has judgment call whether DET~=PST.				
PET	300	Park End Time	JULD_PARK_END JULD_PARK_END_STATUS	Time when float exits from its Park or Drift mission. It may next rise to the surface (AST) or sink to profile depth (DDET)
DDET	400	Deep Descent End Time	JULD_DEEP_DESCENT_END JULD_DEEP_DESCENT_END_STATUS	Time when float first approaches within 3% of the eventual deep drift/profile pressure. This variable is based on pressure only and can be measured or estimated by fall-rate.
DPST	450	Deep Park Start Time	JULD_DEEP_PARK_START JULD_DEEP_PARK_START_STATUS	Time when float transitions to a deep park drift mission. This variable is only defined if the float enters a deep drift phase (i.e. DPST not defined in cases of constant deep pressure due to bottom hits, or buoyancy issues)
DAST	550	Deep Ascent Start Time	JULD_DEEP_ASCENT_START JULD_DEEP_ASCENT_START_STATUS	Time when float begins its rise to drift pressure. Typical for profile-on-descent floats.
AST	500	Ascent Start Time	JULD_ASCENT_START JULD_ASCENT_START_STATUS	Time when float begins to return to the surface.
AET	600	Ascent End Time	JULD_ASCENT_END JULD_ASCENT_END_STATUS	Time when float reaches the surface.
TST	700	Transmission Start Time	JULD_TRANSMISSION_START JULD_TRANSMISSION_START_STATUS	Time when float begins transmitting.
FMT	702	First Message Time	JULD_FIRST_MESSAGE JULD_FIRST_MESSAGE_STATUS	Earliest time of all messages received by telecommunications system
FLT	703	First Location Time	JULD_FIRST_LOCATION JULD_FIRST_LOCATION_STATUS	Earliest location of all float locations.
LLT	703	Last Location Time	JULD_LAST_LOCATION JULD_LAST_LOCATION_STATUS	Latest location of all float locations.
LMT	704	Last Message Time	JULD_LAST_MESSAGE JULD_LAST_MESSAGE_STATUS	Latest time of all messages received by telecommunications system
TET	800	Transmission End Time	JULD_TRANSMISSION_END JULD_TRANSMISSION_END_STATUS	Time when floats stops transmitting.

Table 1: Descriptions of cycle times shown in the previous figure

All these times are in both the N_MEASUREMENT and the N_CYCLE variable groups of the TRAJ file. These times should be included in chronological order in both the cases. This means events may not occur in the same order as in the table above, as it is developed around an Argos float. For an Iridium float with a GPS fix taken before starting Iridium transmission, the measurement codes at the surface might look like this: 703, 700, 702, 704, 800

The main times of a cycle can be separated in two parts:

- Positioning and transmission system times: FMT, FLT, LLT and LMT,
- Times of float events: the other ones.

2.2.1 Positioning system and transmission system times

The FMT, FLT, LLT and LMT times only depend on the positioning system and the transmission system used by the float. Additionally, the order these times and positions occur in chronologically depends on the system being used. The order is completely different for Argos than for Iridium and in some cases of Iridium usage, there is only a single position and time fix during the entire surfacing period. So, in that situation, all the times will be the same.

2.2.1.1 For Argos floats

See Annex A for Argos message time and Argos location time illustration as received from CLS. Status variables should be a "4 (value is determined by satellite)".

2.2.1.1.1 First and last message times

All Argos message times should be collected and the maximum and minimum values stored as FMT and LMT (*we cannot assume that data are received from CLS in chronological order*).

If only one message has been received for a given cycle, its time should be duplicated in FMT and LMT.

NOTE: Some DACs already store FMT and LMT in the TRAJ file but some of them are erroneous because they correspond to ghost Argos messages.

As reliable FMTs and LMTs are crucial for other times estimation (such as APEX DST), we must think of a robust method to reject these ghost messages in real time.

The best method for now is for floats which use a CRC in their Argos messages, to use in FMT and LMT only Argos messages that passed the CRC check. *From AOML: [For known cycle times one can use that information plus an analysis of all cycles in the raw data to identify ghosts. We developed a program for this, but are not yet using it in operations. We could share that program with others once we are convinced it works.]*

2.2.1.1.2 First and last location times

All Argos location times should be collected and the maximum and minimum values stored as FLT and LLT (*we cannot assume that data are received from CLS in chronological order*).

If only one location has been computed for a given cycle, its time should be duplicated in FLT and LLT.

2.2.1.2 For Iridium floats

The chronological order of the events on the surface does not follow the numerical order from the measurement code table which was designed more with the Argos system in mind. For example, usually the GPS fix comes first and then the float begins transmitting to Iridium. Often several

messages are sent to and received from Iridium and then the float stops transmitting. For this situation, following the chronological order of the times, the measurement codes would be 703, 700, 702, 704, 800. .

2.2.1.2.1 First and last message times

For NEMO and SOLO floats

Use the "Time of Session" information, provided in all the Iridium e-mails received for each cycle, as the float message time.

For NAVIS floats

For Iridium, there are two values transmitted that replace the Argos transmission times. When the float reaches the surface, it acquires a GPS position. The time to do this is represented by TTFF (in seconds). After the GPS is acquired, then the Iridium transceiver is activated. The SBDT is the transmission time of the first Iridium packet (housekeeping packet). This is to give an indication of the transmission throughput as the housekeeping is a constant size as opposed to the other packets. After completion of the transmission, a satellite check is done to look for incoming commands. If there is one, it is processed and then the float starts its next profile. Note that SBDT refers to the previous profile, not the current one, as it is calculated AFTER the Iridium transmission takes place.

First Message Time is TST + TTFF.

Last Message Time is the same as LLT.

For NOVA floats

For Iridium, there are two values transmitted that replace the Argos transmission times. When the float reaches the surface, it acquires a GPS position. The time to do this is represented by TTFF (in seconds). After the GPS is acquired, then the Iridium transceiver is activated. The SBDT is the transmission time of the first Iridium packet (housekeeping packet). This is to give an indication of the transmission throughput as the housekeeping is a constant size as opposed to the other packets. After completion of the transmission, a satellite check is done to look for incoming commands. If there is one, it is processed and then the float starts its next profile. Note that SBDT refers to the previous profile, not the current one, as it is calculated AFTER the Iridium transmission takes place.

First Message Time is TST + TTFF.

Last Message Time is the same as the FMT - there is only one GPS fix.

For PROVOR and APEX floats

For Iridium SBD floats, use the "Time of Session" information, provided in all the Iridium e-mails received for each cycle, as the float message time.

For Iridium RUDICS floats, use the packet transmission times as the float message times

2.2.1.2.2 First and last location times

2.2.2 Times of float events

Each float type, and sometimes each model of float type, has different instructions on how to fill in the timing variables in the trajectory file. Remember that all mandatory cycle timing variables must be filled and are in both the N_MEASUREMENT and N_CYCLE arrays. If it is not possible to fill this time, even by an estimation, fill value must be used in both arrays.

2.2.2.1 APEX floats with the APF8 controller board

The cycle timing information transmitted by APEX floats with the APF8 controller board is limited, so **no event times are directly available**.

To compute or estimate the cycle times for APF8 floats, we must use "external" methods based on the float functioning. Some of the same methods can be used for the APF9a or APF9t floats, but should not replace the transmitted times.

Two main methods have developed (from work done for ANDRO) to be efficient and may be robust enough to be implemented in real time.

- The first one, based on float functioning, can be used to estimate TET,
- The second one, based on float transmission strategy, can be used to compute TST.

The first method to compute TET seems less robust for real time application and some DACs may choose not to estimate TET in real time. If this is the case, then the delayed mode operator may choose to estimate a TET in delayed mode where more time can be spent visually inspecting each float's TET estimate. Even though this is a mandatory time, if the DAC or delayed mode operator feels that no time can be estimated accurately enough in delayed mode, the time should be left as fill value.

The second method, based on float transmission strategy, relies on the raw Argos messages the DAC receives. This method is also an estimate, but it is currently being implemented in some manner at the DACs. There are a few different methods to make the estimate and thus, DACs may do it differently. This document is including the recommended method that improves on the one from TWR.

Other event times can be (roughly for some of them) estimated from TET or TST and float parameters and/or in situ data statistical results.

This is the case for:

- DST, DET and PET which are determined from TET,
- AET and AST which are determined from TST.

Again, if the TET is fill value in real time or delayed mode, then DST, DET and PET will be also fill value. This is determined by the DACs and delayed mode operator.

If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET-(N_CYCLE) variable and the JULD_ADJUSTED variables so users know it has been applied.

Argo program measurement codes (MC) for APEX APF8 floats in REAL TIME				
Code (timing)	**APF8 Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or

	from the ship, enter that time and location If the launch time and location are not available, use fill value.			9: value is not immediately known, but believe it can be estimated later
100 (DST)	TET from previous cycle OR Fill Value	If TET is estimated in real time, use the TET from previous cycle. OR If TET is not estimated in real time, use FillValue	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
200 (DET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
250 (PST)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
During the drift phase, the APF8 makes drift measurements. Common codes are listed below. See 3.4.1.1 for CTD measurements during drift for APEX floats				
296	Average pressure Average temperature	Any averaged measurements made during drift	Pressure Temp	9: value is not immediately known, but believe it can be estimated later
297	Minimum pressure Minimum temperature	Minimum value taken during drift	Pressure Temp	9: value is not immediately known, but believe it can be estimated later
298	Maximum pressure Maximum temperature	Maximum value taken during drift	Pressure Temp	9: value is not immediately known, but believe it can be estimated later
End of drift measurements				
300 (PET)	Not available, so use Fill Value CTD performed at end of drift		Time P, T, S	9: value is not immediately known, but believe it can be estimated later
301	Average pressure during drift	Best estimate of drift depth. See section 3.4.3 for more details	Pressure	9: value is not immediately known, but believe it can be estimated later
400 (DDET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
500 (AST)	If PARK and PROFILE depths are equal and TET is estimated in real time: AST(i)=TET(i) – UP TIME OR FillValue	See 3.2.2.1.7	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
501	DownTimeEpoch/UNIX epoch when the down-time expired	Down-time end time – time out	Time	2: value is transmitted by the float
502	Time of profile initiation provided in auxiliary engineering data See 2.2.2.1.7.2 AST_{FL} = $DTET_{FL}$ + TPI minutes		Time	3: value is directly computed from relevant, transmitted float information
600 (AET)	Float does not know when it reaches the surface, so Fill Value		Time	9: value is not immediately known, but believe it can be estimated later

602	Time of MC=701 minus 10 minutes		Time	3: value is directly computed from relevant, transmitted float information
700 (TST)	See section 3.2.2.1.9 & 6.2	Based on Argos messages	Time	3: value is directly computed from relevant, transmitted float information
701 TST sent by APEX floats	TST_{FL} = $DTET_{FL}$ + TOTPI minutes	See 3.2.2.1.9.2	Time	3: value is directly computed from relevant, transmitted float information
702 (FMT)	Earliest time of all Argos messages received	Time	Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	3.2.2.1.1 and Annex B (5.3) OR FillValue	DACs can choose to make this estimate in real time or not. Annex B explains how to make the estimate. 3.2.2.1.1 gives guidance how to implement the method in Annex B	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later

Argo program measurement codes (MC) for APEX APF8 floats in DELAYED MODE				
Code (timing)	**APF8 Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available from the ship, enter that time and location If the launch time and location are not available, use fill value.	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or 9: value is not immediately known, but believe it can be estimated later
100 (DST)	TET from previous cycle OR Fill Value	If TET is estimated in delayed mode, use the TET from previous cycle. OR If TET is not estimated in real time, use FillValue	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
200 (DET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
250 (PST)	PST estimated from 13.1 OR FillValue	See 13.1 for details	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
During the drift phase, the APF8 makes drift measurements. Common codes are listed below. See 3.4.1.1 for CTD measurements during drift for APEX floats				
296	Average pressure Average temperature	Any averaged measurements made during drift	Pressure Temp	9: value is not immediately known, but believe it can be estimated later
297	Minimum pressure	Minimum value taken	Pressure	9: value is not immediately

	Minimum temperature	during drift	Temp	known, but believe it can be estimated later
298	Maximum pressure Maximum temperature	Maximum value taken during drift	Pressure Temp	9: value is not immediately known, but believe it can be estimated later
End of drift measurements				
300 (PET)	PET estimated from 13.2 OR Fill Value CTD performed at end of drift	See 13.2 for details on how to estimate PET	Time P, T, S	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
301	Average pressure during drift	Best estimate of drift depth. See section 3.4.3 for more details	Pressure	9: value is not immediately known, but believe it can be estimated later
400 (DDET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
500 (AST)	If PARK and PROFILE depths are equal and TET is estimated: AST(i)=TET(i) – UP TIME OR AST estimated from 13.3 OR FillValue	See 3.2.2.1.7 OR 13.3	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
501	DownTimeEpoch/UNIX epoch when the down-time expired	Down-time end time – time out	Time	2: value is transmitted by the float
502	Time of profile initiation provided in auxiliary engineering data. See 2.2.2.1.7.2 AST_{FL} = $DTET_{FL}$ + TPI minutes		Time	3: value is directly computed from relevant, transmitted float information
600 (AET)	Float does not know when it reaches the surface, so Fill Value		Time	9: value is not immediately known, but believe it can be estimated later
602	Time of MC=701 minus 10 minutes		Time	3: value is directly computed from relevant, transmitted float information
700 (TST)	See section 3.2.2.1.9 & 6.2	Based on Argos messages	Time	3: value is directly computed from relevant, transmitted float information
701 TST sent by APEX floats	TST_{FL} = $DTET_{FL}$ + TOTPI minutes	See 3.2.2.1.9.2	Time	3: value is directly computed from relevant, transmitted float information
702 (FMT)	Earliest time of all Argos messages received	Time	Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	3.2.2.1.1 and Annex B (5.3)	Delayed mode operators can choose	Time	1: value is estimated using information not transmitted by

	OR FillValue	to make this estimate in real time or not. Annex B explains how to make the estimate. 3.2.2.1.1 gives guidance how to implement the method in Annex B		the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later

2.2.2.1.1 Transmission End Time determination – APEX APF8 floats

The TET can be estimated with the method proposed in §5.3. In this Annex B, two methods are proposed for finding TET depending on whether or not clock offset has been estimated. In order to ensure that these estimation methods are as robust as possible, the metadata information going into them (cycle time, whether the float is a Deep Profile First float, etc) must be correct. A list has been compiled by J.P. Rannou of corrected meta data for floats used in the ANDRO Atlas work. This file, found at ftp://ftp.ifremer.fr/ifremer/argo/etc/coriolis-custom/argo-andro-data/metadata_admt14/) (see http://www.argodatamgt.org/Documentation/Delayed-mode-trajectories-recovered-from-Andro-project for details), may be a good place to start to confirm the metadata for many APEX floats. **If the DAC and/or delayed mode operator wishes to estimate the TET, thefollowing steps should be taken:**

- Correct clock drift at launch for clocks which have not been correctly set,
- Estimate TET using the first algorithm (without clock drift estimation) for the first 32 cycles,
- From cycle #33, estimate the clock drift:
    - If it is less than 20 minutes per year, estimate TET using the second algorithm (with clock drift estimation) for all float cycles,
    - If it is greater than 20 minutes per year, there was an unexpected behavior of the float and the TET should not be estimated.

For more details, refer to Annex B.

Regardless of whether clock offset has been estimated during the TET determination, the resulting values should be stored in the JULD_ADJUSTED variable in the N_MEASUREMENT array with the measurement code set to 800 and STATUS set to 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behavior

N_CYCLE arrays: TET value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 1.

If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

If the DAC and/or delayed mode operator does not choose to estimate the TET, then fill value should be inserted into the JULD variable in the N_MEASUREMENT array with the measurement code set to 800 and the STATUS set to 9.

N_CYCLE arrays: fill value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 9.

2.2.2.1.2 Descent Start Time determination – APEX APF8

DST = TET from previous cycle for all APEX floats.

2.2.2.1.3 Descent End Time determination – APEX APF8

APEX floats do not measure or estimate pressure on their descent, so the time when the float first approaches within 3% of the eventual drift pressure cannot be calculated. Nothing is entered into the JULD variable in the N_MEASUREMENT array because the float cannot measure or estimate the pressure on descent.

If, during delayed mode processing, it is determined that the float overshoots the drift pressure on descent, DET is the time of the overshoot. This time can be entered into the JULD variables in the N_MEASUREMENT array with an MC=200 and a STATUS equal to 2: value is transmitted by the float.

2.2.2.1.4 Park Start Time determination – APEX APF8

For non APF9 APEX floats, the PST can be **roughly** estimated using the DST and the duration of the descent to PARKING depth. One way to estimate the PST is to use the mean descent rate as described in Annex J. DACs and delayed mode operators can choose if they want to use this method to estimate the PST, but it should be done in delayed mode.

If estimated in delayed mode, the PST value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 1 (estimated using procedures that rely on typical float behavior).

If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

In real time or if no estimate is made, fill value should be stored in the JULD variable in the N_MEASUREMENT array with the measurement code set to 250 and the STATUS set to 9.

N_CYCLE arrays: fill value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 9.

2.2.2.1.5 Park End Time determination – APEX APF8

For Argos APEX floats, PET can be computed from TET. This is not done in real time, but can be estimated later. See Annex J for more details.

2.2.2.1.6 Deep Descent End Time determination – APEX APF8

For Argos APEX floats, DDET could be estimated from PET and the mean descent velocity estimated for DET determination.

However:

- Deep descent velocity is not necessarily the same as the mean velocity between the surface and the PARKING depth,
- We have no in situ pressure measurements between PET and DDET from APEX floats,
- DDET is not as important as DET.

Consequently, DDET should not currently be required to be estimated for APEX Argos floats. However, because DDET might be estimated at a later date, fill value should go in the N_MEASUREMENT array with an MC = 400 and STATUS code of 9: value is not immediately known, but believe it can be estimated later

For the N_CYCLE array, JULD_DEEP_DESCENT_END = fill value as does JULD_DEEP_DESCENT_END_STATUS.

.

2.2.2.1.7 Ascent Start Time determination – APEX APF8

2.2.2.1.7.1 Argos APEX floats that do not provide this time

If the PARKING and PROFILE depths are equal for cycle #i, then:

AST(i) = TET(i) - UP TIME

If not, we can however **roughly** estimate AST using AET and the profile duration. See Annex J for more details.

If estimated, the AST value should also be stored in the JULD_ADJUSTED variables with an MC = 500 and STATUS set to 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour. Apply clock offset if it has been determined.

For the N_CYCLE array, the AST value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 1. If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

If the AST value is not estimated, fill value should be stored in the JULD variable with an MC=500 and STATUS set to 9.

N_CYCLE arrays: fill value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 9.

2.2.2.1.7.2 Ascent Start Time provided by APEX floats

Some float directly provide the time at the end of DOWN TIME period ($DTET_{FL}$).

These float versions also provide, in the Auxiliary Engineering Data (AED), the "Time of profile initiation". This information is defined as the time difference, in minutes, between profile start and end of DOWN TIME (negative for start before expiration and positive for start after expiration, thus in this latter case, necessarily when TOD feature has been set).

The Auxiliary Engineering Data are not always transmitted (depending on the remaining space in the last Argos message) but if received, this "Time of profile initiation" (TPI) can be used to compute a second value of AST provided by the float (AST_{FL}).

$AST_{FL} = DTET_{FL}$ + TPI minutes

AST_{FL} value computed from $DTET_{FL}$ (corrected from clock offset) does not need to be corrected from clock offset but the information should be set in the AST_{FL} storage.

AST_{FL} is stored in the JULD_ADJUSTED N_MEASUREMENT arrays with the MC = 502 and the STATUS equal to 3: value is computed from information transmitted by the float. Clock offset has been applied in the $DTET_{FL}$ variable.

2.2.2.1.8 Ascent End Time determination – APEX APF8

For APEX APF8 floats, the AET is not fully determined and cannot be estimated with a simple formula.

Fill value should be stored in the JULD variables in the N_MEASUREMENT array with an MC = 600 and STATUS set to 9.For the N_CYCLE array, fill value should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END set to 9.

2.2.2.1.9 Transmission Start Time determination – APEX APF8

Some APEX floats provide the time at the end of the DOWN TIME period. For these float types, it is possible to compute a TST based on the DOWN TIME and to compute a TST based on Argos transmission or GTS fixes. In real time, it is difficult to know which method to compute TST is better because there may be problems with either the onboard clock (affecting the DOWN TIME) or the actual Argos transmission (affecting the alternate method to calculate TST). Therefore, it is best to compute TST using both methods in real time (going into MC=700 and MC=701). In delayed-mode, an expert can examine the TST values and determine which is best. The best value of TST should always go in MC=700 in delayed mode. This might mean copying over the value in MC=701 if that value is determined to be better in delayed mode. Both methods are outlined below.

2.2.2.1.9.1 Argos APEX floats

The TST can be computed with the method proposed in §6.2.

The TST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 700 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

For the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 3. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.1.9.2 Transmission Start Time provided by APEX Argos floats

Some float versions directly provide the time at the end of the DOWN TIME period ($DTET_{FL}$).

If the float clock offset has been estimated during the TET determination, $DTET_{FL}$ value should first be corrected for clock offset and the information should also be set in the $DTET_{FL}$ storage.

$DTET_{FL}$ is included in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 701 STATUS set to 2: value is transmitted by the float.

These float versions also provide the time, in minutes, of telemetry phase initiation relative to $DTET_{FL}$ (TOTPI).

Thus a TST, provided by the float, can be computed: $TST_{FL} = DTET_{FL}$ + TOTPI minutes

TST_{FL} value computed from $DTET_{FL}$ (corrected for clock offset) does not need to be corrected for clock offset but the information should be set in the TST_{FL} storage.

TST_{FL} is included in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 701 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

3.2.2.2. APEX floats with the APF9a or APF9t controller

APEX floats equipped with APF9a or APF9t controller boards have begun to address the lack of transmitted cycle timing by transmitting timing information both in the mission prelude (cycle 0) and each cycle. Some of this timing information may be in the Auxiliary Engineering data. The four timing specifications that are transmitted during cycle 0 are:

- ParkDescentPeriod
- DeepProfileDescentPeriod
- DownIntervals
- UpIntervals

In addition, two time stamps are transmitted from each cycle:

- DownTimeEpoch
- TelemetryEpoch

These times should be used to calculate other cycle timing variables and stored in the TRAJ file with the appropriate *_STATUS flag to reflect that the timing information is transmitted or calculated from transmitted information.

3.2.2.2.1 Auxiliary Engineering Data (AED)

The Auxiliary Engineering Data (AED) includes the "Time of profile initiation". This information is defined as the time difference, in minutes, between profile start and end of DOWN TIME (negative for start before expiration and positive for start after expiration, thus in this latter case, necessarily when TOD feature has been set).

The AED are not always transmitted (depending on the remaining space in the last Argos message) but if received, this "Time of profile initiation" (TPI) can be used to compute the value of AST provided by the float (AST_{FL}).

AST =DownTimeEpoch(ToD for down-time expiration)minutes + TPI minutes

The AST value computed from DownTimeEpoch (corrected from clock offset) does not need to be corrected from clock offset but the information should be set in the AST storage.

AS_L is stored in the JULD_ADJUSTED N_MEASUREMENT arrays with the MC = 500 and the STATUS equal to 3: value is computed from information transmitted by the float. Clock offset has been applied in the DownTimeEpoch variable.

The descending pressure marks are also included in the Auxiliary Engineering Data and they are measured in bar. These pressure marks can be entered in the JULD (N_MEASUREMENT) variable with a measurement code of 189 or 190 and a STATUS flag of 2: value is transmitted by the float.

APEX APF9a and APF9t floats provide the time at the end of the DOWN TIME period (DownTimeEpoch or ToD for down-time expiration). For these float types, it is possible to compute a TST based on the DOWN TIME and to compute a TST based on Argos transmission (described in 6.2). In real time, it is difficult to know which method to compute TST is better because there may be problems with either the onboard clock (affecting the DOWN TIME) or the actual Argos transmission (affecting the alternate method to calculate TST). Therefore, it is best to compute TST using both methods in real time (going into MC=700 and MC=701). In delayed-mode, an expert can examine the TST values and determine which is best. The best value of TST should always go in MC=700 in delayed mode. This might mean copying over the value in MC=701 if that value is determined to be better in delayed mode.

Argo program measurement codes (MC) for APEX APF9a or APF9t				
Code (timing)	**APF9a or APF9t Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available from the ship, enter that time and location If the launch time and location are not available, use fill value.	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or 9: value is not immediately known, but believe it can be estimated later
100 (DST)	DownTimeEpoch - DownIntervals		Time	3: value is directly computed from relevant, transmitted float information
If an APEX isopycnal float				
189	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours) and following every 60 minutes	Time Pressure (bars)	3: value is directly computed from relevant, transmitted float information OR 9: value is not immediately known, but believe it can be estimated later
Endif an APEX isopycnal float				
190	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours) and following every 60 minutes. See section 2.4.2.2 for more details	Time Pressure (bars)	3: value is directly computed from relevant, transmitted float information OR 9: value is not immediately known, but believe it can be estimated later
200 (DET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
250 (PST)	DST + ParkDescentPeriod	This is a time-out value and does not indicate when the float actually stabilizes at drift pressure	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour
During the drift phase, the APF9a and APF9t floats measure time pressure and temperature hourly, but these are not reported. Instead, a statistical pack of information is sent and the following measurement codes apply:				
296	Average pressure Average temperature	Any averaged measurements made during drift	Pressure Temp	9: value is not immediately known
297	Minimum pressure Minimum temperature	Minimum value taken during drift	Pressure Temp	9: value is not immediately known
298	Maximum pressure Maximum temperature	Maximum value taken during drift	Pressure Temp	9: value is not immediately known
End of drift measurements				
300 (PET)	DownTimeEpoch – DeepProfileDescentPeriod CTD performed at end of drift		Time P, T, S	3: value is directly computed from relevant, transmitted float information
301	Average pressure during drift	Best estimate of drift depth	Pressure	9: value is not immediately known, but believe it can be estimated later

400 (DDET)	Same as AST		Time	2: value is transmitted by the float
500 (AST)	DownTimeEpoch + Time of profile initiation Sent in Auxiliary engineering data which is not always available. See section 3.2.2.2.1	Time that float actually starts ascending; Can be the same as the DownTimeEpoch if the float times out before reaching profile pressure. Otherwise, float begins to ascend as soon as profile pressure is reached	Time	2: value is transmitted by the float
501	DownTimeEpoch/UNIX epoch when the down-time expired/ToD for downtime expiration	Down-time end time – time out	Time (minutes)	2: value is transmitted by the float
600 (AET)	Float does not know when it reaches the surface, so Fill Value		Time	9: value is not immediately known, but believe it can be estimated later
602	701 – 10 minutes		Time	3: value is directly computed from relevant, transmitted float information
700 (TST)	See section 3.2.2.1.9	Based on Argos messages	Time	3: value is directly computed from relevant, transmitted float information
701 TST sent by APEX floats	TST_{FL} = $DTET_{FL}$ + TOTPI minutes	See 3.2.2.1.9.2	Time	3: value is directly computed from relevant, transmitted float information
702 (FMT)	Earliest time of all Argos messages received	Time	Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	DownTimeEpoch + UpIntervals		Time	3: value is directly computed from relevant, transmitted float information

3.2.2.3. APEX floats with the APF9i controller board with a firmware revision date before 072314

APEX floats equipped with the APF9i controller board are Iridium floats and therefore, are able to send the most timing information. Some of this timing information may be in the Auxiliary Engineering data.

For each cycle, three timing specifications are transmitted:

- DeepProfileDescentTime
- DownTime
- ParkDescentTime

In addition, time stamps are recorded and transmitted for several events from each cycle:

- All drift PRES/TEMP measurement
- Profile termination (AET)

- GPS fix

These transmitted times should be used to calculate other cycle timing variables and stored in the TRAJ file with the appropriate *_STATUS flag to reflect that the timing information is either transmitted or calculated from transmitted information.

3.2.2.3.1 Descent Start Time for APF9i

The Descent Start Time (DST) for the APF9i is defined as

DST = PST – ParkDescentTime

PST = Time stamp for the first park-level PRES/TEMP sample.

3.2.2.3.2 Ascent Start Time for APF9i

For APF9i floats the Ascent Start Time (AST) is in the log file if the log file size limit has not been reached and the verbosity configuration parameter permits. The AST of cycle #*N* is retrieved from the log file as the date of the event:

ProfileInit() PrfId:*N* Pressure:*XXX.X*dbar pTable[*YY*]:*ZZZ*dbar

The corresponding pressure (*XXX.X*dbar) is associated to AST.

If the time is in the log files, fill in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 500 and a STATUS code of 2: value is transmitted by the float.

For the N_CYCLE array, fill in JULD_ASCENT_START and set JULD_ASCENT_START_STATUS to 2.

When this time is not transmitted, then the known time of day configuration parameter and down time can be used to estimate the AST. This is:

DST + DownTime/ToD for down time expiration

When the time is estimated, the AST value should also be stored in the JULD_ADJUSTED variables in the N_MEASUREMENT arrays with an MC = 500 and STATUS set to 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour.

For the N_CYCLE array, the AST value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 1. If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

Argo program measurement codes (MC) for APEX APF9i prior to firmware revision date 072314				
Code (timing)	**APF9i Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available from the ship, enter	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or

	that time and location . If no time and location are known, use fill value.			9: value is not immediately known but may be estimated at a later date.
100 (DST)	PST - ParkDescentTime See 3.2.2.3.1		Time	3: value is directly computed from relevant, transmitted float information
If an APEX isopycnal float				
189	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours)and following every 60 minutes	Time Pressure (bars)	2: value is transmitted by the float
Endif an APEX isopycnal float				
190	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours)and following every 60 minutes. See section 2.4.2.2 for more details.	Time Pressure (bars)	2: value is transmitted by the float
200 (DET)	Time stamp for first PRES/TEMP sample within 3% of drift pressure			2: value is transmitted by the float
250 (PST)	TimeStartPark	Start of park phase	Time	2: value is transmitted by the float
290	Hourly pressure and temperature measurements taken during drift. Salinity is occasionally measured.	Series of measurements recorded during drift.	Time Pressure Temp Salinity, if measured	2: value is transmitted by the float
300 (PET)	DST + DownTime – DeepProfileDescentTime CTD performed at end of drift		Time P (bars), T, S	3: value is directly computed from relevant, transmitted float information
400 (DDET)	Same as AST	AST and DDET are the same	Time	2: value is transmitted by the float
500 (AST)	TimeStartProfile Or DST + Downtime Refer to 3.2.2.3.1 Pressure	Time that float actually starts ascending; Can be the same as the DST + DownTime if the float times out before reaching profile pressure. Otherwise, float begins to ascend as soon as profile pressure is reached	Time Pressure	2: value is transmitted by the float Or 3: value is directly computed from relevant, transmitted float information 2: value is transmitted by the float
501	DownTimeEpoch/UNIX epoch when the down-time expired/ToD for downtime expiration	Down-time end time – time out	Time (minutes)	2: value is transmitted by the float
600 (AET)	Fill value as float does not know when it reaches the surface			9: value is not immediately known, but believe it can be estimated later
703 (ST)	All GPS times and locations		Time, Position	2: value is transmitted by the float

700 (TST)	Time of first Iridium message		Time	4: value is determined by satellite
701	time stamp for profile termination	Programmed time for float to change from profile phase to telemetry phase	Time	2: value is transmitted by the float
702 (FMT)	Time of first Iridium message	Time	Time	4: value is determined by satellite
704 (LMT)	Time of last Iridium message		Time	4: value is determined by satellite
800 (TET)	DST for profile n + 1 except for floats with surface measurement phase after transmission end; Fill value for TET for those floats		Time	2: value is transmitted by float

3.2.2.4. APEX floats with the APF9i controller, firmware revision 072314

APEX floats equipped with the newest revision of the APF9i firmware (firmware revision 072314) telemeter 6 time stamps from each cycle that can be slotted directly into 6 Argo cycle timing variables.

For each cycle, the 6 transmitted time stamps are:

1. **TimeStartDescent (DST)**: This timestamp marks the initiation of buoyancy reduction with the intent to descend from the surface. This means the start of the piston retraction at the beginning of the profile cycle. The float cannot detect when it actually descends below the surface. This closely matches DST, but can be wrong in some pathological cases like when a float never descends.

