Iridium Apex Manual
(Apf9i Firmware Revision: 041511)

Dana Swift[*]
School of Oceanography
University of Washington
Seattle, Washington 98195

April 14, 2011

---

[*] swift@ocean.washington.edu, (206) 543-6697

Revision Log.

The following revision log summarizes the history of this Iridium APEX User Manual.

```
$Log: IridiumApex.tex,v $
Revision 1.3  2008/07/14 17:02:19  swift
Added compensator hyper-retraction feature to allow floats to be
parked at mid-water pressures.

Revision 1.2  2006/11/22 02:54:56  swift
Change to facilitate automatic revision control.

Revision 1.1  2006/11/03 19:08:57  swift
Added user manual to CVS control.

Revision 1.1  2006/04/14 23:21:31  swift
First draft of the Iridium-APEX user manual.

Revision 0.8  2006/04/14 23:18:49  swift
Added a section on the remote control facility.

Revision 0.7  2006/04/14 17:32:06  swift
Added a section describing mission configuration facility.

Revision 0.6  2006/04/11 15:34:03  swift
Added a section on decoded and processed data.

Revision 0.5  2006/04/11 14:49:21  swift
Added section on recovery mode operations and functionality.

Revision 0.4  2006/04/10 16:15:42  swift
Finished initial revision of the parametric model of Iridium APEX missions.

Revision 0.3  2006/04/09 16:12:08  swift
Added a section describing the remote host functions and set-up.

Revision 0.2  2005/12/27 23:29:56  swift
Added a section describing the profile cycle model.

Revision 0.1  2005/12/21 17:15:50  swift
Predistribution partial draft.
```

Contents

List of Figures

List of Tables

1 Introduction

WARNING: This Iridium APEX user manual applies only to Apf9i firmware revision 041511.

You should treat this manual as if it were a hint of what the firmware actually does. If your style does not include compulsive skepticism and a neurotic obsession with understanding why things do (or don't) work then my style of float development and techonology transfer might not be for you. When you need a Reference Manual, you should go straight to The Source which was written entirely in the C programming language and is freely available.

2 Controlling Iridium APEX behavior: A parametric model.

The Iridium APEX firmware is highly configurable so that the user can control float behavior by adjusting the values of more than 20 parameters and by selecting several optional modes and features.

2.1 Sample missions.

The ability to configure the float within a 20(plus) dimensional parameter space means that that range of possible float behaviors is practically infinite. However, some general characteristics span the whole parameter space while many potentially useful kinds of missions are excluded entirely. Figures 1, 2, and 3 represent common mission cycles within the usable parameter space.

Figure 1 represents the most general kind of mission cycle and is referred to as *Park-n-Profile* (PnP). The original motivation for PnP was as a mechanism to balance the competing objectives of energy savings versus direct measurement of salinity drift in the deeper water. The basic idea was to collect most profiles from the park level but occasionally execute a deep profile to facilitate evaluation of CTD performance. The "n" in PnP refers to the cycle length of the PnP mechanism; every n^{th} profile is a deep profile.

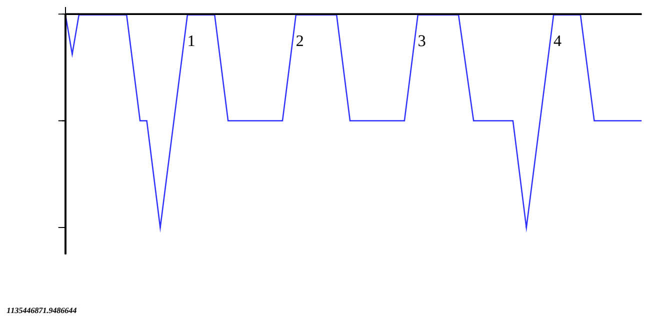

Figure 1: Schematic of a PnP mission with cycle length $n = 4$. The park level is the same for all profiles. Every fourth profile is a deep profile. The shallow blip prior to the (special) first profile represents pressure activation.

The first profile is special because it is executed immediately after the mission prelude, does not drift at the park level, and is also a deep profile. The first profile will be telemetered within 24 hours after the mission is activated. The exact timing will depend on the user's specific parameter selections. This feature was implemented to satisfy the often-heard request for a profile to be executed immediately after deployment.

Figure 2 represents a PnP mission with $n = 1$ (ie., a P1P mission). In this way, PnP firmware can be programmed to collect lagrangian data from a shallower level while still being able to collect deep profiles. As with the P4P mission in Figure 1, the first profile is executed immediately after the mission prelude.

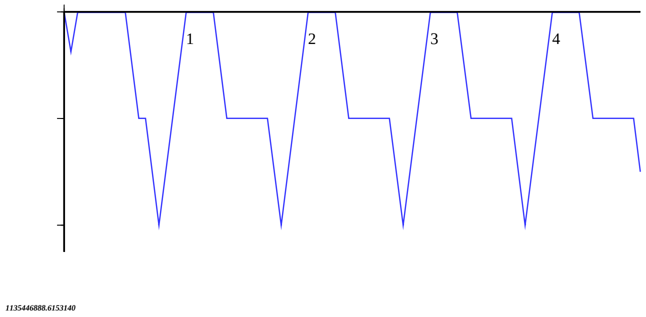

Figure 2: Schematic of a PnP mission with cycle length $n = 1$. Every profile parks shallow but profiles deep. The shallow blip prior to the (special) first profile represents pressure activation.

Figure 3 represents a degenerate case of the PnP model where n is large and the park level has been chosen to be deep. This mission cycle is so common amongst APEX users that it was implemented as a special case. The value $n = 254$ is a special sentinel value that disables the PnP feature so that only park-level mission parameters (ie., park pressure and park piston position) are used for controlling the profile cycle; the profile-level parameters (ie., profile pressure and profile piston position) are ignored.

As with the previous two examples, the first profile is executed immediately after the mission prelude.

2.2 Deconstructing the profile cycle.

Details of the firmware architecture and design are outside the scope of this user manual. However, deconstructing the profile cycle into its constituent elements will give meaning to many of the configuration parameters.

The Apf9i firmware design makes fundamental use of the concept of "sequence points" for controlling the flow of the profile cycle. A sequence point is defined to be a point where one phase of the mission cycle transitions to the next phase. Most of the sequence points are based on time but there are several sequence points that are event-based. Given a properly functioning Apf9i controller, the firmware guarantees the phase transition at each sequence point regardless of the health of any other float component.

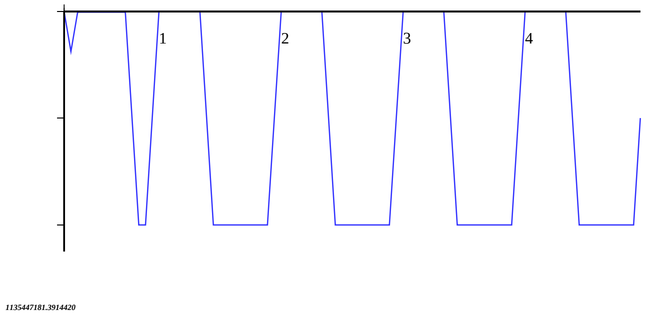

Figure 3: Schematic of a degenerate PnP mission with cycle length n=254. Every cycle parks and profiles from the park level. The shallow blip prior to the (special) first profile represents pressure activation.

The schematics below illustrate different parts of the mission cycle:

1. The pressure-activation phase.
2. The mission prelude.
3. A profile from the park level.
4. A deep profile.

2.2.1 Pressure-activation phase (*optional*).

The pressure-activation feature is an optional phase of the mission that preceeds the mission prelude. It was designed to accomodate requests from ship's crew to be able to deploy the float without being required to start it with a magnet. One event-based sequence point is implemented that activates the mission prelude if the pressure exceeds the activation threshold (ie., 25 decibars).

```
                        |----------------- L ----------------------|
-+----------------+-----+-------+----------------------------------+- Time
P|                 .    |      .                                    .
r|                   .  |     .       Sequence Points                 .
e|                      .    .        -----------------------           .
s|                      B   .         L = Mission prelude                  .
s|                                    B = Pressure-activation
u|
r|
e|
```

Pressure activation mode is not "on" by default. The user must enable pressure activation mode via the interactive user interface (see Section 3). Enabling the pressure activation mode immediately induces the firmware to perform a self-test of the float.

In order for pressure activation to work the float has to be able to sink from the surface down to the activation pressure (ie., 25 decibars). Obviously, if the float is too buoyant to sink then it can never self-activate. Enabling pressure activation mode causes the firmware to put the float into a state of minimum buoyancy; the buoyancy pump is fully retracted to deflate the oil bladder and the air solenoid valve is opened to deflate the air bladder. Then the firmware enters a nonterminating loop where the the CTD is queried for pressure every two hours. If the pressure is less than the activation pressure then the Apf9i controller puts itself back to sleep for another two hours. However, if the pressure exceeds the activation threshold then the firmware launches the mission and enters the mission prelude.

2.2.2 The mission prelude.

The purpose of the mission prelude is mainly to allow the float to transmit a fix of its deployment location and to telemeter its mission programming parameters. The mission prelude is the time period between mission activation and the first descent. The sequence point 'L' is time-based and is the transition between the mission prelude and the first descent. The period of the mission prelude is user-defined (see Section 3).

```
 |--------------------------- L -----------------------------------|
-+-----------------------------------------------------------------+- Time
P|                                                                  .
r|                                              Sequence Points      .
e|                                              -------------------   .
s|                                              L = Mission prelude     .
s|
u|
r|
e|
```

When the mission is launched either manually or else by the pressure activation mechanism, the firmware puts the float into a state of maximum buoyancy by fully extending the buoyancy pump to inflate the oil bladder and then inflating the air bladder.

2.2.3 Profile from park depth.

The profile cycle for a shallow profile consists of four phases (each associated with a time-based sequence point): descent (F), park (K), profile (P), and telemetry (C). Two additional event-based sequence points (S,T) ordinarily cause phase transition before their associated time-outs force the phase transition.

```
 |--------------------------------------------------- C ----------------|
 |------------------------------ P ---------------|                     |
 |----------------- K -------------|              |                     |
 |--- F -----|                     |           S  |                T    |
-+-----------+---------------------+-----------+-------------------+----+- Time
P|.          |                     |         .                      .
r| .         |                     |       .       Sequence Points   .
e|  .        |                     |     .         ---------------    .
s|    .      |                     |   .           F = Descent          .
s|       .   |                     | .             K = Park
u|           . . . . . . . . . . . .               S = Surface detect
r|                                                 P = Profile
e|                                                 T = Telemetry
 |                                                 C = Cycle
```

This is the simplest kind of profile cycle which APEX floats have executed since their initial development. The float sinks to its park depth, drifts for a period of time, profiles to the surface, and then telemeters its data. However, unlike APEX with ARGOS telemetry, the length of time for each profile cycle is not fixed. The profile cycle for Iridium APEX ends as soon as the telemetry is successfully completed or the profile cycle time-out (C) expires, whichever happens first. Typically, an Iridium float is on the surface for only 15 minutes or so before the next profile cycle begins.

For this kind of profile cycle, the end of the park phase (K) coincides with the end of the “down-time” and the beginning of the “up-time”. The sequence point C coincides with the end of the up-time. The (maximum) length of the profile cycle is the down-time plus the up-time.

Descent phase —The profile cycle starts with the descent phase. The float queries the CTD for the surface pressure and then the buoyancy pump is retracted to the park position. The float sinks until the descent period expires (ie., the firmware forces a phase transition at the sequence-point F in the schematic above). Hourly pressure measurements are logged as well as one at the completion of the buoyancy pump retraction. These pressure measurements are referred to as *descent marks* and are telemetered as engineering data.

Special notes for floats with N_2 compensator: The gas pressure (P_{N_2}) in the N_2 compensator plays an important role during the descent phase. While the float is at the surface, the N_2 compensator piston is fully extended which adds ~80 grams of buoyancy to the float. In order to descend back down to the park level, the buoyancy pump must reduce the buoyancy enough to descend deeper than the gas pressure P_{N_2}. To accomplish this, the piston is retracted beyond the park piston position. This is referred to as *compensator hyper-retraction.* Near the end of the descent phase, the piston is extended back to the park position. This allows floats with N_2 compensator to park at any depth greater than 850 dbars. **Warning:** The N_2 compensator renders the float buoyantly unstable in the range of pressures ~400–700 decibars. Therefore, such floats must be parked outside this range.

Park phase —Active ballasting is accomplished and park-level PT measurements are collected during the park phase. The Apf9i wakes once each hour to accomplish these tasks. A PTS sample is collected at the end of the park phase.

Active ballasting: The float wakes each hour to monitor the pressure and make buoyancy adjustments if three **consecutive** measurements violate a 10 decibar dead-band on either side of the user-specified park pressure. Measurements that are within the dead-band or that completely cross the dead-band reset the violation counter and will not induce buoyancy adjustments.

Park-level PT samples: The float collects hourly low-power PT measurements and telemeters them as hydrographic data. Refer to Section 6.1.1 for their data format. Hourly salinity data are not measured due to energy considerations.

Park-level PTS sample: The float collects one PTS sample at the end of the park phase (K). Refer to Section 6.1.2 for its data format.

Profile phase —As might be expected, the profile phase is the most complicated of the profile cycle. Three asynchronous processes are active during the profile cycle: Ascent rate control, hydrographic sampling, and surface detection. These processes operate on a 10 second heartbeat; the Apf9i controller sleeps for 10 seconds then wakes to attend to these processes before going back to sleep.

Ascent rate control: As an initialization step of the profile phase, the buoyancy engine adds a user-specified initial increment of buoyancy to start the float ascending toward the surface. Thereafter, the firmware monitors the pressure at 5 minute intervals to determine if the average ascent rate has been maintained above 0.08 decibars/sec. If the ascent rate falls below this threshold then the buoyancy engine adds a user-specified increment of buoyancy.

Hydrographic sampling: Iridium APEX is designed to collect hydrographic profiles with relatively high vertical resolution. The Sbe41cp CTD has a continuous profiling (CP) mode that runs asynchronously and autonomously from the float's Apf9i controller. When CP-mode is active, the CTD collects 1-Hz samples and stores them internally in non-volatile memory.

The Apf9i shuts down CP-mode 4 decibars below the surface to avoid contaminating the conductivity cell with ingested surface scum. To protect against pressure-sensor drift, the Apf9i commands the Sbe41cp to shut down 4 decibars deeper than the most recent surface pressure measurement. As a fail-safe measure, the Sbe41cp will shut itself down when its pressure sensor reaches 2 decibars (but no attempt is made to account for drift of the pressure sensor).

After the float reaches the surface the Apf9i commands the Sbe41cp to sort the 1-Hz samples into 2 decibar bins and compute the arithmetic mean of the samples in each bin. The resulting bin-averaged profiles have 2 decibar resolution though we often refer to them as "high resolution" or "continuous profiles". These high resolution profiles are telemetered using the format defined in Section 6.1.3.

The Sbe41cp also has a spot-sampled mode that is roughly simlar to the Sbe41 used on ARGOS APEX floats. This Iridium firmware implements an optional mixed-mode sampling strategy in order to save energy in the deep water where gradients are small. At the user's discretion, the float can be programmed to collect spot samples in the deep water and automatically transistion to continuous profiling when the float ascends to the *CP Activation Pressure*. To disable this feature and force continuous profiling from top to bottom then the user should specify the activation pressure to be deeper than the float's operating range. Spot-samples are formated according to Section 6.1.2 and collected according to a pressure table that is hard-coded in firmware.

Surface detection: The surface detection algorithm terminates true when the float ascends to a pressure that is within 4 decibars of the most recent surface pressure measurement[1]. Surface detection causes the profile to be terminated, another increment of buoyancy to be added by the buoyancy pump, and transition to the telemetry phase.

Telemetry phase —If the end of the antenna is even a centimeter below the water's surface then telemetry will not be possible. Therefore, the telemetry phase starts with precise surface detection using its SkySearch algorithm. The heart of this algorithm involves attempting to register the LBT with the Iridium system. If the algorithm terminates true then GPS acquisition and telemetry can proceed; otherwise, the buoyancy engine adds another increment of buoyancy and sleeps for one telemetry-retry period before repeating the attempt.

After determining that the float can see the sky then the LBT is shut down and the GPS engine is used to acquire the float's location. After acquiring the fix then the GPS is shut down and the LBT is reregistered with the Iridium system. The float places a call via the Iridium system to the remote host and logs in using the float's username and password. Once logged into the remote host, the float downloads its new configuration from the remote host and uploads its hydrographic and engineering data to the remote host. Finally, the float logs out of the remote host and reprograms its mission paramters according to the new configuration file that it just received from the remote host.

2.2.4 Deep profile.

For PnP cycle lengths less than 254, the first profile cycle will be a deep one as will profile cycles for which the internal profile counter (PrfId) is an integral multiple of the profile cycle length. For example, a P4P mission will execute a deep profile cycle when the internal profile counter is 1, 4, 8, 12, and so on. The following C source code represents the test executed by firmware to determine if the current profile cycle is a deep one:

```
(PnpCycleLength<254 && ((!(PrfId%PnpCycleLength)) || PrfId==1) ? Yes : No;
```

The profile cycle for a deep profile consists of five phases (each associated with a time-based sequence point): descent (F), park (K), deep-descent (D), profile (P), and cycle (C). Three additional event-based sequence points (Q,S,T) ordinarily cause phase transition before their associated time-outs force the phase transition.

---

[1] Actually, the surface detection algorithm is more complicated than this but the complications handle pathological situations. Refer to the C source code (src/profile.c) in your distribution.

```
 |------------------------------------------------------ C -------------|
 |--------------------------------- P ------------|                     |
 |----------------- D --------|                   |                     |
 |------- K -------------|    |                   |                     |
 |--- F -----|           |    |                S  |                T    |
-+-----------+-----------+----+----------------+-------------------+------- Time
P|.          |           |    |              .                      .
r| .         |           |    |            .       Sequence Points   .
e|  .        |           |    |          .         ---------------    .
s|    .      |           |    |        .           F = Descent          .
s|       .   |           |    |      .             K = Park
u|           . . . . . . .    |    .               Q = DeepProfile
r|                            |  .                 D = DeepDescent
e|                        .   |.                   S = SurfaceDetect
 |                           .                     P = Profile
 |                         .                       T = Telemetry
 |                         Q                       C = Cycle
```

For this kind of profile cycle, the end of the deep-descent phase (D) coincides with the end of the "down-time" and the beginning of the "up-time". The sequence point C coincides with the end of the up-time. The (maximum) length of the profile cycle is the down-time plus the up-time.

Descent phase —Refer to description on page 5.

Park phase —Refer to description on page 5. The park phase is shortened for the deep profile cycle in order to allow time to descend from the park level to the deep target pressure.

Deep descent phase —The purpose of the deep descent phase is to allow the float to descend from the park level (eg., 1000 dbars) to the pressure where the deep profile should begin (eg., 2000 dbars). The maximum time allowed for the deep descent phase is user specified (see Section 3).

The deep descent phase begins by retracting the piston from the park piston position to the profile piston position. During the descent, the pressure is monitored every five minutes to determine of the target pressure has been reached (ie., sequence point Q).

Sequence point Q —If the float descends to its deep target pressure before the deep descent phase times out then the profile piston position is incremented by one count (but no piston extension occurs). The intent is to reduce the descent speed during the next deep profile so that the float will reach the target pressure closer to the end of the down time (ie., sequence point D).

Sequence point D —If the deep descent period times out before the target pressure is reached then the profile piston position is decremented by one count (but no piston retraction occurs). The intent is to increase the descent speed during the next deep profile so that the float will reach the target pressure before the end of the down time (ie., sequence point D).

Transition to the profile phase is forced at either sequence point Q or D.

Profile phase —Refer to description on page 6.

Telemetry phase —Refer to description on page 7.

et

3 Mission configuration.

The deconstruction of the profile cycle in Section 2.2 will provide the framework for understanding how various parameter values determine the nature of the mission. The float's mission is configured according to the following mission parameters:

```
APEX version 032607  sn 0036
User: f5048
Pwd:  0xafb3
Pri:  ATDT0012065550123                         Mhp
Alt:  ATDT0012065551234                         Mha
  0420 ToD for down-time expiration. (Minutes)   Mtc
 01440 Down time. (Minutes)                      Mtd
 00600 Up time. (Minutes)                        Mtu
 00480 Ascent time-out. (Minutes)                Mta
 00359 Deep-profile descent time. (Minutes)      Mtj
 00360 Park descent time. (Minutes)              Mtk
 00360 Mission prelude. (Minutes)                Mtp
 00015 Telemetry retry interval. (Minutes)       Mhr
 00060 Host-connection time-out. (Seconds)       Mht
  1000 Continuous profile activation. (Decibars) Mc
  1000 Park pressure. (Decibars)                 Mk
  2000 Deep-profile pressure. (Decibars)         Mj
   065 Park piston position. (Counts)            Mbp
   000 Compensator hyper-retraction. (Counts)    Mbh
   012 Deep-profile piston position. (Counts)    Mbj
   010 Ascent buoyancy nudge. (Counts)           Mbn
   022 Initial buoyancy nudge (Counts)           Mbi
   004 Park-n-profile cycle length.              Mn
   124 Maximum air bladder pressure. (Counts)    Mfb
   096 OK vacuum threshold. (Counts)             Mfv
   250 Piston full extension. (Counts)           Mff
   100 Piston storage position. (Counts)         Mfs
     2 Logging verbosity. [0-5]                  D
```

A description of each mission parameter follows:

User & Pwd —The user-name and password used by the float to log into the remote host. The display shows an encoded version of the password rather than the password itself.

Pri & Alt —The AT dialstrings used by the Iridium LBT (ie., modem) to dial the primary and alternate remote hosts. Two remote hosts are needed—reliance on only one remote host is dangerous and strongly discouraged.

TimeOfDay —This allows the user to specify that the down-time should expire at a specific time of day (ToD). For example, the ToD feature allows the user to schedule profiles to happen at night.

The ToD is expressed as the number of minutes after midnight (GMT). The valid range is 0-1439 minutes. Any value outside this range will cause the ToD feature to be disabled.

Down-time —The total amount of time allowed for the *descent* and *park* phases of the profile cycle. The sequence points K (Section 2.2.3) and D (Section 2.2.4) mark the end of the down-time. The valid range is 1 minute to 30 days.

Note: If the **TimeOfDay** feature is enabled then the length of the whole profile cycle will turn out to be an integral number of days. The user should specify the down-time to be precisely 1 day less than the desired length of the profile cycle. For example, if profiles are to be executed every 10 days then the down-time should be specified to be 9 days (ie., 12960 minutes).

Up-time —The total amount of time allowed for the *profile* and *telemetry* phases of the profile cycle. Sequence points K [C] (Section 2.2.3) and D [C] (Section 2.2.4) mark the beginning [end] of the up-time. The valid range is 1 minute to 24 hours.

Ascent time-out —The maximum amount of time allowed for the profile phase to complete. Sequence points K [P] (Section 2.2.3) and D [P] (Section 2.2.4) mark the beginning [end] of the ascent time-out period. The valid range is 1 minute to 10 hours.

Deep-profile descent time —The maximum amount of time allowed for the float to descend from the park pressure to the deep target pressure. Sequence points K [D] (Section 2.2.4) mark the beginning [end] of the deep-descent period. The valid range is 0-8 hours.

Park descent time —The amount of time allowed for the float to descend from the surface to the park pressure before the park phase (and active ballasting) begins. The valid range is 1 minute to 8 hours.

Mission prelude —The amount of time allowed after float activation before the float begins its first descent. The valid range is 1 minute to 6 hours.

Telemetry retry interval —The amount of time after initiation of a telemetry attempt before initiating the next attempt (ie,. if the former fails). The valid range is 1 minute to 6 hours.

Host-connection time-out —The maximum amount of time allowed (after sending the AT dialstring) to receive the “CONNECT” response from the remote modem. The valid range is 30 seconds to 5 minutes.

Continuous profile activation —The target pressure for activating the continuous profile. During the profile phase, the firmware will stop collecting spot samples and initiate continuous profiling as soon as the float detects a pressure less than the target pressure. Any finite value is valid.

Park pressure —The target pressure for the active ballasting mechanism. The float firmware will seek to maintain the float at this pressure during the park phase. The valid range is 0-2000 decibars. **Warning:** Floats with N_2 compensator are buoyancy unstable in the range of pressures ∼400–700 decibars and must be parked outside this range. Refer to Section 2.2.3 for more details.

Deep-profile pressure —The target pressure for a deep profile. During the deep-descent phase, the pressure is monitored a 5 minute intervals. The profile phase is initiated when the float detects a pressure greater than this target pressure. The valid range is 0-2000 decibars.

Park piston position —An initialization value for the piston position at the park pressure. The autoballasting mechanism will automatically adjust this value to drive the float to the park pressure. The valid range is 1-254 counts.

Compensator hyper-retraction —This parameter exists solely for floats with N_2 compensator and causes the piston to be temporarily retracted beyond the park position. Near the end of the descent phase, the piston is extended back to the park position. This implements the ability to park such floats at any pressure deeper than 850 decibars. Refer to Section 2.2.3 for more details. The valid range is 0-254 counts. Floats without N_2 compensator should have this parameter set to zero.

Deep-profile piston position —An initialization value for the piston position at the deep-profile pressure. The Apf9i firmware will automatically adjust this value to drive the float to the deep-profile pressure in the time allowed. The valid range is 1-254 counts.

Ascent buoyancy nudge —The amount that the piston is extended when the ascent-control algorithm determines that more buoyancy is needed to maintain the minimum ascent rate of 8 millibars/second. The valid range is 1-254 counts.

Initial buoyancy nudge —The amount that the piston is extended at the beginning of the profile phase to get the float to start ascending. This same extension is also applied when the surface detection algorithm terminates. The valid range is 1-254 counts.

Park-n-profile cycle length —This parameter determines how often deep profiles should be executed. For example, if this value is 4 then profiles 4, 8, 12, and so on will be deep profiles. The valid range is 1-254. The value 254 is a special sentinel value that disables the park-n-profile feature.

Maximum air bladder pressure —This parameter determines the cut-off pressure when inflating the air bladder. The valid range is 1-240 counts.

OK vacuum threshold —This parameter determines the threshold internal pressure during the float's self-test at the beginning of the mission. If the internal pressure exceeds this threshold then the self-test will fail and the mission will be aborted. After the mission starts, this value is never used again. The valid range is 1-254 counts.

Piston full extension —This parameter determines the maximum piston extension allowed to prevent the buoyancy pump from self-destructing. The valid range is 1-254 counts.

Piston storage position —This parameter determines the preferred piston extension during storage and shipment. The valid range is 1-254 counts.

Logging verbosity —An integer in the range [0,5] that determines the logging verbosity with higher values producing more verbose logging. A verbosity of 2 yields standard logging.

The values of all mission parameters can be set via the firmware's command-mode user interface. In addition, a subset of these parameters can be set via the remote control user interface (see Section 4).

3.1 The *Configuration Supervisor.*

The objective of the Configuration Supervisor is to guard against various common classes of misconfigurations. When examining a given mission configuration, the Configuration Supervisor applies ~ 40 tests that seek to detect parameters or interactions between parameters that could be harmful or fatal to a deployed float.

However, **the Configuration Supervisor is not a substitute for a thinking human brain**—misconfigurations exist that can not be detected by firmware but which are effectively fatal to a deployed float. Thorough laboratory simulations of new configurations are strongly encouraged and careful predeployment testing of each float is essential.

Each of the ~ 40 tests is classified as either a constraint or a sanity check. Constraint violations are likely fatal to a deployed float and the Configuration Supervisor will refuse to accept parameters or combinations of parameters that violate a constraint. Sanity checks detect various suspicious conditions that are not likely fatal but that are probably inadvisable or unintended.

Each time the Configuration Supervisor encounters a violation, a verbal description of the violation is given together with the C-source code for the test that was violated. The C-source code is expressed in terms of the mission configuration parameters and can be used to figure out how to correct the problem.