2. **TimeStartPark (PST)**: This timestamp marks the phase transition from the park-descent phase to the park phase. During the park-descent phase, there is no attempt to autoballast the float – it is simply in free fall for a user-specified period of time. Therefore, there is not necessarily any connection to when the float approaches neutral buoyonacy. So it matches up with PST, but not DET.

3. **TimeStartProfileDescent (PET)**: This timestamp is defined only if the Park-n-Profile (PnP) feature is enabled. This timestamp marks the initiation of buoyancy reduction with the intent to descend from the park pressure to the profile pressure. This means the start of the piston retraction at the beginning of the profile-descent phase. The float cannot detect when it actually begins to descend. If PnP is enabled, this timestamp matches PET well.

4. **TimeStartProfile (AST)**: This timestamp marks the transition to the profile phase of the profile cycle. This means the start of the initial piston extension by a user-specified amount. The float cannot detect when it actually begins to ascend. In some pathological cases, the float may continue to descend until the ascent-control algorithm's feedback mechanism compensates for any lingering negative buoyancy. This timestamp comes closest to matching AST, but there are differences.

5. **TimeStopProfile (AET)**: This timestamp marks the termination of the profile phase. The phase transition is induced by one of two conditions: either the surface detection algorithm has returned true or else the ascent timeout period has expired. Typically this phase transition happens a few meters below the sea surface but this is not guaranteed. In unusual cases the phase transition can happen after surfacing or far below the surface. This timestamp matches AET, but it is noted that it has requirement for being at the surface.

6. **TimeStartTelemetry**: This timestamp marks the initiation of the telemetry phase of the profile cycle and it can occur while the float is still subsurface. The telemetry phase consists of well-defined telemetry cycles that are spaced apart by a user-defined period. Each cycle starts with acquisition of a GPS fix and then an attempt to upload data. Cycles continue until all data is successfully uploaded or until the telemetry phase times-out. This most closely matchs TST, but is assigned MC = 701 since it is possible to determine the TST based on the time from the satellite messages.

These transmitted information should be used to calculate other cycle timing variables and stored in the TRAJ file with the appropriate *_STATUS flag to reflect that the timing information is transmitted or calculated from transmitted information

These transmitted times should be used to calculate other cycle timing variables and stored in the TRAJ file with the appropriate *_STATUS flag to reflect that the timing information is either transmitted or calculated from transmitted information.

Argo program measurement codes (MC) for APF9i firmware revision 072314				
Code (timing)	**APF9i (firmware revision 072314) Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available from the ship, enter that time and location. If no time and location are available, use fill value	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or 9: value is not immediately available but may be estimated at a later date
100 (DST)	TimeStartDescent	Descent start time	Time	2: value is transmitted by the float
If an APEX isopycnal float				
189	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours)and following every 60 minutes	Time Pressure (bars)	2: value is transmitted by the float
Endif an APEX isopycnal float				
190	Descent() Pressure: XX.X Found in Auxiliary Engineering data, so sometimes not available every cycle	Descending CTD measurements starting at a programmed time after DST (often six hours)and following every 60 minutes. See section 2.4.2.2 for more details.	Time Pressure (bars)	2: value is transmitted by the float
200 (DET)	Time stamp for first PRES/TEMP sample within 3% of drift pressure			2: value is transmitted by the float
250 (PST)	TimeParkStart	Park start time	Time	2: value is transmitted by the float
290	Hourly pressure and temperature measurements taken during drift. Salinity is	Series of measurements recorded during drift.	Time Pressure Temp Salinity, if measured	2: value is transmitted by the float

	occasionally measured.			
300 (PET)	TimeStartProfileDescent CTD performed at end of drift		Time Pressure (bars), T, S	2: value is transmitted by the float
301	Average of hourly pressure measurements	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	TimeStartProfile	AST and DDET are the same	Time	2: value is transmitted by the float
500 (AST)	TimeStartProfile	AST and DDET are the same	Time	2: value is transmitted by the float
501	DownTimeEpoch/UNIX epoch when the down-time expired	Down-time end time - time out value	Time	2: value is transmitted by the float
600 (AET)	TimeStopProfile		Time	2: value is transmitted by the float
703 (ST)	All GPS times and locations		Time, Position	2: value is transmitted by the float
700 (TST)	Time of first Iridium msg		Time	4: value is determined by satellite
701 (TST)	TimeStartTelemetry		Time	2: value is transmitted by the float
702 (FMT)	Time of first Iridium message	Time	Time	4: value is determined by satellite
704 (LMT)	Time of last Iridium message		Time	4: value is determined by satellite
800 (TET)	DST for profile n + 1 except for floats with surface measurement phase after transmission end; Fill value for TET for those floats		Time	3: value is directly computed from relevant, transmitted float information

2.2.2.2 NAVIS floats

Argo program measurement codes (MC) for NAVIS floats				
Code (timing)	**NAVIS Variable**	**Description**	**Units**	**JULD_STATUS**
0 (launch)	Launch time and location as recorded by deployer If not recorded, use Fill Value	Time and location		0: value is estimated from pre-deployment information found in the metafile 9: value is not immediately known, but believe it can be estimated later
100 (DST)	descentInit	See section 3.2.2.2.1	Time	2: value is transmitted by the float
190	??	Descending CTD measurements		2: value is transmitted by the float
200 (DET)	Time when pressure is within 3% of drift pressure Time stamp for first PRES/TEMP sample within 3% of drift pressure	Can be interpolated from a line fit to the curve of Descent() pressure vs. Time. See section 3.2.2.2.3	Time Time	3: value computed from information transmitted directly by the float 2: value is transmitted by the float
250 (PST)	Last T,P during descent phase?? Some other time when float switches into park??			
During the drift phase, NAVIS floats measure time pressure and temperature				
290	Series of pressure	A series of pressure measurements taken daily during drift. No time can be assigned to these pressures, so use Fill Value in JULD	Pressure	9: value is not immediately known, but believe it can be estimated later
End of drift measurements				
300 (PET)	GoDeepInit()	See section 3.2.2.2.4	Time	2: value is transmitted by the float
301	Average pressure during drift	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	Time when pressure is within 3% of profile pressure	Can be interpolated from a line fit to the curve of ProfileInit() pressure vs time. See section 3.2.2.2.5	Time	3: value is directly computed from relevant, transmitted float information
500 (AST)	Profile() Sample 0 initiated at XXXXX Is this a time out?? Can float start ascending before this time if it reaches the profile pressure??	See section 3.2.2.2.6	Time	2: value is transmitted by the float
600 (AET)	SurfaceDetect()	See section 3.2.2.2.7 Is this a time out value or does float know it is at the surface?	Time	2: value is transmitted by the float

703 (ST)	All GPS times and locations		Time, Position	2: value is transmitted by the float
700 (TST)	TelemetryInit()	See section 3.2.2.2.8	Time	2: value is transmitted by the float
702 (FMT)	Time of first Iridium message		Time	4: value is determined by satellite
704 (LMT)	Time of last Iridium message		Time	4: value is determined by satellite
800 (TET)	Logout() Is this the same as the time of the last Iridium message??	See section 3.2.2.2.9		2: value is transmitted by the float

2.2.2.2.1 Descent Start Time - NAVIS

DescentInit.

The DST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 100 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.2 First Stabilization Time - NAVIS

Unknown, so nothing should be included in the N_MEASUREMENT array.

In the N_CYCLE array, fill value should be stored in the JULD_FIRST_STABILIZATION and the JULD_FIRST_STABILIZATION_STATUS.

2.2.2.2.3 Descent End Time - NAVIS

Can be interpolated from a line fit to the curve of Descent() pressure vs time. Calculate time when pressure is within 3% of drift pressure.

The DET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 200 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_END variable and the JULD_DESCENT_END_STATUS set to 3 (computed from information transmitted directly by the float). If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.4 Park End Time - NAVIS

GoDeepInit().

The PET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 300 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.5 Deep Descent End Time - NAVIS

Can be interpolated from a line fit to the curve of ProfileInit() pressure vs time. Calculate time when pressure is within 3% of profile pressure.

The DDET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 400 and STATUS set to 3: value is computed directly from information transmitted by the float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 3. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.6 Ascent Start Time - NAVIS

Profile() Sample 0 initiated at XXXXX.

The AST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 500 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2: value transmitted by float. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.7 Ascent End Time - NAVIS

SurfaceDetect().

The AET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 600 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2: value transmitted by float. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.8 Transmission Start Time - NAVIS

TelemetryInit().

The TST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 700 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2: value is transmitted by float. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.2.9 Transmission End Time - NAVIS

logout().

The TET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 800 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 2: value is transmitted by float. If the float clock

offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.3 NEMO floats

Argo program measurement codes (MC) for NEMO floats				
Code (timing)	**NEMO Variable**	**Description**	**Units**	**JULD_STATUS**
0 (launch)	Launch time and location as recorded by deployer If not recorded, use Fill Value	Time and location		0: value is estimated from pre-deployment information found in the metafile 9: value is not immediately known, but believe it can be estimated later
100 (DST)	Descent_start_time OR Descent_starttime	See section 3.2.2.3.1	Time Time	2: value is transmitted by the float 2: value is transmitted by the float
200 (DET)	Not usually available, so use Fill Value unless timeout error is triggered. If timeout occurs, enter time of abort	If float doesn't reach parking depth in time, the descent is aborted and a timeout error is reported. If this happens, enter this value into DET. If not, use Fill Value. See section 3.2.2.3.3		9: value is not immediately known, but believe it can be estimated later 2: value is transmitted by the float
250 (PST)	Parking_start_time Not available, so use Fill Value	Only available for newer floats. Do not enter this if timeout error occurs. Use Fill Value in that case. See section 3.2.2.3.3	Time	2: value is transmitted by the float 9: value is not immediately known, but believe it can be estimated later
During the drift phase, NEMO floats measure time pressure and temperature				
290	What kind of drift measurements are made?? A series, an average??			
End of drift measurements				
300 (PET)	Upcast_start_time	Only available for newer floats. See section 3.2.2.3.4		2: value is transmitted by the float
301	Average pressure during drift	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
500 (AST)	Ascent_start_time Or Ascent_starttime	See section 3.2.2.3.6 Is this a time out value?? Does float start ascending if it hits profile pressure?	Time Time	2: value is transmitted by the float 2: value is transmitted by the float
600 (AET)	Surfacingtime Or	See section 3.2.2.3.7 Is this a time out	Time	2: value is transmitted by the float

	Ascent_end_time	value or does float know it is at the surface?	Time	2: value is transmitted by the float
700 (TST)	End_of_profile_time Or Surface_start_time	See section 3.2.2.3.8	Time Time	2: value is transmitted by the float 2: value is transmitted by the float
702 (FMT)	Earliest time of all Argos messages received		Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	Not available, so use Fill Value	See section 3.2.2.3.9		9: value is not immediately known, but believe it can be estimated later

2.2.2.3.1 Descent Start Time - NEMO

DST is called **descent_start_time** or **descent_starttime**.

The DST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 100 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.3.2 First Stabilization Time - NEMO

FST is not measured by the float and should be excluded from the N_MEASUREMENT array.

In the N_CYCLE array, the value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to fill value.

2.2.2.3.3 Descent End Time & Park Start Time - NEMO

PST is called **parking_start_time** and is not available for all floats. For newer floats with serial numbers >113 this time will be in the recorded technical data. The time recorded here is either when the floats reaches programmed parking depth under a controlled descent (actually measuring the pressure) or at a programmed count of the pump for a parkcount descent. Both of these times are based either on an actual measurement of pressure or on a descent timer, so they are characterized as Park Start Time rather than Descent End Time. The PST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 250 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

The only exception is for a timeout error when the float, during a controlled descent, is not able to reach the parking depth and aborts the procedure after a predetermined time. In this case the time of the abort is recorded. If this timeout occurs, the time of abort should be recorded in the Descent End Time variable as it is a timeout value. The DET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 200 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_END variable and the JULD_DESCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

In the N_CYCLE array, fill value should be stored in the JULD_DESCENT_END, JULD_DESCENT_END_STATUS, JULD_PARK_START and JULD_PARK_START_STATUS variables.

2.2.2.3.4 Park End Time - NEMO

PET is called **upcast_start_time**. This variable is available for all floats with serial number >113. While this may seem like an odd name, the float manufacturer chose this following their internal logic.

The PET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 300 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

If the **upcast_start_time** is not available, it should be excluded from the N_MEASUREMENT array.

In the N_CYCLE array, fill value should be stored in the JULD_PARK_END and the JULD_PARK_END_STATUS variables.

2.2.2.3.5 Deep Descent End Time - NEMO

DDET is not an event for this float, so it should be excluded from the N_MEASUREMENT array.

In the N_CYCLE array, fill value should be stored in the JULD_DEEP_DESCENT_END and the JULD_DEEP_DESCENT_END_STATUS variables.

2.2.2.3.6 Ascent Start Time - NEMO

AST is called **ascent_start_time** or **ascent_starttime**.

The AST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 500 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.3.7 Ascent End Time - NEMO

AET is called **surfacingtime** or **ascent_end_time**.

The AET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 600 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.3.8 Transmission Start Time - NEMO

TST is called **end_of_profile_time** or **surface_start_time**.

The TST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 700 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.3.9 Transmission End Time - NEMO

TET is not known for the actual profile, the floats starts descending immediately after transmission. But the DST for the next profile will be the TET of the current profile. Before this time is known, the TET is a fill value with a status of "9".

The TET should be set to fill value for the current cycle in the JULD variable in the N_MEASUREMENT array with an MC = 800 and STATUS set to 9: value is not immediately known, but believe it can be estimated later.

When the next cycle arrives, the TET should be set to the DST of current profile in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with an MC = 800 and STATUS set to 2: value is transmitted by float.

For the N_CYCLE array, fill value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 9 for the current cycle.

Once the DST of the next profile occurs, and hence the TET of the previous profile is known, the TET can be filled in the previous cycle.

The value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4 NINJA floats

There are two types of NINJA floats: those deployed in 2002 - 2007 and those deployed in 2008. For the floats deployed in 2002 - 2007, some times are directly provided by the float (the day (day number in the current month), hours, minutes and seconds of the event are transmitted). Other times must be computed from technical information.

All these times must be corrected for clock offset before storage in the N_CYCLE and N_MEASUREMENT arrays.

Argo program measurement codes (MC) for NINJA 300001, 300002, 300003 floats				
Code (timing)	**NINJA Variable**	**Description**	**Units**	**JULD_STATUS**
0 (launch)	Launch time and location as recorded by deployer If not recorded, use Fill Value	Time and location		0: value is estimated from pre-deployment information found in the metafile 9: value is not immediately available but may be estimated at a later date
100 (DST)	Descent_Start_Day	See section 3.2.2.4.1.1	Day number in the month, hours, minutes and seconds	2: value is transmitted by the float
150 (FST)	First Stabilization Time Pressure provided with time	First of three stabilization times provided as hours and minutes elapsed since DST. See section 3.2.2.4.1.2	Time (hours and minutes since DST) Pressure	2: value is transmitted by the float 2: value is transmitted by the float
189	Second and Third Stabilization Times Pressure provided with time.	Next two stabilization times given as hours and minutes elapsed since DST. See section 3.2.2.4.1.2	Time (hours and minutes since DST) Pressure	2: value is transmitted by the float 2: value is transmitted by the float
200 (DET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
250 (PST)	Parking_Depth_in_Time Pressure	See section 3.2.2.4.1.3	Day number in the month, hours, minutes and seconds Pressure	2: value is transmitted by the float 2: value is transmitted by the float
During the drift phase, NINJA 300001, 300002, 300003 floats measure pressure daily.				
290	Series of pressure	A series of pressure measurements taken daily during drift. No time can be assigned to these pressures, so use	Pressure	2: value is transmitted by the float

		Fill Value in JULD		
End of drift measurements				
300 (PET)	Not available, so use Fill Value Pressure	See section 3.2.2.4.1.4	 Pressure	9: value is not immediately known, but believe it can be estimated later
301	Average pressure during drift	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	Not available, so use Fill Value	See section 3.2.2.4.1.5		9: value is not immediately known, but believe it can be estimated later
500 (AST)	Ascent_Start_Day	See section 3.2.2.4.1.6	Day number in the month, hours, minutes and seconds	2: value is transmitted by the float
590	Times associated with ascending CTD measurements	Transmitted data is of the elapsed time for each vertical slice of ascent (from the max pressure to 2000 db for the first slice; and for each 100 dbar think slice until the surface)	Time	2: value is transmitted by the float
600 (AET)	AST + profile duration	See section 3.2.2.4.1.6	Day number in the month, hours, minutes and seconds	3: value is directly computed from relevant, transmitted float information
700 (TST)	ARGOS_Start_day	See section 3.2.2.4.1.8	Day number in the month, hours, minutes and seconds	2: value is transmitted by the float
702 (FMT)	Earliest time of all Argos messages received		Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	Not available, so use Fill Value	See section 3.2.2.4.1.9		9: value is not immediately known, but believe it can be estimated later

2.2.2.4.1 Dated events for NINJA 300001, 300002 and 300003 versions

2.2.2.4.1.1 Descent Start Time - NINJA

The DST is directly provided by these NINJA versions (**Descent_Start_Day**): the day (day number in the month), hours, minutes and seconds of DST are transmitted.

The DST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 100 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.2 First Stabilization Time - NINJA

Three Stabilization Times are provided by these NINJA versions (as hours and minutes elapsed since DST) with the associated pressures.

The **three** Stabilization Times and pressures should be stored in the JULD (or JULD_ADJUSTED if clock offset has been applied) and PRES variables in the N_MEASUREMENT array with the MC set to 150 for the first stabilization and 189 for the next two. STATUS should be set to 2: value transmitted by float.

In the N_CYCLE array, the first stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.3 Park Start Time - NINJA

The PST is directly provided by these NINJA versions (**Parking_Depth_in_Time**): the day (day number in the month), hours, minutes and seconds of DST are transmitted.

The associated pressure is also transmitted.

The time and pressure should be stored in the JULD (or JULD_ADJUSTED if clock offset has been applied) and PRES variables in the N_MEASUREMENT array with the MC set to 250 and STATUS set to 2: value transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.4 Park End Time - NINJA

NINJA 30001 can only observe given information on the parking depth (parking depth = profile depth) just after it is deployed. Then, the times from PET to AST are not in the technical message. NINJA 30002 and 300003 can observe from profile depth which is deeper than parking depth, but their firmware are only updated a little from 30001. The times from PET to AST are not added in the technical message. Therefore, we cannot know the times from PET to AST.

Unfortunately, there is no constant amount of time from AST. Since the PET is an event that occurs for the float, without the time known, it will be fill value in the JULD & JULD_ADJUSTED variables in the N_MEASUREMENT array with an MC = 300 and STATUS set to 9 as it might be estimated at a later time.

In the N_CYCLE array, fill value should be stored in the JULD_PARK_END and the JULD_PARK_END_STATUS should be set to 9.

2.2.2.4.1.5 Deep Descent End Time - NINJA

NINJA floats do not record this time, so use fill value in the JULD and JULD_ADJUSTED variables in the N_MEASUREMENT array with an MC=400 and JULD_STATUS and JULD_ADJUSTED_STATUS of ‘9’.

In the N_CYCLE array, fill value should be stored in the JULD_DEEP_DESCENT_END variable and the JULD_DEEP_DESCENT_END_STATUS should be set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.6 Ascent Start Time - NINJA

The AST is directly provided by these NINJA versions (**Ascent_Start_Day**): the day (day number in the month), hours, minutes and seconds of AST are transmitted.

The AST should be stored in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with the MC set to 500 and STATUS set to 2: value transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.7 Ascent End Time - NINJA

The AET is not directly provided by these NINJA versions.

However, these floats provide the elapsed time for each vertical slice of ascent (from the max pressure to 2000 dbar for the first slice; and for each 100 dbar thick other slices until the surface).

The cumulative sum of these times is thus the profile duration and can be used to compute AET from AST.

The AET should be stored in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with the MC set to 600 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 3. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

AET is stored as Ascent_0db_time

2.2.2.4.1.8 Transmission Start Time - NINJA

The TST is directly provided by these NINJA versions: the day (day number in the month), hours, minutes and seconds of TST are transmitted. The variable is called “ARGOS_START_Day”.

The TST should be stored in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with the MC set to 700 and STATUS set to 2: value transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.1.9 Transmission End Time - NINJA

TET is not known. Since the TET is an event that occurs for the float, without the time known, it will be fill value in the JULD & JULD_ADJUSTED variables in the N_MEASUREMENT array with an MC = 800 and STATUS set to 9 as it might be estimated at a later time.

In the N_CYCLE array, fill value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.4.2 2008 NINJA floats

No timing information is available in real time or in delayed mode. Nothing should be included in the N_MEASUREMENT array. All cycle timing variables and their status flags should be fill value in the N_CYCLE array.

2.2.2.5 Deep NINJA floats

Deep NINJA floats profile to deeper than 2000db and the accuracy of the CTD data below 2000 db is not yet well understood. Currently, Argo is labeling these data with lower quality flags (2 and 3) in real time. So, if any pressure measurements included in the trajectory file are below 2000 db, they should be flagged with a 2 or 3 in real time.

Argo program measurement codes (MC) for DeepNINJA floats				
Code (timing)	**DeepNINJA Variable**	**Description**	**Units**	**JULD_STATUS**
0 (launch)	Launch time and location as recorded by deployer If not recorded, use Fill Value	Time and location		0: value is estimated from pre-deployment information found in the metafile 9: value is not immediately available but may be estimated at a later date
100 (DST)	Descent_Start_Time Pressure provided with time.	Time when float starts descending to the parking depth from sea surface.	Date and Time Pressure (dbar)	2: value is transmitted by the float 2: value is transmitted by the float
150 (FST)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
200 (DET)	Descent_End_Time Pressure provided with time.	Time when float reaches the parking depth and start drifting	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
250 (PST)	Use DET	Time when float reaches the parking depth and start drifting	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
During the drift phase, DeepNINJA floats measure time pressure, temperature and salinity hourly.				
290	Series of time, pressure, temperature and salinity measured during drift	A series of pressure measurements taken daily during drift.	Date and Time Pressure (dbar) Temperature (°C) Salinity(psu)	2: value is transmitted by the float
End of drift measurements				
300 (PET)	Deep_Descent_Start_Time Pressure provided with	Time when float start descending from the parking depth to the profile depth.	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the

	time.			float
301	Average pressure during drift	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	Deep_Descent_End_Time Pressure provided with time.	Time when float reaches the profile depth.	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
500 (AST)	Ascent_Start_Time Pressure provided with time.	Time when float starts ascending to the sea surface	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
600 (AET)	Ascent_End_Time Pressure provided with time.	Time when float reaches the sea surface and stop ascending.	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
703 (ST)	GPS fix		Time, Position	2: value is transmitted by the float
700 (TST)	First Message Time Pressure provided with time	Time when the float transmits the first message	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float
702 (FMT)	Time when the first message is received		Date and Time	4: value is determined by satellite
704 (LMT)	Time when the last message is received		Date and Time	4: value is determined by satellite
800 (TET)	Transmit_END_Time Pressure provided with time.	Time when float stops transmitting.	Date and Time Pressure(dbar)	2: value is transmitted by the float 2: value is transmitted by the float

2.2.2.6 NOVA floats

The housekeeping data packet has most of the cycle timing variables in it. The variable names and how they are calculated are described below.

Argo program measurement codes (MC) for NOVA floats				
Code (timing)	**NOVA Variable**	**Description**	**Units**	**JULD_STATUS**
0 (launch)	Time of last GPS fix – PARAM 12 setting	Launch time and location. See section	Time (seconds), position	3: value is directly computed from relevant, transmitted float

	(accurate to +/-5 minutes) Time of activation (within one hour of GPS fix)	3.2.2.5.1 Launch time and location. See section 3.2.2.5.1	 Time (seconds), position	information 2: value is transmitted by the float
100 (DST)	NVS/3 + time stamp of previous Iridium transmission	See section 3.2.2.5.2 DST not transmitted by float	NVS (no unit) Time of Iridium message	3: value is directly computed from relevant, transmitted float information
150 (FST)	FST	See section 3.2.2.5.3	Time (hours)	2: value is transmitted by the float
190	CTD taken during first descent after activation	CTD only taken on descent only after activation	Time (hours) Pressure (dbar) Temp (deg C) Salinity (psu)	2: value is transmitted by the float
200 (DET)	Deepest Temp,Pres pair taken during first descent after activation	CTD only taken on descent only after activation	Time (hours) Pressure (dbar) Temp (deg C) Salinity (psu)	2: value is transmitted by the float
250 (PST)	EDT	Float recognizes when it has stabilized at depth and changes into park phase. See section 3.2.2.5.4	Time (hours)	2: value is transmitted by the float
During the drift phase, NOVA floats measure time pressure and temperature at variable times. Choose the measurement code below that most appropriately describes what types of measurements are taken:				
290	Series of pressure Series of temperature Series of salinity	A series of CTD measurements taken during drift at user specified times	Time (hours) Pressure (dbar) Temp (deg C) Salinity (psu)	2: value is transmitted by the float
297	Minimum pressure	Minimum pressure recorded during drift phase	Pressure (dbar)	2: value is transmitted by the float
298	Maximum pressure	Maximum pressure recorded during drift phase	Pressure (dbar)	2: value is transmitted by the float
End of drift measurements				
300 (PET)	DDST	See section 3.2.2.5.5	Time (hours)	3: value is directly computed from relevant, transmitted float information
301	Average pressure during drift	Best estimate of drift depth	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	DDET	See section 3.2.2.5.6	Time (hours)	2: value is transmitted by the float
500 (AST)	SAT	See section 3.2.2.5.7	Time (hours)	2: value is transmitted by the float
600 (AET)	EAT	See section 3.2.2.5.8	Time (hours)	2: value is transmitted by the float
703 (surface fixes)	All GPS fixes		Time, Position	2: value is transmitted by the float
700 (TST)	AET + SBDT from previous message	See section 3.2.2.5.9	Time (AET in hours, SBDT in	3: value is directly computed from relevant, transmitted float

			seconds)	information
702 (FMT)	AET + SBDT from previous message	See section 3.2.2.5.9	Time (AET in hours, SBDT in seconds)	3: value is directly computed from relevant, transmitted float information
704 (LMT)	TST + SBDT from previous cycle	See section 3.2.2.5.10	Time (TST in hours, SBDT in seconds)	3: value is directly computed from relevant, transmitted float information
800 (TET)	TST + SBDT from previous cycle	See section 3.2.2.5.10	Time (TST in hours, SBDT in seconds)	3: value is directly computed from relevant, transmitted float information

2.2.2.6.1 Launch time

Once the magnet is removed, the float looks at the PARAMETER 12 setting (the delay before mission which can be set to between 0 and up to 1 hour). Once that time has elapsed, the float will acquire a GPS location and other diagnostic information and send a housekeeping message (time and location stamped) prior to beginning its mission. The message can be identified by its CYCLE COUNT = 255. Depending on how accurate you want to be:

a) Activation can be calculated from the housekeeping message:
Time of Last GPS fix – PARAM 12 setting = activation time to +/-5 minutes
b) The time of the activation will be within 1 hour of housekeeping message for cycle 255

2.2.2.6.2 Descent Start Time - NOVA

DST is not transmitted directly by the float, but must be calculated from the NVS variable. NVS is the number of valve activations at the surface. There is no unit on this and the minimum value is zero and the maximum value is 255. The start byte is 10 and the bit length is 8. The decoding equation is y = x.

DST = NVS/3 + time stamp of previous Iridium transmission.

The DST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 100 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 3.. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.3 First Stabilization Time - NOVA

FST is called FST and is the time in the day when the float first activated the valve during its descent. It is measured in hours, had a minimum value of zero and a maximum value of 23.9. The start byte is 5 and the bit length is 8. The decoding equation is y = 0.1 * x.

The FST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 150 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the first stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.4 Park Start Time - NOVA

PST is called EDT and is the time in the day when the float ended its descent to parking. It is measured in hours, has a minimum value of zero and a maximum value of 23.9. The start byte is 4 and the bit length is 8. The equation to calculate it is y = 0.1 * x. This time can change from cycle to cycle because the float recognizes it is stable at the parking depth.

The PST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 250 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.5 Park End Time - NOVA

PET is called PET and is the time in the day when the float started its descent to profile depth. The unit is hours, has a minimum value of zero and a maximum value of 23.9. The start byte is 6 and the bit length is 8. The equation to calculate it is y = 0.1 * x.

The PET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 300 and STATUS set to 2: value is transmitted by float.

2.2.2.6.6 Deep Descent End Time - NOVA

DDET is called DDET and is the time in the day when the float achieved its profile depth. The unit is hours, has a minimum value of zero and a maximum value of 23.9. The start byte is 7 and the bit length is 8. The decoding equation is y = 0.1 * x. The float recognizes when it is at the profile pressure and and reports this time as DDET. It can change from profile to profile.

The DDET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 400 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_DEEP_DESCENT_END variable and the JULD_DEEP_DESCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.7 Ascent Start Time - NOVA

AST is called SAT and is the time in the day when the float started its ascending profile. The unit is hours, has a minimum value of zero and a maximum value of 23.9. The start byte is 8 and the bit length is 8. The decoding equation is y = 0.1 * x. This time is set in PARAMETER 2, but should be sufficiently after DDET.

The AST should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 500 and STATUS set to 2: value is transmitted by float.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.8 Ascent End Time - NOVA

AET is called EAT and is the time in the day when the float ended its ascending profile. The unit is hours, has a minimum value of zero and a maximum value of 23.9. The start byte is 9 and the bit length is 8. The decoding equation is y = 0.1 * x. Once the CTD stops profiling at 6 db, the internal bladder is emptied and the float rises to the surface. At that time, the EAT is recorded. Afterwards, GPS acquisition starts and then Iridium transmission.

The AET should be in the JULD (or JULD_ADJUSTED if clock offset has been applied) variables in the N_MEASUREMENT array with an MC = 600 and STATUS set to 2: value is transmitted by float.

2.2.2.6.9 Transmission Start Time & First Message Time- NOVA

For Iridium, there are two values transmitted. When the float reaches the surface, it acquires a GPS position. The time to do this is represented by TTFF (in seconds). After the GPS is acquired, then the Iridium transceiver is activated. The time to do this is represented by SBDT (again in seconds). After completion of the transmission, a satellite check is done to look for incoming commands. If there is one, it is processed and then the float starts its next profile. Note that SBDT refers to the previous profile, not the current one, as it is calculated AFTER the Iridium transmission takes place. TST and FMT should be the same for NOVA floats

TST = AET + SBDT from previous cycle.

The TST & FMT should be set to fill value for the current cycle in the JULD variable in the N_MEASUREMENT array with an MC = 700 and STATUS set to 9: value is not immediately known, but believe it can be estimated later.

When the next cycle arrives, the TST & FMT should be filled in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with an MC = 700 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

For the N_CYCLE array, fill value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 9 for the current cycle.

Once the next profile occurs, and hence the TST & FMT of the previous profile is known, the TST & FMT can be filled in the previous cycle.

The value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 3. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.6.10 Transmission End Time & Last Message Time- NOVA

TET = TST + SBDT from previous cycle.

The TETand the LMT should be set to fill value for the current cycle in the JULD variable in the N_MEASUREMENT array with an MC = 800 and STATUS set to 9: value is not immediately known, but believe it can be estimated later.

When the next cycle arrives, the TET & LMT should be filled in the JULD (or JULD_ADJUSTED if clock offset has been applied) variable in the N_MEASUREMENT array with an MC = 800 and STATUS set to 3: value is directly computed from relevant, transmitted float information.

For the N_CYCLE array, fill value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 9 for the current cycle.

Once the next profile occurs, and hence the TET & LMT of the previous profile is known, the TET & LMTcan be filled in the previous cycle.

The value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS set to 3. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7 PROVOR/ARVOR floats

PROVOR floats directly provide, in the technical message, cycle timing information.

More precisely the provided times can be decoded to obtain the hours and minutes of a timed event but the corresponding day must be obtained by other means (see §3.2.2.6.8).

For PROVOR Argos floats, these times must be corrected for clock offset whereas for PROVOR Iridium floats the clock is set each cycle, thus the clock offset can be neglected.