3.1.1 Missions impossible.

Constraint violations represent missions that are not possible or else potentially fatal to a deployed float. The Configuration Supervisor will reject mission configurations that violate constraints. The following is a description of each constraint:
Range constraints are applied to most mission parameters:

0	≤	Verbosity	≤	5
1 Count	≤	AirBladderMaxP	≤	240 Counts
1 Count	≤	OkVacuumCount	≤	254 Counts
1 Count	≤	PistonBuoyancyNudge	≤	254 Counts
1 Count	≤	DeepProfilePistonPos	≤	254 Counts
1 Count	≤	PistonFullExtension	≤	254 Counts
1 Count	≤	PistonInitialBuoyancyNudge	≤	254 Counts
0 Count	≤	PistonParkHyperRetraction	≤	254 Counts
1 Count	≤	ParkPistonPos	≤	254 Counts
1 Count	≤	PistonStoragePosition	≤	254 Counts
1	≤	PnPCycleLen	≤	254
0 Decibars	≤	ParkPressure	≤	2000 Decibars
0 Decibars	≤	DeepProfilePressure	≤	2000 Decibars
0 Minute	≤	DeepProfileDescentTime	≤	8 Hours
0 Minute	<	DownTime	≤	30 Days
1 Minute	≤	AscentTimeOut	≤	10 Hours
0 Minute	<	ParkDescentTime	≤	8 Hours
0 Minute	<	TimePrelude	≤	6 Hours
1 Minute	≤	TelemetryRetry	≤	6 Hours
0 Minutes	<	UpTime	≤	24 Hours
30 Seconds	≤	ConnectTimeOut	≤	5 Minutes

The up-time must allow for a deep profile plus 2 hours for telemetry:
$mission.TimeUp >= (mission.PressureProfile/dPdt) + 2 * Hour$

The up-time must allow for a park profile plus 2 hours for telemetry:
$mission.TimeUp >= (mission.PressurePark/dPdt) + 2 * Hour$

The up-time has to be greater than the ascent time-out period:
$mission.TimeUp > mission.TimeOutAscent$

The down-time has to be greater than the park-descent time plus deep-profile descent time:
$mission.TimeDown > mission.TimeParkDescent + mission.TimeDeepProfileDescent$

The profile pressure must be greater than (or equal to) the park pressure:
$mission.PressureProfile >= mission.PressurePark$

The primary dial command must begin with AT:
!strncmp(mission.at, AT,2)

The alternate dial command must begin with AT:
!strncmp(mission.alt,AT,2)

3.1.2 Missions insane.

The Configuration Supervisor will warn the operator about violations of sanity checks but will not reject the configuration. The following is a list of each sanity check:

SBE41CP not designed for spot-sampling the main thermocline - CP mode recommended.
$mission.PressureCP >= 750$

Ascent time should be sufficient for a deep profile:
$mission.TimeOutAscent >= (mission.PressureProfile/dPdt) + 1 * Hour$

Ascent time should be sufficient for a park profile:
$mission.TimeOutAscent >= (mission.PressurePark/dPdt) + 1 * Hour$

Up time should be sufficient to guarantee at least 2 hours for telemetry:
$mission.TimeUp >= mission.TimeOutAscent + 2 * Hour$

Park descent period should be compatible with park pressure:
$mission.TimeParkDescent >= mission.PressurePark/dPdt$

Park descent period should not be excessive:
$mission.TimeParkDescent <= 1.5 * mission.PressurePark/dPdt + 1 * Hour$

Deep-profile descent period should be compatible with profile pressure:
$mission.TimeDeepProfileDescent >= (mission.PressureProfile - mission.PressurePark)/dPdt)$

Down time should be sufficient for active ballasting algorithm to adjust buoyancy:
$mission.TimeDown > mission.TimeDeepProfileDescent + mission.TimeParkDescent + 2 * Hour$

Deep-profile descent period should not be excessive:
$mission.TimeDeepProfileDescent <= 1.5 * (mission.PressureProfile - mission.PressurePark)/dPdt + Hour$

Profile piston position should be compatible with park piston position:
$mission.PistonDeepProfilePosition <= mission.PistonParkPosition$

Compensator instability: N2 float should be parked deeper. This sanity check applies only if the compensator hyper-retraction is nonzero.
$mission.PressurePark >= MinN2ParkPressure$

Maximum air-bladder pressure seems insane:
$120 \leq mission.MaxAirBladder \leq 128$

The float serial number should be greater than zero:
$mission.FloatId > 0$

4 Remote control (a.k.a 2-way commands).

The ability to accomplish 2-way communication and remote control via the Iridium system was the major motivator for implementing remotely configurable operation.

WARNING: Remote control of Iridium floats is a new and advanced feature that requires a careful and knowledgeable operator. For example, it is quite possible to send the float remote commands that will render it incapable of re-establishing a communications session with the remote host. Without physical possession of the float, this condition is not recoverable and therefore the float would be effectively killed. The *Configuration Supervisor* (see Section 3.1) attempts to protect against some common classes of misconfigurations. However, there is no substitute for a careful, knowledgeable, prudent, and conservative operator. Furthermore, **it is my advice that new mission configurations should always first be tried on a laboratory simulator before being applied to a float in the field**.

Remote control of the mission is accomplished by creating a configuration file, "mission.cfg", on the float's remote-host computer. The name of the configuration file can not be changed and the syntax of configuration files is tightly controlled and accomplished through the use of "configurators". A lexical analyzer is implemented in firmware to parse the configuration file and install the configurator arguments as the float's new mission configuration.

Strict syntax rules are rigidly enforced as a protective measure against accidental and perhaps fatal misconfiguration. Every line in the configuration file must be either a blank line (ie., all white space), a comment (first non-whitespace character must be '#'), or a well-formed configurator. Configurators have a fixed syntax:

```
ParameterName(argument) [CRC]
```

where ParameterName satisfies the regex "[a-zA-Z0-9]{1,31}" (ie., maximum of 31 characters), argument satisfies the regex ".*", and the [CRC] field is optional (but strongly encouraged) and, if present, must satisfy the regex "[(0x[0-9a-fA-F]{1,4})]". That is, the opening and closing brackets are literal characters "[]" that bracket a string that represents a 4-16 bit hexidecimal number. If the CRC field is present then it represents the 16-bit CRC of the configurator: "ParameterName(argument)". The CRC of the configurator is computed and checked against the CRC specified in the configurator.

The CRCs must have the same value or else the configuration attempt fails. The CRC is generated by the CCITT polynomial[2].

It is important to note that any white space in the argument is treated as potentially significant. Every byte (including white space) between the parentheses is considered to be a non-negligible part of the argument. In cases where the argument string is converted to a number then the presence of extraneous white space won't matter. However, if the argument represents, say, a login name or a passord then extraneous space would be fatal.

Only one configurator per line is allowed and the configurator must be the left-most text on the line except that it can be preceeded by an arbitrary amount of whitespace. No text, except for an arbitrary amount of white space, is allowed to the right of the rightmost closing parenthesis. The maximum length of a line (including white space) is 126 bytes and the maximum length of the ParameterName is 31 bytes.

The order that configurators are given in the file does not matter except that if configurators are repeated then only the last one is relevant. The ParameterName of the configurator is not case sensitive. However, the argument is potentially case sensitive as, for example, a user name or password.

If any syntax error is detected in the configuration file or if the argument of a configurator fails a range check then the configuration attempt fails in its entirety. In this case then the new configuration attempt is completely disregarded and the previous configuration remains active.

Configurators virtually never come in complete sets. It is normal to adjust the values of some mission parameters while leaving others unchanged. However, certain mission parameters interact with each other and it is very important that the float operator understand the details of these interactions because float operation can be significantly affected. Moreover, be mindful that the float itself will change the values of some mission parameters (eg., park piston position, deep profile piston position) during the course of the mission.

The following is an example of a valid configuration file, **mission.cfg**:

```
# Activate continuous profiling at 1000dbars (spot sampling in deep water).
CpActivationP(1000)     [0xF2CC]

# Allow 5 hours to descend from the surface to the park pressure.
ParkDescentTime(300)    [0xB880]

# Set the park pressure to be 1000dbars.
ParkPressure(1000)      [0x899C]

# Set the park-n-profile cycle length to 4.
PnPCycleLen(4)          [0x2825]

# Set the down-time to 5 days
DownTime(7200)          [0xBC7F]
```

A description of each configurator follows:

---

[2] Ssee the comment section of the C source file, “crc16bit.c”, for details of the CRC generator.

ActivateRecoveryMode() —This configurator induces the float into recovery mode and initiate telemetry at regular intervals given by the telemetry retry period. This configurator requires no argument.

AirBladderMaxP(Counts) —The cut-off pressure (in A/D counts) for air-bladder inflation. The air pump will be deactivated when the air bladder pressure exceeds the cut-off. The valid range of the argument is 1-240 counts. This configurator is for disaster recovery only and should rarely be necessary.

AscentTimeOut(Minutes) —The initial segment of the uptime that is designated for profiling and vertical ascent. If the surface has not been detected by the time this timeout expires then the profile will be aborted and the telemetry phase will begin. The valid range of the argument is 1 minute to 10 hours.

AtDialCmd() —The modem AT dialstring used to connect to the primary host computer. Be sure to include "ATDT" as the leading part of the string. Changing both AtDialCmd() and AltDialCmd() at the same time is dangerous and strongly discouraged.

AltDialCmd() —The modem AT dialstring used to connect to the alternate host computer. Be sure to include "ATDT" as the leading part of the string. Changing both AtDialCmd() and AltDialCmd() at the same time is dangerous and strongly discouraged.

BuoyancyNudge(Counts) —The piston extension (counts) applied each time the ascent rate falls below the user-specified minimum. This adds buoyancy to the float in order to maintain the ascent rate.

ConnectTimeOut(Seconds) —The number of seconds allowed after dialing for a connection to be established with the host computer. The valid range of the argument is 30-300 sec.

CpActivationP(Decibars) —The pressure where the Apf9i firmware transitions from subsampling the water column (in the deep water) to where the continuous profiling mode of the SBE41CP is activated for a high resolution profile to the surface.

WARNING: The SBE41CP is not designed for subsampling in the presence of significant temperature gradients. The pump period for spot samples is insufficient to drive thermal mass errors down to an acceptable level and will result in degraded hydrographic data. An activation pressure deeper than the main thermocline is strongly encouraged. An minimum activation pressure of 1000 dbar is recommended and a sanity-check violation will be encountered if the activation pressure is less than 750 dbars.

DeepProfileDescentTime(Minutes) —This time determines the maximum amount of time allowed for the float to descent from the park pressure to the deep profile pressure. The deep profile is initiated when the deep profile descent time expires or else the float reaches the deep profile pressure, whichever occurs first. The valid range of the argument is 0-8 hours.

DeepProfilePistonPos(Counts) —The Apf9i firmware retracts the piston to the deep profile piston position in order to descend from the park pressure to the deep profile pressure. The deep profile piston position should be set so that the float can reach the deep profile pressure before the deep profile descent period expires. The valid range of the argument is 1-254 counts.

DeepProfilePressure(Decibars) —This is the target pressure for deep profiles. The valid range of the argument is 0-2000 decibars.

CompensatorHyperRetraction(Counts) —Floats with N2-compensators require the piston to be hyper-retracted in order to descend from the surface to the park level. The valid range is 0-254 counts.

DownTime(Minutes) —This determines the length of time that the float drifts at the park pressure before initiating a profile. The valid range of the argument is 1 minute to 30 days.

Note: If the **TimeOfDay** feature is enabled then the length of the whole profile cycle will turn out to be an integral number of days. The user should specify the down-time to be precisely 1 day less than the desired length of the profile cycle. For example, if profiles are to be executed every 10 days then the down-time should be specified to be 9 days (ie., 12960 minutes).

FlashErase() —This command requires no argument and causes the FLASH memory chip to be reformatted. **WARNING:** All contents of the FLASH file system will be destroyed.

FlashCreate() —This command requires no argument and causes the FLASH file system to be reinitialized. This command is time consuming (30 minutes) and energy-expensive. The process involves writing a test pattern to each 8KB block of the FLASH ram and then re-reading the contents to ensure that the test pattern matches what was written. If bad blocks are discovered then they are added to a bad-block list. Blocks identified in the bad block list are not used for storage. **WARNING:** All contents of the FLASH file system will be destroyed.

FloatId() —The 4-digit float identifier. This configurator is for disaster recovery only and should never be necessary.

MaxLogKb(Kilobytes) —The maximum size of the logfile in kilobytes. Once the log grows beyond this size, logging is inhibited and the logfile will be automatically deleted at the start of the next profile. The valid range of the argument is 5-60 kilobytes.

ParkDescentTime(Minutes) —This time determines the maximum amount of time allowed for the float to descent from the surface to the park pressure. The active ballasting phase is initiated when the park descent time expires. The valid range of the argument is 0-8 hours.

ParkPistonPos(Counts) —The Apf9i firmware retracts the piston to the park piston position in order to descend from the surface to the park pressure. The park piston position should be set so that the float will become neutrally buoyant at the park pressure. The valid range of the argument is 1-254 counts.

ParkPressure(Decibars) —This is the target pressure for the active ballasting algorithm during the park phase of the mission cycle. The valid range of the argument is 0-2000 decibars.

PnPCycleLen() —A deep profile is initiated when the internal profile counter is an integral multiple of park-n-profile cycle length. All other profiles will be collected from the park pressure to the surface.

Pwd() —The password used to login to the host computer. This configurator is dangerous and intended for disaster recovery only—its use is strongly discouraged.

TelemetryRetry(Minutes) —This determines the time period between attempts to successfully complete telemetry tasks after each profile. The valid range of the argument is 1 minute to 6 hours.

TimeOfDay(Minutes) —This allows the user to specify that the down-time should expire at a specific time of day (ToD). For example, the ToD feature allows the user to schedule profiles to happen at night.

The ToD is expressed as the number of minutes after midnight (GMT). The valid range is 0-1439 minutes. Any value outside this range will cause the ToD feature to be disabled.

UpTime(Minutes) —This determines the maximum amount time allowed to execute the profile and complete telemetry. The valid range of the argument is 1 minute to 1 day.

User() —The login name on the host computer that the float uses to upload and download data. This configurator is dangerous and intended for disaster recovery only—its use is strongly discouraged.

Verbosity() —An integer in the range [0,4] that determines the logging verbosity with higher values producing more verbose logging. A verbosity of 2 yields standard logging. Increased verbosity will probably require increased logging capacity via the **MaxLogKb()** configurator.

4.1 The (linux) *chkconfig* utility.

The ocean is very skilled at finding and exploiting the weaknesses of both the float and its operator. The remote control feature offers new and useful applications for floats but it also necessarily introduces new weaknesses.

One particularly worrisome weakness is the potential for accidental misconfiguration of a float via 2-way commands as described in Secion 4. The *chkconfig* utility helps to protect against common kinds of misconfigurations by subjecting a mission configuration file to the *Configuration Supervisor* (see Section 3.1). The *chkconfig* utility reads a proposed mission configuration file and merges its parameters with the float's existing configuration. The merged configuration is then subjected to the *Configuration Supervisor* to determine if the merged configuration is valid.

For example, suppose that the float's current configuration is represented by the configurators in **mission.current**:

```
AscentTimeOut(540)
AtDialCmd(ATDT0012066859312)
AtDialCmd(ATDT0012066163256)
BuoyancyNudge(5)
CompensatorHyperRetraction(0)
ConnectTimeOut(60)
CpActivationP(1000)
DeepProfileDescentTime(300)
DeepProfilePistonPos(16)
DeepProfilePressure(2000)
DownTime(1440)
InitialBuoyancyNudge(22)
MaxLogKb(40)
ParkDescentTime(300)
ParkPistonPos(24)
```

```
ParkPressure(1000)
PnPCycleLen(1)
TelemetryRetry(15)
TimeOfDay(-1)
UpTime(660)
Verbosity(2)
```

Next suppose that the mission is to be configured for rapid cycling by applying a single configurator in **mission.cfg**:

```
# Configure the down-time for 8 hours
DownTime(480) [0x2493]
```

To check the validity of the proposed configuration for rapid cycling, execute the command:

```
chkconfig if=mission.cfg cfg=mission.current
```

```
(Apr 14 2006 22:17:37) chkconfig            Validating the float's current configuration.
[snippage]
(Apr 14 2006 22:17:37) chkconfig            The float's current configuration is accepted.
(Apr 14 2006 22:17:37) chkconfig            Validating the float's new configuration.
(Apr 14 2006 22:17:37) configure()          Parsing configurators in "mission.cfg".
(Apr 14 2006 22:17:37) configure()             DownTime(480) [0x605A] [DownTime(480)].
(Apr 14 2006 22:17:37) configure()          Configuration CRCs and syntax OK.
(Apr 14 2006 22:17:37) ConfigSupervisor()   Constraint violated: cfg->TimeDown > cfg->TimeParkDescent+cfg->TimeDeepProfileDescent
(Apr 14 2006 22:17:37) ConfigSupervisor()   Sanity check violated: cfg->TimeDown > cfg->TimeDeepProfileDescent + cfg->TimeParkDescent + 2*Hour
(Apr 14 2006 22:17:37) ConfigSupervisor()   Configuration rejected.
(Apr 14 2006 22:17:37) configure()          Configuration rejected by configuration supervisor.
(Apr 14 2006 22:17:37) chkconfig            Configuration file invalid.
```

The *Configuration Supervisor* detected violations of one constraint and one sanity check—the proposed configuration is rejected on the basis of the constraint violation.

The constraint violation indicates that the down-time must be (strictly) greater than the park descent period plus the deep-profile descent period. The definition of the sequence points in Section 2.2 requires that the down-time includes the park-descent phase, the park phase, and the deep-descent phase. Since 480 minutes of down-time does not allow for 300 minutes for each of the two descent periods then the proposed configuration is an example of an impossible mission.

If the down-time is lengthed to 601 minutes to make the mission possible then the *chkconfig* command responds with

```
(Apr 14 2006 22:54:22) chkconfig            Validating the float's current configuration.
[snippage]
(Apr 14 2006 22:54:22) chkconfig            The float's current configuration is accepted.
(Apr 14 2006 22:54:22) chkconfig            Validating the float's new configuration.
(Apr 14 2006 22:54:22) configure()          Parsing configurators in "mission.cfg".
(Apr 14 2006 22:54:22) configure()             DownTime(601) [CRC=0x17A2] [DownTime(601)].
(Apr 14 2006 22:54:22) configure()          Configuration CRCs and syntax OK.
(Apr 14 2006 22:54:22) ConfigSupervisor()   Sanity check violated: cfg->TimeDown > cfg->TimeDeepProfileDescent + cfg->TimeParkDescent + 2*Hour
(Apr 14 2006 22:54:22) ConfigSupervisor()   Configuration accepted.
[snippage]
(Apr 14 2006 22:54:22) ../bin/chkconfig     Configuration file OK.
```

This fixed the constraint violation but the *Configuration Supervisor* still warns of a sanity check violation. The sanity check indicates that the proposed configuration does not allow sufficient time for the active ballasting mechanism to make any buoyancy adjustments. This condition is not likely to be fatal to the float. However, the float will not likely to be able to perform the intended mission because the active ballasting mechanism will not drive the float to the programmed park pressure.

4.2 Group-wise or fleet-wise remote control.

As an advanced technique, it is possible to write configurations suitable for uniform control of groups or fleets of floats. Such techniques facilitate some kinds of field experiments while obviously limiting some kinds of flexibility or individualization. This technique is still experimental and beyond the scope of this manual (for now). Contact the author for more details.

5 Recovery mode.

As its name suggests, recovery mode is intended primarily to facilitate post-deployment recovery of the float from the ocean. However, its operational behaviors are general enough to allow for many other useful applications, too. Recovery mode is fundamentally a remote control feature. Section 4 describes the facilility for remote control of Iridium floats using 2-way commands. The **ActivateRecoveryMode()** command is used to both initiate and maintain recovery mode for as long as the operator desires.

The recovery mode cycle operates on the telemetry-retry period. Each cycle starts by ensuring that the piston is fully extended and the air bladder is fully inflated. Then a GPS fix is obtained and telemetry is initiated to upload the GPS fix and a small amount of engineering data to the remote host. The pathname for the file has the pattern *FloatId.YYMMDDhhmm* where *FloatId* is the 4-digit float identifier and *YYMMDDhhmm* represents the date & time when the recovery cycle was initiated. It is a simple matter to arrange for the remote host to automatically relay the GPS fix to an Iridium handset on-board the ship. This allows recovery operations to be conducted without on-shore aid.

The new mission configuration file will also be downloaded from the remote host. If the new mission configuration file contains the **ActivateRecoveryMode()** command then the float will go to sleep for one telemetry-retry period and then wake up to repeat the recovery mode cycle. If the new mission configuration file does not contain the **ActivateRecoveryMode()** command then subsequent float behavior depends upon what the float was doing before recovery mode was initiated. If the float was in its mission prelude then the mission prelude is re-initiated[3].

On the other hand, if the float's mission was in progress when recovery mode was initiated then upon termination of recovery mode the mission resumes where it left off. That is, if profile *N* had been telemetered just prior to initiation of recovery mode then the descent phase of profile cycle *N+1* will begin as soon as recovery mode is terminated.

6 Telemetered data.

Iridium floats telemeter two kinds of data for each profile cycle and these data are transferred as separate files: message files and log files. Message files contain hydrographic data and follow a naming convention *FloatId.ProfileId*.msg where *FloatId* is the 4-digit serial number of the float controller and *ProfileId* is the 3-digit profile counter. Log files contain detailed engineering data

---

[3] The mission prelude is not merely continued where it left off when recovery mode was activated; the mission prelude is completely restarted

with time-stamped diagnostics of float operations. Examples of message files and log files from actual floats can be found in Appendixes A and C. It will be helpful to refer to these examples as you read this section.

6.1 Format specification for APF9i firmware.

This section summarizes the format specification for hydrographic data telemetered by Iridium floats. The “official” format specification can be found in the “src” directory of your distribution. Any discrepancy between **src/FormatNotes** and the information in this manual should be resolved in favor of the former.

Iridium message files end with a ”.msg” extension. Each iridium message file consists of blocks of similar data presented in the order that they were collected during the profile cycle. This firmware revision includes five blocks of data:

1. Park-phase PT samples: These are hourly low-power PT samples collected during the park phase of the profile cycle.
2. Low resolution PTS samples: The deep parts of the profile can be represented using low-resolution spot samples collected at predetermined pressures. Low resolution spot sampling in the deep water was implemented as an energy savings measure.
3. High resolution PTS samples: The shallower parts of the profile can be represented with high resolution (ie., 2 decibar) bin-averaged PTS samples. In continuous profiling mode, the CTD samples at 1Hz and stores the data for later binning and averaging.
4. GPS fixes: After the profile is executed and the float reaches the surface, the location of the profile is determined via GPS.
5. Biographical and engineering data: Various kinds of biographical and engineering data are collected at various times during the profile cycle.

Usually, only one telemetry cycle is required to upload the data to the remote host computer. However, sometimes the iridium connection is broken or the quality of the connection is so poor that the float will abort the telemetry attempt, wait a few minutes, and then try again. Data blocks 4 and 5 will be repeated for each telemetry cycle of a given profile.

A description of the format for each of these blocks of data follows. A sample Iridium-message file is available in Appendix A.

6.1.1 Format for park-phase PT samples.

Hourly low-power PT samples are collected during the park phase of the profile cycle. The park phase is also when active ballasting is done. Each sample includes the date and time of the sample, the unix epoch (ie., the number of seconds since 00:00:00 on Jan 1, 1970), the mission time (ie., the number of seconds since the start of the current profile cycle), the pressure (decibars), and the temperature (°C). For example:

```
           |------- date -----|   UnixEpoch   MTime       P       T
ParkPt:    Jul 03 2006 18:37:34  1151951854   14414  988.18  7.4971
ParkPt:    Jul 03 2006 19:37:31  1151955451   18011  992.15  7.3613
ParkPt:    Jul 03 2006 20:37:31  1151959051   21611  998.23  7.3428
ParkPt:    Jul 03 2006 21:37:31  1151962651   25211 1000.38  7.2806
ParkPt:    Jul 03 2006 22:37:31  1151966251   28811 1003.01  7.2844
```

6.1.2 Format for low resolution PTS samples.

The SBE41CP that is used on iridium floats has features that enable subsampling of the water column (similar to the SBE41) as well as the ability to bin-average a continuous sampling of the water column. For subsampled data, the values of pressure, temperature, and salinity are not encoded but are given in conventional units (decibars, °C, PSU). For example:

```
$ Discrete samples: 6
$       p        t        s
  1002.59    3.912  34.4573 (Park Sample)
  1000.11    3.929  34.4547
   947.36    4.035  34.4454
   897.56    4.163  34.4303
   847.54    4.344  34.4104
   798.09    4.478  34.3934
```

6.1.3 Format for high resolution PTS samples.

For conintuously sampled data, 2-decibar bins are used for bin-averaging. These data are encoded as three 16-bit integers (PTS) and then an 8-bit integer that represents the number of samples in the bin:

```
# Nov 05 2006 23:38:59 Sbe41cpSerNo[1520] NSample[11134] NBin[495]
00000000000000[2]
002B5CE0885115
003C5CE188511F
00505CE088511E
00645CE088511D
00785CDF88521C
008C5CDF88521B
00A05CDF88511A
00B45CDC88511A
00C85CAD885A1A
00DC5C8D886418
00F05C88886919
01045C88886A19
01185C74887618
```

```
012C5C4E88BB15
[snippage...]
2684144D874814
2698145C874D14
```

The first 4-bytes of the encoded sample represents the pressure in centibars. The second 4-bytes represents the temperature in millidegrees. The third 4-bytes represent the salinity in parts per million. The final 2-bytes represent the number of samples collected in the 2dbar pressure bin.

For example, the encoding: 2698145C874D14 represents a bin with (0x14=) 20 samples where the mean pressure was (0x2698=) 988.0dbars, the mean temperature was (0x0EDE=) 3.806C, and the mean salinity was (0x86A0=) 34.464PSU. The PTS values were encoded as 16-bit hex integers according to the C-source code found in Appendix D.

Integers in square brackets '[]' indicate replicates of the same encoded line. For example, a line that looks like: 00000000000000[2] indicates that there were 2 adjacent lines with the same encoding....all zeros in this case.

6.1.4 Format for GPS fixes.

Each telemetry cycle begins with the float attempting to acquire a GPS fix. The fix includes the amount of time required to acquire the fix, the longitude and latitude (degrees), the date and time of the fix, and the number of satellites used to determine the fix. For example:

```
# GPS fix obtained in 98 seconds.
#         lon     lat mm/dd/yyyy hhmmss nsat
Fix: -152.945  22.544 09/01/2005 104710    8
```

Positive values of longitude, latitude represent east, north hemispheres, respectively. Negative values of longitude, latitude represent west, south hemispheres, respectively. The date is given in month-day-year format and the time is given in hours-minutes-seconds format.

If no fix was acquired then the following note is entered into the iridium message:

```
# Attempt to get GPS fix failed after 600 seconds.
```

6.1.5 Format for biographical and engineering data.

These data have the format, "key"="value", as shown in the following examples:

```
ActiveBallastAdjustments=5
AirBladderPressure=119
AirPumpAmps=91
```

```
AirPumpVolts=192
BuoyancyPumpOnTime=1539
```

Interpretation of these data requires detailed knowledge of firmware implementations and is generally beyond the scope of this manual. However, there are two 16-bit status-words (status, Sbe41cpStatus) whose asserted bits indicate special conditions as described below

The definition of each bit mask for the 16-bit "status" word are given in the table below:

`DeepPrf`	`0x0001`	`The current profile is a deep profile.`
`ShallowWaterTrap`	`0x0002`	`Shallow water trap detected`
`Obs25Min`	`0x0004`	`Sample time-out (25 min) expired.`
`PistonFullExt`	`0x0008`	`Piston fully extended before surface-detection algorithm terminated.`
`AscentTimeOut`	`0x0010`	`Ascent time-out expired.`
`DownLoadCfg`	`0x0020`	`The mission.cfg file has not been downloaded yet.`
`BadSeqPnt`	`0x0100`	`Invalid sequence point detected.`
`Sbe41cpPFail`	`0x0200`	`Sbe41cp(P) exception.`
`Sbe41cpPtFail`	`0x0400`	`Sbe41cp(PT) exception.`
`Sbe41cpPtsFail`	`0x0800`	`Sbe41cp(PTS) exception.`
`Sbe41cpPUnreliable`	`0x1000`	`Sbe41cp(P) unreliable.`
`AirSysLimit`	`0x2000`	`Pneumatic inflation limits imposed; excessive energy consumption.`
`WatchDogAlarm`	`0x4000`	`Wake-up by watchdog alarm.`
`PrfIdOverflow`	`0x8000`	`The 8-bit profile counter overflowed.`

The definition of each bit mask for the 16-bit "Sbe41cpStatus" word (Pneumonic: SBE41) are given in the table below:

`Sbe41cpPedanticExceptn`	`0x0001`	`An exception was detected while parsing the p-only pedantic regex.`
`Sbe41cpPedanticFail`	`0x0002`	`The SBE41 response to p-only measurement failed the pedantic regex.`
`Sbe41cpRegexFail`	`0x0004`	`The SBE41 response to p-only measurement failed the nonpedantic regex.`
`Sbe41cpNullArg`	`0x0008`	`NULL argument detected during p-only measurement.`
`Sbe41cpRegExceptn`	`0x0010`	`An exception was detected while parsing the p-only nonpedantic regex.`
`Sbe41cpNoResponse`	`0x0020`	`No response detected from SBE41 for p-only request.`
	`0x0040`	`Not used yet.`
	`0x0080`	`Not used yet.`
`Sbe41cpPedanticExceptn`	`0x0100`	`An exception was detected while parsing the pts pedantic regex.`
`Sbe41cpPedanticFail`	`0x0200`	`The SBE41 response to pts measurement failed the pedantic regex.`
`Sbe41cpRegexFail`	`0x0400`	`The SBE41 response to pts measurement failed the nonpedantic regex.`
`Sbe41cpNullArg`	`0x0800`	`NULL argument detected during  pts measurement.`
`Sbe41cpRegExceptn`	`0x1000`	`An exception was detected while parsing the pts nonpedantic regex.`
`Sbe41cpNoResponse`	`0x2000`	`No response detected from SBE41 for pts request.`
	`0x4000`	`Not used yet.`
	`0x8000`	`Not used yet.`

6.2 Engineering log files.

The engineering log files contain time-stamped entries of what the float was doing at any given time. Every nook and cranny of the float firmware has self-monitoring features built in that are a synthesis of self-adaptive and user-controlled behaviors. The self-adaptive nature stems from the fact that if the float firmware detects problems or difficulties then engineering log entries are automatically generated as an aid to on-shore diagnostics. The user-controlled nature stems from the fact that the user can remotely adjust the verbosity of the engineering logs using the 2-way **Verbosity** command. Refer to Section 4 for information about 2-way (ie., remote) float configuration.