Most of the times have an unusual time resolution (see §3.2.2.6.9) which must be stored in the data.

PROVOR floats do not provide a quick, easy real time estimate for Transmission End Time, but the float actually experiences this event. Therefore, in the N_MEASUREMENT array TET (MC = 800) should be fill value and its status flag should be a "9".

In the N_CYCLE array, fill value should be stored in the JULD_TRANSMISSION_END variable and the JULD_TRANSMISSION_END_STATUS should be set to 9.

PROVOR floats do experience both JULD_DESCENT_END (DET) and JULD_DEEP_DESCENT_END (DDET), but there is no way to know these times right now. Therefore, in the N_MEASUREMENT array DET (MC = 200) and DDET (MC = 400) should be fill value and its status flag should be a "9".

In the N_CYCLE array, fill value should be stored in the JULD_DESCENT_END and JULD_DEEP_DESCENT_END variables and JULD_DESCENT_END_STATUS and JULD_DEEP_DESCENT_END_STATUS should be set to 9.

2.2.2.7.1 Timed events for PROVOR 101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018 and 101019 versions

All status flags should be a "2" since they come directly from the float.

2.2.2.7.1.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "descent start time".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "float First Stabilization Time" (or "first float First Stabilization Time" for PROVOR 101018 and 101019 versions).

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "float stabilization pressure" (or "first float stabilization pressure" for PROVOR 101018 and 101019 versions).

The stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "end of descent time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.4 Park End Time

The hours and minutes of the PET are provided, in the technical message, by the technical parameter "profile descent start time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.5 Deep Park Start Time

The hours and minutes of the DPST are provided, in the technical message, by the technical parameter "profile descent stop time".

In the N_CYCLE array, the value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.6 Ascent Start Time

The hours and minutes of the AST are provided, in the technical message, by the technical parameter "profile ascent start time".

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.1.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "time at end of ascent".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018 and 101019 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Descent Start Time without clock offset applied (3.2.2.6.1.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.1.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	First Stabilization Time without clock offset applied (3.2.2.6.1.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.1.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Park Start Time without clock offset applied (3.2.2.6.1.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.1.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Park End Time without clock offset applied (3.2.2.6.1.4)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.1.4)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Deep Park Start Time without clock offset applied (3.2.2.6.1.5)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Deep Park Start Time with clock offset applied (3.2.2.6.1.5)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Ascent Start Time without clock offset applied (3.2.2.6.1.6)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.1.6)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	101011, 101012, 101014, 101015, 101013, 100001, 101017,	Ascent End Time without clock offset applied (3.2.2.6.1.7)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent End Time with clock offset applied (3.2.2.6.1.7)	2: value is transmitted by float	

	101018, 101019			
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Transmission Start Time without clock offset applied (3.2.2.6.1.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.1.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	101011, 101012, 101014, 101015, 101013, 100001, 101017, 101018, 101019	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.2 Timed events for PROVOR 102002, 102003 and 102004 versions

2.2.2.7.2.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "descent start time".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "float First Stabilization Time".

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "float stabilization pressure" (except for PROVOR 102004 version).

In the N_CYCLE array, the stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "end of descent time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.4 Park End Time

The hours and minutes of the PET are provided, in the technical message, by the technical parameter "profile descent start time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.5 Deep Park Start Time

The hours and minutes of the DPST are provided, in the technical message, by the technical parameter "profile descent stop time".

In the N_CYCLE array, the value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.6 Ascent Start Time

The hours and minutes of the AST are provided, in the technical message, by the technical parameter "profile ascent start time".

In the N_CYCLE array, the corresponding number should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 14 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.2.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "time at end of ascent".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 102002, 102003 and 102004 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	102002, 102003, 102004	Descent Start Time without clock offset applied (3.2.2.6.2.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.2.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	102002, 102003, 102004	First Stabilization Time without clock offset applied (3.2.2.6.2.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.2.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	102002, 102003, 102004	Park Start Time without clock offset applied (3.2.2.6.2.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	

		Park Start Time with clock offset applied (3.2.2.6.2.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	102002, 102003, 102004	Park End Time without clock offset applied (3.2.2.6.2.4)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.2.4)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	102002, 102003, 102004	Deep Park Start Time without clock offset applied (3.2.2.6.2.5)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Deep Park Start Time with clock offset applied (3.2.2.6.2.5)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	102002, 102003, 102004	Ascent Start Time without clock offset applied (3.2.2.6.2.6)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.2.6)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	102002, 102003, 102004	Ascent End Time without clock offset applied (3.2.2.6.2.7	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent End Time with clock offset applied (3.2.2.6.2.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	102002, 102003, 102004	Transmission Start Time without clock offset applied (3.2.2.6.2.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.2.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	102002, 102003, 102004	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.3 Timed events for PROVOR 101009, 101006, 101008 and 101010 versions

2.2.2.7.3.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "descent start time".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "float First Stabilization Time".

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "float stabilization pressure".

In the N_CYCLE array, the stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "end of descent time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.4 Park End Time

The PET is deduced from AST by the following relation: PET = AST - DELAI

where DELAI is a programmed meta-data parameter that determines the maximum amount of time given to the float for diving from PARKING to PROFILE depth.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.5 Deep Park Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the NC_CYCLE array, fill value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.6 Ascent Start Time

The hours and minutes of the AST are provided, in the technical message, by the technical parameter "profile ascent start time".

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.3.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "time at end of ascent".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 101009, 101006, 101008 and 101010 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	101009, 101006, 101008, 101010	Descent Start Time without clock offset applied (3.2.2.6.3.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.3.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	101009, 101006, 101008, 101010	First Stabilization Time without clock offset applied (3.2.2.6.3.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.3.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	101009, 101006, 101008, 101010	Park Start Time without clock offset applied (3.2.2.6.3.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.3.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	101009, 101006, 101008, 101010	Park End Time without clock offset applied (3.2.2.6.3.4)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.3.4)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	101009, 101006, 101008, 101010	Fill value (3.2.2.6.3.5)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.3.5)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	101009, 101006, 101008, 101010	Ascent Start Time without clock offset applied (3.2.2.6.3.6)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.3.6)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	101009, 101006, 101008, 101010	Ascent End Time without clock offset applied (3.2.2.6.3.7)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent End Time with clock offset applied (3.2.2.6.3.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	101009, 101006, 101008, 101010	Transmission Start Time without clock offset applied (3.2.2.6.3.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.3.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	101009, 101006, 101008, 101010	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.4 Timed events for PROVOR 100006, 100005, 100004, 100008 and 100003 versions

2.2.2.7.4.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "descent start time".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "First Stabilization Time".

In the N_CYCLE array, the first stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "end of descent time".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.4 Park End Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the N_CYCLE array, fill value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.5 Deep Park Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

Fill value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.6 Ascent Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the N_CYCLE array, fill value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.4.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "end of resurfacing time".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 100006, 100005, 100004, 100008 and 100003 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	100006, 100005, 100004, 100008, 100003	Descent Start Time without clock offset applied (3.2.2.6.4.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.4.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	100006, 100005, 100004, 100008, 100003	First Stabilization Time without clock offset applied (3.2.2.6.4.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.4.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	100006, 100005, 100004, 100008, 100003	Park Start Time without clock offset applied (3.2.2.6.4.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.4.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	100006, 100005, 100004, 100008, 100003	Fill value (3.2.2.6.4.4)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.4.4)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	100006, 100005, 100004, 100008, 100003	Fill value (3.2.2.6.4.5)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.4.5)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	100006, 100005, 100004, 100008, 100003	Fill value (3.2.2.6.4.6)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.4.6)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	100006, 100005, 100004, 100008, 100003	Ascent End Time without clock offset applied (3.2.2.6.4.7)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent End Time with clock offset applied (3.2.2.6.4.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	100006, 100005, 100004, 100008, 100003	Transmission Start Time without clock offset applied (3.2.2.6.4.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.4.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	100006, 100005, 100004, 100008, 100003	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.5 Timed events for PROVOR 101007 version

2.2.2.7.5.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "heure début de plongée".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "heure de première stabilisation".

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "pression de première stabilisation".

In the N_CYCLE array, the stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "heure de fin de descente".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.4 Park End Time

The PET is deduced from AST by the following relation: PET = AST - DELAI

where DELAI is a programmed meta-data parameter that determines the maximum amount of time given to the float for diving from PARKING to PROFILE depth.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.5 Deep Park Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the N_CYCLE array, fill value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.6 Ascent Start Time

The hours and minutes of the AST are provided, in the technical message, by the technical parameter "heure de début profil remontée".

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.5.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "heure de fin de remontée à la surface".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 101007 version				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	101007	Descent Start Time without clock offset applied (3.2.2.6.5.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.5.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	101007	First Stabilization Time without clock offset applied (3.2.2.6.5.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.5.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	101007	Park Start Time without clock offset applied (3.2.2.6.5.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.5.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	101007	Park End Time without clock offset applied (3.2.2.6.5.4)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.5.4)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	101007	Fill value (3.2.2.6.5.5)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.5.5)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	101007	Ascent Start Time without clock offset applied (3.2.2.6.5.6)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.5.6)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	101007	Ascent End Time without clock offset applied (3.2.2.6.5.7)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	

		Ascent End Time with clock offset applied (3.2.2.6.5.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	101007	Transmission Start Time without clock offset applied (3.2.2.6.5.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.5.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	101007	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.6 Timed events for PROVOR 101002, 101005 and 100002 versions

2.2.2.7.6.1 Descent Start Time

The hours and minutes of the DST are provided, in the technical message, by the technical parameter "heure début de plongée".

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "heure de première stabilisation".

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "pression de première stabilisation". This pressure should go in the PRES variable with an MC of 150.

In the N_CYCLE array, the stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "heure de fin de descente".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.4 Park End Time

The PET is deduced from AST by the following relation: PET = AST - *DELAI*

where *DELAI* is a programmed meta-data parameter that determines the maximum amount of time given to the float for diving from PARKING to PROFILE depth.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.5 Deep Park Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the N_CYCLE array, fill value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.6 Ascent Start Time

The AST is computed from TSD in float time (TSD_{FT}) (i.e. not already corrected from clock offset).

AST_{FT} = floor((TSD_{FT} - *MinProfDuration*)*24)/24

where:

- AST_{FT} is the AST in float time,
- *MinProfDuration* is the minimum profile duration, given by the latest CTD measurement time of the profile (these times are relative to AST_{FT}).

We thus assume that AST is programmed at a given hour (in float time), which is the case.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.6.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "heure de fin de remontée à la surface".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 101002, 101005 and 100002 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	101002, 101005, 100002	Descent Start Time without clock offset applied (3.2.2.6.6.1)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.6.1)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	101002, 101005, 100002	First Stabilization Time without clock offset applied (3.2.2.6.6.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.6.2)	2: value is transmitted by float	

MC	Float type	JULD	JULD_STATUS	
250 PST	101002, 101005, 100002	Park Start Time without clock offset applied (3.2.2.6.6.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.6.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	101002, 101005, 100002	Park End Time without clock offset applied (3.2.2.6.6.4)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.6.4)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	101002, 101005, 100002	Fill value (3.2.2.6.6.5)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.6.5)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	101002, 101005, 100002	Ascent Start Time without clock offset applied (3.2.2.6.6.6)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.6.6)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	101002, 101005, 100002	Ascent End Time without clock offset applied (3.2.2.6.6.7)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent End Time with clock offset applied (3.2.2.6.6.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	101002, 101005, 100002	Transmission Start Time without clock offset applied (3.2.2.6.6.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.6.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	101002, 101005, 100002	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.7 Timed events for PROVOR 101003 and 101004 versions

2.2.2.7.7.1 Descent Start Time

The hours and minutes of the Buoyancy Reduction Start Time (BRST) are provided, in the technical message, by the technical parameter "heure début de plongée".

The Number of Valve Actions at the Surface (NVAS) is provided, in the technical message, by the technical parameter "nombre d'actions EV en surface".

The DST is computed from BRST and NVAS:

DST = BRST + NVAS*130 seconds

In the N_CYCLE array, the value should be stored in the JULD_DESCENT_START variable and the JULD_DESCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.2 First Stabilization Time

The hours and minutes of the FST are provided, in the technical message, by the technical parameter "heure de première stabilisation".

The associated pressure (**in bars**) is also provided, in the technical message, by the technical parameter "pression de première stabilisation". This pressure should be stored in PRES with an MC code equal to 150.

In the N_CYCLE array, the stabilization value should be stored in the JULD_FIRST_STABILIZATION variable and the JULD_FIRST_STABILIZATION_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.3 Park Start Time

The hours and minutes of the PST are provided, in the technical message, by the technical parameter "heure de fin de descente".

In the N_CYCLE array, the value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.4 Park End Time

The PET is deduced from AST by the following relation: PET = AST - *DELAI*

where *DELAI* is a programmed meta-data parameter that determines the maximum amount of time given to the float for diving from PARKING to PROFILE depth.

In the N_CYCLE array, the value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.5 Deep Park Start Time

There is no easy way to get this time for this PROVOR float, so fill value should be used.

In the N_CYCLE array, fill value should be stored in the JULD_DEEP_PARK_START variable and the JULD_DEEP_PARK_START_STATUS set to 9. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.6 Ascent Start Time

The AST is computed from TSD in float time (TSD_{FT}) (i.e. not already corrected from clock offset).

AST_{FT} = floor((TSD_{FT} - *MinProfDuration*)*24)/24

where:

- AST_{FT} is the AST in float time,
- *MinProfDuration* is the minimum profile duration, given by the latest CTD measurement time of the profile (these times are relative to ASTFT).

We thus assume that AST is programmed at a given hour (in float time), which is the case.

In the N_CYCLE array, the value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.7 Ascent End Time

The AET is deduced from TST by the following relation: AET = TST - 16 minutes

In the N_CYCLE array, the corresponding sum should be stored in the JULD_ASCENT_END variable and the JULD_ASCENT_END_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

2.2.2.7.7.8 Transmission Start Time

The hours and minutes of the TST are provided, in the technical message, by the technical parameter "heure de fin de remontée à la surface".

In the N_CYCLE array, the value should be stored in the JULD_TRANSMISSION_START variable and the JULD_TRANSMISSION_START_STATUS set to 2. If the float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

N_MEASUREMENT Array				
PROVOR floats 101003 and 101004 versions				
MC	**Float type**	**JULD**	**JULD_STATUS**	
100 DST	101003 101004	Descent Start Time without clock offset applied (3.2.2.6.7.1)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Descent Start Time with clock offset applied (3.2.2.6.7.1)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
150 FST	101003 101004	First Stabilization Time without clock offset applied (3.2.2.6.7.2)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		First Stabilization Time with clock offset applied (3.2.2.6.7.2)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
250 PST	101003 101004	Park Start Time without clock offset applied (3.2.2.6.7.3)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park Start Time with clock offset applied (3.2.2.6.7.3)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
300 PET	101003 101004	Park End Time without clock offset applied (3.2.2.6.7.4)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Park End Time with clock offset applied (3.2.2.6.7.4)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
450 DPST	101003 101004	Fill value (3.2.2.6.7.5)	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value (3.2.2.6.7.5)	9: not immediately know, but believe value can be estimated later	
MC	**Float type**	**JULD**	**JULD_STATUS**	
500 AST	101003 101004	Ascent Start Time without clock offset applied (3.2.2.6.7.6)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Ascent Start Time with clock offset applied (3.2.2.6.7.6)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
600 AET	101003 101004	Ascent End Time without clock offset applied (3.2.2.6.7.7)	3: value is directly computed from relevant, transmitted float information	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	

		Ascent End Time with clock offset applied (3.2.2.6.7.7)	3: value is directly computed from relevant, transmitted float information	
MC	**Float type**	**JULD**	**JULD_STATUS**	
700 TST	101003 101004	Transmission Start Time without clock offset applied (3.2.2.6.7.8)	2: value is transmitted by float	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Transmission Start Time with clock offset applied (3.2.2.6.7.8)	2: value is transmitted by float	
MC	**Float type**	**JULD**	**JULD_STATUS**	
800 TET	101003 101004	Fill value	9: not immediately know, but believe value can be estimated later	
		JULD_ADJUSTED	**JULD_ADJUSTED_STATUS**	
		Fill value	9: not immediately know, but believe value can be estimated later	

2.2.2.7.8 From day, hours and minutes to time

The hours and minutes of the event times are obtained from technical message information.

The associated day can be obtained by the following algorithms.

The day of TST is determined using FMT:

1. Convert FMT in Float Time (FMT_{FT} = FMT + *FloatClockDrift*),
2. Convert the hours and minutes of FMT_{FT} in Technical Message time (in tenths of and our after truncation) to obtain FMT_{FTTM},
3. Compare the resulting FMT_{FTTM} with TST to determine the day of TST (remembering that $FMT_{FTTM} \geq TST$).

The day of AET is determined using TST.

The day of AST is determined using AET and the assumption that: AET-AST < 24 h.

The day of DDET is determined using AST and the assumption that: AST- DDET < 24 h.

The day of PET is determined using DDET and the assumption that: DDET - PET < 24 h.

The day of DST is determined using a Reference Date (RD) which can be:

- For cycle #0: the day of the first descent (meta-data parameter needed for data decoding),
- For a given cycle #*N* (*N* > 0):
  - If cycle #*N-1* exists: RD is the LMT of the cycle #*N-1*,
  - Otherwise, RD is computed from the last received Argos **CTD** message (LMT_{CTD}): RD = LMT_{CTD} - *CycleDuration*.

The obtained RD is then used to determine the day of DST:

1. Convert RD in float time (RD_{FT} = RD + *FloatClockDrift*),
2. Convert the hours and minutes of RD_{FT} in Technical Message time (in tenths of and our after truncation) to obtain RD_{FTTM},
3. Compare the resulting RD_{FTTM} with DST to determine the day of DST (remembering that $RD_{FTTM} \leq DST$).

The day part of FST is determined using DST and the assumption that: FST-DST < 24 h.

The day part of DET is determined using FST and the assumption that: FST-DET < 24 h.

2.2.2.7.9 Technical time resolution

For PROVOR 102004 version, technical times are given in minutes.

For all other PROVOR float versions, technical times are given in tenths of an hour; moreover they are resulting from a truncation of raw measurements.

Consequently, for example, an event dated 13:36 by the float occurred in the time interval [13:36 - 13:42[.

To take this characteristic into account and to have a statistical mean estimate of the event time, in the decoding process we must add 3 minutes to DST, FST, DET, DDET, AET and TST.

Note however that PET and AST should not be modified because they are resulting from float programmed actions (at a specific hour).

2.2.2.8 SOLO floats

Real time:

All cycle times (DST, FST, DET, PST, PET, DDET, DPST, AST, AET, TST, TET) cannot be filled in real time for SOLO floats and should be filled with fill value. The corresponding status variables for these timing variables should all be a "9" for time unknown. No times should be filled from information provided in the meta files.

Delayed mode:

The SIO SOLO returns little timing information directly measured by the float. However, estimation of float surfacing and sinking from the surface, as well as most other important timing events can be successfully carried out in DMQC. If the standard amount of data is returned by the float, timing can be estimated within 10 minutes for most float events.

DMQC for timing is best done when all data has been received from the float (the float has stopped reporting). SIO has followed the following procedure for DMQC of SIO Argos float timing estimation.

1) The surface time for the SIO SOLO is FIXED. This is built into the firmware. Surface time CANNOT vary. Since the float resets it's clock each cycle, it is very unlikely clock drift will be an issue.
2) Assume that changes in cycle time are slowly evolving over the life of the float. Here 'slowly varying' is defined as slower than the float's cycle time.
3) Collect each surface interval's first and last Argos data times. These times will not have a width greater than the floats programmed surface time. If the measured surface interval does exceed the programmed surface time, the likely culpret is a 'ghost message', and the Argos dataset will need to be edited. It is best to use the times of all Argos data, and not only position times.
4) Fit a slowly varying envelope of 'surface time' width to the Argos first/last times.
5) Some 'jumps' may be necessary depending on float behavior.
6) To determine other float timing (off the surface), add forwards and backwards from the determined AET and DST times, using float parameters.

The exact method of determining the envelope is left to the PI.

The Dtraj netCDF files from SIO have been filled using the above procedure. The estimated time of AET has been transferred to the DMQC profile netCDF file variable JULD. When this is done JULD_QC has been changed to '8', meaning interpolated.

2.2.2.9 SOLO-II floats

Solo-II floats are all newer, Iridium floats and as such, their cycle timing variables follow a different chronological order than many of the other floats included in this document. J. Gilson has created a table from which one can match all the desired measurement codes with what the SOLO-II float measures, referring back to the float's documentation. The documentation is all available on the SIO-Argo website (http://sio-argo.ucsd.edu/manuals.html).

With SOLO-II float version 2.0 and later, data is returned which closely labels the data sent by the float to the Measurement Code discussed here. In the below table the code retuned by the float is

described as 'Variable Code'. The Variable Code value does not equate to the Measurement Code value.

Argo program measurement codes (MC)				
Code (timing)	**SOLO II Variable**	**Description**	**Units**	**JULD_STATUS**
0	Cy 0: GPS ID=0x00 (Variable Code=1)	GPS fix from surfacing after short ~100dbar test dive	Time,position	1
100 (DST)	Cy>0: Fall ID=0x40 (Variable Code=1)	Typically, thefirst T,P pair [taken as valve opened to leave surface]	Time,P(0.04db)	2
199	Cy=0: Eng ID=0xe0 (Variable Code=7)	P,T,S triplet taken when float realizes it is under the surface and pumps to return to the surface (Eng ID=0xe0 bytes 39-47)	P(0.04db),T(0.001°C), S(0.001psu)	2
139/140	Cy>0: Fall ID=0x40	All pre-FST T,P Fall pairs not assigned to other MC (139 used for buoyancy adjustments)	Time,P(0.04db)	2
150 (FST)	Cy>0: Fall ID=0x40 (Variable Code=2)	T,P Fall pair ~ 100dbar	Time,P(0.04db)	2
189/190	Cy>0:Fall ID=0x40	All other pre-DET T,P Fall pairs (189 used for buoyancy adjustments)	Time,P(0.04db)	2
200 (DET)	Cy=0:Rise ID=0x50	Typically, Deepest T,P pair	Time,P(0.04db)	2
	Cy>0: Fall ID=0x40	Choice of T,P pair that is first within 3% of pressure at beginning of drift (see Eng ID=0xe2 bytes 63-65)	Time,P(0.04db)	2
n=239/240	Cy>0: Fall ID=0x40	All post DET T,P pairs. MC239 are used for buoyancy adjustments. If n is the number of stabilizations (see Argo ID=0xf0), the T,P n+1 from end of Fall record is a stabilization. Each later T,P pair excluding the last will be an additional stabilization. Note: for some floats there are stabilizations during drift.	Time,P(0.04db)	2
if there is a drift phase (drift pressure defined) (common to cycles > 1)				
250 (PST)	Cy>0: Fall ID=0x40 (Variable Code=4)	Last T,P Fall pair	Time,P(0.04db)	2
296	Cy>0: Eng ID=0xe2	Drift broken into two averaged halves. Stored in Eng ID=0xe2 bytes 63-80; Time estimated from the last Fall ID=0x40 T,P pair [note: not DET] and first Rise ID=0x50 T,P pair	P(0.04db),T(0.001°C), S(0.001psu)	2
290	Cy>0 with park phase Drift ID 0x98	The SOLOII can return the raw drift measurements. Time is not returned, but can be estimated within a few seconds.	P(0.04db),T(0.001°C), S(0.001psu)	3
300 (PET)	Cy>0: Rise ID=0x50	First T,P Rise pair [taken as valve opened]	Time,P(0.04db)	2
301		Best estimate of drift depth (average of two averaged halves)	Pressure	1

Endif if there is a deep dive (profile pressure > drift pressure and drift pressure defined)				
389/390	Cy>1: Rise ID=0x50	All pre-DDET T,P Rise pairs (389 indicates time of buoyancy adjustedment)	Time,P(0.04db)	2
400 (DDET)	Cy>1: Rise ID=0x50	DDET is determined by a) 2nd derivative of Rise pair series or b) within 3% of profile depth (see Eng ID=0xe2 bytes 39-41).	Time,P(0.04db)	2
489/490	Cy>1: Rise ID=0x50	All post-DDET/pre-AST T,P Rise pairs (489 indicates time of buoyancy adjustment)	Time,P(0.04db)	2
500 (AST)	Cy>1: Rise ID=0x50; Eng ID=0xe2 (Typically Variable Code =7)	AST is determined by 2nd derivative of Rise pair series.	Time,P(0.04db); P(0.04db),T(0.001°C), S(0.001psu)	2
Else				
500 (AST)	Cy=0: Rise ID=0x50; (Variable Code=7) Cy=1 Eng ID=0xe2 (Typically Variable Code =7)	First T,P Rise pair [taken as valve opened] AST is determined by 2nd derivative of Rise pair series	Time,P(0.04db); P(0.04db),T(0.001°C), S(0.001psu)	2 2
Endif				
589/590	Cy>=0: Rise ID=0x50	All T,P Rise pairs post AST excluding last or last two. 589 indicates buoyancy adjustment.	Time,P(0.04db)	2
599	Cy=0: Eng ID=0xe0 Cy>0: Eng ID=0xe2	last P,T,S triplet taken before turning off CTD (Eng ID=0xe0 bytes 48-56) last P,T,S triplet taken before turning off CTD (Eng ID=0xe2 bytes 45-50)	P(0.04db),T(0.001°C), S(0.001psu) P(0.04db),T(0.001°C), S(0.001psu)	2 2
600 (AET)	Cy>-1: Rise ID=0x50 (Variable Code=8)	Last or 2nd to last T,P Rise pair	Time,P(0.04db)	2
703	Cy=0: GPS ID=0x00 Cy>0: GPS ID=0x02	GPS Fix GPS Fix	Time, Position Time, Position	2 2
700 (TST) 702 (FMT)	 Time in SBD email	TST is not recorded by the float, but it is within a minute of the first message Time of first SBD message	Time Time	1 4
704 (LMT) 800 (TET)	Time in SBD email	Time of last SBD message TET is not recorded by the float, but it is within a few seconds of the last message	Time Time	4 3
703	Cy>0: GPS ID=0x01	GPS Fix: May be multiple GPS fixes, depending on float settings	Time, Position	2

2.3 Guidelines for Argos message selection

2.3.1 Argos float message selection

Ideally, every DAC should use the same method for Argos message selection for each float type. Some floats are transmitting a CRC (Cyclic Redundancy Check) done on board the float and others are not. Additionally, not all CRC have the same reliability. Recommendations were issued at ADMT 10, but inconsistencies still exist between DACs.

Each float types message selection strategy will be listed below:

Argos message selection done at Coriolis for PROVOR/ARVOR

Technical message selection

1. If only one technical message is received with a good CRC, use it,
2. If more than one technical message is received, all with good CRCs, use the"first received one"
3. If no technical message is received with a good CRC, no technical message is used. In this case, times provided by the float are missing and, consequently, the order of the drift CTD measurements cannot be determined.

CTD data message selection

Received messages are processed by type (type 4: "descent profile CTD message", type 5: "submerged drift CTD message" and type 6: "ascent profile CTD message").

For each type, the Id of the received message is computed.

- For type 4 or type 6 messages, the Id is defined by the date and the pressure of the first CTD measurement of the message,
- For type 5 messages, the Id is defined by the date and the time of the first CTD measurement of the message.

The selection process must lead to (at most) one message for a given Id.

For a given type, all messages of a given Id are processed:

1. If only one message is received with a good CRC, use it,
2. If more than one message is received all with good CRCs, use the "first received one",
3. If no message is received with a good CRC:
    a. If 1 or 2 copies of the message has been received, no message is used for this Id,
    b. If more than 2 copies of the message have been received:
        i. If an even number of copies of the message have been received, reject the "first received one",
        ii. The possibly emitted message is computed from received copies (each bit of the message is defined by selecting the "most redundant" received one),
        iii. A CRC check is done on this "reconstructed" message:
            1. If it succeeds, use this "reconstructed" message,
            2. If it fails, no message is used for this Id.

2.4 CTD measurements

2.4.1 CTD measurements sampled during the drift phase at parking depth

2.4.1.1 APEX floats

2.4.1.1.1 CTD measurement sampled at the end of the drift phase at parking depth

All APEX float versions provide a CTD measurement (P, T and S) sampled at the end of the drift phase at parking depth, generally called park (or bottom) measurement. The corresponding time of the measurement is provided by Iridium float versions only.

This measurement should be associated to PET (MC 300) or AST (MC 500) if theoretical PARKING and PROFILE depths are equal for the corresponding cycle (see §3.2.2.1.5).

2.4.1.1.2 CTD measurements regularly sampled during the drift phase at parking depth

Some APEX float versions provide CTD measurements (only P and T generally) regularly sampled during the drift phase at parking depth. The corresponding times of the measurements are provided by Iridium float versions only.

For regularly sampled CTD measurements, use MC minus 10. Usually this will be 290 because the float is transitioning towards PET which is 300. If the float is transitioning towards a different MC, subtract four from that MC.

For averaged sampled CTD measurements, use MC minus 4. Usually this will be 296 because the float is transitioning towards PET which is 300. If the float is transitioning towards a different MC, subtract four from that MC.

For Argos float versions these measurements need to be dated in a post-processing procedure. For that we need to know additional meta-data parameters.

The following six sampled strategies can be encountered.

2.4.1.1.2.1 Normal float behavior

For most of the APEX floats, the first CTD measurement is sampled 8 hours after DST (*is this really DST?*) and the following ones with a programmed theoretical period.

Thus, for these floats, to compute the CTD measurement times we must first know:

- The DST (see §3.2.2.1.2),
- The theoretical period of the drift measurements.

2.4.1.1.2.2 Floats with daily CTD measurements

Some APEX floats provide a daily measurement, the first one sampled 24 hours after DST.

2.4.1.1.2.3 Floats providing only averaged values

Some floats provide *N* averages of hourly sampled CTD measurements. *N* is a programmed meta-data parameter.

The times of the averaged value should be computed to be regularly set between DET and PET.

The time of the average #*i* should be: time(*i*) = DET + (2**i* - 1)*(PET - DET)/(2**N*)

2.4.1.1.2.4 Isopycnal floats behavior

Isopycnal APEX floats generally provide two sets of CTD measurements.

The first one corresponds to the stabilization at the target sigma-theta value.

The first CTD measurement is sampled 6 hours after DST and the following ones with a 1.5 hour period.

See §3.4.2.3 for the storage of these data.

The second set of CTD measurements corresponds to the drift at the target sigma-theta value.

The first CTD measurement is sampled 6 hours after the last measurement of the first set and the following ones with a 6 hour period.

These are series of measurements, so the MC should be MC-10.

2.4.1.1.2.5 Old versions of isopycnal floats

Some (old) versions of isopycnal floats provide CTD measurement sampled at isopycnal depth but we do not know how to compute their corresponding times.

2.4.1.1.2.6 RAFOS floats behavior

RAFOS floats generally provide a daily CTD measurement sampled at the end of the last listening window of the day.

Thus, for these floats, to compute the CTD measurement times we must first know:

- The hour of the last listening window,
- The length of the listening windows.

The decoded and dated (if possible) CTD measurements should be stored in the N_MEASUREMENT arrays with:

- JULD set to the computed times,
- JULD_QC: set to 0,
- JULD_STATUS set to 3 (computed directly from information sent by the float)
- CYCLE_NUMBER set to corresponding cycle number,
- MEASUREMENT_CODE set to 290,
- <PARAM> set to decoded CTD values,
- <PARAM>_QCs set to 0,
- All other variables set to _FillValue.

For Iridium floats, the decoded times should be corrected for clock offset and stored in the JULD_ADJUSTED variable. CLOCK_OFFSET should also be filled in the N_CYCLE array.

For Argos floats, the times are computed from DST, thus clock offset is already taken into account. The times should still be stored in JULD_ADJUSTED since clock offset has been taken into account.

For Argos floats, we must consider missing Argos messages to correctly set CTD measurement dates (i.e. we must take into account the missing CTD measurements, due to not received data, to correctly apply the periodicity of the dates).