7 Processed data.

Decoding and processing the message files from this Iridium implementation is relatively easy because the data are ASCII and mostly self-describing. Only the high resolution hydrographic data are encoded although some of the engineering data must be processed through calibration equations. Appendix D contains the C source code used to encode the high resolution data and Appendix B contains an example profile after decoding and processing has been applied.

8 The remote UNIX host.

This Iridium implementation uses a modem-to-modem communications model. The float initiates a telephone call to a remote host computer, logs into the remote host with a username and password, executes a sequence of commands to transfer data, and then logs out. The communications session is float-driven

With respect to the remote host, there is no difference between the float logging in and a human logging in. The communications session is initiated and fully controlled by the float. On the other hand, the float is not naturally adaptable or interactive like a human would be and so an unusual amount of fault tolerance has been built into both sides of the communications session.

An important fault tolerance measure is redundancy in the form of two similarly configured remote hosts each with its own dedicated telephone line. This is optional but recommended. Ideally, these two remote hosts should be separated far enough from each other that power outages or telephone outages are not likely to simultaneously affect both remote hosts. The float firmware is designed to automatically switch to the alternate remote host if with the primary remote host appears to be out of service.

8.1 System requirements.

This Iridium implementation is strongly tied to the use of a UNIX computer as the remote host (ie., Microsoft operating systems are not suitable). **The most important "system requirement" is a system administrator that is familiar, comfortable, and competent in a UNIX environment.** While many different flavors of UNIX could be made to work, development was done using RedHat Linux (versions 7-9). RedHat Linux (version 9) will be assumed for the remainder of this section.

The **mgetty** package must be installed and configured to monitor a Hayes-compatible external modem attached to one of the serial ports. For information on how to install and configure the **mgetty** package, refer to the **mgetty** documentation supplied with RedHat Linux. If you customize the login prompt, make sure that it includes the phrase “login:”. Similarly, make sure that the password prompt includes the phrase “Password:”. The float will not successfully log in if these two phrases are not present.

Once **mgetty** is installed and configured properly, you should be able to log into the remote host via a modem-to-modem connection from another computer. You should test this using the following communications parameters: 4800baud, no parity, 8-bit data, 1 stop-bit.

8.2 Remote host set up.

Once each telemetry cycle, the float downloads “mission.cfg” from the home directory where the float logs in and this new mission configuration becomes active as the last step before the telemetry cycle terminates (see Section 4). In the context of a UNIX environment, this simple mechanism allows for great flexibility for remotely controlling floats individually, in groups, or fleet-wise. It is also flexible in that it is possible to switch which model is used even after floats have been deployed. Finally, a UNIX-based remote host facilitates easy speciation of floats as well as for new float developments with no requirement for backward compatibility.

8.2.1 Setting up the *default user* on the remote host.

Another fault tolerance measure requires creation of a *default user* on the remote host. Begin by creating a new *iridium* group to which the *default user* and all floats will belong. As root, execute the command:

```
groupadd -g1000 iridium
```

Next, create an account for the *default user* using *iridium* as the username:

```
adduser -s/bin/tcsh -c”Iridium Apex Drifter” -g”iridium” -u1000 -d/home/iridium iridium
```

Then give the new user a password by executing (as root):

```
passwd iridium
```

For the convenience of the float manager, you might also want to change the permissions on the float’s home directory:

```
chmod 750 ˜iridium.
```

The file, */etc/passwd*, will contain the following entry:

```
iridium:x:1000:1000:Iridium Apex Drifter:/home/iridium:/bin/tcsh
```

The remainder of the set-up for this float should be done while logged into the remote host as the *default user* (ie., *iridium*). Create two directories:

```
mkdir ~/bin ~/logs
```

and populate the *~/bin* directory with the *SwiftWare* xmodem utilities **rx** and **sx** as well as the **chkconfig** utility. These three files are in the **support** directory of your distribution.

Finally, use **emacs** to create the following three ascii files: **.cshrc**, **.rxrc**, and **.sxrc**:

.cshrc: This file configures the t-shell at login time. You can modify the configuration to suit yourself so long as your customizations do not interfere with the effects that the three commands below have. In particular, it is important that the float's **bin** directory be in the *path* before any of the system directories. This will ensure that the float's version of the utilities **chkconfig**, **rx**, and **sx** will be used rather than the system's utilities with these same names.

```
# set the hostname
set hostname=‘hostname‘

# add directories for local commands
set path = (. ~/bin /bin /sbin /usr/sbin /usr/local/bin)

# set the prompt
set prompt=""$hostname":[$cwd]> "
```

.rxrc: This is the configuration file for the *SwiftWare* implementation the xmodem receive utility. *SwiftWare* **rx** implements the standard xmodem protocol except that a nonstandard 16-bit CRC is used. Beware that the float will not be able to transfer any hydrographic or engineering data to the remote host using the system version of **rx**. Make sure that the *LogPath* references the *default user's* **logs** directory or else potentially valuable logging/debugging information will be irretrievably lost.

```
# This is the configuration file for ’rx’, the
# SwiftWare xmodem receive utility.

# set the default debug level (range: 0-4)
Verbosity=5

# specify the name of the log file
LogPath=/home/iridium/logs/rxlog

# enable (AutoLog!=0) or disable (AutoLog==0) the auto-log feature
AutoLog=1
```

```
# specify ascii mode (BinaryMode==0) or binary mode (BinaryMode!=0)
BinaryMode=0

# specify CRC mode (16bit or 8bit)
CrcMode=16bit
```

.sxrc: This is the configuration file for the *SwiftWare* implementation the xmodem send utility. *SwiftWare* **sx** implements the standard xmodem protocol except that a nonstandard 16-bit CRC is used. Beware that new mission configurations will not be downloaded from the remote host to the float if system version of **sx** is used. Make sure that the *LogPath* references the *default user's* **logs** directory or else potentially valuable logging/debugging information will be irretrievably lost.

```
# This is the configuration file for 'sx', the
# SwiftWare xmodem send utility.

# set the default debug level (range: 0-4)
Verbosity=5

# specify the name of the log file
LogPath=/home/iridium/logs/sxlog

# enable (AutoLog!=0) or disable (AutoLog==0) the auto-log feature
AutoLog=1

# specify fixed packet type (128b or 1k)
# PktType=1k
```

8.2.2 Setting up the remote host for individualized remote control.

The ability to individualize each float is implemented by each float having its own account on the remote host. The steps to set up the remote host are analagous to those for setting up the *default user* (see Section 8.2.1). For example, to create an account for float 5047 then make sure the **iridium** group exists (see Section 8.2.1) and then execute the following command (as root):

adduser -s/bin/tcsh -c"Iridium Apex Drifter" -g"iridium" -u15047 -d/home/f5047 f5047

Then give the new user a password and change the permissions of the float's home directory as shown for the *default user*. **Be sure to configure the float to use this username and password** (see Section 3). The file, **/etc/passwd**, will contain the following entry:

```
f5047:x:15047:1000:Iridium Apex Drifter:/home/f5047:/bin/tcsh
```

The remainder of the set-up for this float follows very closely that of the *default user* and should be done while logged into the remote host as the float (ie., *f5047*). Create **bin** and **logs** directories in the float's home directory and populate the **bin** directory with the *SwiftWare* xmodem utilities **rx** and **sx** as well as the **chkconfig** utility.

Finally, copy the three ascii files **.cshrc**, **.rxrc**, and **.sxrc** from the *default user's* home directory to the float's home directory. Be sure to edit these files so that the *LogPath* points to the float's **logs** directory or else potentially valuable logging/debugging information will be irretrievably lost.

8.2.3 Setting up the remote host for fleet-wise remote control.

The flexibility inherent with individualized float control necessarily increases the level of operational management required—each float has to be considered and controlled individually. However, fleet-wise management of floats is also made possible by configuring the float to use a fleet-wise username. This is in contrast to Section 8.2.2 where each float was configured with a unique username (based on the float serial number). The steps to set up the remote host for fleet-wise control are virtually the same as those in Sections 8.2.1 & 8.2.2 except that the username and password are fleet-wise parameters. Be sure to configure each float in the fleet with the fleet-wise username and password.

A Sample: Iridium message file.

The following is an example Iridium-message file (5135.009.msg) from profile 9 of UW float 5135. The blocks with park-phase PT data and high resolution PTS data have been snipped to save space. The actual message file is included in the *manual* directory your distribution.

```
ParkPt:    Oct 27 2006 09:12:21  1161940341   21614  991.13  5.0624
ParkPt:    Oct 27 2006 10:12:18  1161943938   25211  997.14  5.0385
ParkPt:    Oct 27 2006 11:12:18  1161947538   28811  994.93  5.0252
[snippage...]
ParkPt:    Nov 05 2006 12:12:18  1162728738  810011  999.88  5.0748
ParkPt:    Nov 05 2006 13:12:18  1162732338  813611  999.87  5.0815
ParkPt:    Nov 05 2006 14:12:18  1162735938  817211 1001.27  5.1691
$ Profile 5135.009 terminated: Sun Nov  5 23:33:42 2006
$ Discrete samples: 22
$       p       t       s
   999.76   5.1389  34.6385 (Park Sample)
  1995.95   2.4262  34.7183
  1947.22   2.5096  34.7160
  1898.18   2.5933  34.7124
  [snippage...]
  1098.41   4.7406  34.6413
  1047.72   4.9210  34.6328
   997.88   5.1765  34.6345
# Nov 05 2006 23:38:59 Sbe41cpSerNo[1520] NSample[11134] NBin[495]
00000000000000[2]
002B5CE0885115
003C5CE188511F
00505CE088511E
00645CE088511D
00785CDF88521C
```

```
008C5CDF88521B
00A05CDF88511A
00B45CDC88511A
[snippage...]
2670144B874714
2684144D874814
2698145C874D14
# GPS fix obtained in 65 seconds.
#         lon     lat mm/dd/yyyy hhmmss nsat
Fix:  106.738 -19.100 11/05/2006 234740    7
Apf9iFwRev=051906
ActiveBallastAdjustments=0
AirBladderPressure=124
AirPumpAmps=69
AirPumpVolts=191
BuoyancyPumpOnTime=1899
BuoyancyPumpAmps=216
BuoyancyPumpVolts=172
CurrentPistonPosition=199
DeepProfilePistonPosition=16
GpsFixTime=65
FloatId=5135
ParkDescentPCnt=7
ParkDescentP[0]=5
ParkDescentP[1]=45
ParkDescentP[2]=78
ParkDescentP[3]=95
ParkDescentP[4]=98
ParkDescentP[5]=99
ParkDescentP[6]=99
ParkPistonPosition=71
ParkObs={ 999.8dbar,    5.139C,  34.6385PSU }
ProfileId=009
ObsIndex=21
QuiescentAmps=6
QuiescentVolts=198
RtcSkew=8
Sbe41cpAmps=12
Sbe41cpVolts=190
Sbe41cpStatus=0x0000
status=0x0001
SurfacePistonPosition=177
SurfacePressure=0.01
Vacuum=77

<EOT>
```

B Sample: Decoded and processed data.

The following is an example of decoded and processed data for profile 9 of UW float 5135. The blocks with park-phase PT data and PTS data have been snipped to save space.

```
$  APEX-Seabird (051906) Iridium Message Parser & Calibration Applicator [SwiftWare]
$  $Revision: 1.2 $  $Date: 2006/11/24 23:58:15 $
$  Cmd Line: /www/argo/bin/ApexSbeIr051906-parser if=5135.009.msg of=5135.009.edf
$
$ E    lat     lon       date   time  zbot  zmax sh co           stnid      n
$ H -19.10  106.74 11/05/2006 23.794     *  1996  *  *        5135.009    515
$
$  Processed engineering data:
$  BatteryVoltage=15.7V  VoltCount=198
$  AirBladderPressure=7.0"Hg  AirBladderPressureCount=124
$  Vacuum=-7.0"Hg  VacuumCount=77
$  BottomPistonPosition=16
$  IAirPump=266mA  AmpCount=69
$  IHighPressurePump=858mA  AmpCount=216
$  IQuiescent=12.4mA  AmpCount=6
$  ISbe41cp=37mA  AmpCount=12
$  VAirPump=15.2V  VoltCount=191
$  VHighPressurePump=13.7V  VoltCount=172
$  VLoad=13.7V  VoltCount=172
$  VQuiescent=15.7V  VoltCount=198
$  VSbe41cp=15.1V  VoltCount=190
$  PumpMotorSeconds=1899 sec
$  DeepProfilePistonPosition=16
$  ParkPistonPosition=71
$  SurfacePistonPosition=177
$  SurfacePressure=0 dbar
$  ProfileTermination=0x0001 : Deep profile.
$  Sbe41cpStatus=0x0000
$
$  Raw engineering meta data:
$  ActiveBallastAdjustments=0
$  AirBladderPressure=124
$  AirPumpAmps=69
$  AirPumpVolts=191
$  Apf9iFwRev=051906
$  BuoyancyPumpAmps=216
$  BuoyancyPumpOnTime=1899
$  BuoyancyPumpVolts=172
$  CurrentPistonPosition=199
$  DeepProfilePistonPosition=16
$  FloatId=5135
$  GpsFixTime=65
$  ObsIndex=21
$  ParkDescentPCnt=7
$  ParkDescentP[0]=5
$  ParkDescentP[1]=45
```

```
$  ParkDescentP[2]=78
$  ParkDescentP[3]=95
$  ParkDescentP[4]=98
$  ParkDescentP[5]=99
$  ParkDescentP[6]=99
$  ParkObs={ 999.8dbar,    5.139C,  34.6385PSU }
$  ParkPistonPosition=71
$  ProfileId=009
$  QuiescentAmps=6
$  QuiescentVolts=198
$  RtcSkew=8
$  Sbe41cpAmps=12
$  Sbe41cpStatus=0x0000
$  Sbe41cpVolts=190
$  SurfacePistonPosition=177
$  SurfacePressure=0.01
$  Vacuum=77
$  status=0x0001
$
$  NFix=1  //      lon     lat Julian-sec       date   hour nsat FixTime
$  Fix(First): 106.738 -19.100 1162799260 11-05-2006 23.794    7      65
$ F %7.2f %7.4f %7.4f %7.4f %7.4f %5.1f %4u
$ T     p        t        s    theta    sigma  speed    n
     4.30  23.7760  34.8970  23.7751  23.6314    9.5   21
     6.00  23.7770  34.8970  23.7757  23.6313    6.5   31
     8.00  23.7760  34.8970  23.7743  23.6317    6.7   30
    10.00  23.7760  34.8970  23.7739  23.6318    6.9   29
    12.00  23.7750  34.8980  23.7725  23.6330    7.1   28
    14.00  23.7750  34.8980  23.7721  23.6331    7.4   27
    16.00  23.7750  34.8970  23.7717  23.6325    7.7   26
    18.00  23.7720  34.8970  23.7682  23.6335    7.7   26
    20.00  23.7250  34.9060  23.7208  23.6542    7.7   26
[snippage...]
   986.00   5.1970  34.6320   5.1133  27.3705   10.0   20
   988.00   5.2120  34.6370   5.1280  27.3727   10.0   20
   997.88   5.1765  34.6345   5.0919  27.3749      *    1
   999.76   5.1389  34.6385   5.0544  27.3825      *    1
  1047.72   4.9210  34.6328   4.8338  27.4034      *    1
  1098.41   4.7406  34.6413   4.6502  27.4308      *    1
[snippage...]
  1898.18   2.5933  34.7124   2.4563  27.7037      *    1
  1947.22   2.5096  34.7160   2.3696  27.7140      *    1
  1995.95   2.4262  34.7183   2.2832  27.7230      *    1
$
$  Park-phase PT time-series: 222 samples
$                 UnixEpoch  MTime       p       t  |-------- Date --------|
$  ParkPt [001]:  1161940341  21614   991.1  5.0624  Fri Oct 27 02:12:21 2006
$  ParkPt [002]:  1161943938  25211   997.1  5.0385  Fri Oct 27 03:12:18 2006
$  ParkPt [003]:  1161947538  28811   994.9  5.0252  Fri Oct 27 04:12:18 2006
$  ParkPt [004]:  1161951138  32411   997.3  5.1130  Fri Oct 27 05:12:18 2006
$  ParkPt [005]:  1161954738  36011   988.9  5.0522  Fri Oct 27 06:12:18 2006
  [snippage...]
```

```
$  ParkPt [219]:  1162725138 806411  1001.6  5.1068  Sun Nov  5 03:12:18 2006
$  ParkPt [220]:  1162728738 810011   999.9  5.0748  Sun Nov  5 04:12:18 2006
$  ParkPt [221]:  1162732338 813611   999.9  5.0815  Sun Nov  5 05:12:18 2006
$  ParkPt [222]:  1162735938 817211  1001.3  5.1691  Sun Nov  5 06:12:18 2006
```

C Sample: Iridium engineering log file.