2.4.1.2 PROVOR floats

2.4.1.2.1 CTD measurements regularly sampled during the drift phase at parking depth

All PROVOR floats have the capability to achieve and provide CTD measurements (P, T and S) regularly sampled during the drift phase at parking depth but this capability must be enabled by the operator in the programmed float mission. Regularly sampled CTD measurements are represented by the MC towards which the float is transitioning - 10. Usually the float is transitioning towards PET or 300, making the MC code 290.

For PROVOR 100001 version, we have no information, in the transmitted data, about the drift measurement times.

For PROVOR 100006, 100005, 100004, 100008 and 100003 versions, the time of the first drift measurement is provided in the technical message as well as the drift data sampling period.

For all other float versions, the time of the first drift measurement of each Argos message (or Iridium packet) is transmitted in the data message. The day number of this time is relative to the day of the first descent (cycle #0).

Moreover, for some float versions, to minimize the impact of the loss of a drift CTD message, drift measurements are transmitted using an interleaving scheme:

- Measurements #1, #3, #5, #7, … are transmitted in a first message,
- Measurements #2, #4, #6, #8, … are transmitted in a second message.

In this case, it is very important to determine the **theoretical** time of the first drift measurement (particularly when this measurement is not received from the float).

This time depends on float version.

2.4.1.2.1.1 Drift measurement times determination for PROVOR 101011, 102002, 101012, 101014, 101015, 102003, 101013 and 100001 versions

For these float versions, measurements are done at round hours.

The theoretical First Drift Measurement Time (FDMT) is relative to the hour of the DET (which must be rounded down).

If DET is given as a Julian day.

If DSP is the Drift Sampling Period (in hours).

Then FDMT = floor(DET*24)/24 + DSP/24.

For PROVOR 101011, 101012, 101014, 101015 and 101013 versions, drift measurements are transmitted using an interleaving scheme.

This is not the case for PROVOR 102002, 102003 and 100001 versions.

2.4.1.2.1.2 Drift measurement times determination for PROVOR 101009, 101006, 101008, 101007, 101010, 101002, 101005, 101003, 101004 and 100002 versions

For these float versions, the theoretical First Drift Measurement Time (FDMT) is relative to the day of DET.

If DET is given in Gregorian time as "MM/DD/YYYY hh:mm:ss".

If DSP is the Drift Sampling Period (in hours).

Then FDMT = "MM/DD/YYYY 00:00:00" + *N**DSP/24

where *N* is the minimum integer value for which

"MM/DD/YYYY 00:00:00" + *N**DSP/24 > "MM/DD/YYYY hh:mm:ss"

For PROVOR 101009, 101006, 101008, 101007, 101010, 101002, 101005 and 100002 versions, drift measurements are transmitted using an interleaving scheme.

This is not the case for PROVOR 101003 and 101004 versions.

2.4.1.3 NINJA floats

2.4.1.3.1 CTD measurements for NINJA 300001, 300002 and 300003 versions

2.4.1.3.1.1 CTD measurement sampled at the beginning and end of the drift phase at parking depth

These NINJA versions provide a pressure measurement sampled at the beginning and at the end of the parking phase.

The first pressure should be stored in the N_MEASUREMENT array in association with PST (thus with a MEASUREMENT_CODE set to 250).

The second pressure should be stored in the N_MEASUREMENT with a MEASUREMENT_CODE set to 300 for PET.

The time associated with these pressure measurements cannot be estimated, so JULD should be fill value and JULD_STATUS should be ‘9’.

2.4.1.3.1.2 CTD measurements regularly sampled during the drift phase at parking depth

These NINJA versions provide pressure measurements daily sampled during the drift phase at parking depth.

These pressure measurements should be stored in the N_MEASUREMENT with a MEASUREMENT_CODE set to 290 if the float is transitioning towards PET. Even though we don't know how to compute the time for these measurements, they can be included in the N_MEASUREMENT array with the appropriate MC code.

The time associated with these pressure measurements cannot be estimated, so JULD should be fill value and JULD_STATUS should be ‘9’.

2.4.1.3.1.3. CTD measurements for NINJA 300004 version

These NINJA versions provide CTD measurements (P, T and S) sampled during the drift phase at parking depth.

These CTD measurements should be stored in the N_MEASUREMENT with a MEASUREMENT_CODE set to 290 if transitioning towards PET.

The time associated with these pressure measurements cannot be estimated, so JULD should be fill value and JULD_STATUS should be ‘9’.

2.4.1.4 SOLO-II and SOLO floats

Measurement code	SOLO II Variable	Description	Units
296	Cy>0: Eng ID=0xe2	The drift is broken into two averaged halves. Stored in Eng ID=0xe2 bytes 67-78; Time estimated from the last Fall ID=0x40 T,P pair [note: not DET] and first Rise ID=0x50 T,P pair	P(0.04db),T(0.001°C), S(0.001psu)

SOLO floats perform the same drift measurements as SOLO-II floats. Use MC=296 for the two drift measurements reported by the SOLO.

2.4.1.5 NOVA floats

Measurement code	NOVA Variable	Description	Units
290	Series of pressure Series of temperature Series of salinity	A series of CTD measurements taken during drift at user specified times	Pressure Temp Salinity
297	Minimum pressure	Minimum pressure recorded during drift phase	Pressure
298	Maximum pressure	Maximum pressure recorded during drift phase	Pressure

NOVA has PARAMETER 5 PARKING SAMPLING PERIOD which ranges from 0 – 24 hours and is user configurable. It comes set to 24, which means a CTD measurement is recorded and transmitted per time period per cycle (normally 9 measurements).

2.4.2 Miscellaneous CTD measurements

2.4.2.1 Stabilization CTD measurements

2.4.2.1.1 PROVOR floats

All PROVOR versions provide the First Stabilization Time but only some of them provide the associated pressure (**in bars**).

Detailed information can be found in paragraphs 3.2.2.6.1.2, 3.2.2.6.2.2, 3.2.2.6.3.2, 3.2.2.6.4.2, 3.2.2.6.5.2, 3.2.2.6.6.2 and 3.2.2.6.7.2.

2.4.2.1.2 NINJA floats

Some NINJA versions provide three First Stabilization Times with the associated pressures (see §3.2.2.4.1.2).

The **three** First Stabilization Times and pressures should be stored in the N_MEASUREMENT arrays with the MEASUREMENT_CODE set to 150 for the first stabilization and 189 for the second and third stabilizations.

2.4.2.2 APEX descending pressure marks

Some APEX float versions (see ANNEX I: Cookbook entry point) provide pressure marks hourly sampled during descent from the surface to the PARKING depth. These can be assigned MC = 190.

2.4.2.2.1 APEX Argos floats

For APEX Argos floats, these descending pressure marks are provided in the Auxiliary Engineering Data.

The first measurement is done at the end of the piston retraction, thus at DST, the following are done with a 1 hour period.

Note also that the descending pressure marks are **in bars**.

For APF9 APEX floats, it appears to be DST plus one hour. Here is a real example:

(Jun 12 2012 22:19:36, 15 sec) DescentInit() Surface pressure: -0.1dbar. IER: 0x00

(Jun 12 2012 22:19:41, 20 sec) PistonMoveAbsWTO() 226->066 225 224 223 222 221 220 219 218 217 216 215 214 213 212 211 210 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191 190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162 161 160 159 158 157 156 155 154 153 152 151 150 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133 132 131 130 129 128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 099 098 097 096 095 094 093 092 091 090 089 088 087 086 085 084 083 082 081 080 079 078 077 076 075 074 073 072 071 070 069 068 067 066 [873sec, 13.4Volts, 0.314Amps, CPT:873sec]

(Jun 12 2012 22:34:16, 895 sec) Descent() Pressure: 3.5

(Jun 12 2012 22:34:17, 896 sec) SetAlarm() Success: itimer=896 sec, ialarm=3600 sec

(Jun 12 2012 23:19:23, 3602 sec) Apf9Init() Wake-up initiated by interval-timer alarm signal.

(Jun 12 2012 23:19:26, 3604 sec) Descent() Pressure: 282.1

What is the time of the first descent pressure mark of the APF9a floats? The end of the piston retraction corresponds to TET, DST or another time?

The pressure mark times are computed from DST (already corrected for clock offset); thus they do not need to be corrected for clock offset but the information should be set.

The descending pressure marks are not always transmitted (depending on the remaining space in the last Argos message) but if received, the decoded and dated pressures should be stored in the N_MEASUREMENT arrays with:

- JULD or JULD_ADJUSTED, if clock offset has been applied, set to the computed times,
- JULD_QC: set to 0,
- JULD_STATUS set to 2
- CYCLE_NUMBER set to corresponding cycle number,
- MEASUREMENT_CODE set to 190,
- PRES set to decoded pressure mark values,
- PRES:resolution should be deleted (see §1.2.2)
- PRES_QCs set to 0,
- All other variables set to _FillValue.

2.4.2.2.2 APEX Iridium floats

For APEX Iridium floats, the descending pressure marks and the associated times can be retrieved from log file as the pressure values and the times of the events:

Descent() Pressure: XX.X

Descent() Pressure: YYY.Y

…

Descent() Pressure: ZZZ.Z

associated with the concerned cycle.

The pressure mark times should be first corrected from clock offset, then stored in the N_MEASUREMENT arrays with:

- JULD set to the retrieved dates or JULD_ADJUSTED if clock offset has been applied,
- JULD_QC/JULD_ADJUSTED_QC: set to 0,
- JULD_STATUS/JULD_ADJUSTED_STATUS set to 2,
- CYCLE_NUMBER set to corresponding cycle number,
- MEASUREMENT_CODE set to 190,
- PRES set to retrieved pressure mark values,
- PRES:resolution should be deleted (see §1.2.2)
- PRES_QCs set to 0,
- All other variables set to _FillValue.

2.4.2.3 APEX isopycnal pre-stabilization measurements

Isopycnal APEX floats generally provide a set of CTD measurements sampled just before the stabilization at the target sigma-theta value.

The first CTD measurement is sampled 6 hours after DST and the following ones with a 1.5 hour period.

The decoded and dated CTD measurements should be stored in the N_MEASUREMENT arrays with:

- JULD set to the computed times of JULD_ADJUSTED if clock offset has been applied,
- JULD_QC/JULD_ADJUSTED_QC: set to 0,
- JULD_STATUS/JULD_ADJUSTED_STATUS set to 2,
- CYCLE_NUMBER set to corresponding cycle number,
- MEASUREMENT_CODE set to 189,
- <PARAM> set to decoded CTD values,
- <PARAM>_QCs set to 0,
- All other variables set to _FillValue.

For Iridium floats, the decoded times should be corrected for clock offset.

For Argos floats, the dates are computed from DST, thus clock offset is already taken into account.

For Argos floats, we must consider missing Argos messages to correctly set CTD measurement times (i.e. we must take into account the missing CTD measurements, due to not received data, to correctly apply the periodicity of the times).

2.4.2.4 Dated bins of descending/ascending profiles

2.4.2.4.1 PROVOR floats

PROVOR float transmits profile data through specific Argos messages (or Iridium packets). Only the first CTD measurement of each Argos message is transmitted with its associated time. Thus, after decoding, because of the interleaving scheme used to pack the profile bin measurements, some profiles bins (around one over four or five, depending of the PROVOR version) are dated.

The dated bins of the descending (or ascending) profiles should be stored in the N_MEASUREMENT arrays (CTD measurements and associated times) with a MEASUREMENT_CODE set to 190 or 590.

2.4.2.4.2 NINJA floats

Some NINJA versions provide time information recorded during the ascent phase.

The transmitted data consist of the elapsed time for each vertical slice of ascent (from the max pressure to 2000 dbar for the first slice; and for each 100 dbar thick other slices until the surface).

These times can be used to (roughly) compute the time of one bin each 100 dbar.

The obtained dated bins of the ascending profiles should be stored in the N_MEASUREMENT arrays (CTD measurements and associated time) with a MEASUREMENT_CODE set to 590.

2.4.2.4.3 SOLO-II floats

SOLO-II floats provide timed pressure measurements on descent and ascent. The dated pressures of both the descending and ascending profiles should be stored in the N_MEASUREMENT arrays (CTD measurements and associated times) with a measurement code set to 190 for descending and 590 for ascending.

2.4.2.5 Deepest descending/ascending CTD measurements

The profile CTD measurements are stored in the PROF file. However, a copy of the deepest bin CTD measurements should be stored in the TRAJ file.

The deepest bin CTD measurements of a descending profile should be stored in the N_MEASUREMENT arrays with a MEASUREMENT_CODE set to 203.

The deepest bin CTD measurements of an ascending profile should be stored in the N_MEASUREMENT arrays with a MEASUREMENT_CODE set to 503.

2.4.2.6 Min/max pressure during drift at PARKING depth

Some APEX and PROVOR float versions, along with NOVA floats, provide the minimum and maximum values of the pressure regularly sampled during the drift at PARKING depth.

These values should be stored in the N_MEASUREMENT arrays with a MEASUREMENT_CODE set to 297 or 298.

For PROVOR floats these values are given **in bars**, the pressure resolution should be set accordingly (see §1.2.2).

2.4.2.7 Max pressure during descent to PARKING depth

Some PROVOR float versions provide the maximum pressure experienced by the float during the descent to PARKING depth. This value is given **in bars**, the pressure resolution should be set accordingly (see §1.2.2).

This pressure should be stored in the N_MEASUREMENT arrays with the MEASUREMENT_CODE set to 198.

2.4.2.8 Min/max pressure during drift at PROFILE depth

Some PROVOR float versions provide the minimum and maximum values of the pressure regularly sampled during the drift at PROFILE depth.

These values should be stored in the N_MEASUREMENT arrays with a MEASUREMENT_CODE set to 497 and 498.

These values are given **in bars**, the pressure resolution should be set accordingly (see §1.2.2).

2.4.2.9 Max pressure during descent to PROFILE depth

Some PROVOR float versions provide the maximum pressure experienced by the float during the descent to PROFILE depth. This value is given **in bars**, the pressure resolution should be set accordingly (see §1.2.2).

This pressure should be stored in the N_MEASUREMENT arrays with the MEASUREMENT_CODE set to 398.

2.4.2.10 Max pressure of the cycle

Some NINJA float versions provide the maximum pressure experienced by the float during the cycle.

This pressure should be stored in the N_MEASUREMENT arrays with the MEASUREMENT_CODE set to 498.

2.4.2.11 PROVOR Iridium spy data

Some PROVOR Iridium versions provide pressure values (**in bars**) versus time sampled during the three vertical phases of the cycle (from surface to PARKING depth, from PARKING depth to PROFILE depth and from PROFILE depth to surface).

These dated pressure measurements should be stored in the N_MEASUREMENT arrays with a MEASUREMENT_CODE set to 189 or 389 or 589 depending on the phase.

2.4.3 REPRESENTATIVE_PARK_PRESSURE

The REPRESENTATIVE_PARK_PRESSURE and REPRESENTATIVE_PARK_PRESSURE_STATUS variables in the N_CYCLE array are to include one pressure value for the drift period of the current cycle. These values can be filled in real time, but should be confirmed/updated in delayed mode. The STATUS flags are clear as to how the value is calculated and should be done for each float. Flag '1' involves finding a weighted average of regularly sampled pressures during drift (MC = 290). Flag '2' is the mean value directly provided by the floats of pressure measurements regularly sampled during drift (MC = 296). Flag '3' is the median value, directly provided by the float of pressure measurement regularly sampled during drift. Flag '4' is the pressure measured at PET. Flag '5' is the average of the min and max pressure measurements sampled during drift (MCs = 297 and 298). Flag '6' and '7' is the PARKING_PRESSURE meta-data value for floats that for some reason either missed the pressure measurement ('6') or do not make pressure measurements ('7') during drift. Flag '8' is the value estimated in Delayed Mode from float behavior/data. This may include profile limits, data from other cycles, temperature at drift, etc.

As an example here is the algorithm used to compute the RPP for ANDRO files:

The RPP is computed for each cycle and depends on the measurements sampled during the drift phase at parking depth. We start from the most reliable RPP (STEP #1) and, if the needed data are not present, we try the next step and so on until the last step (STEP #).

STEP #1:

If we have isopycnal pre-stabilization CTD measurements and CTD measurement regularly sampled during the drift phase at parking depth: the RPP is the average value weighted by the time (thus with a weight of 1.5 for the pre-stabilization CTD measurements and 6 for the other ones).

STEP #2:

If we have CTD measurement regularly sampled during the drift phase at parking depth: the RPP is the average value of these measurements (note that the measurement done at PET which is generally also present is not used in this case).

STEP #3:

If we directly have the mean value of (generally hourly) regularly sampled CTD measurements: the RPP is this mean value (note that the measurement done at PET which is generally also present is not used in this case).

STEP #4:

If we directly have the median value of regularly sampled CTD measurements: the RPP is this median value.

STEP #5:

If we have a CTD measurement done at PET: the RPP is this measurement.

STEP #6:

If we have the Min and Max pressure values of the measurements done during the drift phase at parking depth: the RPP is the mean of this two values.

STEP #7:

If we have multiple profiles during the cycle: the RPP is the average of mean and max profile values, weighted by the time spent in profile and in drift between profiles (this case is specific to APEX BOUNCE cycles).

STEP #8:

If we have the PARKING_PRESSURE meta-data value: the RPP is this value.

2.5 GROUNDED Flags

The updated GROUNDED flags can be found in Reference table 20 in the Users Manual. Here is how they should be applied both in real time (RT) and delayed mode (DM).

Y in RT: float reports it is grounded

Y in DM: DM operator decides float is grounded due to a bathymetry check or understanding of the float's behavior

N in RT: float reports it is not grounded

N in DM: DM operator decides float is not grounded due to a bathymetry check or understanding of the float's behavior

B in RT: float is grounded based on real time check with outside bathymetry database

S in RT or DM: float is known to be drifting at a shallower depth than originally programmed

U in RT: unknown

3 ANNEX A: Some definitions

Here are some definitions about elements mentioned in this document, if some of them remain unclear, please ask for a new or updated definition (argo@ucsd.edu, support@argo.net).

3.1 Definitions of Argos raw data contents

The following definitions can be found in the Argos User's manual (http://www.argos-system.org/manual/).

Let us consider Argos raw data provided in a PRV/DS command output format.

Argos float messages are collected by a given satellite during a satellite pass. A header of the satellite pass (in underlined bold in the two following examples) is added to the data by CLS.

In this header, one can find:

- The number of lines of data relative to the satellite pass header (including the header line),
- The name of the satellite,
- If a location has been computed from the data collected during the satellite pass (example 2):
  - The location class of the location,
  - The date of the location,
  - The latitude and longitude of the location.

The Argos float messages, collected during the satellite pass, follow the header.

For each we find:

- The Argos message date (time of reception of the message by the satellite),
- The Argos message redundancy,
- The Argos message content.

Example 1: A satellite pass without Argos location.

```
02412 63706 17 31 L
2007-04-24 02:40:16 2 64 A2 56 BA
B2 3C 8D 7C
AF 9F 85 AD
72 ED D5 4E
65 09 F7 5D
5C 1E B9 52
D0 CE AA 61
9A 30 00
2007-04-24 02:40:58 1 67 09 D5 CB
5F 31 75 7C
23 8D 3D 82
73 AA 30 8E
5C 46 A1 C7
68 D0 F9 91
D9 60 B9 7E
EB 38 00
```

Example 2: A satellite pass with Argos location.

```
02412 63706 33 31 D 2 2007-04-24 05:30:15 -32.189 11.405 0.000 401651871
2007-04-24 05:27:35 1 51 C9 1B A6
F4 0B 5B 5F
2E 83 F4 7F
DF E0 E4 06
```

```
1F 99 80 1E
94 6A 80 FD
FE 10 39 1E
A7 F4 00
2007-04-24 05:30:15 1 05 08 16 92
0F 83 AE 18
40 20 90 0A
20 00 19 9E
04 15 A6 00
0C 39 05 85
89 01 8E 04
C9 68 20
2007-04-24 05:30:52 1 58 A3 7D 66
F4 8B 16 DF
33 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00
2007-04-24 05:32:55 1 69 F5 58 B9
28 72 27 88
72 A5 59 91
54 B5 B4 2A
95 02 43 52
A8 49 2A 5C
E9 DD 4C F7
62 00 00
```

3.2 Cyclic Redundancy Check

All Argos floats (except SIO SOLO) have an error detection code embedded in their Argos messages. Checking this code, called Cyclic Redundancy Check (CRC), can theoretically enhance the reliability of the data by rejecting messages possibly corrupted by transmission error.

3.3 Float clock drift and clock offset

Some Argo float versions provide times for dated events or dated measurements. Over time, the float's clock may drift. Clock drift can be defined as the drift of the clock in hours/ minutes/ seconds per year. To correct for this, we must apply a clock offset where clock offset is defined as a measurement, done at a given time, of the offset of the clock due to clock drift. Thus a clock offset should be estimated for each of these float times.

Note that, in this document, float clock offset can also embrace a clock that has not been correctly set or a clock that has been set in local time. Of course, in these cases clock offset is not only revealing a drift of the float clock...

Float clock offset is defined as: Float clock offset = Float time - UTC time.

A good estimate of the clock offset can be obtained when the float transmits its Real Time Clock (RTC) time in the technical data. It can then be compared to the time from Argos of the corresponding message to compute a clock offset for all the float times of the concerned cycle.

Unfortunately this is not always the case, some floats do not transmit their RTC time and even if they do, this RTC time is not always received.

3.4 APEX Argos test/data messages

APEX Argos test messages are transmitted by an APEX float during the six hours period spent at the surface prior to its first dive. The test message contains programmed mission parameters and technical data.

After each deep cycle, APEX floats transmit the collected data in the ARGOS data messages.

3.5 APEX Deep Profile First floats

Some APEX floats are programmed to achieve their first profile shortly after deployment (for comparison to conventional CTD cast from the ship).

In the Deep Profile First (DPF) cycle the firmware is set to complete the first profile within 24 hours of deployment. The float descends to park, parks then descends to profiling depth. Then the profile commences. The duration of park is 5 minutes for Iridium floats and 60 minutes for ARGOS floats. The DPF also ignores any time of day setting. It is unclear how long the two descents are. Experience has shown that both the descent time out configuration settings are activated when the float fails to reach target depths.

3.6 APEX Time Of Day feature

Some APEX floats have the capability to schedule profiles so that the float surfaces at a particular Time Of Day (TOD).

When the TOD feature is enabled, the float RTC is used to dynamically set the end of the DOWN TIME period to a (user programmed) number of minutes after midnight.

The time of day feature is ignored by Deep Profile First floats.

3.7 APEX Auxiliary Engineering Data

For some APEX floats, the remaining space of the last Argos data message is filled with Auxiliary Engineering Data (AED).

The AED are considered to be of lower priority and will never cause an additional Argos data message to be generated.

4 ANNEX B: Transmission End Time estimation for an APEX Argos float

The methods used to estimate TET for an Apex Argos float are all based on the float's theoretical functioning.

4.1 Apex float theoretical functioning

The core cycle of an APEX float is based on four main parameters as illustrated in Figure 2:

1. DOWN TIME: The time period including the descent and the drift at depth,
2. UP TIME: The time period including the ascent and the drift at surface,
3. PARKING PRESSURE: The aimed depth for the drift phase,
4. DEEPEST PRESSURE: The aimed depth for the start of the profile (also called PROFILE PRESSURE in the document).

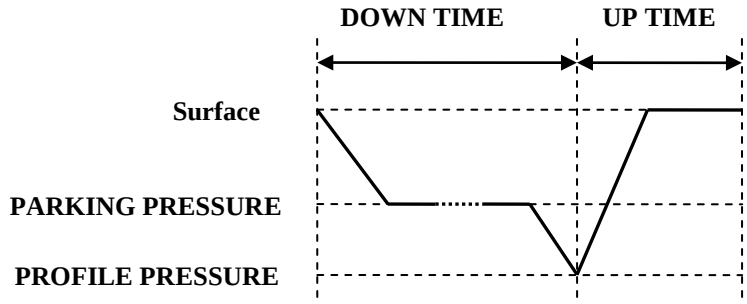


Figure 2: Main parameters of an APEX cycle

The float can also be programmed to achieve a deep profile once over *N* cycles (in this case, *N* is called the Park and Profile parameter).

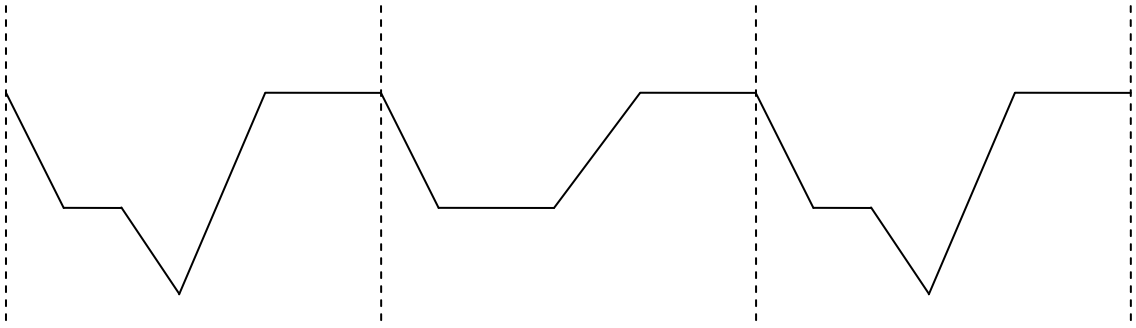


Figure 3: Example of programmed mission with a PnP parameter of 2 (deep profile every second cycle)

Finally, the float can also be programmed to achieve its first profile shortly after deployment (for comparison to conventional CTD cast from the ship).

During its first cycle, this Deep Profile First (DPF) float directly descends to the PROFILE PRESSURE and immediately achieves its first profile.

4.2 The Park et al. method

The Park et al. method is based on the assumption that the Apex Argos float always starts to profile at the end of the DOWN TIME. Unfortunately, this is only the case for floats when PARKING and PROFILE pressures are equal.

The float starts its descent to PROFILE PRESSURE hours before the end of the DOWN TIME. The length of time to descend from the drift pressure to the profile pressure varies. It is user specified for APF9 floats. Often users choose somewhere between 4 and 6 hours. Six hours is shown in the diagram below.

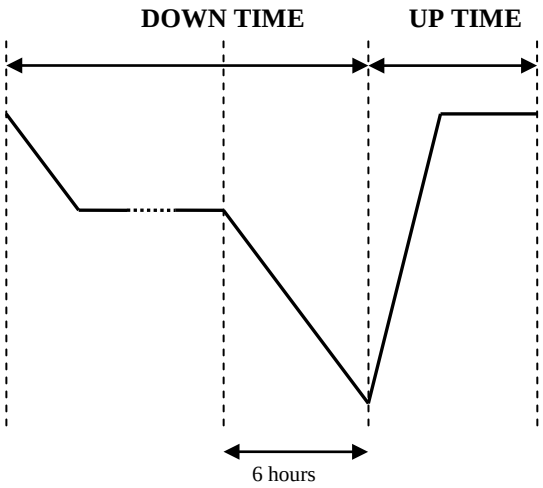


Figure 4: Schematics of a cycle where PARKING and PROFILE pressures differ

The float then starts to profile when one of the following conditions is met:

- If the PROFILE PRESSURE depth is reached,
- If the end of the DOWN TIME period is reached.

Note that the second assumption is only true for floats without Time Of Day (TOD) capability or with the TOD feature disabled.

In the first case, the PROFILE depth can be reached before the end of the DOWN TIME period implying an arrival at the surface earlier than for the theoretical case (under the assumption of a constant ascent rate), see next figure.

After such a cycle, if it reached the surface before the end of the DOWN TIME period, the float increases the value of its profile piston position count to decrease its descent rate from PARKING depth to PROFILE depth for the next cycle.

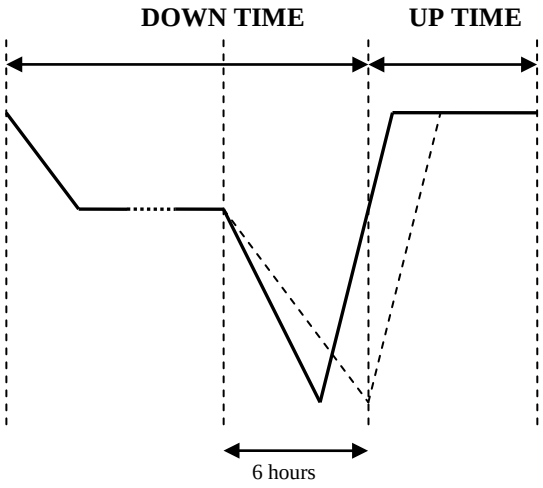


Figure 5: First case, PROFILE depth reached before end of the DOWN TIME period

In the second case, at the end of the DOWN TIME, the float has not necessarily reached the PROFILE depth implying an arrival at the surface earlier than for the theoretical case but later than for the first case (under the assumption of a constant ascent rate), see next Figure.

After such a cycle, the float decreases the value of its profile piston position count to increase its descent rate from PARKING depth to PROFILE depth for the next cycle.

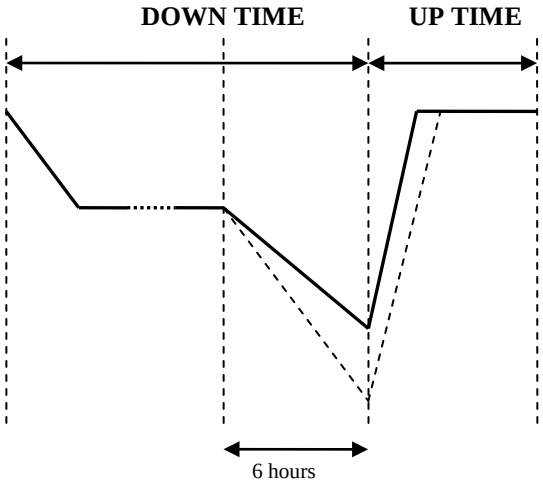


Figure 6: Second case, PROFILE depth not reached before end of DOWN TIME period

This behavior produces continual variation of AETs; 2 main values can roughly be distinguished as illustrated in the next figure.

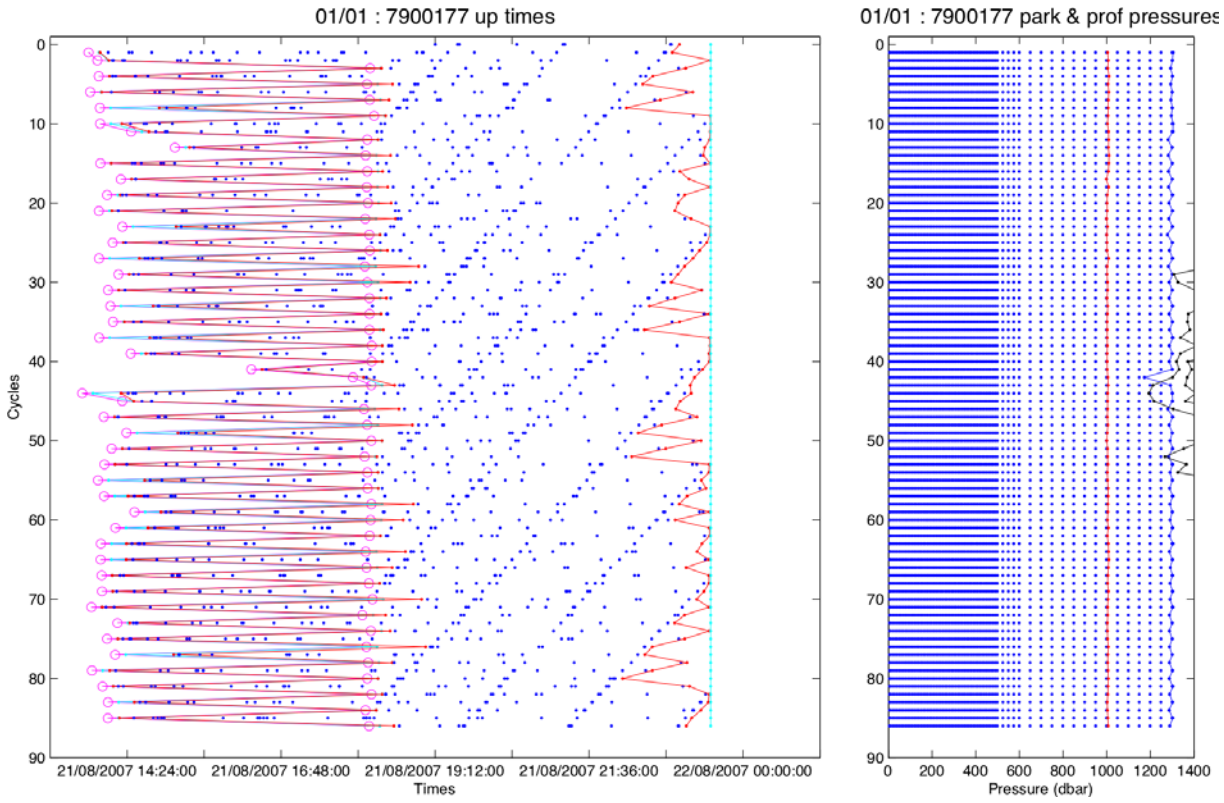


Figure 7: Example of behaviour of a float which drift and profile at different depths

In this figure, we can see on the left part **the times** and on the right part **the pressures of the CTD measurements**.

The times are drawn in reference to cycle #0. For a given cycle, we can find, in chronological order: the AET (pink circle), the TST (light blue dot), the FMT (red dot), the location times (blue dots), the LMT (red dot) and the TET (light blue dot).

We clearly see 2 sets of AETs: the first one around 21/08/2007 14:16:00 and the second one around 21/08/2007 18:14:00 (in reference to cycle #0).

The pressures of the CTD measurements are drawn in blue dots for the profile measured pressures and in red dots for the Representative Parking Pressure (RPP) pressures. The black dots represent the local bathymetry provided by the ETOPO2 atlas.

We see that this float (WMO #7900177) has drift around 1000 dbar and profiled from 1300 dbar.

More detailed information can be found in the Kobayashi and Nakajima paper.

In the DEP data set, only 22.5% (i.e. 806 floats) of the 3622 APEX Argos floats drift and profile at the same depth and are then eligible for the Park et al. method.

Consequently, we decided to forget this method and to specify another one applicable to all APEX Argos floats.

4.3 The proposed method

The proposed method is based on two algorithms that can be alternatively used, depending of the number of cycles available for a given float.

The first algorithm uses half of the Park et al. method to determine the maximum envelope of the LMTs.

The second algorithm also estimates TETs but takes into account the drift of the float clock.

These algorithms are based on the duration of the APEX Argos cycles which are always known CYCLE TIME = DOWN TIME + UP TIME.

We cannot say that these durations are constant but we can say that their theoretical values are known (i.e. predictable).

In the ANDRO data set, these cycle durations vary only for two float versions:

- For APEX bounce floats: bounce cycle (even numbered cycles) duration is smaller than usual cycle (odd numbered cycles) duration (but both theoretical values are known),
- For APEX "seasonal" floats: the cycle duration is two (known) values depending of the day in the year (concerned versions are 001046 and 001055).

4.3.1 First algorithm: Transmission End Times estimated from the maximum envelope of the Last Message Times

The maximum envelope of the LMTs is a lower bound of the TET as illustrated in the following figure.

The first algorithm is the following:

1. Identify the reference cycle (number *N*).
   The reference cycle is the first received cycle.
   For DPF floats however the reference cycle can't be cycle #0 it is then the first received cycle with a number > 0.
2. Compute the reference value of the LMT of each cycle (LMT_{RV}).
   For cycle #*i*:
   $LMT_{RV}(i) = LMT(i) - duration(N, i)$
   Where duration(*N*, *i*) is the theoretical duration between cycle #*N* and cycle #*i*.
   If the cycle theoretical duration is constant and equal to CYCLE TIME (all floats except bounce and "seasonal" ones) duration(*N*, *i*) = (*i* - *N*)*CYCLE TIME and then
   $LMT_{RV}(i) = LMT(i) - (i - N)$*CYCLE TIME
3. Find the maximum value of the obtained $LMT_{RV}(i)$ values.
4. Compute the TET of each cycle.
   $TET(i) = max(LMT_{RV}) + duration(N, i)$
   Here again if the cycle theoretical duration is constant we obtain
   $TET(i) = max(LMT_{RV}) + (i - N)$*CYCLE TIME

Note that for DPF floats we can choose to set TET equal to LMT for cycle #0 (i.e. for DPF floats: TET(0) = LMT(0)).

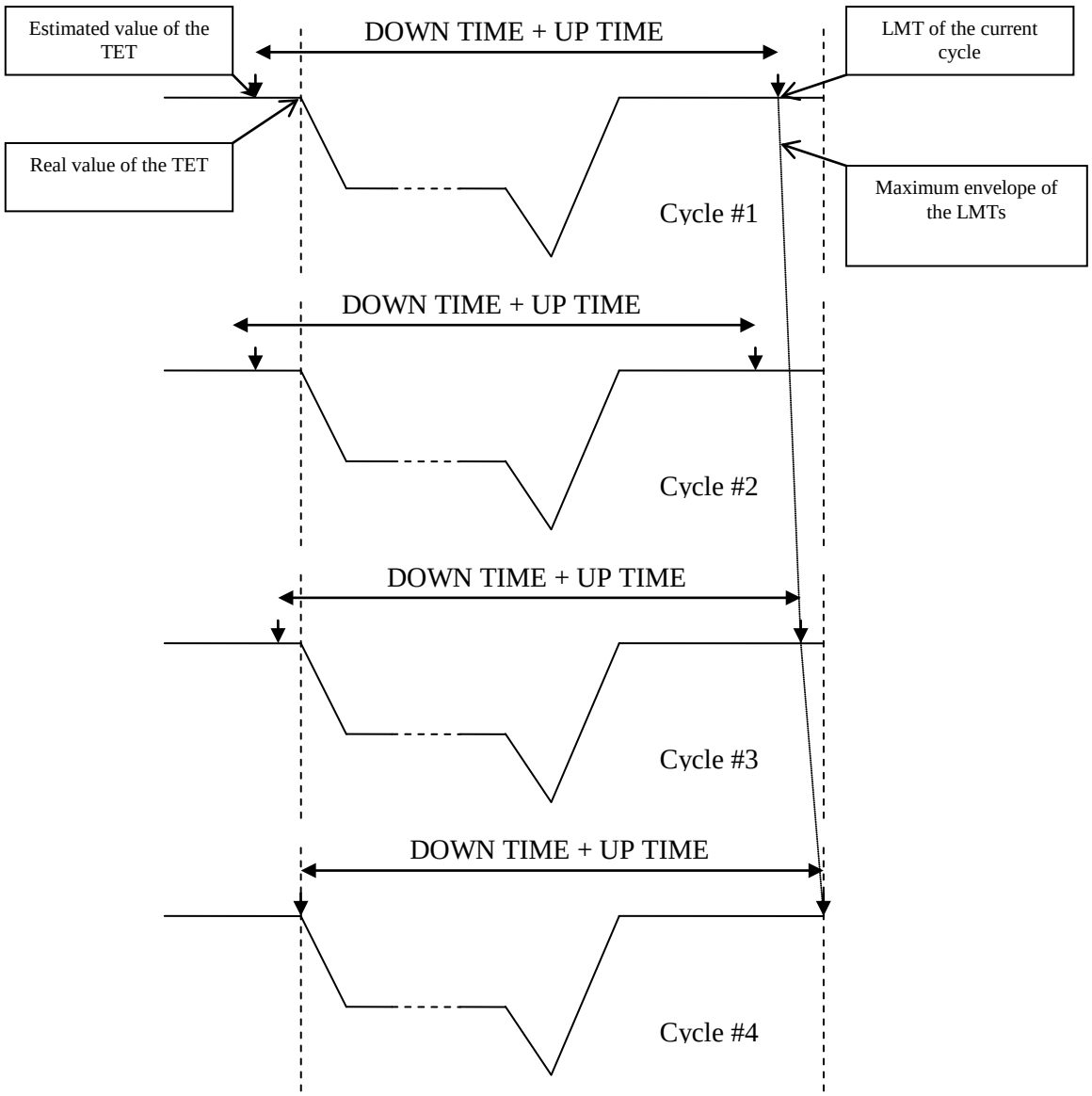


Figure 8: Estimation of the TETs from the maximum envelope of the LMTs

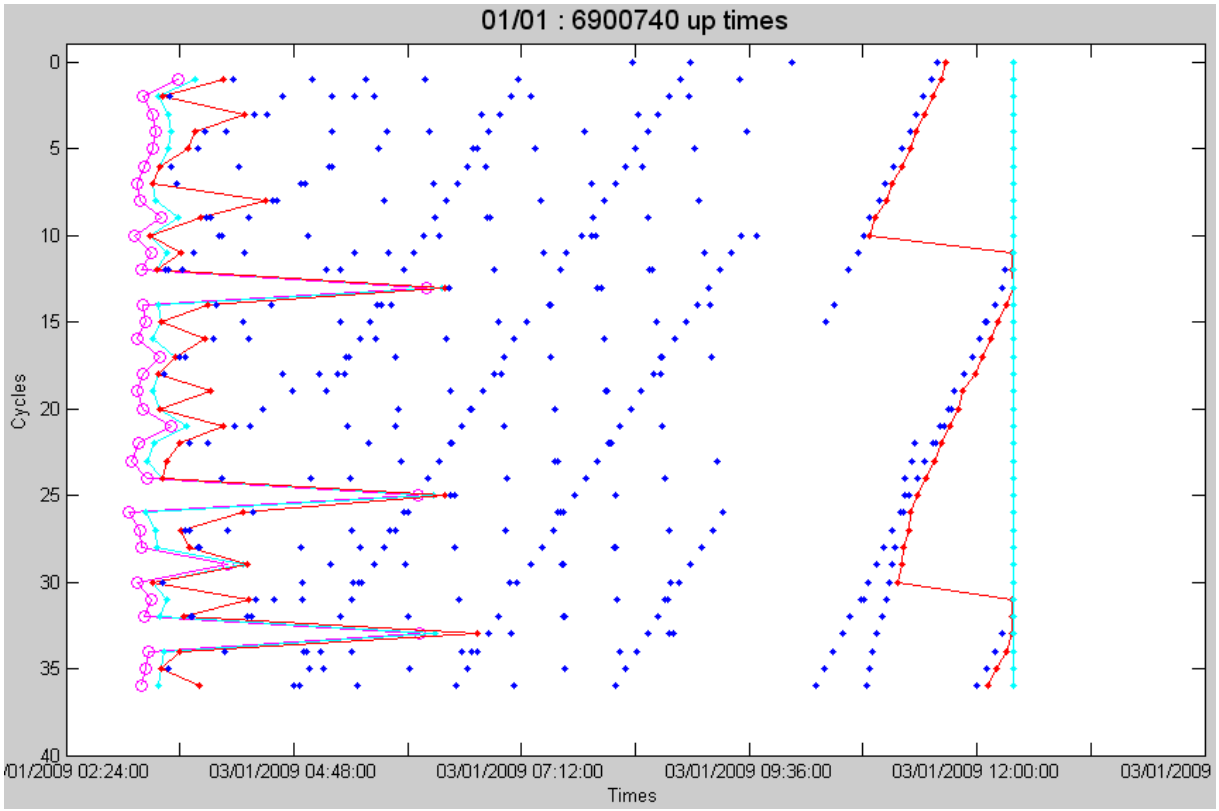


Figure 9: Example of estimation of TETs with the maximum envelope of the LMTs

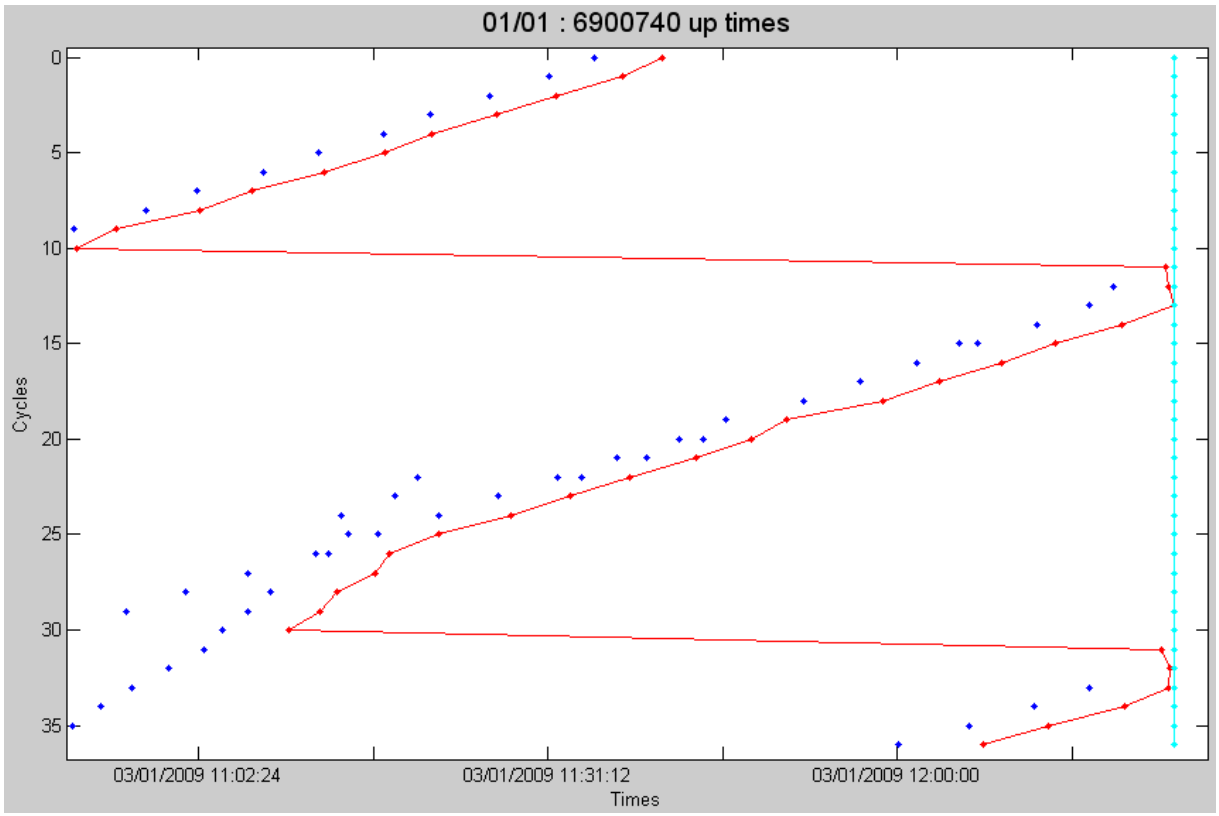


Figure 10: A zoom of the previous figure shows that estimated TETs are defined by the LMT of cycle #13

4.3.2 Second algorithm: Transmission End Times estimated by a method that takes the float clock offset into account

The TETs obtained so far have been estimated under the assumption that the theoretical CYCLE TIME value is the real duration of the cycles.

However, the on board float clock can drift during float life implying variations of the cycle durations.

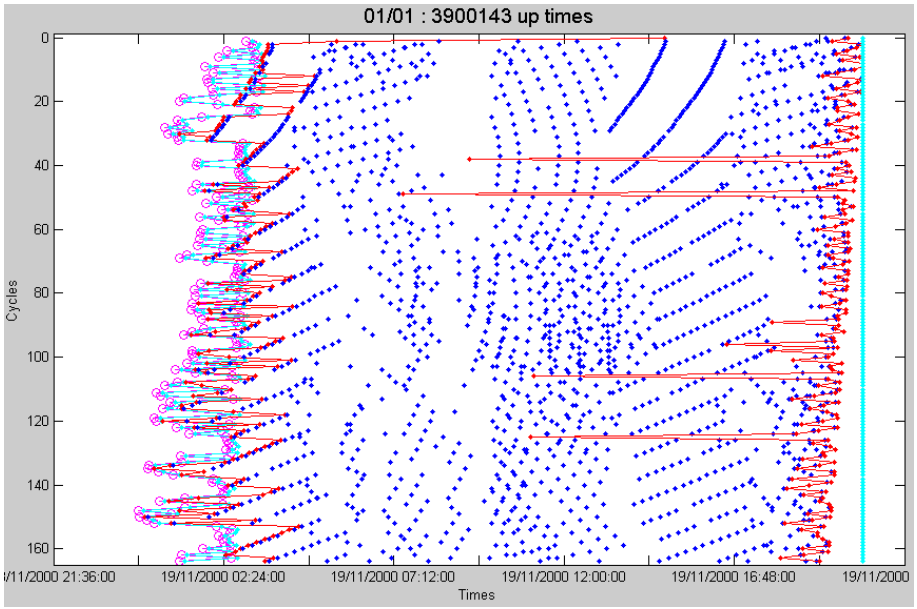


Figure 11: Example of a rather important negative clock drift (estimated to - 00:13:59 per year which implies a maximum difference of 00:58:34 between TETs estimated with or without taking the clock drift into account). We clearly see a regular decrease of cycle duration.

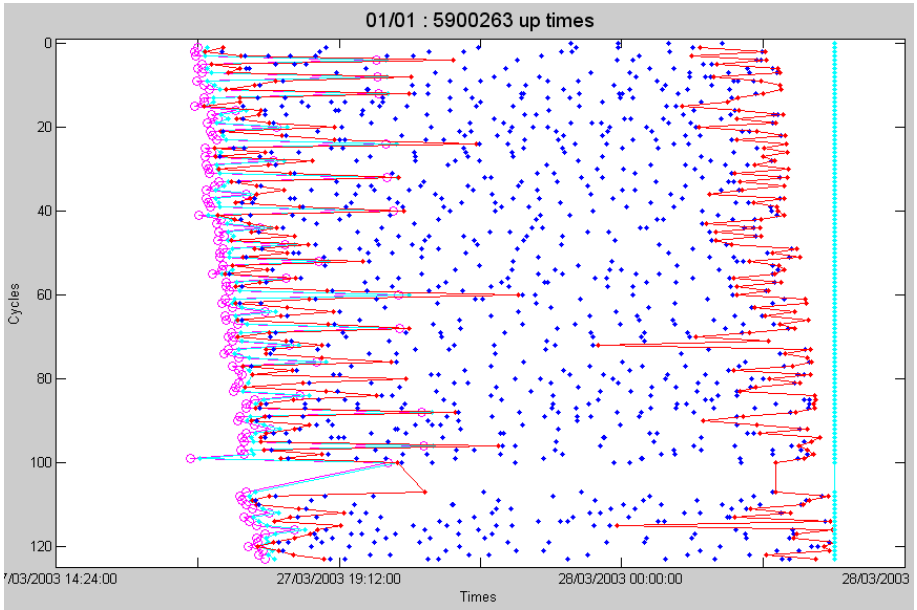


Figure 12: Example of a rather important positive clock drift (estimated to + 00:17:15 per year which implies a maximum difference of 00:58:56 between TETs estimated with or without taking the clock drift into account). We clearly see a regular increase of cycle duration.

The proposed algorithm can also take into account erroneous theoretical CYCLE TIME values.

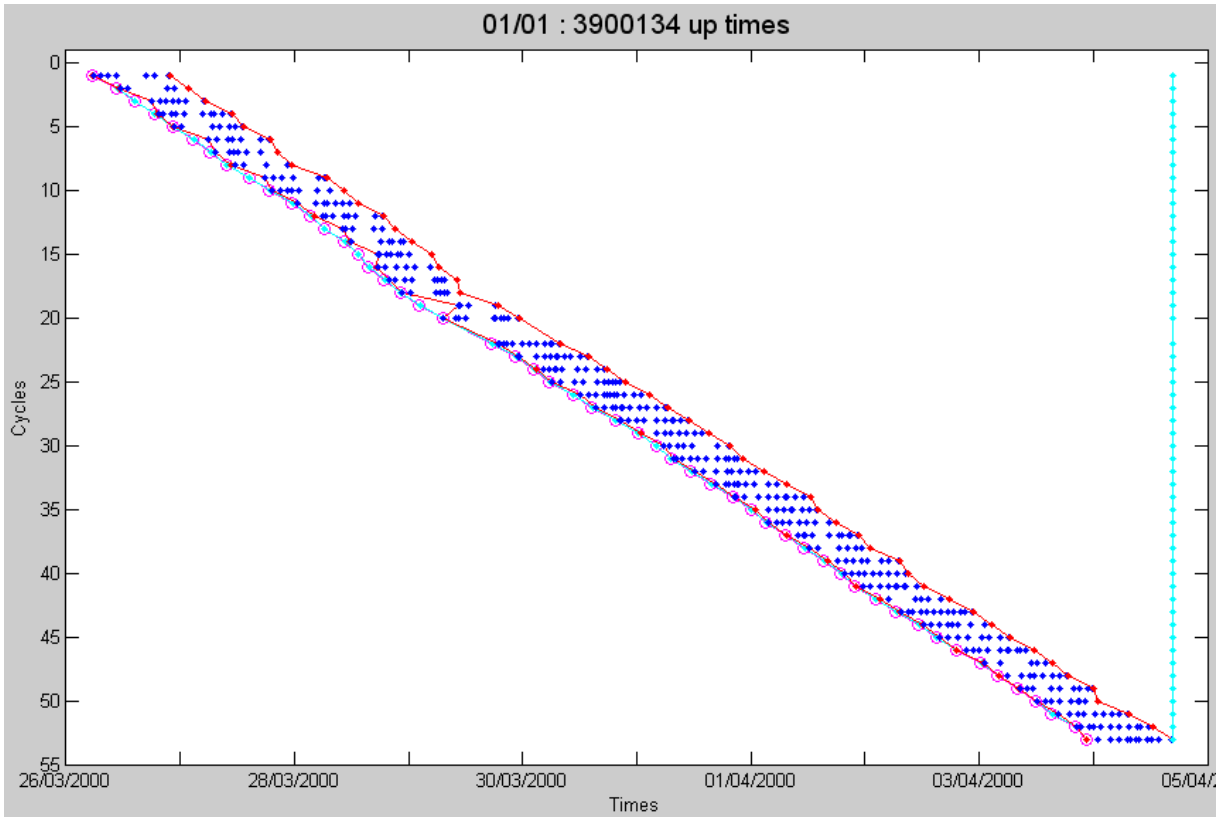


Figure 13: Example of an erroneous theoretical cycle time (DOWN TIME + UP TIME = 228 + 22 = 240 hours whereas the cycle time is around 244 hours) seen as a very important positive clock drift (estimated to + 147:44:36 per year which implies a maximum difference of 209:26:54 between TETs estimated with or without taking the clock drift into account).

In the second algorithm we **linearly estimate the float clock offset** to take it into account in the TETs estimation.

The algorithm stands in six steps illustrated in the following paragraphs.

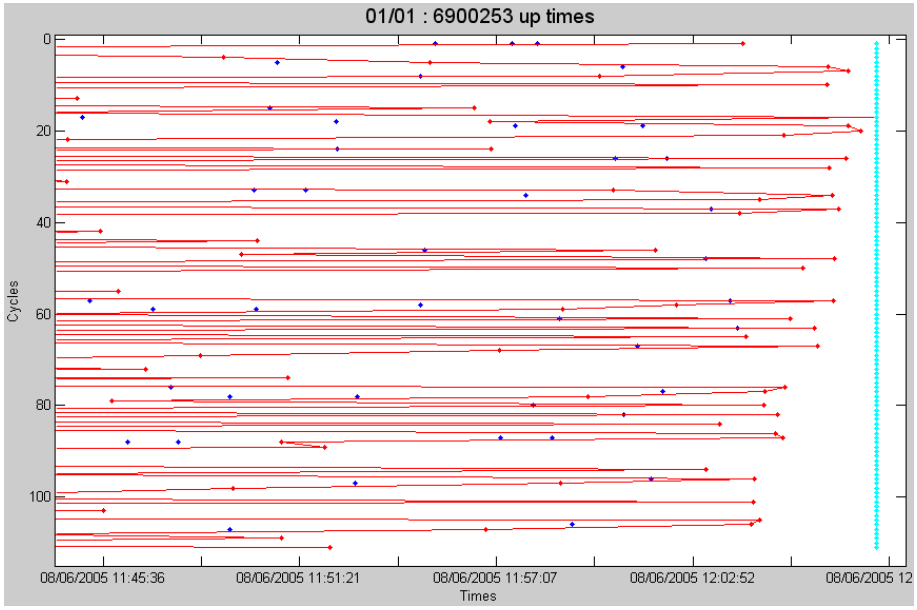


Figure 14: Example of TETs estimated with the maximum envelope of the LMTs (first algorithm), we want to estimate the TETs taking into account the float clock offset.

4.3.2.1 Step #1

First determine the convex envelope of the LMTs.

This can be done by various algorithms; the one we used is in section 5.3.2.7.

Next, obtain a subset of the LMTs (the base points) that define the convex envelope of all the LMTs as illustrated in the next figure.

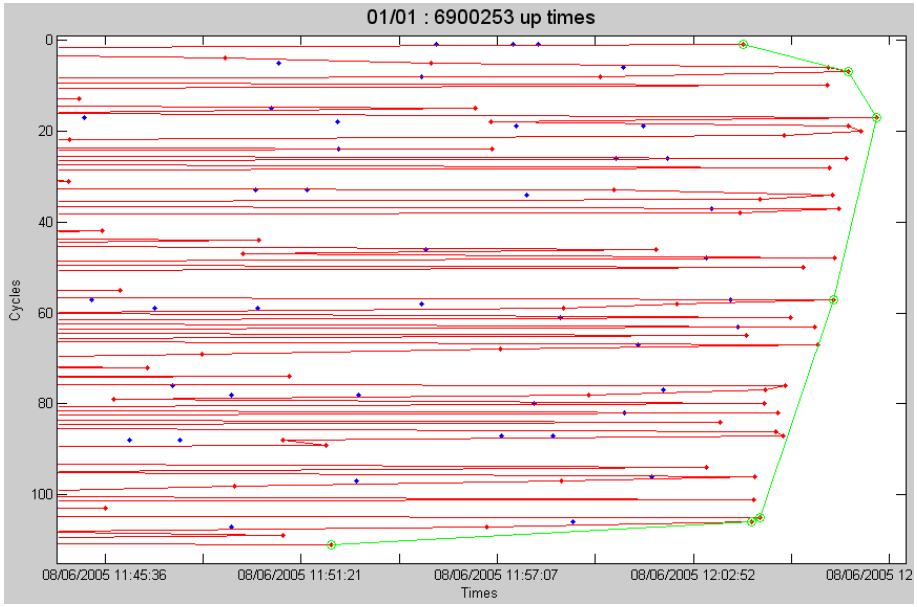


Figure 15: The convex envelope of the LMTs (green line) defined by the base points (green circles)

4.3.2.2 Step #2

The second step consists in setting a point on the convex envelope for all received cycles.

We thus obtain a set of points on the convex envelope.

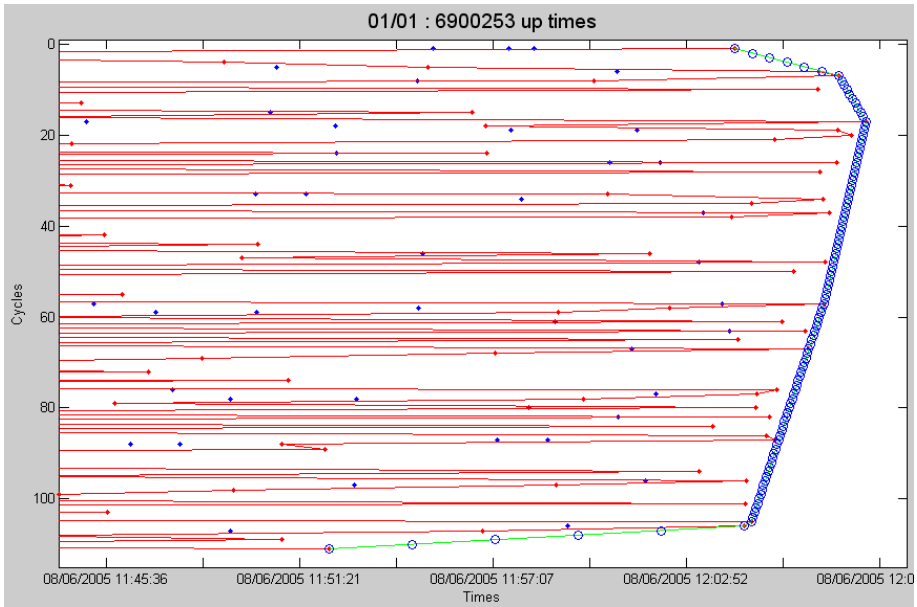


Figure 16: The 111 received cycles are set on the convex envelope (blue circles)

4.3.2.3 Step #3

The third step consists in deleting some first and last points on the convex envelope.

2/5 of the points on the convex envelope are deleted: the first 1/5 of the point and the last 1/5 of the points.

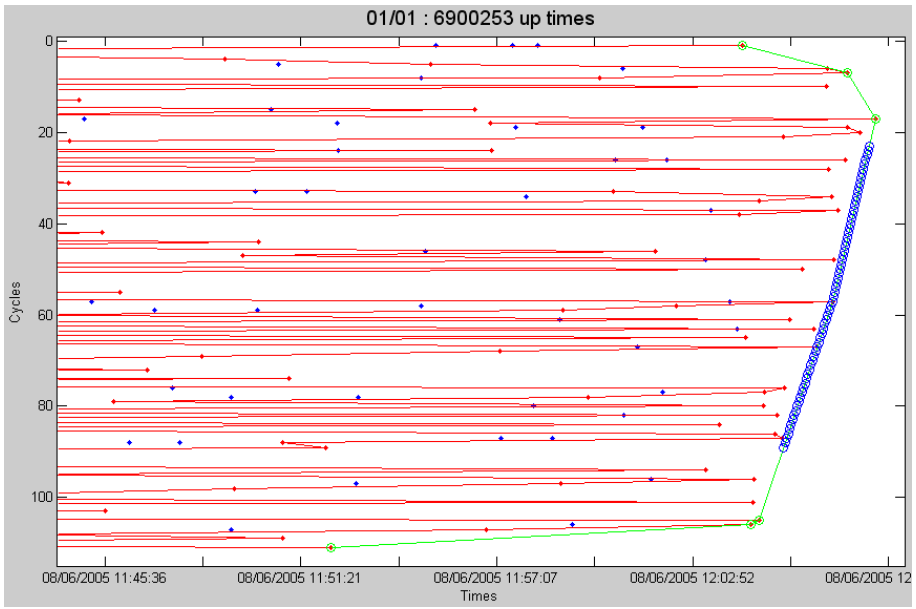


Figure 17: The first and last 22 points are ignored (only the 67 central points are preserved)

4.3.2.4 Step #4

The fourth step consists in linearly fitting the points on the convex envelope (in a least squares sense).

We used the "polyfit" Matlab function to do that.

The slope of the resulting line is our estimated clock offset.

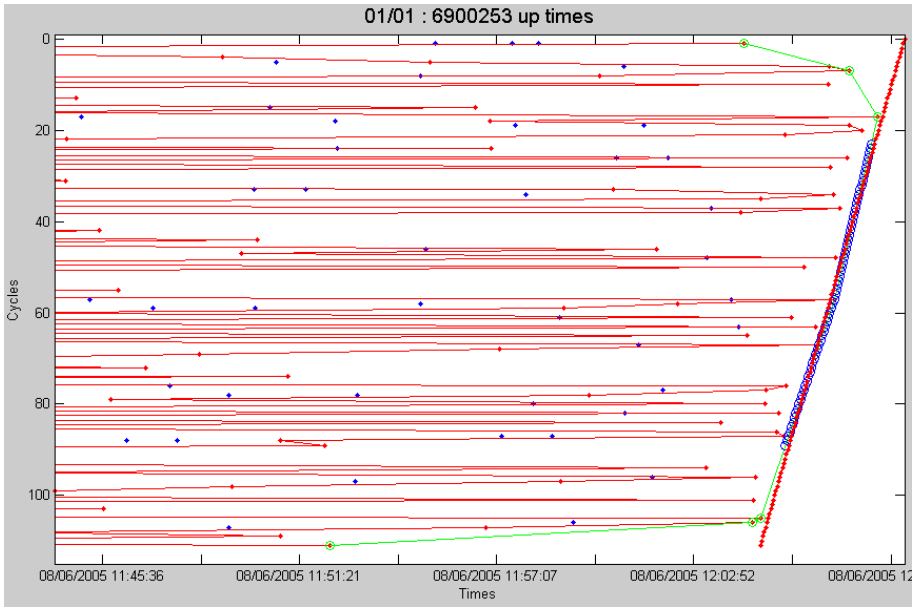


Figure 18: The blue circles are linearly fitted (red line)

4.3.2.5 Step #5

The fifth step consists of adjusting the estimated clock offset on the convex envelope.