The following is an example engineering log file (5135.009.log) from profile 9 of UW Iridium float 5135. You will note that the log file starts with entries of the telemetry of profile 8 and then continues with entries of the execution of profile 9. Obviously, engineering data regarding telemetry is collected while telemetry is happening and therefore knowledge of these data aren't completely known until the telemetry has finished. This explains why the engineering telemetry data for profile cycle 8 is included in the engineering logs of profile cycle 9.

```
(Oct 27 2006 02:54:53,  851789 sec) TelemetryInit()      Profile 8. (Apf9i FwRev: 051906)
(Oct 27 2006 03:01:59,  852215 sec) AirSystem()          Battery [186cnt, 14.4V]  Current [78cnt, 31.4mA]  Barometer [122cnt, 5.4"Hg]  Run-Time [51s]
(Oct 27 2006 03:02:32,  852249 sec) GpsServices()        GPS almanac is current.
(Oct 27 2006 03:02:32,  852249 sec) GpsServices()        Initiating GPS fix acquisition.
(Oct 27 2006 03:02:48,  852265 sec) gga()                $GPGGA,030657,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*40
(Oct 27 2006 03:02:58,  852275 sec) gga()                $GPGGA,030707,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*44
(Oct 27 2006 03:03:08,  852285 sec) gga()                $GPGGA,030717,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*45
(Oct 27 2006 03:03:18,  852295 sec) gga()                $GPGGA,030727,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*46
(Oct 27 2006 03:03:28,  852305 sec) gga()                $GPGGA,030737,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*47
(Oct 27 2006 03:03:38,  852315 sec) gga()                $GPGGA,030747,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*40
(Oct 27 2006 03:03:48,  852325 sec) gga()                $GPGGA,030757,1957.2228,S,10647.7051,E,0,00,,,M,,M,,*41
(Oct 27 2006 03:04:13,  852349 sec) gga()                $GPGGA,030340,1932.7209,S,10643.1988,E,1,10,0.9,,M,-30.5,M,,*5B
(Oct 27 2006 03:04:13,  852350 sec) GpsServices()        Profile 8 GPS fix obtained in 101 seconds.
(Oct 27 2006 03:04:13,  852350 sec) GpsServices()                  lon     lat mm/dd/yyyy hhmmss nsat
(Oct 27 2006 03:04:13,  852350 sec) GpsServices()        Fix:  106.720 -19.545 10/27/2006 030340   10
(Oct 27 2006 03:04:41,  852377 sec) gga()                $GPGGA,030410,1932.7103,S,10643.1936,E,1,10,0.9,,M,-30.5,M,,*55
(Oct 27 2006 03:04:46,  852383 sec) gga()                $GPGGA,030440,1932.7006,S,10643.1882,E,1,10,0.9,,M,-30.5,M,,*5A
(Oct 27 2006 03:04:46,  852383 sec) GpsServices()        APF9 RTC skew (6s) OK.
(Oct 27 2006 03:04:47,  852383 sec) LogNmeaSentences()   E,4.3,M,,M,6.5,M*04
(Oct 27 2006 03:04:47,  852383 sec) LogNmeaSentences()   $PGRMT,GPS 15-L Ver. 2.80,P,P,R,R,P,,,R,P*74
(Oct 27 2006 03:04:47,  852384 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Oct 27 2006 03:04:47,  852384 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Oct 27 2006 03:04:56,  852393 sec) LogNmeaSentences()   $GPRMC,030450,A,1932.6972,S,10643.1875,E,001.0,334.7,271006,001.1,W*73
(Oct 27 2006 03:04:57,  852393 sec) LogNmeaSentences()   $GPGGA,030450,1932.6972,S,10643.1875,E,1,10,0.9,,M,-30.5,M,,*58
(Oct 27 2006 03:04:57,  852394 sec) LogNmeaSentences()   $GPGSA,M,2,01,03,07,11,14,19,20,,23,25,31,,1.3,0.9,*14
(Oct 27 2006 03:04:58,  852394 sec) LogNmeaSentences()   $GPGSV,3,3,11,23,11,338,43,25,15,047,44,31,11,054,44*44
(Oct 27 2006 03:04:58,  852394 sec) LogNmeaSentences()   $PGRME,4.3,M,,M,6.5,M*04
(Oct 27 2006 03:04:58,  852395 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Oct 27 2006 03:04:58,  852395 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Oct 27 2006 03:05:06,  852403 sec) LogNmeaSentences()   $GPRMC,030500,A,1932.6954,S,10643.1857,E,000.8,311.3,271006,001.1,W*79
(Oct 27 2006 03:05:07,  852403 sec) LogNmeaSentences()   $GPGGA,030500,1932.6954,S,10643.1857,E,1,10,0.9,,M,-30.5,M,,*58
(Oct 27 2006 03:05:07,  852404 sec) LogNmeaSentences()   $GPGSA,M,2,01,03,07,11,14,19,20,,23,25,31,,1.3,0.9,*14
(Oct 27 2006 03:05:08,  852404 sec) LogNmeaSentences()   $GPGSV,3,1,11,01,64,070,47,03,55,017,46,07,11,032,40,11,33,218,43*75
(Oct 27 2006 03:05:08,  852405 sec) LogNmeaSentences()   $PGRME,4.3,M,,M,6.5,M*04
(Oct 27 2006 03:05:08,  852405 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Oct 27 2006 03:05:09,  852405 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Oct 27 2006 03:05:16,  852413 sec) LogNmeaSentences()   $GPRMC,030510,A,1932.6926,S,10643.1829,E,003.3,332.2,271006,001.1,W*7C
(Oct 27 2006 03:05:17,  852413 sec) LogNmeaSentences()   $GPGGA,030510,1932.6926,S,10643.1829,E,1,10,0.9,,M,-30.5,M,,*55
(Oct 27 2006 03:05:17,  852414 sec) LogNmeaSentences()   $GPGSA,M,2,01,03,07,11,14,19,20,,23,25,31,,1.3,0.9,*14
(Oct 27 2006 03:05:18,  852414 sec) GpsServices()        GPS services complete.
(Oct 27 2006 03:05:42,  852439 sec) CLogin()             Connecting to primary host.
(Oct 27 2006 03:05:56,  852453 sec) CLogin()             Connection 1 established in 14 seconds.
(Oct 27 2006 03:06:11,  852467 sec) login()              Login successful.
(Oct 27 2006 03:06:13,  852470 sec) CLogin()             Logged in to host.  [Login required 17 seconds]
(Oct 27 2006 03:06:18,  852474 sec) RxConfig()           Downloading "mission.cfg" from host.
(Oct 27 2006 03:06:18,  852474 sec) Rx()                 Initiating transfer. [0x43]
(Oct 27 2006 03:06:24,  852480 sec) Rx()                 Truncated packet received - retrying.
(Oct 27 2006 03:06:24,  852480 sec) LogPacket()          0x02 0x01 0xfe  []   0x00 0x00
(Oct 27 2006 03:06:25,  852481 sec) RxStartByte()        Sync errors encountered: 370
(Oct 27 2006 03:06:30,  852487 sec) Rx()                 Pad character [0x1a] found in ascii mode - truncating packet.
(Oct 27 2006 03:06:34,  852490 sec) Rx()                 Received EOT - transfer complete. [1 packets, 1024 bytes, 16 sec, 64.0 bps]
(Oct 27 2006 03:06:34,  852490 sec) RxConfig()           Download successful.
(Oct 27 2006 03:06:34,  852490 sec) WriteVitals()        Writing vitals to "5135.008.msg".
(Oct 27 2006 03:06:38,  852494 sec) UpLoadFile()         Uploading "5135.008.msg" to host as "5135.008.msg".
(Oct 27 2006 03:06:39,  852496 sec) Tx()                 CRC negotiation successful. [16-bit CRC]
(Oct 27 2006 03:06:45,  852502 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:06:46,  852502 sec) TxPacket()           NAK received - resending packet. [PktNum=0x01]
(Oct 27 2006 03:06:46,  852502 sec) TxPacket()           History of packet transmission failures: [01110001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:06:46,  852502 sec) LogPacket()          0x02 0x01 0xfe  []   0x68 0x3a
(Oct 27 2006 03:07:20,  852537 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:07:21,  852537 sec) TxPacket()           NAK received - resending packet. [PktNum=0x07]
(Oct 27 2006 03:07:21,  852537 sec) TxPacket()           History of packet transmission failures: [10000001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:07:21,  852537 sec) LogPacket()          0x02 0x07 0xf8  []   0xae 0x27
(Oct 27 2006 03:08:25,  852601 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:08:25,  852601 sec) TxPacket()           NAK received - resending packet. [PktNum=0x12]
```

```
(Oct 27 2006 03:08:25,  852601 sec) TxPacket()          History of packet transmission failures: [00000001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:08:25,  852601 sec) LogPacket()         0x02 0x12 0xed  []   0xe4 0x0d
(Oct 27 2006 03:08:51,  852627 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:08:51,  852627 sec) TxPacket()          NAK received - resending packet. [PktNum=0x16]
(Oct 27 2006 03:08:51,  852627 sec) TxPacket()          History of packet transmission failures: [00100001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:08:51,  852627 sec) LogPacket()         0x02 0x16 0xe9  []   0xcb 0x2f
(Oct 27 2006 03:09:14,  852651 sec) Tx()                Transmission completed successfully [25 packets, 23949 bytes, 155 sec, 159.4 bps]
(Oct 27 2006 03:09:15,  852651 sec) UpLoadFile()        Upload successful.
(Oct 27 2006 03:09:18,  852655 sec) UpLoadFile()        Uploading "5135.008.log" to host as "5135.008.log".
(Oct 27 2006 03:09:20,  852656 sec) Tx()                CRC negotiation successful. [16-bit CRC]
(Oct 27 2006 03:09:25,  852662 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:09:26,  852662 sec) TxPacket()          NAK received - resending packet. [PktNum=0x01]
(Oct 27 2006 03:09:26,  852662 sec) TxPacket()          History of packet transmission failures: [00100001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:09:26,  852662 sec) LogPacket()         0x02 0x01 0xfe  []   0x9d 0x82
(Oct 27 2006 03:09:29,  852666 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:09:29,  852666 sec) TxPacket()          NAK received - resending packet. [PktNum=0x01]
(Oct 27 2006 03:09:30,  852666 sec) TxPacket()          History of packet transmission failures: [01000011]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:09:30,  852666 sec) LogPacket()         0x02 0x01 0xfe  []   0x9d 0x82
(Oct 27 2006 03:10:38,  852734 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:10:38,  852734 sec) TxPacket()          NAK received - resending packet. [PktNum=0x0d]
(Oct 27 2006 03:10:38,  852734 sec) TxPacket()          History of packet transmission failures: [00000001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:10:38,  852734 sec) LogPacket()         0x02 0x0d 0xf2  []   0x37 0x24
(Oct 27 2006 03:10:45,  852742 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:10:46,  852742 sec) TxPacket()          NAK received - resending packet. [PktNum=0x0d]
(Oct 27 2006 03:10:46,  852742 sec) TxPacket()          History of packet transmission failures: [00000011]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:10:46,  852742 sec) LogPacket()         0x02 0x0d 0xf2  []   0x37 0x24
(Oct 27 2006 03:11:06,  852763 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:11:07,  852763 sec) TxPacket()          NAK received - resending packet. [PktNum=0x10]
(Oct 27 2006 03:11:07,  852763 sec) TxPacket()          History of packet transmission failures: [00110001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:11:07,  852763 sec) LogPacket()         0x02 0x10 0xef  []   0xf6 0x41
(Oct 27 2006 03:11:23,  852780 sec) GetReceiverResponse( Return value: 21
(Oct 27 2006 03:11:24,  852780 sec) TxPacket()          NAK received - resending packet. [PktNum=0x12]
(Oct 27 2006 03:11:24,  852780 sec) TxPacket()          History of packet transmission failures: [10001001]  (0:xmit-ok, 1:xmit-failed).
(Oct 27 2006 03:11:24,  852780 sec) LogPacket()         0x02 0x12 0xed  []   0x24 0x19
(Oct 27 2006 03:11:52,  852809 sec) Tx()                Transmission completed successfully [22 packets, 22148 bytes, 152 sec, 148.2 bps]
(Oct 27 2006 03:11:53,  852809 sec) UpLoadFile()        Upload successful.
(Oct 27 2006 03:11:53,  852809 sec) UpLoad()            Files successfully uploaded: 2
(Oct 27 2006 03:11:53,  852809 sec) UpLoad()            Upload complete.
(Oct 27 2006 03:11:55,  852812 sec) logout()            Log-out successful.
(Oct 27 2006 03:11:56,  852813 sec) Telemetry()         Telemetry cycle complete: PrfId=8 ConnectionAttempts=1  Connections=1
(Oct 27 2006 03:11:57,  852813 sec) TelemetryTerminate() Parsing new mission configuration.
(Oct 27 2006 03:11:57,  852814 sec) configure()         Parsing configurators in "mission.cfg".
(Oct 27 2006 03:12:02,  852818 sec) configure()            TelemetryRetry(15)      [0x7BBC] [TelemetryRetry(15)].
(Oct 27 2006 03:12:06,  852823 sec) configure()            DownTime(14000)         [0x5593] [DownTime(14000)].
(Oct 27 2006 03:12:07,  852823 sec) configure()         Configuration CRCs and syntax OK.
(Oct 27 2006 03:12:07,  852823 sec) ConfigSupervisor()  Configuration accepted.
(Oct 27 2006 03:12:07,  852823 sec) TelemetryTerminate() Reconditioning the file system.
(Oct 27 2006 03:12:07,       1 sec) DescentInit()       Deep profile 9 initiated at mission-time 852823sec.
(Oct 27 2006 03:12:10,       3 sec) DescentInit()       Surface pressure: 0.0dbars.
(Oct 27 2006 03:12:15,       8 sec) PistonMoveAbsWTO()   195->071 194 193 192 191 190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162 161 160 15
(Oct 27 2006 03:22:35,     629 sec) Descent()           Pressure: 46.1
(Oct 27 2006 04:12:10,    3604 sec) Descent()           Pressure: 445.1
(Oct 27 2006 05:12:10,    7204 sec) Descent()           Pressure: 779.0
(Oct 27 2006 06:12:10,   10804 sec) Descent()           Pressure: 952.3
(Oct 27 2006 07:12:10,   14404 sec) Descent()           Pressure: 982.4
(Oct 27 2006 08:12:10,   18004 sec) Descent()           Pressure: 987.7
(Oct 27 2006 09:12:10,   21604 sec) Descent()           Pressure: 991.1
(Oct 27 2006 09:12:11,   21604 sec) ParkInit()
(Nov 05 2006 14:32:42,  818435 sec) CtdPower()          [  999.68,  5.1401, 34.6389]
(Nov 05 2006 14:32:42,  818436 sec) CtdPower()          CTD Power consumption [190VCnt 12ACnt]: 14.728Volts * 0.048Amps = 0.71Watts.
(Nov 05 2006 14:32:42,  818436 sec) ParkTerminate()     Piston Position:71 Vacuum:77 Vq:198 Aq:6 Vsbe:190 Asbe:12
(Nov 05 2006 14:33:14,  818468 sec) ParkTerminate()     PTS: 999.8dbars 5.1389C 34.6385PSU
(Nov 05 2006 14:33:15,  818468 sec) GoDeepInit()        Moving piston.
(Nov 05 2006 14:33:15,  818469 sec) PistonMoveAbsWTO()   071->015 070 069 068 067 066 065 064 063 062 061 060 059 058 057 056 055 054 053 052 051 050 049 048 047 046 045 044 043 042 041 040 039 038 037 036 03
(Nov 05 2006 17:22:10,  828604 sec) GoDeep()            Sequence point detected at 2003.1dbar.
(Nov 05 2006 17:22:13,  828607 sec) ProfileInit()       PrfId:009  Pressure:2003.1dbar  pTable[0]:2000dbar
(Nov 05 2006 17:22:24,  828617 sec) PistonMoveAbsWTO()   015->037 016 017 018 019 020 [30sec, 13.7Volts, 0.874Amps, CPT:916sec]
(Nov 05 2006 17:23:04,  828657 sec) PistonMoveAbsWTO()   020->037 021 022 023 024 025 [30sec, 13.4Volts, 0.822Amps, CPT:946sec]
(Nov 05 2006 17:23:44,  828697 sec) PistonMoveAbsWTO()   025->037 026 027 028 029 [30sec, 13.3Volts, 0.890Amps, CPT:976sec]
(Nov 05 2006 17:24:24,  828737 sec) PistonMoveAbsWTO()   029->037 030 031 032 033 034 [30sec, 13.3Volts, 0.862Amps, CPT:1006sec]
(Nov 05 2006 17:25:35,  828809 sec) Profile()           Sample 0 initiated at 1998.9dbars for bin 0 [2000dbars].  PTS: 1996.0dbars 2.4262C 34.7183PSU
(Nov 05 2006 17:25:36,  828809 sec) PistonMoveAbsWTO()   034->037 035 036 037 [17sec, 13.3Volts, 0.870Amps, CPT:1023sec]
(Nov 05 2006 17:33:24,  829278 sec) Profile()           Sample 1 initiated at 1950.1dbars for bin 1 [1950dbars].  PTS: 1947.2dbars 2.5096C 34.7160PSU
(Nov 05 2006 17:42:09,  829803 sec) Profile()           Sample 2 initiated at 1900.8dbars for bin 2 [1900dbars].  PTS: 1898.2dbars 2.5933C 34.7124PSU
(Nov 05 2006 17:52:37,  830431 sec) Profile()           Sample 3 initiated at 1850.8dbars for bin 3 [1850dbars].  PTS: 1848.7dbars 2.6898C 34.7063PSU
(Nov 05 2006 17:52:40,  830434 sec) AscentControlAgent() Bouyancy nudge to 47 (v=0.074dbar/sec).
(Nov 05 2006 17:52:40,  830434 sec) PistonMoveAbsWTO()   037->047 038 039 040 041 042 [30sec, 13.6Volts, 0.782Amps, CPT:1053sec]
(Nov 05 2006 17:53:22,  830475 sec) PistonMoveAbsWTO()   042->047 043 044 045 046 [30sec, 13.4Volts, 0.802Amps, CPT:1083sec]
(Nov 05 2006 17:54:02,  830515 sec) PistonMoveAbsWTO()   046->047 047 [3sec, 13.5Volts, 0.810Amps, CPT:1086sec]
(Nov 05 2006 18:01:19,  830953 sec) Profile()           Sample 4 initiated at 1800.8dbars for bin 4 [1800dbars].  PTS: 1798.0dbars 2.8293C 34.6978PSU
(Nov 05 2006 18:10:17,  831491 sec) Profile()           Sample 5 initiated at 1750.8dbars for bin 5 [1750dbars].  PTS: 1748.0dbars 2.9263C 34.6937PSU
(Nov 05 2006 18:19:25,  832039 sec) Profile()           Sample 6 initiated at 1700.9dbars for bin 6 [1700dbars].  PTS: 1698.2dbars 3.0365C 34.6872PSU
(Nov 05 2006 18:28:53,  832607 sec) Profile()           Sample 7 initiated at 1650.2dbars for bin 7 [1650dbars].  PTS: 1647.6dbars 3.1362C 34.6827PSU
(Nov 05 2006 18:38:19,  833173 sec) AscentControlAgent() Bouyancy nudge to 57 (v=0.079dbar/sec).
(Nov 05 2006 18:38:19,  833173 sec) PistonMoveAbsWTO()   047->057 048 049 050 051 052 [30sec, 13.6Volts, 0.729Amps, CPT:1116sec]
(Nov 05 2006 18:39:32,  833246 sec) Profile()           Sample 8 initiated at 1598.1dbars for bin 8 [1600dbars].  PTS: 1595.3dbars 3.2713C 34.6748PSU
(Nov 05 2006 18:39:33,  833246 sec) PistonMoveAbsWTO()   052->057 053 054 055 056 057 [29sec, 13.6Volts, 0.677Amps, CPT:1145sec]
(Nov 05 2006 18:47:06,  833700 sec) Profile()           Sample 9 initiated at 1550.1dbars for bin 9 [1550dbars].  PTS: 1547.1dbars 3.3872C 34.6718PSU
(Nov 05 2006 18:55:44,  834218 sec) Profile()           Sample 10 initiated at 1500.8dbars for bin 10 [1500dbars].  PTS: 1498.2dbars 3.5018C 34.6652PSU
(Nov 05 2006 19:03:40,  834694 sec) AscentControlAgent() Bouyancy nudge to 67 (v=0.079dbar/sec).
(Nov 05 2006 19:03:40,  834694 sec) PistonMoveAbsWTO()   057->067 058 059 060 061 062 [30sec, 13.7Volts, 0.697Amps, CPT:1175sec]
(Nov 05 2006 19:04:22,  834735 sec) PistonMoveAbsWTO()   062->067 063 064 065 066 067 [29sec, 13.6Volts, 0.616Amps, CPT:1204sec]
(Nov 05 2006 19:05:42,  834816 sec) Profile()           Sample 11 initiated at 1450.7dbars for bin 11 [1450dbars].  PTS: 1447.5dbars 3.6770C 34.6643PSU
(Nov 05 2006 19:13:27,  835281 sec) Profile()           Sample 12 initiated at 1400.8dbars for bin 12 [1400dbars].  PTS: 1397.8dbars 3.8251C 34.6594PSU
(Nov 05 2006 19:21:45,  835779 sec) Profile()           Sample 13 initiated at 1350.4dbars for bin 13 [1350dbars].  PTS: 1347.5dbars 4.0108C 34.6557PSU
```

```
(Nov 05 2006 19:30:33,  836307 sec) Profile()            Sample 14 initiated at 1300.5dbars for bin 14 [1300dbars].  PTS: 1297.9dbars 4.1146C 34.6515PSU
(Nov 05 2006 19:40:11,  836885 sec) Profile()            Sample 15 initiated at 1250.8dbars for bin 15 [1250dbars].  PTS: 1248.5dbars 4.2443C 34.6510PSU
(Nov 05 2006 19:44:14,  837128 sec) AscentControlAgent() Bouyancy nudge to 77 (v=0.078dbar/sec).
(Nov 05 2006 19:44:14,  837128 sec) PistonMoveAbsWTO()    067->077 068 069 070 071 072 [30sec, 13.8Volts, 0.616Amps, CPT:1234sec]
(Nov 05 2006 19:44:56,  837169 sec) PistonMoveAbsWTO()    072->077 073 074 075 076 077 [27sec, 13.7Volts, 0.556Amps, CPT:1261sec]
(Nov 05 2006 19:49:24,  837438 sec) Profile()            Sample 16 initiated at 1200.9dbars for bin 16 [1200dbars].  PTS: 1197.8dbars 4.3623C 34.6440PSU
(Nov 05 2006 19:58:02,  837956 sec) Profile()            Sample 17 initiated at 1150.3dbars for bin 17 [1150dbars].  PTS: 1147.7dbars 4.5327C 34.6408PSU
(Nov 05 2006 20:08:10,  838564 sec) Profile()            Sample 18 initiated at 1100.6dbars for bin 18 [1100dbars].  PTS: 1098.4dbars 4.7406C 34.6413PSU
(Nov 05 2006 20:09:43,  838657 sec) AscentControlAgent() Bouyancy nudge to 87 (v=0.077dbar/sec).
(Nov 05 2006 20:09:43,  838657 sec) PistonMoveAbsWTO()    077->087 078 079 080 081 082 [30sec, 13.8Volts, 0.568Amps, CPT:1291sec]
(Nov 05 2006 20:10:25,  838698 sec) PistonMoveAbsWTO()    082->087 083 084 085 086 087 [27sec, 13.7Volts, 0.552Amps, CPT:1318sec]
(Nov 05 2006 20:16:46,  839080 sec) Profile()            Sample 19 initiated at 1050.8dbars for bin 19 [1050dbars].  PTS: 1047.7dbars 4.9210C 34.6328PSU
(Nov 05 2006 20:24:51,  839565 sec) Profile()            Sample 20 initiated at 1000.9dbars for bin 20 [1000dbars].  PTS: 997.9dbars 5.1765C 34.6345PSU
(Nov 05 2006 20:26:19,  839652 sec) Sbe41cpStartCP()     Continuous profile started.
(Nov 05 2006 20:46:43,  840877 sec) AscentControlAgent() Bouyancy nudge to 97 (v=0.079dbar/sec).
(Nov 05 2006 20:46:44,  840877 sec) PistonMoveAbsWTO()    087->097 088 089 090 091 092 [30sec, 13.8Volts, 0.504Amps, CPT:1348sec]
(Nov 05 2006 20:47:24,  840917 sec) PistonMoveAbsWTO()    092->097 093 094 095 096 097 [26sec, 13.6Volts, 0.532Amps, CPT:1374sec]
(Nov 05 2006 21:18:26,  842780 sec) AscentControlAgent() Bouyancy nudge to 107 (v=0.079dbar/sec).
(Nov 05 2006 21:18:27,  842780 sec) PistonMoveAbsWTO()    097->107 098 099 100 101 102 [30sec, 13.9Volts, 0.419Amps, CPT:1404sec]
(Nov 05 2006 21:19:07,  842820 sec) PistonMoveAbsWTO()    102->107 103 104 105 106 107 [25sec, 13.7Volts, 0.459Amps, CPT:1429sec]
(Nov 05 2006 21:50:07,  844681 sec) AscentControlAgent() Bouyancy nudge to 117 (v=0.074dbar/sec).
(Nov 05 2006 21:50:08,  844681 sec) PistonMoveAbsWTO()    107->117 108 109 110 111 112 [30sec, 14.0Volts, 0.371Amps, CPT:1459sec]
(Nov 05 2006 21:50:48,  844721 sec) PistonMoveAbsWTO()    112->117 113 114 115 116 117 [24sec, 13.9Volts, 0.379Amps, CPT:1483sec]
(Nov 05 2006 22:16:42,  846276 sec) AscentControlAgent() Bouyancy nudge to 127 (v=0.074dbar/sec).
(Nov 05 2006 22:16:43,  846276 sec) PistonMoveAbsWTO()    117->127 118 119 120 121 122 [30sec, 14.0Volts, 0.342Amps, CPT:1513sec]
(Nov 05 2006 22:17:23,  846316 sec) PistonMoveAbsWTO()    122->127 123 124 125 126 127 [23sec, 14.0Volts, 0.326Amps, CPT:1536sec]
(Nov 05 2006 22:43:15,  847869 sec) AscentControlAgent() Bouyancy nudge to 137 (v=0.079dbar/sec).
(Nov 05 2006 22:43:16,  847869 sec) PistonMoveAbsWTO()    127->137 128 129 130 131 132 [30sec, 14.1Volts, 0.286Amps, CPT:1566sec]
(Nov 05 2006 22:43:55,  847909 sec) PistonMoveAbsWTO()    132->137 133 134 135 136 137 [22sec, 14.0Volts, 0.282Amps, CPT:1588sec]
(Nov 05 2006 22:54:30,  848544 sec) AscentControlAgent() Bouyancy nudge to 147 (v=0.078dbar/sec).
(Nov 05 2006 22:54:31,  848544 sec) PistonMoveAbsWTO()    137->147 138 139 140 141 142 [30sec, 14.1Volts, 0.262Amps, CPT:1618sec]
(Nov 05 2006 22:55:10,  848584 sec) PistonMoveAbsWTO()    143->147 144 145 146 147 [22sec, 14.1Volts, 0.266Amps, CPT:1640sec]
(Nov 05 2006 23:05:44,  849218 sec) AscentControlAgent() Bouyancy nudge to 157 (v=0.077dbar/sec).
(Nov 05 2006 23:05:45,  849218 sec) PistonMoveAbsWTO()    147->157 148 149 150 151 152 153 [30sec, 14.2Volts, 0.246Amps, CPT:1670sec]
(Nov 05 2006 23:06:24,  849258 sec) PistonMoveAbsWTO()    153->157 154 155 156 157 [20sec, 14.1Volts, 0.246Amps, CPT:1690sec]
(Nov 05 2006 23:16:56,  849890 sec) AscentControlAgent() Bouyancy nudge to 167 (v=0.051dbar/sec).
(Nov 05 2006 23:16:57,  849890 sec) PistonMoveAbsWTO()    157->167 158 159 160 161 162 [30sec, 14.2Volts, 0.234Amps, CPT:1720sec]
(Nov 05 2006 23:17:36,  849930 sec) PistonMoveAbsWTO()    163->167 164 165 166 167 [20sec, 14.2Volts, 0.234Amps, CPT:1740sec]
(Nov 05 2006 23:23:02,  850256 sec) AscentControlAgent() Bouyancy nudge to 177 (v=0.074dbar/sec).
(Nov 05 2006 23:23:03,  850256 sec) PistonMoveAbsWTO()    167->177 168 169 170 171 172 [30sec, 14.2Volts, 0.218Amps, CPT:1770sec]
(Nov 05 2006 23:23:42,  850296 sec) PistonMoveAbsWTO()    172->177 173 174 175 176 177 [21sec, 14.2Volts, 0.218Amps, CPT:1791sec]
(Nov 05 2006 23:31:48,  850782 sec) SurfaceDetect()      SurfacePressure:0.0dbars Pressure:3.8dbars PistonPosition:177
(Nov 05 2006 23:31:49,  850783 sec) Sbe41cpStopCP()      Continuous profile stopped.
(Nov 05 2006 23:31:53,  850787 sec) PistonMoveAbsWTO()    177->199 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 [108sec, 14.2Volts, 0.181Amps, CPT:1899sec]
(Nov 05 2006 23:38:28,  851181 sec) Sbe41cpBinAverage()  Finished averaging 11134 samples in 281 seconds.
(Nov 05 2006 23:38:33,  851186 sec) Sbe41cpUploadCP()    Sbe41cpSerNo[1520] NSample[11134] NBin[495]
(Nov 05 2006 23:39:32,  851246 sec) Sbe41cpUploadCP()    Continuous profile uploaded [495 lines].

<EOT>
```