Obviously, the contact point(s) is(are) base point(s), thus we try each base point of the convex envelope.

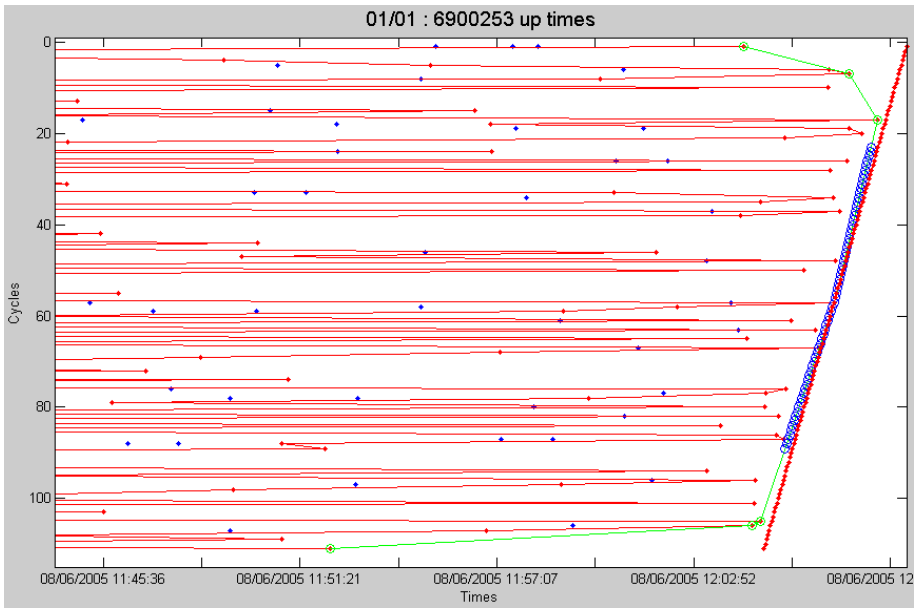


Figure 19: The red line is adjusted on the convex envelope

4.3.2.6 Step #6

For each received cycle, we compute the estimated TETs situated on the adjusted line of estimated clock offset.

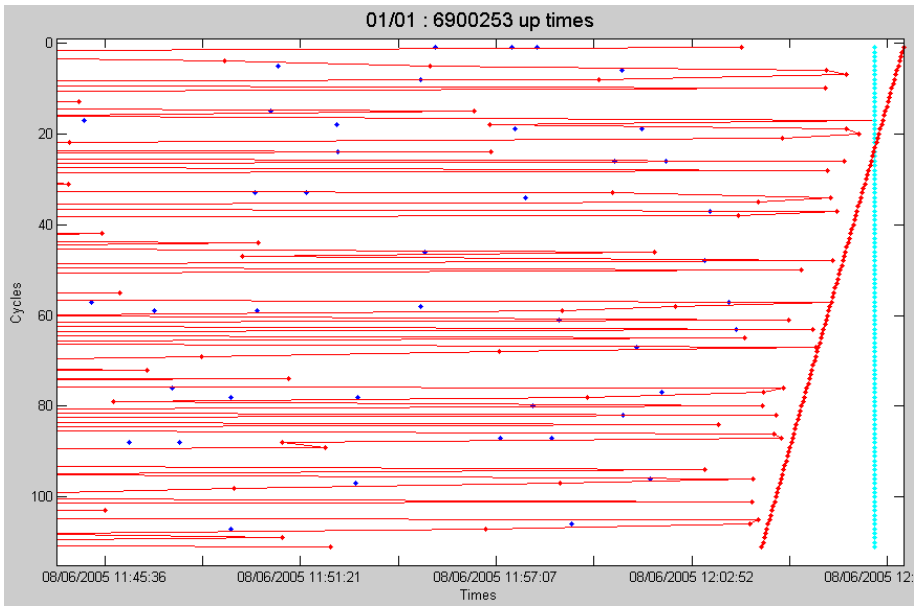


Figure 20: Estimated TETs with (red points on the red line) or without (light blue points on the light blue line) taking float clock drift into account. In this example, clock drift is estimated to -00:00:55 per year which implies a maximum difference of 00:03:19 between the red and the light blue lines

5.3.2.7. Example of implementation of the second algorithm

```
% Extract of the "dep_add_apx_descent_start_with_clock_drift.m" program used in
% the ANDRO toolbox.
% AUTHORS  : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% Inputs:
% cycles: cycle numbers
% cycleTime : cycle durations
% tabLastPosDate: date of the last Argos location for each cycle
% tabLastMsgDate: date of the last received Argos message for each cycle

% Due to possible anomalies in cycle duration (DPF floats, clock jumps, etc),
% the cycles have to be processed by slices.
% A slice is defined by a number of consecutive cycles.
% For each slice we can define the following parameters:
%    - tabFirstId/tabLastId: first/last Id of the cycles that define the slice
%    - tabNbMinPts: minimum number of cycles needed to estimate the clock drift
%      for a given slice (set to 33 by default)

% Outputs:
% tabArgosStopFinal: the max envelope of the LMTs
% tabSavEstClockDrift: the estimated clock drift for each slice


% max envelope of the LMTs
tabArgosStopFinal = ones(max(cycles)+1, 1)*g_dateDef;

% estimated clock drifts
tabSavEstClockDrift = ones(length(tabFirstId), 1)*g_dateDef;

% processing of each slice
tabIdSlice = [];
tabCoef = [];
tabNbPts = [];
tabCycleTime = [];
for idSlice = 1:length(tabFirstId)

   firstId = tabFirstId(idSlice);
   lastId = tabLastId(idSlice);
   fprintf('   Slice #%d: cycles %d to %d\n', ...
      idSlice, firstId-1, lastId-1);
   cycleSlice = cycles(firstId:lastId);
   nbMinPts = tabNbMinPts(idSlice);

   % last Argos message date (or last location date) of each cycle
   convexEnv = tabLastMsgDate(firstId:lastId);
   argosLastLoc = tabLastPosDate(firstId:lastId);
   idNotDated = find(convexEnv == g_dateDef);
   convexEnv(idNotDated) = argosLastLoc(idNotDated);
   oriDates = convexEnv;

   % maximum envelope without clock drift estimation
   maxEnv = convexEnv;
   idDated = find(maxEnv ~= g_dateDef);
```

```
  maxEnvDates = maxEnv(idDated);
  maxEnvCycles = cycleSlice(idDated);
  if (~isempty(maxEnvCycles))
    maxEnvDates = maxEnvDates - compute_duration(maxEnvCycles, maxEnvCycles(1),
cycleTime);
    maxDate = max(maxEnvDates);
    maxEnv(idDated) = maxDate + compute_duration(maxEnvCycles, maxEnvCycles(1), cycleTime);

    tabArgosStopFinal(firstId:lastId) = maxEnv;
  end

  % compute the convex envelope of the LMTs
  oneMore = 1;
  while (oneMore)
    oneMore = 0;
    idDated = find(convexEnv ~= g_dateDef);
    if (~isempty(idDated))
      idStart = idDated(1);
      stop = 0;
      while (~stop)
        stop = 1;
        idIdDated = find(idDated > idStart);
        for id = 1:length(idIdDated)-1
          idFirst = idStart;
          xFirst = compute_duration(cycleSlice(idStart), cycleSlice(1), cycleTime);
          yFirst = convexEnv(idFirst);
          idCheck = idDated(idIdDated(id));
          xCheck = compute_duration(cycleSlice(idCheck), cycleSlice(1), cycleTime);
          yCheck = convexEnv(idCheck);
          idLast = idDated(idIdDated(id+1));
          xLast = compute_duration(cycleSlice(idLast), cycleSlice(1), cycleTime);
          yLast = convexEnv(idLast);

          coefA = (yFirst - yLast)/(xFirst - xLast);
          coefB = yFirst - coefA*xFirst;

          expValue = coefA*xCheck + coefB;
          if (yCheck <= expValue)
            % this point is not part of the convex envelope
            convexEnv(idCheck) = g_dateDef;
            % one point has been excluded, a new iteration is needed
            oneMore = 1;
          else
            % we must start from this base point (possibly in the convex
            % envelope)
            idStart = idCheck;
            stop = 0;
            break;
          end
        end
      end
    end
  end

  % the dates of the excluded points are set on the convex envelope
```

```
  convexEnvAll = convexEnv;
  idDated = find(convexEnv ~= g_dateDef);
  for id = 1:length(idDated)-1
    idFirst = idDated(id);
    xFirst = compute_duration(cycleSlice(idFirst), cycleSlice(1), cycleTime);
    yFirst = convexEnv(idFirst);
    idLast = idDated(id+1);
    xLast = compute_duration(cycleSlice(idLast), cycleSlice(1), cycleTime);
    yLast = convexEnv(idLast);

    polyCoef = polyfit([xFirst xLast], [yFirst yLast], 1);

    idNew = [idFirst+1:idLast-1];
    idNew(find(oriDates(idNew) == g_dateDef)) = [];

    convexEnvAll(idNew) = polyval(polyCoef, compute_duration(cycleSlice(idNew), cycleSlice(1),
cycleTime));
  end

  % clock drift estimation
  idDated = find(convexEnvAll ~= g_dateDef);
  if (length(idDated) < nbMinPts)
    fprintf('      Not enough points (%d) => no clock drift estimation\n', ...
      length(idDated));
  else

    % delete the first and last 1/5 of the points
    nbTotal = length(idDated);
    nbToDel = fix(length(idDated)/5);
    fprintf('      Number of points total:del/used/del %d:%d/%d/%d\n', ...
      length(idDated), nbToDel, length(idDated)-2*nbToDel, nbToDel);
    convexEnvAll(idDated(1:nbToDel)) = g_dateDef;
    convexEnvAll(idDated(end-(nbToDel-1):end)) = g_dateDef;

    % linear estimation of the clock drift with the remaining points
    idDated = find(convexEnvAll ~= g_dateDef);
    xVal = compute_duration(cycleSlice(idDated), cycleSlice(1), cycleTime);
    yVal = convexEnvAll(idDated);

    polyCoef = polyfit(xVal, yVal, 1);

    if (length(unique(cycleTime(cycleSlice+1))) == 1)
      tabIdSlice = [tabIdSlice; idSlice];
      tabCoef = [tabCoef; polyCoef(1)];
      tabNbPts = [tabNbPts; length(idDated)];
      tabCycleTime = [tabCycleTime; unique(cycleTime(cycleSlice+1))];
    end

    % set the estimate on the convex envelope (it is a base point of the
    % envelope)
    basePoint = [];
    idDated = find(convexEnv ~= g_dateDef);
    for id = 1:length(idDated)
      idPoint = idDated(id);
      xPoint = compute_duration(cycleSlice(idPoint), cycleSlice(1), cycleTime);
```

```
      yPoint = convexEnv(idPoint);

      coefB = yPoint - polyCoef(1)*xPoint;
      curPolyCoefCur = [polyCoef(1) coefB];

      yAll = polyval(curPolyCoefCur, compute_duration(cycleSlice(idDated), cycleSlice(1),
cycleTime));

      idKo = find(convexEnv(idDated) > yAll);
      if (isempty(idKo) || ((length(idKo) == 1) && (idKo == id)))
        basePoint = [basePoint; idPoint];
      end
    end

    if (length(basePoint) > 1)
      comment = sprintf(' %d', cycleSlice(basePoint));
      fprintf('      WARNING : %d base points (cycles %s)\n', ...
        length(basePoint), comment);
      basePoint = basePoint(1);
    end

    if (isempty(basePoint))
      fprintf('      ERROR : no base point => nothing done\n');
    else
      % compute the clock drift

      idPoint = basePoint;
      xPoint = compute_duration(cycleSlice(idPoint), cycleSlice(1), cycleTime);
      yPoint = convexEnv(idPoint);

      coefB = yPoint - polyCoef(1)*xPoint;
      polyCoefFinal = [polyCoef(1) coefB];

      clockDrift = polyval(polyCoefFinal, compute_duration(cycleSlice, cycleSlice(1), cycleTime));
      clockDrift(find(oriDates == g_dateDef)) = g_dateDef;

      idDated = find(clockDrift ~= g_dateDef);
      totalClockDrift = clockDrift(idDated(end)) - ...
        (clockDrift(idDated(1)) + compute_duration(cycleSlice(idDated(end)), cycleSlice(idDated(1)),
cycleTime));
      yearClockDrift = totalClockDrift*365/(compute_duration(cycleSlice(idDated(end)),
cycleSlice(idDated(1)), cycleTime));
      fprintf('      Clock drift (per year): %s\n', ...
        format_time(yearClockDrift*24));

      if (~isnan(yearClockDrift))
        tabClockDrift = [tabClockDrift; yearClockDrift];
        tabSavEstClockDrift(idSlice) = yearClockDrift;
      end

      idDated = find((maxEnv ~= g_dateDef) & (clockDrift ~= g_dateDef));
      fprintf('      Max diff: %s\n', ...
        format_time(max(abs(maxEnv(idDated) - clockDrift(idDated)))*24));

      tabArgosStopFinal(firstId:lastId) = clockDrift;
```

```
    end
  end
end

% same processing for slices where clock drift estimation has not been done (we
% use a weighted average of the already computed estimates)
if ((length(tabIdSlice) ~= length(tabFirstId)) && ~isempty(tabIdSlice))

  for idSlice = 1:length(tabFirstId)
    if (isempty(find(tabIdSlice == idSlice, 1)))

      firstId = tabFirstId(idSlice);
      lastId = tabLastId(idSlice);
      fprintf('   Phase2, slice #%d: cycles %d à %d\n', ...
        idSlice, firstId-1, lastId-1);
      cycleSlice = cycles(firstId:lastId);

      if (length(cycleSlice) == 1)
        continue;
      end

      % compute the mean coefficient (weighted average)
      % be careful, we must use only the coefficients available for this cycle
      % time (case of the seasonal APEXs)
      if (length(unique(cycleTime(cycleSlice+1))) == 1)
        cycTime = unique(cycleTime(cycleSlice+1));
        idMead = find(tabCycleTime == cycTime);
        if (~isempty(idMead))
          meanCoef = sum(tabCoef(idMead).*tabNbPts(idMead))/sum(tabNbPts(idMead));
        else
          continue;
        end
      end

      % last Argos message date (or last location date) of each cycle
      convexEnv = tabLastMsgDate(firstId:lastId);
      argosLastLoc = tabLastPosDate(firstId:lastId);
      idNotDated = find(convexEnv == g_dateDef);
      convexEnv(idNotDated) = argosLastLoc(idNotDated);
      oriDates = convexEnv;

      % maximum envelope without clock drift estimation
      maxEnv = convexEnv;
      idDated = find(maxEnv ~= g_dateDef);
      maxEnvDates = maxEnv(idDated);
      maxEnvCycles = cycleSlice(idDated);
      if (~isempty(maxEnvCycles))
        maxEnvDates = maxEnvDates - compute_duration(maxEnvCycles, maxEnvCycles(1),
cycleTime);
        maxDate = max(maxEnvDates);
        maxEnv(idDated) = maxDate + compute_duration(maxEnvCycles, maxEnvCycles(1),
cycleTime);
      end

      % compute the convex envelope of the LMTs
```

```
oneMore = 1;
while (oneMore)
  oneMore = 0;
  idDated = find(convexEnv ~= g_dateDef);
  if (~isempty(idDated))
    idStart = idDated(1);
    stop = 0;
    while (~stop)
      stop = 1;
      idIdDated = find(idDated > idStart);
      for id = 1:length(idIdDated)-1
        idFirst = idStart;
        xFirst = compute_duration(cycleSlice(idStart), cycleSlice(1), cycleTime);
        yFirst = convexEnv(idFirst);
        idCheck = idDated(idIdDated(id));
        xCheck = compute_duration(cycleSlice(idCheck), cycleSlice(1), cycleTime);
        yCheck = convexEnv(idCheck);
        idLast = idDated(idIdDated(id+1));
        xLast = compute_duration(cycleSlice(idLast), cycleSlice(1), cycleTime);
        yLast = convexEnv(idLast);

        coefA = (yFirst - yLast)/(xFirst - xLast);
        coefB = yFirst - coefA*xFirst;

        expValue = coefA*xCheck + coefB;
        if (yCheck <= expValue)
          % this point is not part of the convex envelope
          convexEnv(idCheck) = g_dateDef;
          % one point has been excluded, a new iteration is needed
          oneMore = 1;
        else
          % we must start from this base point (possibly in the
          % convex envelope)
          idStart = idCheck;
          stop = 0;
          break;
        end
      end
    end
  end
end

% the dates of the excluded points are set on the convex envelope
convexEnvAll = convexEnv;
idDated = find(convexEnv ~= g_dateDef);
for id = 1:length(idDated)-1
  idFirst = idDated(id);
  xFirst = compute_duration(cycleSlice(idFirst), cycleSlice(1), cycleTime);
  yFirst = convexEnv(idFirst);
  idLast = idDated(id+1);
  xLast = compute_duration(cycleSlice(idLast), cycleSlice(1), cycleTime);
  yLast = convexEnv(idLast);

  polyCoef = polyfit([xFirst xLast], [yFirst yLast], 1);
```

```
      idNew = [idFirst+1:idLast-1];
      idNew(find(oriDates(idNew) == g_dateDef)) = [];

      convexEnvAll(idNew) = polyval(polyCoef, compute_duration(cycleSlice(idNew),
cycleSlice(1), cycleTime));
    end

    % use the mean coefficient
    basePoint = [];
    idDated = find(convexEnv ~= g_dateDef);
    for id = 1:length(idDated)
      idPoint = idDated(id);
      xPoint = compute_duration(cycleSlice(idPoint), cycleSlice(1), cycleTime);
      yPoint = convexEnv(idPoint);

      coefB = yPoint - meanCoef*xPoint;
      curPolyCoefCur = [meanCoef coefB];

      yAll = polyval(curPolyCoefCur, compute_duration(cycleSlice(idDated), cycleSlice(1),
cycleTime));

      idKo = find(convexEnv(idDated) > yAll);
      if (isempty(idKo) || ((length(idKo) == 1) && (idKo == id)))
        basePoint = [basePoint; idPoint];
      end
    end

    if (length(basePoint) > 1)
      comment = sprintf(' %d', cycleSlice(basePoint));
      fprintf('      WARNING : %d base points (cycles %s)\n', ...
        length(basePoint), comment);
      basePoint = basePoint(1);
    end

    if (isempty(basePoint))
      fprintf('      ERROR : no base point => nothing done\n');
    else
      % compute the clock drift

      idPoint = basePoint;
      xPoint = compute_duration(cycleSlice(idPoint), cycleSlice(1), cycleTime);
      yPoint = convexEnv(idPoint);

      coefB = yPoint - meanCoef*xPoint;
      polyCoefFinal = [meanCoef coefB];

      clockDrift = polyval(polyCoefFinal, compute_duration(cycleSlice, cycleSlice(1), cycleTime));
      clockDrift(find(oriDates == g_dateDef)) = g_dateDef;

      idDated = find(clockDrift ~= g_dateDef);
      totalClockDrift = clockDrift(idDated(end)) - ...
        (clockDrift(idDated(1)) + compute_duration(cycleSlice(idDated(end)),
cycleSlice(idDated(1)), cycleTime));
      yearClockDrift = totalClockDrift*365/(compute_duration(cycleSlice(idDated(end)),
cycleSlice(idDated(1)), cycleTime));
```

```
          fprintf('      Clock drift (per year): %s\n', ...
            format_time(yearClockDrift*24));

          idDated = find((maxEnv ~= g_dateDef) & (clockDrift ~= g_dateDef));
          fprintf('      Max diff: %s\n', ...
            format_time(max(abs(maxEnv(idDated) - clockDrift(idDated)))*24));

          tabArgosStopFinal(firstId:lastId) = clockDrift;
        end
      end
    end
end

% ------------------------------------------------------------------------------
% Compute the duration between 2 cycles.
%
% SYNTAX :
%  [o_duration] = compute_duration(a_tabEndCyNum, a_startCyNum, a_cycleTime)
%
% INPUT PARAMETERS :
%   a_tabEndCyNum : final cycle numbers
%   a_startCyNum  : first cycle number
%   a_cycleTime   : cycle durations
%
% OUTPUT PARAMETERS :
%   o_duration : computed durations
%
% EXAMPLES :
%
% SEE ALSO :
% AUTHORS  : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)
% ------------------------------------------------------------------------------
% RELEASES :
%   27/10/2009 - RNU - creation
% ------------------------------------------------------------------------------
function [o_duration] = compute_duration(a_tabEndCyNum, a_startCyNum, a_cycleTime)

global g_durationDef;

% default values initialization
init_valdef;

o_duration = ones(length(a_tabEndCyNum), 1)*g_durationDef;

for id = 1:length(a_tabEndCyNum)
  % number of the cycles for which the duration is wanted
  cyNum = [a_startCyNum+1:a_tabEndCyNum(id)];
  if (~isempty(cyNum))
    % duration computation
    o_duration(id) = sum(a_cycleTime(cyNum+1));
  else
    o_duration(id) = 0;
  end
end
```

o_duration = o_duration/24;

return;

4.3.3 Final improvement: taking the cycle duration anomalies into account

Some Apex floats have experienced anomalies in their cycle duration, an example is provided in next figure.

These floats have been processed by slices.

A new slice is defined for each set of cycles with a constant cycle duration. Each slice is then processed with the proposed method.

This is also the way we have processed DPF floats. We created a first slice with cycle #0 and a second one with all the other cycles (see Figure 22).

Within the 3622 APEX Argos floats of the ANDRO data set, 777 floats have been processed by slices: 717 floats only because they are DPF floats and 60 floats because they have cycle duration anomalies.

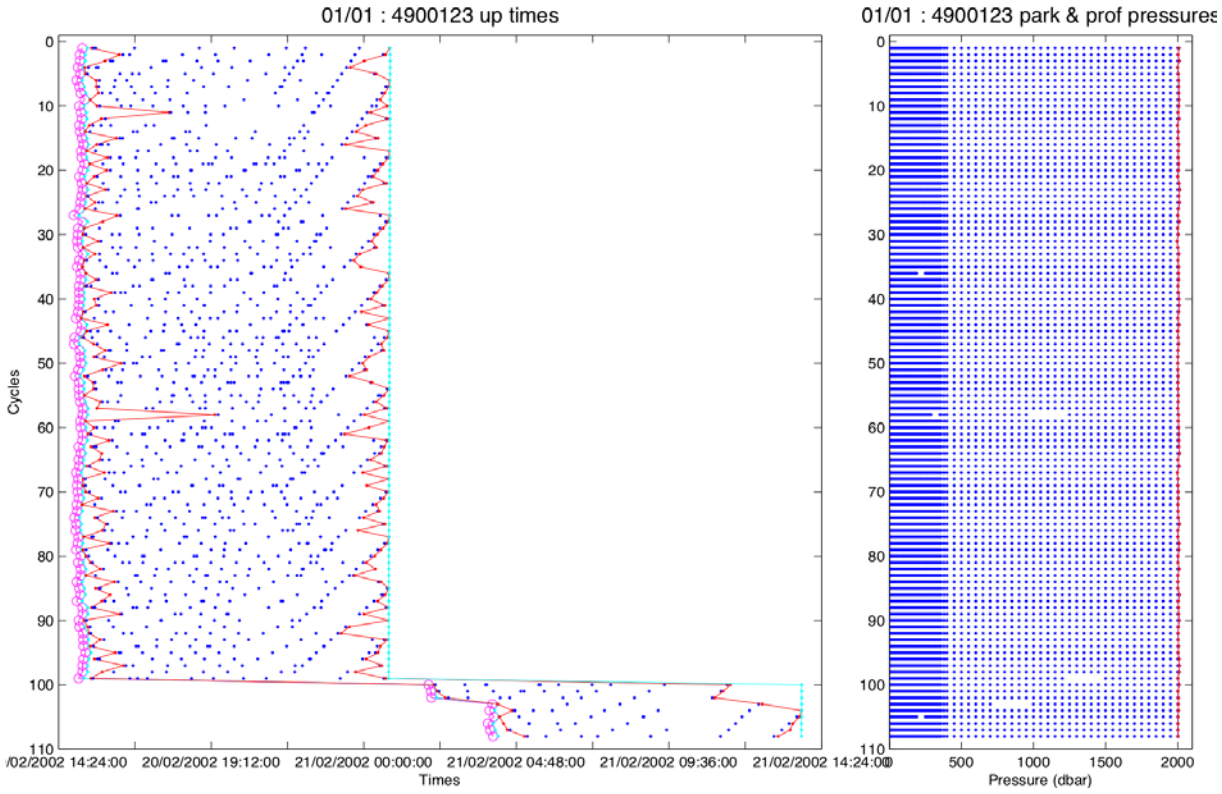


Figure 21: Example of a cycle duration anomaly (cycle #100 duration is 250 hours whereas others cycles have the expected duration i.e. 240 hours)

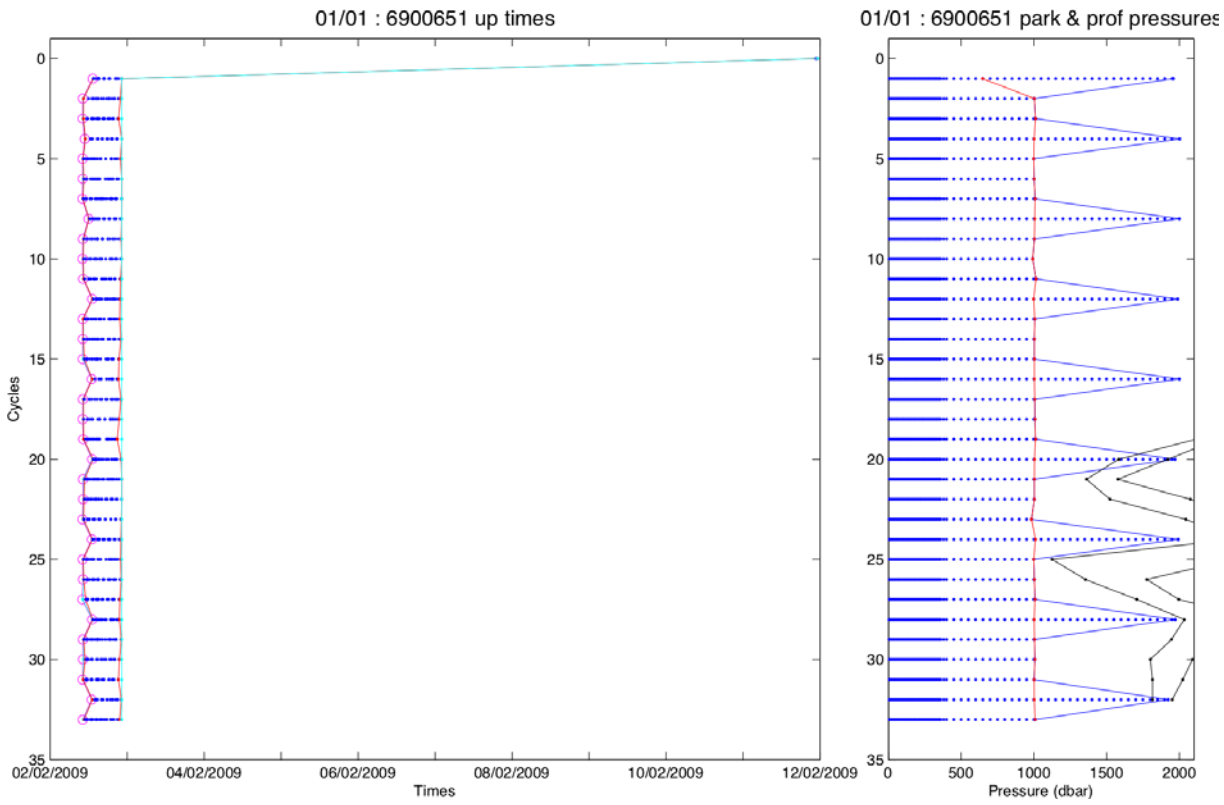


Figure 22: Example of DPF float, the first profile is a deep profile (whereas with a PnP parameter of 4, it should otherwise be a shallow profile) and the first cycle is shorter than the other ones

4.3.4 Results obtained in the ANDRO data set

For the processing of the ANDRO data set we have chosen to estimate the float clock offset only if we have more than 20 points to fit in step #4; thus only if we have received at least 33 cycles for the float.

The estimated TETs of the 3622 floats have then been visually checked and the following parameter modifications done:

- The estimation using the second algorithm has been canceled for 53 floats (i.e. we use the first algorithm results even if more than 33 cycles has been received). In most of these cases, bad results are due to too few cycles used in the estimate of the clock offset (between 21 and ~40),
- For 191 floats, the process needed additional customization (modification of the number of deleted points in step #3).

Thus we are convinced that the proposed method is robust enough to be implemented in real time (except for cycle duration anomalies which can only be detected by visual inspection).

Some Apex float versions provide the time of the end of the DOWN TIME period. We have estimated the float clock offset from these times and successfully compared it with the one obtained by the second algorithm.

Thus we are convinced that the second algorithm method is reliable for TETs estimation.

Most of the estimated clock offsets are in the [-00:10:00; +00:10:00] interval (the [-00:2:30; +00:00:30] interval for new floats) and they imply corrections of less than 80 minutes.

4.3.5 Recommended method for real time processing

To estimate the TETs of an APEX Argos float you need to know:

- Its theoretical CYCLE TIME duration(s),
- If the float is a DPF float.

If we think that clock offset can be neglected for real time processing, we recommend using only the first algorithm.

If not, we recommend using both algorithms:

- First algorithm when we have received less than 33 cycles from the float,
- Second algorithm when we have received at least 33 cycles from the float.

When using the second algorithm, if the absolute value of the estimated clock offset is greater than 00:20:00 per year, we must be sure that the float is not a DPF one; otherwise the float has probably experienced a cycle duration anomaly and the TETs should not be estimated in real time (neither by the first nor by the second algorithm).

The value 33 should be discussed.

5 ANNEX C: Computing Transmission Start Time for and APEX Argos float

The number of Argos messages needed to transmit the data collected at depth can vary between cycles (it mainly depends on profile length).

Starting when the float arrives at the surface, these *M* Argos messages are transmitted sequentially (from #1 to #*M*) and repeatedly until the end of the UP TIME period.

If a complete set of the *M* message is called a **block** of data, thus *B* blocks of *M* messages are transmitted.

Note however that the last block is not necessarily complete (because the transmission stops at the end of the UP TIME, not at the end of a block).

All Argos messages are numbered. Moreover, message #1 gives the block number.

Since all messages received by the ARGOS satellite are dated, we get the times of transmission of the messages received and their numbers.

From messages #1, we get also the block numbers to which they belong.

5.1 Teledyne Webb Research proposed method

This method is explained in the APEX user's manual and illustrated in the following figure.

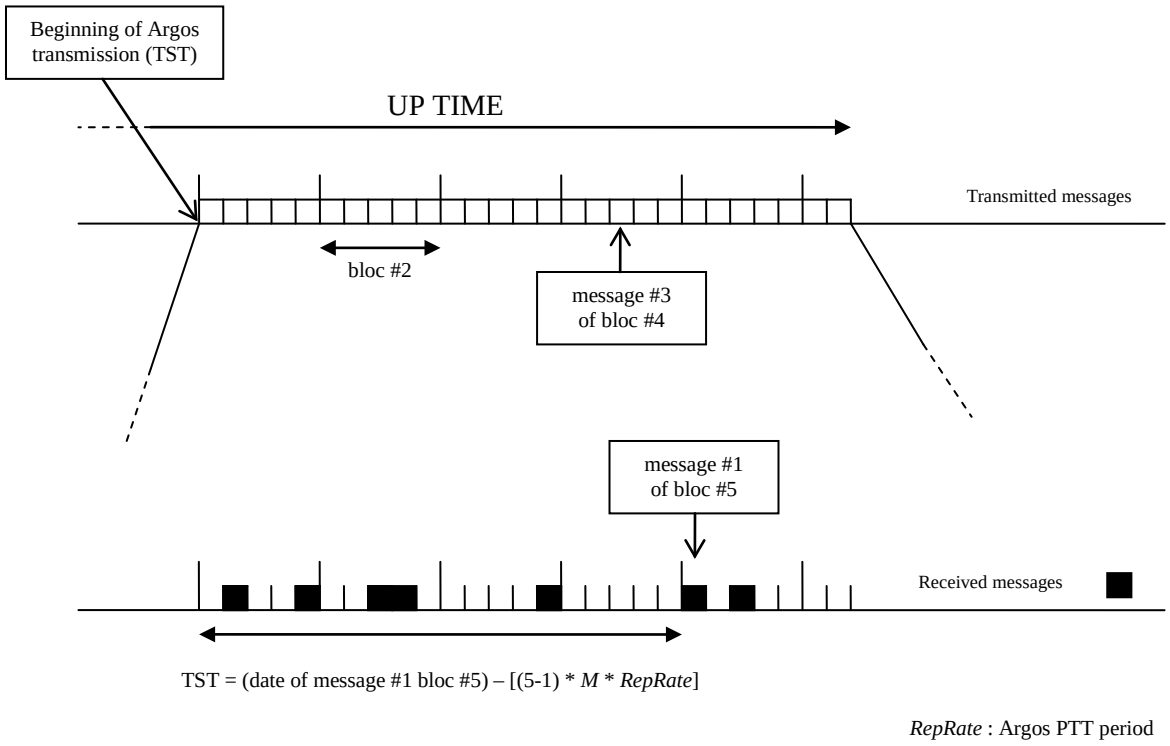


Figure 23: Teledyne Webb Research method to compute TST for an APEX Argos float

If **at least one message #1** is received, its corresponding block number (B_N) can be determined.

The number of transmitted messages since TST is then: (B_N-1)*M and

TST = (Argos time of received message #1) - [(B_N-1)M***RepRate*]**

where *RepRate* is the period of the float Argos PTT.

This method thus implies:

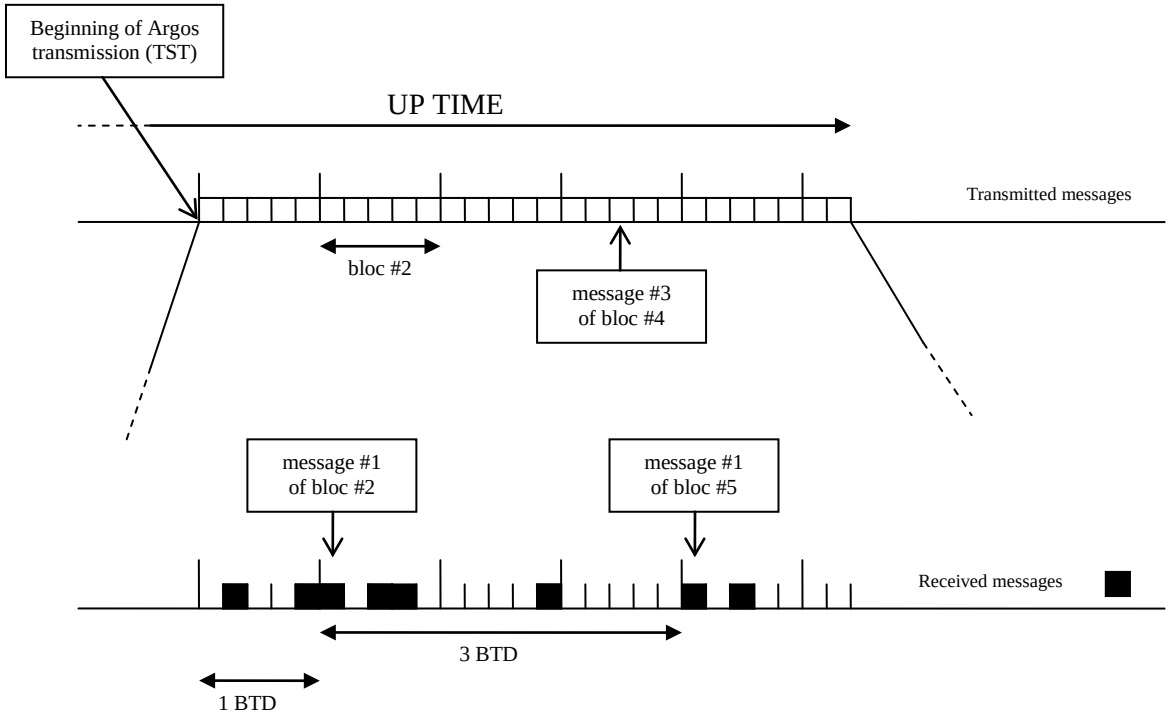
- To receive at least one message #1,
- To know the period of the Argos PTT (*RepRate*),
- To know the total number of transmitted Argos messages (*M*).

The *RepRate* parameter is a meta-data variable, thus possibly erroneous (it can however be checked from received message times).

The *M* value can sometimes be difficult to compute. Each float format must be carefully studied; *M* can be computed from the variable parts of the Argos messages (i.e. profile length but also to the number of PTS measurement sampled during the drift phase).

5.2 An improved proposed method

To get rid of the *M* value determination, a second method can be used, as illustrated in the following figure.



BTD = Block Transmission Duration = [(date of message #1 bloc #5) – (date of message #1 bloc #2)]/(5-2)

Figure 24: An improved method to compute TST for an APEX Argos float

If **at least two messages #1** (dated *T1* and *T2* respectively) **belonging to two different blocks** (B_{N1} and B_{N2} respectively) are received, the transmission duration of a block can be determined:

Transmission Duration of one Block =*BTD* = (*T2* - *T1*)/(B_{N2} - B_{N1})

But ***BTD* = *M***RepRate***, then

TST = *T1* - [(B_{N1}-1)BTD*] or TST = *T2* - [(B_{N2}-1)**BTD*]**

The following strategy is then suggested:

1. If at least two messages belonging to two different blocks are received, use the improved method.
   In this case all (in fact at most 100) values of TST are computed (from all combinations of *Ti* and *Tj* messages #1 received) and the most redundant result is chosen for TST.
   At the beginning of our work, we started to compute all the values of TST but in case of shallow profiles (or no profile at all when the float stays at the surface) i.e. when the M value is only 1 or 2 and then B value is very important, the method doesn't work (the histogram of TST computed values has many redundant values and the most redundant one can be inconsistent). We have not understood why (it can be an interesting question for TWR) but we think this is due to inconsistencies in the transmitted data, we thus decided to compute "only" 100 values of the TST and to choose the most redundant value.
2. If only one message #1 is received, the TWR method is used with the assumption (sometimes erroneous) that *M* is equal to the maximal number of the received Argos messages (i.e. that the last Argos message has been received).

6 ANNEX D: Apex float vertical velocities

6.1 APEX float descending velocity

To estimate the descending APEX Argos velocity, 463 APEX floats which provide pressure marks hourly sampled during the descent to PARKING depth were analyzed.

At most 7 pressure marks are transmitted by the floats. Thus, at most 6 averaged hourly descent rates were computed (remember that the first pressure mark is sampled at the end of the piston retraction, thus it is not dated and useless for this estimation), and a global averaged descent rate for the 6 first hours of the descent.

The mean descent rate depends on the float PARKING pressure; the averages for 5 parking depths (250 dbar, 500 dbar, 1000 dbar, 1500 dbar and 2000 dbar) were computed. The RPP has been used to associate a descent rate to the corresponding depth.

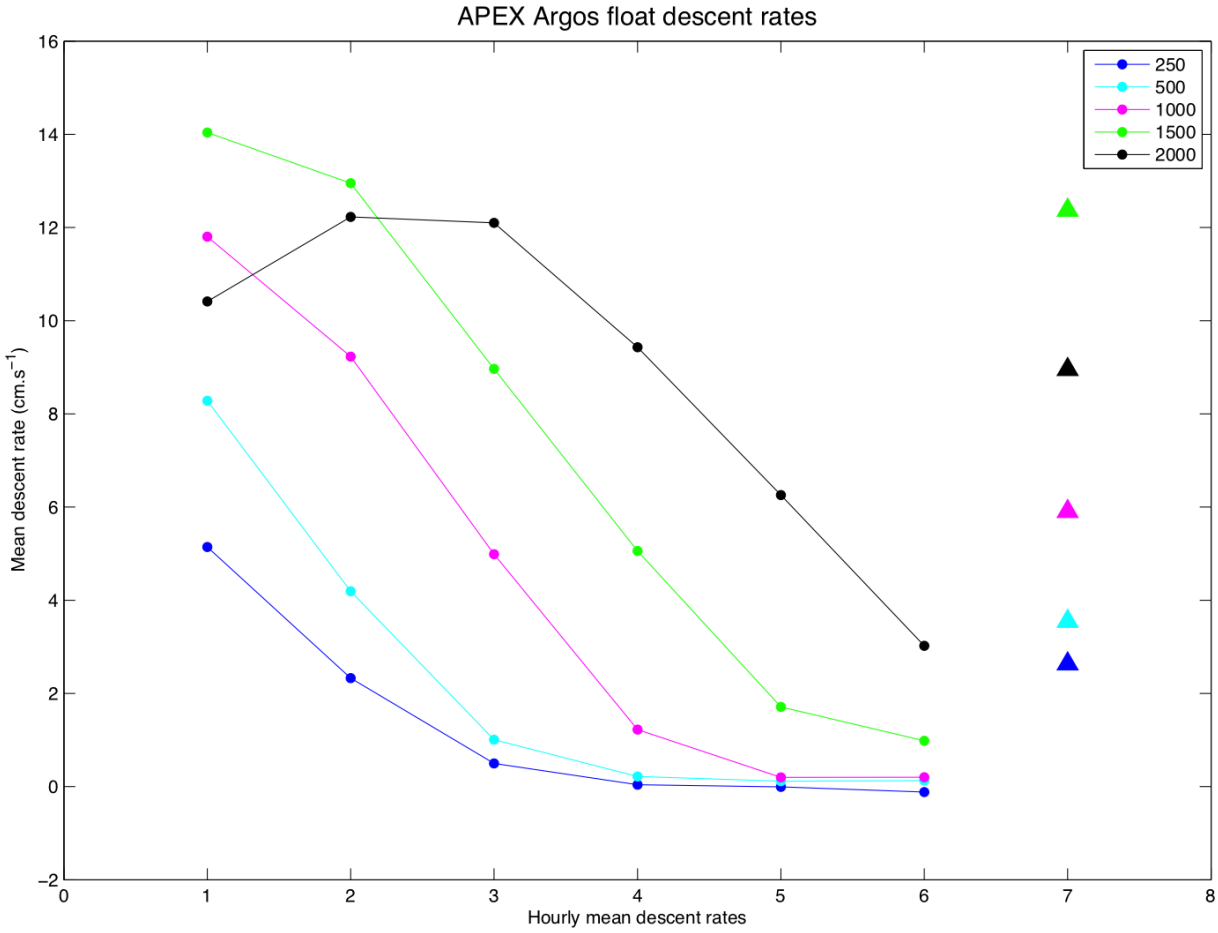


Figure 25: Mean hourly descent rates of APEX Argos floats. For each PARKING pressure the 6 mean hourly descent rates (dots) and the global mean descent rate of the 6 first hours of the descent (triangle) are shown.

The following table gives the number of samples used to compute the averages.

	1st hour	2nd hour	3rd hour	4th hour	5th hour	6th hour	Global
250 dbar	373	333	250	208	155	86	1 405
500 dbar	580	549	526	471	345	197	2 668
1000 dbar	15 010	14 852	14 762	14 646	8 762	3 827	71 859
1500 dbar	389	144	141	119	91	52	936
2000 dbar	2 114	2 105	2 091	2 059	2 040	1 981	12 390

Table 2: Number of samples used to compute mean descent rates

The global mean descent rate values are provided in the following table.

PARKING depth	250 dbar	500 dbar	1000 dbar	1500 dbar	2000 dbar
Mean descent rate ($cm*s^{-1}$)	2.64	3.55	5.91	12.37	8.95
Associated standard deviation	1.87	2.54	1.23	3.02	1.25

Table 3: Global mean descent rate values

6.2 APEX float ascending velocity

To estimate the ascending APEX Argos velocity, the two following data samples were used:

- The 298 APEX floats which provide the AST_{float} and TST_{float} (thus AET_{float}) (see §3.2.2.1.7.2 and 3.2.2.1.9.2),
- The 1497 APEX floats for which the following are available:
  - The AST, estimated as TET - UP TIME for cycles where PARKING and PROFILE pressure are equal (see §3.2.2.1.7.1)
  - The AET, computed from TST obtained with the method explained in Annex C (see §6.2)

The data of the first set are more reliable because they are measured and transmitted by the floats (whereas the data of the second set come from estimations).

To compute a mean ascent rate, a reliable deepest profile pressure value is needed. For that, it is important that the Argos message of the first (deepest) profile bin PTS measurement has been received. This information can be provided by the APEX Argos decoders.

As this information was not stored in the DEP data set, only profiles for which the following is true could be used:

| deepest bin pressure - PROFILE pressure | < 150 dbar.

(these data are in green in the following figures).

If | deepest bin pressure - PROFILE pressure | > 150 dbar, the cycle can be flagged "grounded" (blue stars) or not (red stars).

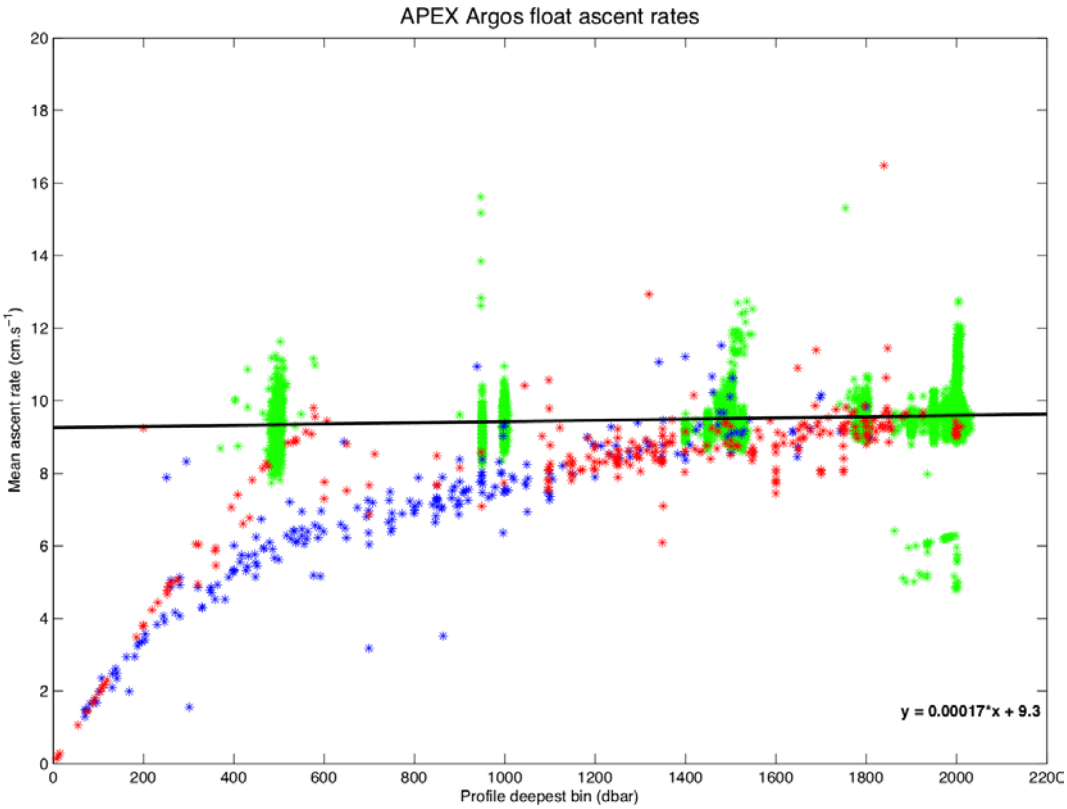


Figure 26: Ascent rates computed from measured data

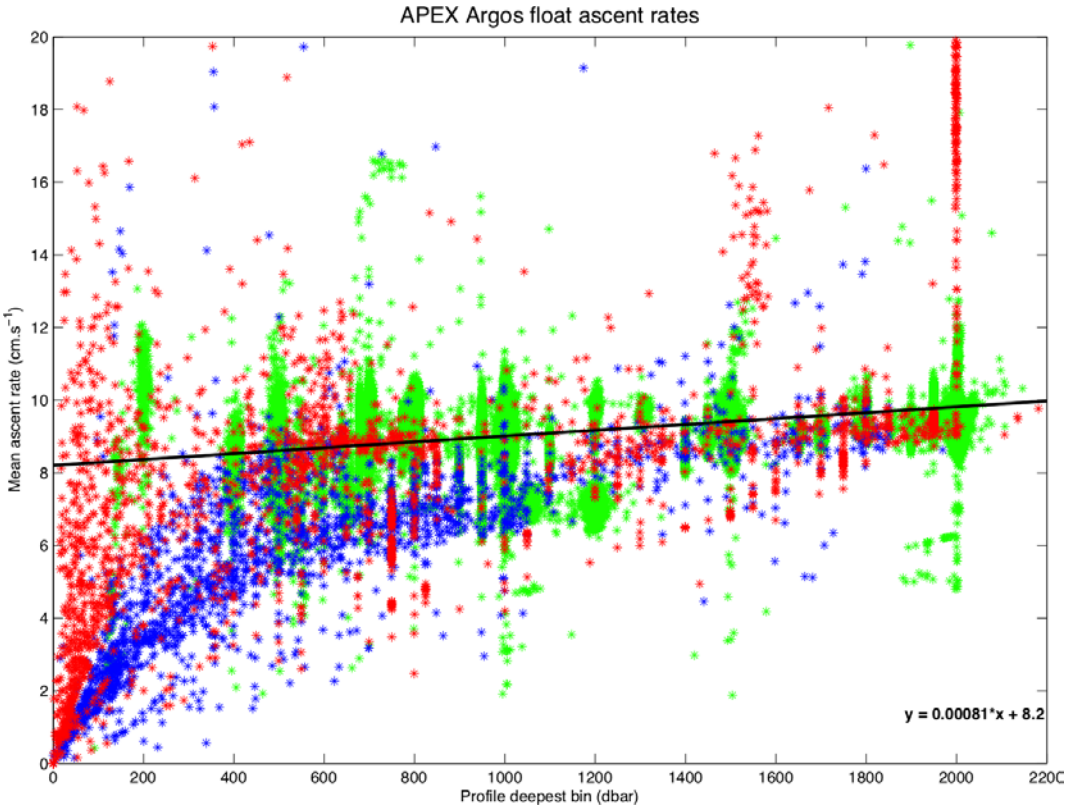


Figure 27: Ascent rates computed from estimated data

In the first figure, the green stars can be linearly fitted by the Y = 0.00017*X + 9.3 function.

In the second figure, the green stars can be linearly fitted by the Y = 0.00081*X + 8.3 function.

Thus the mean ascent rate is around 9.5 cm/s in the first case and between 9.0 and 10 cm/s in the second case.

Therefore, it is recommended to use **a mean ascent rate of 9.5 cm/s for APEX Argos floats**.

7 ANNEX E: Input parameters

Most of the specifications given in this document need input parameters. These parameters are part of the values used to program the float mission.

Unfortunately these values can be difficult to collect; some of them are transmitted by the instrument (in APEX float test message or PROVOR/APEX Iridium technical message) but the others can only be found in float operator notes.

It is thus important to ask each float PI to collect the programmed float parameters and to send them to the concerned DAC.

Some input parameters, gathered in the framework of the ANDRO project, are joined in electronic form to this document.

In the Excel file *CorrectedMetadata.xlsx* you can find, for the 5967 ANDRO floats, the corrected meta-data values for:

- Float PTT,
- All existing float missions:
  A mission is defined by:
  - Its repetition rate,
  - Its duration, given by:
    - The UP_TIME and DOWN_TIME period for APEX floats,
    - The CYLE_TIME for other floats.
  - Its parking pressure,
  - Its profile pressure,
- The float launch time and position,
- The startup time of the float.

In the Excel file *_provor_floats_information.xls*, you can find the corrected meta-data values used to decode the PROVOR Argos floats.

You can find in particular:

- The DELAI parameter values (maximum amount of time given to the float for diving from PARKING to PROFILE depth),
- The reference day ("day of the first descent") used to decode the transmitted times.

In the Excel file *DPDP_values.xlsx*, you can find the decoded values of the "Deep profile descent period" transmitted in the test message by some version of APEX floats.

8 ANNEX F: Measurement code table

8.1 General Measurement Code Table Key

Measurement code type	Definition
Any code evenly divisible by 100 (e.g. 100, 200, 300, etc.)	**Primary Measurement Codes (MC)**. Each marks a mandatory-to-fill cycle timing variable. These are very important for determining trajectory estimates. All are found in both the N_MEASUREMENT and N_CYCLE data arrays.
Any code evenly divisible by 50 but not evenly divisible by 100 (e.g. 150, 250, 450, et)	**Secondary Measurement Codes (MC)**. Each marks a suggested-to-fill cycle timing variable. Secondary MC are not always applicable to all floats, but are very useful in determining trajectory estimates.
Any code that falls in between any Primary or Secondary Measurement Code (span of 50 values). These codes describe data that are important cycle timing information but are not as important as the primary or secondary timing variables. The value span is subdivided into two halves. Measurement codes in this section will be described relative to the values of the Primary and Secondary codes.	**Relative Generic Codes**. Values spanning from MC minus 24 to MC minus 1: Measurement codes that have lower value and within 24 of a Primary or Secondary Measurement Code. These code definitions are phrased generally, so can be attached to data from many different floats. These code values (MC minus 24 to MC minus 1) are assigned when a float records a measurement while transitioning TOWARDS the MC. The definitions of the MC from MC minus 24 to MC minus 1 are repeated for all Primary and Secondary MC. An example, most floats record pressure/temperature/salinity during drift. The float is transitioning towards PET (MC=300) during this period. Thus the pressure/temperature/salinity measurements will have an MC between MC minus 24 and MC minus 1 where MC=300 (thus between MC=276 and MC=299). Which value is chosen is determined by the measurement itself (See table below). **Relative Specific Codes**. Values spanning from MC plus 1 to MC plus 25: These are specific measurements that are generally NOT recorded by multiple float types. They are believed to be valuable enough in trajectory estimation that they are defined here, and not within the generically defined MC minus 24 to MC minus 1 span. MC codes in this span will be specific to the MC code, and will NOT be repeated for other Primary and Secondary MCs. An example, APEX floats report the “Down-time end date”, which is important in determining the start of ascent (MC=500). The MC for “Down-time end date” is recorded with MC plus 1 (MC=501).

8.2 Relative Generic Code Table Key (from MC minus 24 to MC minus 1)

This table pertains to any measurement code that has lower value and within 24 of a Primary or Secondary Measurement Code (see below). These definitions apply relative to every Primary and Secondary code. For example, AST (time of ascent start, MC=500) and AET (time of ascent end, MC=600) are both Primary MCs. There exists a measurement code MC minus 4 for both AST and AET which is assigned to any averaged measurement that is taken while transitioning towards the MC. If an averaged measurement is recorded while transitioning towards AST, the correct MC=496. If an averaged measurement is recorded while transitioning towards AET, the correct MC=596.

Relative Measurement code	Meaning
MC minus 1	Any single measurement transitioning towards MC (see MC-10 for a 'series' of measurements)
MC minus 2	Maximum value while float is transitioning towards an MC (e.g. pressure)
MC minus 3	Minimum value while float is transitioning towards an MC (e.g. pressure)
MC minus 4	Any averaged measurements made during transition to MC
MC minus 5	Median value while float is transitioning towards an MC
MC minus 6	Standard deviation of measurements taken during transition towards an MC
MC minus 7 to MC minus 9	currently unassigned
MC minus 10	Any “series” of measurements recorded while transitioning towards MC. (e.g. Provor 'spy' measurements, SOLOII pressure-time pairs, etc).
MC minus 11	Active adjustment to buoyancy made at this time
MC minus 12	Any supporting measurements for the maximum value (MC minus 2)

MC minus 13	Any supporting measurements for the minimum value (MC minus 3)
MC minus 14	Any supporting measurements for the averaged value (MC minus 4)
MC minus 15	Any supporting measurements for the median value (MC minus 5)
MC minus 16 to MC minus 24	currently unassigned

8.3 Measurement Code Table

Measure-ment code	Variable	Meaning	Transmitted by listed float type. Value can be estimated in other floats
0		Launch time and location of the float	All float types
76-99	see above table	Any measurement recorded during transition towards DST	
100	**DST**	**All measurements made when float leaves the surface, beginning descent.** Time (JULD_DESCENT_START)	Time: PROVOR, ARVOR, SOLO-II,WHOI SOLOIR, NEMO, NEMOIR, APEX APF9, APEXIR APF9, Deep NINJA
101	DM Traj file only	This MC is used in the DM Traj file when new cycles have been recovered during DM operations (the TECH file, where surface pressure measurements usually belong, is not updated during TRAJ DM). The PRES variable should contain the measurement provided by the float (after 5dbar subtraction when needed). For APEX floats, this measurement is used to compute (see procedure 3.2.1 of Argo QC manual) a pressure offset applied to all pressure measurements. This offset should be stored in the PRES_ADJUSTED variable. No information should be stored with this MC in Real Time.	
102-125	unassigned	Reserved for specific timing events around DST.	
126-149	see above table	Any measurement recorded during transition towards FST	
150	**FST**	**All measurements made at time when a float first becomes water-neutral.** Time (JULD_FIRST_STABILIZATION)	PROVOR, ARVOR
151-175	unassigned	Reserved for specific timing events around FST.	
176-199	see above table	Any measurement recorded during transition towards DET	
200	**DET**	**All measurements made at time when float first approaches within 3% of the eventual drift pressure.** Float may be transitioning from the surface or from a deep profile. This variable is based on measured or estimated pressure only In the case of a float that overshoots the drift pressure on descent, DET is the time of the overshoot. Time (JULD_DESCENT_END)	Time: PROVOR, ARVOR, SOLO-II, NEMO, NEMOIR, DeepNINJA
201-202 & 204-225	unassigned	Reserved for specific timing events around DET.	
203		Deepest bin reached during descending profile	
226-249	see above table	Any measurement recorded during transition towards PST	
250	**PST**	**All measurements made at time when float transitions to its Park or Drift mission**. This variable is based on float logic based on a descent timer (i.e. SOLO), or be based on measurements of pressure (i.e. Provor).	APEX non APF9, APEX APF9, APEX APF9i, SIO SOLO, SOLO-II, NEMO, NEMOIR CTD: WHOI SOLO

		Time(JULD_PARK_START)	NINJA
251-275	unassigned	Reserved for specific timing events around PST.	
276-299	see above table	Any measurement recorded during transition towards PET	
300	**PET**	**All measurements made at time when float exits from its Park or Drift mission.** It may next rise to the surface (AST) or sink to profile depth Time (JULD_PARK_END)	Time: PROVOR (excluding PROVOR MT), ARVOR, SOLO-II, NEMO, NEMOIR, POPS CTD: WHOI SOLO
301		Representative Park <PARAM> found either from measurements taken during drift or from metafile information	
302-325	unassigned	Reserved for specific timing events around PET.	
376-399	see above table	Any measurement recorded during transition towards DDET	
400	**DDET**	**All measurements made at time when float first approaches within 3% of the eventual deep drift/profile pressure**. This variable is based on pressure only and can be measured or estimated. Time (JULD_DEEP_DESCENT_END)	Time: APEX APF9a or APF9t, APF9i, PROVOR CTS3, ARVOR, SOLO-II, POPSm , DeepNINJA
401-425	unassigned	Reserved for specific timing events around DDET.	
426-449	see above table	Any measurement recorded during transition towards DPST	
450	**DPST**	**All measurements made at time when float transitions to a deep park drift mission.** This variable is only defined if the float enters a deep drift phase (i.e. DPST not defined in cases of constant deep pressure due to bottom hits, or buoyancy issues).	
451-475	unassigned	Reserved for specific timing events around DPST.	
476-499	see above table	Any measurement recorded during transition towards AST	
500	**AST**	**All measurements made at the start of the float's ascent to the surface** Time (JULD_ASCENT_START)	Time: APEX APF9, PROVOR, ARVOR, SOLO-II, NEMO, NEMOIR, POPS, DeepNINJA
501		Down-time end time: end date of the down-time parameter reported by APEX floats	APEX
502		Ascent start time directly transmitted by APEX floats	APEX
503		Deepest bin reached during ascending profile	
504-525	unassigned	Reserved for specific timing events around AST.	
526-549	see above table	Any measurement recorded during transition towards DAST	
550	**DAST**	**All measurements made at the start of the float's ascent from profile pressure to drift pressure**. Used for floats that profile on descent and then move back up to drift pressure. Time (JULD_DEEP_ASCENT_START)	Time: Deep SOLO-II
551-575	unassigned	Reserved for specific timing events around DAST.	
576-599	see above table	Any measurement recorded during transition towards AET	
600	**AET**	**All measurements made at the end of ascent.** Time (JULD_ASCENT_END)	PROVOR, ARVOR, SOLO-II, NEMO, NEMOIR, POPS, DeepNINJA
601-625	unassigned	Reserved for specific timing events around AET.	
676-699	see above table	Any measurement recorded during transition towards TST	

700	**TST**	**Time and location of the start of transmission for the float.** Time (JULD_TRANSMISSION_START)	APEX APF9, APEXIR APF9, PROVOR, ARVOR, SOLO-II, NEMO, NEMOIR, POPS, DeepNINJA
701		Transmission start time directly transmitted by APEX float	APEX
702	**FMT**	**Earliest time of all messages received by telecommunications system** – may or may not have a location fix. Time (JULD_FIRST_MESSAGE)	All floats
703		Surface times and locations (if available) during surface drift. Should be listed in chronological order.	All floats
704	**LMT**	**Latest time of all messages received by telecommunications system** – may or may not have a location fix. Time (JULD_LAST_MESSAGE)	All floats
705-725	unassigned	Reserved for specific timing events around TST	
776-799	see above table	Any measurement recorded during transition towards TET	
800	**TET**	**Time and location of the end of transmission for the float.** Time (JULD_TRANSMISSION_END)	PROVOR, ARVOR, SOLO-II, APEXIR APF9, DeepNINJA
801-825	unassigned	Reserved for specific timing events around TET	
901		Grounded flag Configuration phase	
902		Last time before float recovery. For floats that have been recovered, it is important to know when this occurred. This time in the JULD array will be the last time before the float was recovered. Determined by inspection of data	
903		Pressure offset used to correct APEX pressure measurements	APEX
1100	In Air Measurements	Indicates the float is taking measurements in air. All relative measurement codes apply	O2 sensors
1076-1099	See table above	Reserved for specific timing events in air	

9 ANNEX G: Implementation of the JAMSTEC trajectory quality control method

The JAMSTEC trajectory quality control method is described in Nakamura et al (2008), "Quality control method of Argo float position data", JAMSTEC Report of Research and Development, Vol. 7, 11-18 (http://www.godac.jamstec.go.jp/catalog/data/doc_catalog/media/JAM_RandD07_02.pdf).

This method checks the surface trajectory of an Argos float by considering the speeds induced by the successive Argos fixes. The test can flag the surface position as '3' or '4'.

In the following, we propose a detailed description of the algorithm to implement.

9.1 Inputs

The inputs of the algorithm are:

- The surface trajectory to be checked (N Argos location dates, latitudes, longitudes and classes),
- The last good (flagged as '1') surface location of the (already checked) previous (received) cycle.

9.2 Algorithm

Assuming that the location dates have not been flagged as bad by the test #2 "Impossible date test", we first chronologically sort the surface positions.

The whole surface trajectory is used to initialize the (checked) current trajectory.

The current trajectory is processed in an infinite loop in which the following steps are performed:

9.2.1 Step 1

The subsurface drift speed is computed between the last good surface position of the previous cycle and the first position of the current trajectory.

If this speed is greater than 3 m/s, the first position of the current trajectory is flagged as '4', this position is then excluded from the current trajectory and a new iteration of the infinite loop starts.