D Encoding of hydrographic data.

The C source code below is used in APEX firmware to encode the hydrographic data before it is telemetered to the remote host.

```
#ifndef ENCODE_H
#define ENCODE_H

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * $Id: EncodeCSourceCode.tex,v 1.1 2006/11/03 19:08:57 swift Exp $
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/** RCS log of revisions to the C source code.
 *
 * \begin[verbatim]
 * $Log: EncodeCSourceCode.tex,v $
 * Revision 1.1  2006/11/03 19:08:57  swift
 * Added user manual to CVS control.
 *
 * Revision 1.2  2006/07/10 22:24:49  swift
 * Modifications to bring the manual up to date with
 * changes to the SeaBird CTD firmware (v1.1c).
```

```
 *
 * Revision 1.3  2006/02/08 20:17:28  swift
 * Modifications to shorten PTS encoding from 20-bits down to 16-bits and
 * to shorten the encoding of the number of samples from 16-bits to 8-bits.
 *
 * Revision 1.2  2003/09/10 16:50:04  swift
 * Added change-log tracking macro.
 *
 * Revision 1.1  2003/09/10 16:47:17  swift
 * Initial revision
 * \end[verbatim]
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
#define encodeChangeLog "$RCSfile: EncodeCSourceCode.tex,v $  $Revision: 1.1 $   $Date: 2006/11/03 19:08:57 $"

/* function prototypes */
unsigned char EncodeN(unsigned int NSample);
unsigned int  EncodeO(float o2);
unsigned int  EncodeP(float p);
unsigned int  EncodeS(float s);
unsigned int  EncodeT(float t);

#endif /* ENCODE_H */

#include <assert.h>
#include <nan.h>

/*------------------------------------------------------------------------*/
/* function to encode the number of samples as an 8-bit unsigned integer  */
/*------------------------------------------------------------------------*/
/**
   This function encodes the number samples in the bin average as an 8-bit
   unsigned integer with protection against overflow.  The encoding accounts
   for the full range of 16-bit unsigned integers but only values in the
   open range: 0<NSample<255 are representable.  This encoding makes full
   use of all 8-bits.

      \begin[verbatim]
      input:
         NSample ... The number of samples in the bin-average.

      output:
         1) Values greater than or equal to 255 are mapped to 0xff.

         2) All other values are expressed as an 8-bit unsigned integer.
      \end[verbatim]
*/
unsigned char EncodeN(unsigned int NSample)
{
   /* prevent overflow of the sample counter */
   unsigned int N = (NSample>=255) ? 0xff : NSample;

   return N;
```

```
}

/*------------------------------------------------------------------------*/
/* function to encode oxygen as a 2-byte unsigned integer                 */
/*------------------------------------------------------------------------*/
/**
   This function implements the hex-encoding of IEEE-formattted floating
   point oxygen data into 16-bit unsigned integers with 2's-complement
   representation.  The encoding formula accounts for the full range of
   32-bit IEEE floating point values but only values in the open range:
   -4095<o2<61439 are representable.  This encoding makes full use of all
   16-bits.

      \begin[verbatim]
      input:
         o2 ... The oxygen (o2-freq) expressed as a floating point value.

      output:
         1) Nonfinite values (Inf, -Inf, NaN) are mapped to the sentinel hex
            value: 0xf000.

         2) Oxygen frequency values less than -4095 are mapped to the
            sentinel hex value: 0xf001.

         3) Oxygen frequency values greater than 61439 are mapped to the
            sentinel hex value: 0xefff.

         4) All other values are to the nearest integer and expressed as a
            16-bit signed integer in 2's-complement form.
      \end[verbatim]

   Important Note: This function is not portable to C-implementations for
   which unsigned integers do not have exactly two bytes.  For the APF9
   controller, this function has been fully tested over the full range of
   oxygen.
*/
unsigned int EncodeO(float o2)
{
   /* initialize with the mapping for a nonfinite oxygen */
   long int O2 = 0xf000;

   /* make sure long ints are at least 3 bytes long */
   assert(sizeof(long int)>=3);

   if (finite(o2))
   {
      /* assign out-of-range values to sentinel values */
      if (o2>=61439) O2=0xefff; else if (o2<=-4095) O2=0xf001;

      /* encode the oxygen frequency (rounded) */
      else O2 = (unsigned int)(o2 + ((o2<0) ? -0.5 : 0.5));
```

```
      /* express in 16-bit 2's-complement form */
      if (O2<0) O2+=0x10000L;
   }

   return O2;
}

/*------------------------------------------------------------------------*/
/* function to encode pressure as a 2-byte unsigned integer               */
/*------------------------------------------------------------------------*/
/**
   This function implements the hex-encoding of IEEE-formattted floating
   point pressure data into 2-byte signed integers with 2's complement
   representation.  The encoding formula accounts for the full range of
   32-bit IEEE floating point values but only values in the open range:
   -3276.7<p<3276.7 are representable.  This encoding makes full use of all
   16-bits.

      \begin[verbatim]
      input:
         p ... The pressure (decibars) expressed as a floating point value.

      output:
         1) Nonfinite values (Inf, -Inf, NaN) are mapped to the sentinel hex
            value: 0x8000.

         2) Pressure values less than -3276.7 are mapped to the sentinel
            value: 0x8001.

         3) Pressure values greater than 3276.7 are mapped to the sentinel
            value: 0x7fff.

         4) All other values are expressed in millibars rounded to the
            nearest integer and expressed as a 16-bit signed integer in
            2's-complement form.
      \end[verbatim]
*/
unsigned int EncodeP(float p)
{
   /* initialize with the mapping for a nonfinite pressure */
   long int P = 0x8000L;

   /* make sure long ints are at least 3 bytes long */
   assert(sizeof(long int)>=3);

   if (finite(p))
   {
      /* assign out-of-range values to sentinel values */
      if (p>=3276.7) P=0x7fffL; else if (p<=-3276.7) P=0x8001L;

      /* encode the pressure as the number of centibars (rounded) */
      else P = (long int)(10*(p + ((p<0) ? -0.05 : 0.05)));
```

```
      /* express in 16-bit 2's-complement form */
      if (P<0) P+=0x10000L;
   }

   return P;
}

/*------------------------------------------------------------------------*/
/* function to encode salinity as a 2-byte unsigned long integer          */
/*------------------------------------------------------------------------*/
/**
   This function implements the hex-encoding of IEEE-formattted floating
   point salinity data into 16-bit unsigned integers with 2's complement
   representation.  The encoding formula accounts for the full range of
   32-bit IEEE floating point values but only values in the open range:
   -4.095<s<61.439 are representable.  This encoding makes full use of all
   16-bits.

      \begin[verbatim]
      input:
         s ... The salinity (PSU) expressed as a floating point value.

      output:
         1) Nonfinite values (Inf, -Inf, NaN) are mapped to the sentinel hex
            value: 0xf000.

         2) Salinity values less than -4.095 are mapped to the sentinel
            value: 0xf001.

         3) Salinity values greater than 61.439 are mapped to the sentinel
            value: 0xefff.

         4) All other values are expressed in parts-per-ten-million
            rounded to the nearest integer and expressed as a 16-bit
            signed integer in 2's-complement form.
      \end[verbatim]
*/
unsigned int EncodeS(float s)
{
   /* initialize with the mapping for a nonfinite salinity */
   long int S = 0xf000L;

   /* make sure that long integers have at least three bytes */
   assert(sizeof(long int)>=3);

   if (finite(s))
   {
      /* assign out-of-range values to sentinel values */
      if (s>=61.439) S=0xefffL; else if (s<=-4.095) S=0xf001L;

      /* encode the salinity as the number of parts-per-ten-million (rounded) */
```

```
      else S = (long int)(1000*(s + ((s<0) ? -0.0005 : 0.0005)));

      /* express in 16-bit 2's-complement form */
      if (S<0) S+=0x10000L;
   }

   return S;
}

/*------------------------------------------------------------------------*/
/* function to encode temperature as a 2-byte unsigned integer            */
/*------------------------------------------------------------------------*/
/**
   This function implements the hex-encoding of IEEE-formattted floating
   point temperature data into 16-bit unsigned integers with 2's complement
   representation.  The encoding formula accounts for the full range of
   32-bit IEEE floating point values but only values in the open range:
   -4.095<t<61.439 are representable.  This encoding makes full use of all
   16-bits.

      \begin[verbatim]
      input:
         t ... The temperature (C) expressed as a floating point value.

      output:
         1) Nonfinite values (Inf, -Inf, NaN) are mapped to the sentinel hex
            value: 0xf000.

         2) Temperature values less than -6.5535 are mapped to the sentinel
            value: 0xf001.

         3) Temperature values greater than 98.3039 are mapped to the sentinel
            value: 0xefff.

         4) All other values are expressed in tenths of millidegrees Celsius
            rounded to the nearest integer and expressed as a 16-bit signed
            integer in 2's-complement form.
      \end[verbatim]
*/
unsigned int EncodeT(float t)
{
   /* initialize with the mapping for a nonfinite temperature */
   long int T = 0xf000L;

   /* make sure that long integers have at least three bytes */
   assert(sizeof(long int)>=3);

   if (finite(t))
   {
      /* assign out-of-range values to sentinel values */
      if (t>=61.439) T=0xefffL; else if (t<=-4.095) T=0xf001L;
```

```
      /* encode the temperature as the number of tenths of millidegrees (rounded) */
      else T = (long int)(1000*(t + ((t<0) ? -0.0005 : 0.0005)));

      /* express in 16-bit 2's-complement form */
      if (T<0) T+=0x10000L;
   }

   return T;
}
```