9.2.2 Step 2

Speeds are computed for the second position to the last position of the current trajectory. Each speed is computed between position #i-1 and position #i and affected to position #i.

In case of duplicated positions (i.e. if position #i-1 and position #i have the same latitude, longitude and date): the position #i is flagged as '4', it is then excluded from the current trajectory and a new iteration of the infinite loop starts.

In case of an erroneous cycle number of the position #i (i.e. if the times difference between position #i and position #i-1 is greater than one day): the position #i is flagged as '4', it is then excluded from the current trajectory and a new iteration of the infinite loop starts.

9.2.3 Step 3

The position #iMax is found as the position with the maximum speed.

If this maximum speed is greater than 3 m/s, the position #iMax is 'questionable' and the **speed test** (see §11.3) is performed on it over the current trajectory.

The **speed test** should lead to define position #iMax or/and position #iMax-1 as 'abnormal'.

9.2.4 Step 4

If the **distance test** (see §11.4) between position #iMax and position #iMax-1 is verified, the 'abnormal' position(s) is (are) flagged as '3'.

The 'abnormal' position(s) is(are) then deleted from the current trajectory (even when the distance test is not verified) and a new iteration of the infinite loop starts.

The infinite loop ends when no 'abnormal' position has been detected or when the current trajectory has less than 2 positions.

9.3 Speed test

The speed test is performed on a 'questionable' position over a given trajectory.

The 'questionable' position (called B in the following) can be all but the first position of the trajectory. The position which precedes B on the trajectory is called A in the following.

9.3.1 Case of different Argos classes

If positions A and B have different Argos classes, the position with the less accurate Argos class is defined as 'abnormal' by the speed test.

Remember that the accuracy of the Argos location classes is the following:

more accurate <= 3, 2, 1, 0, A, B, Z => less accurate

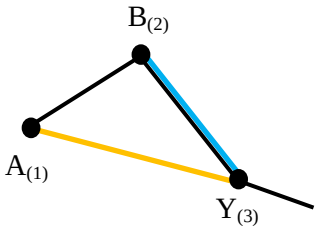
9.3.2 Case of identical Argos classes

If positions A and B have the same Argos classes:

- If the trajectory only comprises the two positions A and B, both positions are defined as 'abnormal' by the speed test,
- Otherwise the speed test depends on the position of the location B on the trajectory, 3 cases are possible.

Case 1: If B is the second position of the trajectory

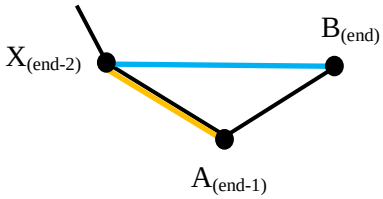
In this case: A is the first position, B the second one and there is a position Y following the position B on the trajectory.



Speeds on the segments A-Y (orange) and B-Y (blue) are computed: if $speed_{A-Y}$ is greater than $speed_{B-Y}$, the position A is defined as 'abnormal' by the speed test otherwise B is defined as 'abnormal' by the speed test.

Case 2: If B is the last position of the trajectory

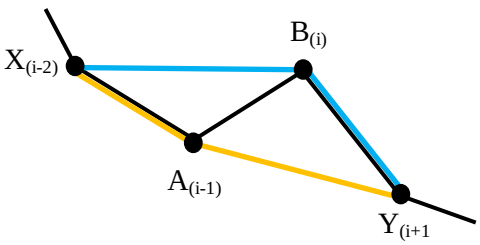
In this case: A is the last but one position, B is the last position and there is a position X preceding the position A on the trajectory.



Speeds on the segments X-A (orange) and X-B (blue) are computed: if $speed_{X-A}$ is greater than $speed_{X-B}$, the position A is defined as 'abnormal' by the speed test otherwise B is defined as 'abnormal' by the speed test.

Case 3: we are not in case 1 or 2

In this case: there is a position X preceding the position A on the trajectory and a position Y following the position B on the trajectory.



Speeds on the segments X-A-Y (orange trajectory) and X-B-Y (blue trajectory) are computed. If $speed_{X\text{-}A\text{-}Y}$ is greater than $speed_{X\text{-}B\text{-}Y}$, the position A is defined as 'abnormal' by the speed test otherwise B is defined as 'abnormal' by the speed test.

9.4 Distance test

The distance test is performed on two Argos locations A and B.

The distance test is verified if the distance between locations A and B is greater or equal to

$1.0 \times \sqrt{Er_A{}^2 + Er_B{}^2}$ where Er_A and Er_B are the radii of position error for locations A and B respectively.

These position errors, deduced from the position classes, are 150 m, 350 m and 1000 m for Argos class 3, 2 and 1 respectively. Moreover we have associated a position error of 1500 m, 1501 m, 1502 m and 1503 m for Argos classes 0, A, B and Z respectively.

9.5 Distance computation

As far as distance and speed are concerned in this trajectory QC method, we must specify an algorithm to compute distance between positions of the surface trajectory. This algorithm must be common to all the DACs so that the trajectory QC results will not depend on DAC's distance computation method.

We propose to use the distance algorithm from the *Laboratoire de Physiques des Océans* (*LPO*) at IFREMER.

This algorithm computes distance between points on the earth using the WGS 1984 ellipsoid, its Matlab implementation and some test points are provided below.

9.5.1 Matlab implementation of the LPO distance algorithm

```
function [range, A12, A21] = distance_lpo(lat, long)
%
% Computes distance and bearing between points on the earth using WGS 1984
% ellipsoid
%
% [range, A12, A21] = distance_lpo(lat, long) computes the ranges RANGE between
% points specified in the LAT and LONG vectors (decimal degrees with positive
% indicating north/east). Forward and reverse bearings (degrees) are returned
% in AF, AR.
%
% Ellipsoid formulas are recommended for distance d<2000 km,
% but can be used for longer distances.
%
% GIVEN THE LATITUDES AND LONGITUDES (IN DEG.) IT ASSUMES THE IAU SPHERO
% DEFINED IN THE NOTES ON PAGE 523 OF THE EXPLANATORY SUPPLEMENT TO THE
% AMERICAN EPHEMERIS.
%
% THIS PROGRAM COMPUTES THE DISTANCE ALONG THE NORMAL
% SECTION (IN M.) OF A SPECIFIED REFERENCE SPHEROID GIVEN
% THE GEODETIC LATITUDES AND LONGITUDES OF THE END POINTS
% *** IN DECIMAL DEGREES ***
%
% IT USES ROBBIN'S FORMULA, AS GIVEN BY BOMFORD, GEODESY,
% FOURTH EDITION, P. 122.  CORRECT TO ONE PART IN 10**8
% AT 1600 KM.  ERRORS OF 20 M AT 5000 KM.
%
% CHECK:  SMITHSONIAN METEOROLOGICAL TABLES, PP. 483 AND 484,
% GIVES LENGTHS OF ONE DEGREE OF LATITUDE AND LONGITUDE
% AS A FUNCTION OF LATITUDE. (SO DOES THE EPHEMERIS ABOVE)
%
% PETER WORCESTER, AS TOLD TO BRUCE CORNUELLE...1983 MAY 27
```

```
%

% On 09/11/1988, Peter Worcester gave me the constants for the
% WGS84 spheroid, and he gave A (semi-major axis), F = (A-B)/A
% (flattening) (where B is the semi-minor axis), and E is the
% eccentricity, E = ( (A**2 - B**2)**.5 )/ A
% the numbers from peter are: A=6378137.; 1/F = 298.257223563
% E = 0.081819191
A = 6378137.;
E = 0.081819191;
B = sqrt(A.^2 - (A*E).^2);
EPS = E*E/(1.-E*E);

NN = max(size(lat));
if (NN ~= max(size(long))),
   error('dist: Lat, Long vectors of different sizes!');
end

if (NN == size(lat))
   rowvec = 0;  % it is easier if things are column vectors,
else
   rowvec = 1; % but we have to fix things before returning!
end;

% convert to radians
lat = lat(:)*pi/180;
long = long(:)*pi/180;

% fixes some nasty 0/0 cases in the geodesics stuff
lat(lat == 0) = eps*ones(sum(lat == 0), 1);

% endpoints of each segment
PHI1 = lat(1:NN-1);
XLAM1 = long(1:NN-1);
PHI2 = lat(2:NN);
XLAM2 = long(2:NN);

% wiggle lines of constant lat to prevent numerical problems.
if (any(PHI1 == PHI2))
   for ii = 1:NN-1
      if (PHI1(ii) == PHI2(ii))
         PHI2(ii) = PHI2(ii) + 1e-14;
      end
   end
end
% wiggle lines of constant long to prevent numerical problems.
if (any(XLAM1 == XLAM2))
   for ii = 1:NN-1
      if (XLAM1(ii) == XLAM2(ii))
         XLAM2(ii) = XLAM2(ii) + 1e-14;
      end
   end
end

% COMPUTE THE RADIUS OF CURVATURE IN THE PRIME VERTICAL FOR EACH POINT
xnu = A./sqrt(1.0-(E*sin(lat)).^2);
xnu1 = xnu(1:NN-1);
xnu2 = xnu(2:NN);

% COMPUTE THE AZIMUTHS.
% A12 (A21) IS THE AZIMUTH AT POINT 1 (2) OF THE NORMAL SECTION CONTAINING THE POINT 2 (1)
TPSI2 = (1.-E*E)*tan(PHI2) + E*E*xnu1.*sin(PHI1)./(xnu2.*cos(PHI2));
PSI2 = atan(TPSI2);

% SOME FORM OF ANGLE DIFFERENCE COMPUTED HERE??
DPHI2 = PHI2-PSI2;
DLAM = XLAM2-XLAM1;
CTA12 = (cos(PHI1).*TPSI2 - sin(PHI1).*cos(DLAM))./sin(DLAM);
A12 = atan((1.)./CTA12);
CTA21P = (sin(PSI2).*cos(DLAM) - cos(PSI2).*tan(PHI1))./sin(DLAM);
A21P = atan((1.)./CTA21P);

% GET THE QUADRANT RIGHT
DLAM2 = (abs(DLAM)<pi).*DLAM + (DLAM>=pi).*(-2*pi+DLAM) + (DLAM<=-pi).*(2*pi+DLAM);
A12 = A12 + (A12<-pi)*2*pi-(A12>=pi)*2*pi;
A12 = A12 + pi*sign(-A12).*(sign(A12) ~= sign(DLAM2));
A21P = A21P + (A21P<-pi)*2*pi - (A21P>=pi)*2*pi;
```

```
A21P = A21P + pi*sign(-A21P).*(sign(A21P) ~= sign(-DLAM2));
% A12*180/pi
% A21P*180/pi

SSIG = sin(DLAM).*cos(PSI2)./sin(A12);

% At this point we are OK if the angle < 90 but otherwise
% we get the wrong branch of asin!
% This fudge will correct every case on a sphere, and *almost*
% every case on an ellipsoid (wrong handling will be when
% angle is almost exactly 90 degrees)
dd2 = [cos(long).*cos(lat) sin(long).*cos(lat) sin(lat)];
dd2 = sum((diff(dd2).*diff(dd2))')';
if (any(abs(dd2-2) < 2*((B-A)/A))^2),
   disp('dist: Warning...point(s) too close to 90 degrees apart');
end
bigbrnch = dd2>2;

SIG = asin(SSIG).*(bigbrnch==0) + (pi-asin(SSIG)).*bigbrnch;
A21 = A21P - DPHI2.*sin(A21P).*tan(SIG/2.0);

% COMPUTE RANGE
G2 = EPS*(sin(PHI1)).^2;
G = sqrt(G2);
H2 = EPS*(cos(PHI1).*cos(A12)).^2;
H = sqrt(H2);
TERM1 = -SIG.*SIG.*H2.*(1.0-H2)/6.0;
TERM2 = (SIG.^3).*G.*H.*(1.0-2.0*H2)/8.0;
TERM3 = (SIG.^4).*(H2.*(4.0-7.0*H2)-3.0*G2.*(1.0-7.0*H2))/120.0;
TERM4 = -(SIG.^5).*G.*H/48.0;

range = xnu1.*SIG.*(1.0 + TERM1 + TERM2 + TERM3 + TERM4);

% CONVERT TO DECIMAL DEGREES
A12 = A12*180/pi;
A21 = A21*180/pi;
if (rowvec),
   range = range';
   A12 = A12';
   A21 = A21';
end
```

9.5.2 Test points

The following table provides results of calculation distances from the LPO distance algorithm.

Test #	Longitude point #1	Latitude point #1	Longitude point #2	Latitude point #2	Distance (m)
1	59.137	81.450	132.862	-71.971	17452769.38
2	245.057	-75.309	331.764	-77.086	2110391.35
3	185.622	87.327	183.692	-17.999	11689986.02
4	182.640	20.009	49.196	5.048	14227739.39
5	150.579	41.603	208.973	39.188	4868529.07
6	0.000	0.000	332.341	19.629	3717195.47
7	356.228	79.610	254.896	-47.763	15364005.55
8	199.871	88.917	70.224	52.035	4312751.18
9	287.193	-35.107	200.803	52.926	12831368.01
10	102.486	-83.242	312.077	75.131	18753227.55
11	69.797	88.120	207.543	18.708	8087967.56
12	93.492	-16.942	304.265	20.978	16765984.94
13	199.115	-39.885	182.679	60.574	11263499.39
14	303.234	77.720	332.681	-0.149	8830419.21
15	152.391	-4.042	179.072	-21.859	3490115.84

16	38.772	-90.000	252.147	9.952	11097348.67
17	170.518	85.414	311.396	-28.009	13474193.18
18	83.708	44.039	273.558	48.297	9728568.10
19	325.393	4.457	60.402	-18.541	10702629.73
20	201.680	15.173	142.753	-29.194	8013018.54

10 ANNEX H: Cookbook entry point

Given the large amount of information included in this cookbook and the way it changes for all the different float types, there needs to be an easy way for DACs to use this cookbook to make calculations for the trajectory file. This Annex has been created to make it easy for DACs to find out what calculations are needed based on float version. The Annex consists of tables including all float versions versus all measurement codes which are needed to fill the trajectory files. This should prevent against forgetting anything.

In the final version of the cookbook, the cells should be filled:

- By N/A (for Not Applicable) if the concerned data cannot (float capability) be produced from the given float version,
- Otherwise, the list of paragraphs in the cookbook that explain how to process.

These tables thus provide an overview of all the data expected to be in the TRAJ file for a given float version and a direct access (through the ability to jump to linked paragraphs (CTRL+Click)) to the concerned specifications.

These table are updated through 2014 for PROVOR, ARVOR, NINJA and SOLO floats.

All other tables are not updated or usable at this time.

10.1 PROVOR floats

Format Id	Code 0 Launch	Code 100 DST	Code 150 FST	Code 200 DET	Code 250 PST	Code 300 PET	Code 400 DDET
101001							
101002	3.1.1	3.2.2.6.6.1	3.2.2.6.6.2	FillValue	3.2.2.6.6.3	3.2.2.6.6.4	FillValue
101003	3.1.1	3.2.2.6.7.1	3.2.2.6.7.2	FillValue	3.2.2.6.7.3	3.2.2.6.7.4	FillValue
101004	3.1.1	3.2.2.6.7.1	3.2.2.6.7.2	FillValue	3.2.2.6.7.3	3.2.2.6.7.4	FillValue
101005	3.1.1	3.2.2.6.6.1	3.2.2.6.6.2	FillValue	3.2.2.6.6.3	3.2.2.6.6.4	FillValue

101006	3.1.1	3.2.2.6.3.1	3.2.2.6.3.2	FillValue	3.2.2.6.3.3	3.2.2.6.3.4	FillValue
101007	3.1.1	3.2.2.6.5.1	3.2.2.6.5.2	FillValue	3.2.2.6.5.3	3.2.2.6.5.4	FillValue
101008 - 101010	3.1.1	3.2.2.6.3.1	3.2.2.6.3.2	FillValue	3.2.2.6.3.3	3.2.2.6.3.4	FillValue
101011 - 101015	3.1.1	3.2.2.6.1.1	3.2.2.6.1.2	FillValue	3.2.2.6.1.3	3.2.2.6.1.4	FillValue
101016							
101017 - 101019	3.1.1	3.2.2.6.1.1	3.2.2.6.1.2	FillValue	3.2.2.6.1.3	3.2.2.6.1.4	FillValue
101020							

Format Id	Code 450 DPST	Code 500 AST	Code 550 DAST	Code 600 AET	Code 700 TST	Codes 702-704 FMT, LMT	Code 800 TET
101001							
101002	3.2.2.6.6.5	3.2.2.6.6.6	FillValue	3.2.2.6.6.7	3.2.2.6.6.8	3.2.1.1.1	FillValue
101003	3.2.2.6.7.5	3.2.2.6.7.6	FillValue	3.2.2.6.7.7	3.2.2.6.7.8	3.2.1.1.1	FillValue
101004	3.2.2.6.7.5	3.2.2.6.7.6	FillValue	3.2.2.6.7.7	3.2.2.6.7.8	3.2.1.1.1	FillValue
101005	3.2.2.6.6.5	3.2.2.6.6.6	FillValue	3.2.2.6.6.7	3.2.2.6.6.8	3.2.1.1.1	FillValue
101006	3.2.2.6.3.5	3.2.2.6.3.6	FillValue	3.2.2.6.3.7	3.2.2.6.3.8	3.2.1.1.1	FillValue
101007	3.2.2.6.5.5	3.2.2.6.5.6	FillValue	3.2.2.6.5.7	3.2.2.6.5.8	3.2.1.1.1	FillValue
101008 - 101010	3.2.2.6.3.5	3.2.2.6.3.6	FillValue	3.2.2.6.3.7	3.2.2.6.3.8	3.2.1.1.1	FillValue
101011 - 101015	3.2.2.6.1.5	3.2.2.6.1.6	FillValue	3.2.2.6.1.7	3.2.2.6.1.8	3.2.1.1.1	FillValue
101016							
101017 - 101019	3.2.2.6.1.5	3.2.2.6.1.6	FillValue	3.2.2.6.1.7	3.2.2.6.1.8	3.2.1.2.1	FillValue
101020							

Format Id	Code 189	Code 190	Code 198	Code 203	Code 290	Code 297	Code 298
101001							
101002 - 101005		3.4.2.4.1		3.4.2.5	3.4.1.2.1		
101006 - 101010		3.4.2.4.1		3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101011		3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101012		3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101013		3.4.2.4.1		3.4.2.5	3.4.1.2.1		
101014		3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101015		3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101016							
101017 - 101019	3.4.2.11	3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6
101020							

Format Id	Code 301 RPP	Code 389	Code 398	Code 503	Code 589	Code 590	Code 901 GRND
101001							
101002 - 101015	3.4.3			3.4.2.5		3.4.2.4.1	3.5
101016							
101017 - 101019	3.4.3	3.4.2.11	3.4.2.9	3.4.2.5	3.4.2.11	3.4.2.4.1	3.5
101020							

10.2 PROVOR-MT floats

Format Id	Code 0 Launch	Code 100 DST	Code 150 FST	Code 200 DET	Code 250 PST	Code 300 PET	Code 400 DDET
??????							
100001	3.1.1	3.2.2.6.1.1	3.2.2.6.1.2	FillValue	3.2.2.6.1.3	3.2.2.6.1.4	FillValue
100002	3.1.1	3.2.2.6.6.1	3.2.2.6.6.2	FillValue	3.2.2.6.6.3	3.2.2.6.6.4	FillValue
100003 - 100006	3.1.1	3.2.2.6.4.1	3.2.2.6.4.2	FillValue	3.2.2.6.4.3	3.2.2.6.4.4	FillValue
100007							
100008	3.1.1	3.2.2.6.4.1	3.2.2.6.4.2	FillValue	3.2.2.6.4.3	3.2.2.6.4.4	FillValue
100009							

Format Id	Code 450 DPST	Code 500 AST	Code 550 DAST	Code 600 AET	Code 700 TST	Codes 702-704 FMT, LMT	Code 800 TET
??????							
100001	3.2.2.6.1.5	3.2.2.6.1.6	FillValue	3.2.2.6.1.7	3.2.2.6.1.8	3.2.1.1.1	FillValue
100002	3.2.2.6.6.5	3.2.2.6.6.6	FillValue	3.2.2.6.6.7	3.2.2.6.6.8	3.2.1.1.1	FillValue
100003 - 100006	3.2.2.6.4.5	3.2.2.6.4.6	FillValue	3.2.2.6.4.7	3.2.2.6.4.8	3.2.1.1.1	FillValue
100007							
100008	3.2.2.6.4.5	3.2.2.6.4.6	FillValue	3.2.2.6.4.7	3.2.2.6.4.8	3.2.1.1.1	FillValue
100009							

Format Id	Code 198	Code 203	Code 290	Code 297	Code 298	Code 301 RPP
??????						
100001	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6	3.4.3
100002 - 100006		3.4.2.5	3.4.1.2.1			3.4.3
100007						
100008		3.4.2.5	3.4.1.2.1			3.4.3
100009			3.4.1.2.1			

Format Id	Code 390	Code 398	Code 497	Code 498	Code 503	Code 590	Code 901 GRND
??????							
100001 - 100006					3.4.2.5		3.5
100007							
100008					3.4.2.5		3.5
100009							

10.3 ARVOR floats

Format Id	Code 0 Launch	Code 100 DST	Code 150 FST	Code 200 DET	Code 250 PST	Code 300 PET	Code 400 DDET
102001							
102002 - 102004	3.1.1	3.2.2.6.2.1	3.2.2.6.2.2	FillValue	3.2.2.6.2.3	3.2.2.6.2.4	FillValue

Format Id	Code 450 DPST	Code 500 AST	Code 550 DAST	Code 600 AET	Code 700 TST	Codes 702-704 FMT, LMT	Code 800 TET
102001							
102002	3.2.2.6.2.5	3.2.2.6.2.6	FillValue	3.2.2.6.2.7	3.2.2.6.2.8	3.2.1.1.1	FillValue
102003	3.2.2.6.2.5	3.2.2.6.2.6	FillValue	3.2.2.6.2.7	3.2.2.6.2.8	3.2.1.1.1	FillValue
102004	3.2.2.6.2.5	3.2.2.6.2.6	FillValue	3.2.2.6.2.7	3.2.2.6.2.8	3.2.1.2.1	FillValue

Format Id	Code 190	Code 198	Code 203	Code 290	Code 297	Code 298	Code 301 RPP
102001							
102002 - 102004	3.4.2.4.1	3.4.2.7	3.4.2.5	3.4.1.2.1	3.4.2.6	3.4.2.6	3.4.3

Format Id	Code 390	Code 398	Code 497	Code 498	Code 503	Code 590	Code 901 GRND
102001							
102002					3.4.2.5	3.4.2.4.1	3.5
102003					3.4.2.5	3.4.2.4.1	3.5
102004		3.4.2.9			3.4.2.5	3.4.2.4.1	3.5

10.4 NINJA floats

Format Id	Code 0 Launch	Code 100 DST	Code 150 FST	Code 200 DET	Code 250 PST	Code 300 PET	Code 400 DDET
300001 - 300003	3.1.1	3.2.2.4.1.1	3.2.2.4.1.2	FillValue	3.2.2.4.1.3	3.2.2.4.1.4	3.2.2.4.1.5
300004	3.1.1	3.2.2.4.2	3.2.2.4.2	FillValue	3.2.2.4.2	3.2.2.4.2	3.2.2.4.2

Format Id	Code 450 DPST	Code 500 AST	Code 550 DAST	Code 600 AET	Code 700 TST	Code 702-704 FMT, LMT	Code 800 TET
300001 - 300003	FillValue	3.2.2.4.1.6	FillValue	3.2.2.4.1.7	3.2.2.4.1.8	3.2.1.1.1	3.2.2.4.1.9
300004	FillValue	3.2.2.4.2	FillValue	3.2.2.4.2	3.2.2.4.2	3.2.1.1.1	3.2.2.4.2

Format Id	Code 290	Code 301 RPP	Code 498	Code 503	Code 590	Code 901 GRND
300001 - 300003	3.4.1.3.1.2	3.4.3	3.4.2.10	3.4.2.5	3.4.2.4.2	3.5

300004	3.4.1.3.1.2	3.4.3		3.4.2.5		3.5

11 ANNEX I: APEX APF8 Estimation methods for PST, PET, AST

11.1. Park Start Time (PST)

The mean descent rate to use depends on the PARKING depth, the recommended values are provided in the following table (see also §7.1).

PARKING depth	250 dbar	500 dbar	1000 dbar	1500 dbar	2000 dbar
Mean descent rate (cm/s)	2.6	3.6	5.9	12.4	9.0

Table 4: Recommended descent rates

Thus for cycle #i:

PST(i) = DST(i) + (PARKING_PRESSURE(i) * 100 * 36)/(mean descent rate * 864)

where PARKING_PRESSURE(i) is the theoretical PARKING_PRESSURE of cycle #i.

The PST value (MC=250) should be stored in the N_MEASUREMENT arrays only in the JULD_ADJUSTED variable since the time is estimated based on float behavior. The STATUS should be set to 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour. Float clock offset corrections can also be applied.

The JULD variables should be fill value.

The PST value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 1 (estimated using procedures that rely on typical float behavior).

If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

If no estimate is made, fill value should be stored in the JULD variable in the N_MEASUREMENT array with the measurement code set to 250 and the STATUS set to 9.

N_CYCLE arrays: fill value should be stored in the JULD_PARK_START variable and the JULD_PARK_START_STATUS set to 9.

11.2. Park End Time (PET)

We must check first that, for the corresponding cycle, the theoretical PARKING and PROFILE depths differ (be careful with PnP floats, see §5.1).

If not, there is no PET, do not include it in the N_MEASUREMENT array and put fill value in the N_CYCLE array JULD_PARK_END and JULD_PARK_END_STATUS variables.

Otherwise PET = TET - UP_TIME - DPDP hours.

Where DPDP is the value of the Deep Profile Descent Period, a programmed meta-data parameter that determines the maximum amount of time given to the float for diving from PARKING to PROFILE depth. In older floats without this metadata, DPDP is often between 4 and 6 hours. For the newer APF9 firmware, this time period is user-specified.

If the float clock offset has been estimated during the TET determination, the CLOCK_OFFSET (N_CYCLE) variable should also be filled. Place PET in the JULD_ADJUSTED

(N_MEASUREMENT) variables with MC=300 and a STATUS of 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour.

For the N_CYCLE array, PET value should be stored in the JULD_PARK_END variable and the JULD_PARK_END_STATUS set to. If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

11.3. Ascent Start Time (AST)

If the PARKING and PROFILE depths are equal for cycle #i, then:

(1) : AST(i) = TET(i) - UP TIME

If not, we can however **roughly** estimate AST using AET and the profile duration.

(2) : AST(i) = AET(i) - duration of profile #i

The duration of profile #i can be estimated with the profile deepest pressure (ProfMaxPres(i)) and a mean ascent rate.

ProfMaxPres(i)) is the maximum pressure of the profile **if the Argos message of the first profile measurement has been received (otherwise, AST(i) should not be estimated).**

The mean Ascent rate to use can be **9.5 cm/s** (see §7.2).

Thus AST(i) = AET(i) - (PARKING_ ProfMaxPres (i) * 100 * 36)/(9.5 * 864)

We can also verify that AST(i) is in the interval

[TET(i) - UP TIME - DPDP hours; TET(i) - UP TIME].

Note that AST estimated in (1) is much more reliable than AST estimated in (2), associated JULD_QC should reflect it.

The AST value should also be stored in the JULD_ADJUSTED variables with an MC = 500 and STATUS set to 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour. Apply clock offset if it has been determined.

For the N_CYCLE array, the AST value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 1. If float clock offset has been estimated and applied, make sure to fill in the CLOCK_OFFSET (N_CYCLE) variable so users know it has been applied.

If the AST value is not estimated, fill value should be stored in the JULD variable with an MC=500 and STATUS set to 9.

N_CYCLE arrays: fill value should be stored in the JULD_ASCENT_START variable and the JULD_ASCENT_START_STATUS set to 9.

Ascent Start Time provided by the float

Some float versions (see ANNEX H: Cookbook entry point) directly provide the time at the end of DOWN TIME period ($DTET_{FL}$).

These float versions also provide, in the Auxiliary Engineering Data (AED), the "Time of profile initiation". This information is defined as the time difference, in minutes, between profile start and end

of DOWN TIME (negative for start before expiration and positive for start after expiration, thus in this latter case, necessarily when TOD feature has been set).

The AED are not always transmitted (depending on the remaining space in the last Argos message) but if received, this "Time of profile initiation" (TPI) can be used to compute a second value of AST provided by the float (AST_{FL}).

$AST_{FL} = DTET_{FL}$ + TPI minutes

AST_{FL} value computed from $DTET_{FL}$ (corrected from clock offset) does not need to be corrected from clock offset but the information should be set in the AST_{FL} storage.

AST_{FL} is stored in the JULD_ADJUSTED N_MEASUREMENT arrays with the MC = 502 and the STATUS equal to 3: value is computed from information transmitted by the float. Clock offset has been applied in the $DTET_{FL}$ variable.

Argo program measurement codes (MC) for APEX APF8 floats in REAL TIME				
Code (timing)	**APF8 Variable**	**Description**	**Units**	**JULD_STATUS**
0	Float does not know when it is launched. If the launch time and location are available from the ship, enter that time and location If the launch time and location are not available, use fill value.	Launch time and location	Time, position	0: value is estimated from pre-deployment information found in the metafile Or 9: value is not immediately known, but believe it can be estimated later
100 (DST)	TET from previous cycle OR Fill Value	If TET is estimated in real time, use the TET from previous cycle. OR If TET is not estimated in real time, use FillValue	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
200 (DET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
250 (PST)	Not available, so use Fill Value OR Park Start Time estimated in 11.1			9: value is not immediately known, but believe it can be estimated later OR 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour
During the drift phase, the APF8 makes drift measurements. Common codes are listed below. See 3.4.1.1 for CTD measurements during drift for APEX floats				
296	Average pressure Average temperature	Any averaged measurements made during drift	Pressure Temp	2: value is transmitted by the float
297	Minimum pressure Minimum temperature	Minimum value taken during drift	Pressure Temp	2: value is transmitted by the float
298	Maximum pressure Maximum temperature	Maximum value taken during drift	Pressure Temp	2: value is transmitted by the float
End of drift measurements				

300 (PET)	Not available, so use Fill Value OR Park End Time estimated using 11.1 CTD performed at end of drift		Time Time P, T, S	9: value is not immediately known, but believe it can be estimated later OR 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour
301	Average pressure during drift	Best estimate of drift depth. See section 3.4.3 for more details	Pressure	3: value is directly computed from relevant, transmitted float information
400 (DDET)	Not available, so use Fill Value			9: value is not immediately known, but believe it can be estimated later
500 (AST)	If PARK and PROFILE depths are equal and TET is estimated in real time: AST(i)=TET(i) – UP TIME OR Ascent Start Time estimated in 11.1 OR FillValue	See 3.2.2.1.7	Time	1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 1: value is estimated using information not transmitted by the float or by procedures that rely on typical float behaviour OR 9: value is not immediately known, but believe it can be estimated later
501	DownTimeEpoch/UNIX epoch when the down-time expired	Down-time end time – time out	Time	2: value is transmitted by the float
600 (AET)	Float does not know when it reaches the surface, so Fill Value		Time	9: value is not immediately known, but believe it can be estimated later
700 (TST)	See section 3.2.2.1.9 & 6.2	Based on Argos messages	Time	3: value is directly computed from relevant, transmitted float information
701 TST sent by APEX floats	TST_{FL} = $DTET_{FL}$ + TOTPI minutes	See 3.2.2.1.9.2	Time	3: value is directly computed from relevant, transmitted float information
702 (FMT)	Earliest time of all Argos messages received	Time	Time	4: value is determined by satellite
703 (ST)	All Argos times and locations		Time, Position	4: value is determined by satellite
704 (LMT)	Latest time of all Argos messages received		Time	4: value is determined by satellite
800 (TET)	3.2.2.1.1 and Annex B (5.3) OR FillValue	DACs can choose to make this estimate in real time or not. Annex B explains how to make the estimate. 3.2.2.1.1 gives guidance how to implement the method in Annex B	Time	3: value is directly computed from relevant, transmitted float information OR 9: value is not immediately known, but believe it can be estimated later