User Manual: Iridium Apex with Ice Dection/Evasion (Apf9i Firmware Revision: 070207)

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

July 2, 2007

*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  2006/11/22 02:30:25  swift
Change name of user manual to include ice detection/avoidance.

Revision 1.2  2006/11/22 02:17:40  swift
Modification to facilitate automatic firmware revision management.

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 070207.

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.

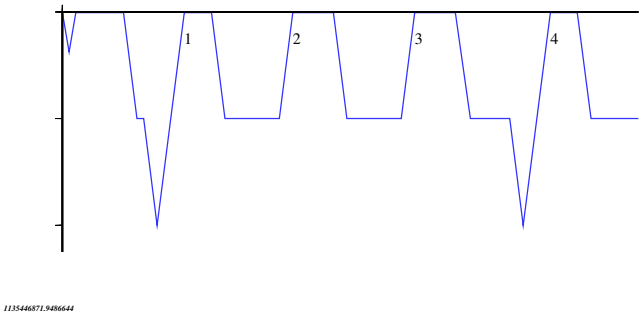


Figure 1: Schematic of a PnP mission with cycle length $n = 4$. The park level is the same for all profiles. Every fourth profile is a deep profile. The shallow blip prior to the (special) first profile represents pressure activation.

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.

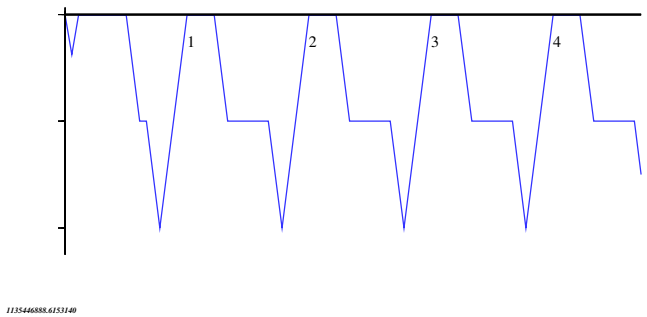


Figure 2: Schematic of a PnP mission with cycle length $n = 1$. Every profile parks shallow but profiles deep. The shallow blip prior to the (special) first profile represents pressure activation.

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.

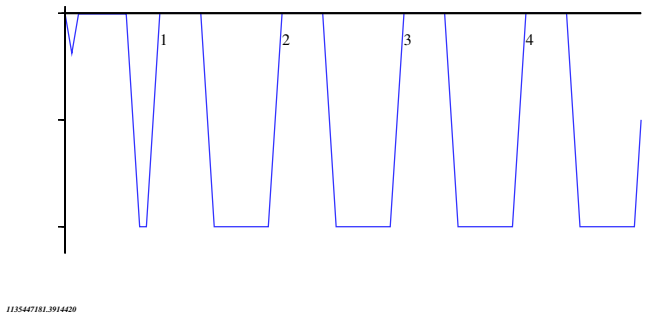


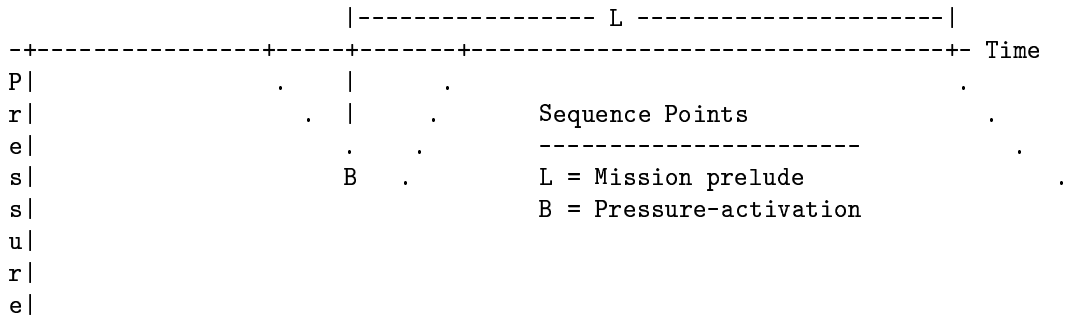
Figure 3: Schematic of a degenerate PnP mission with cycle length n=254. Every cycle parks and profiles from the park level. The shallow blip prior to the (special) first profile represents pressure activation.

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).

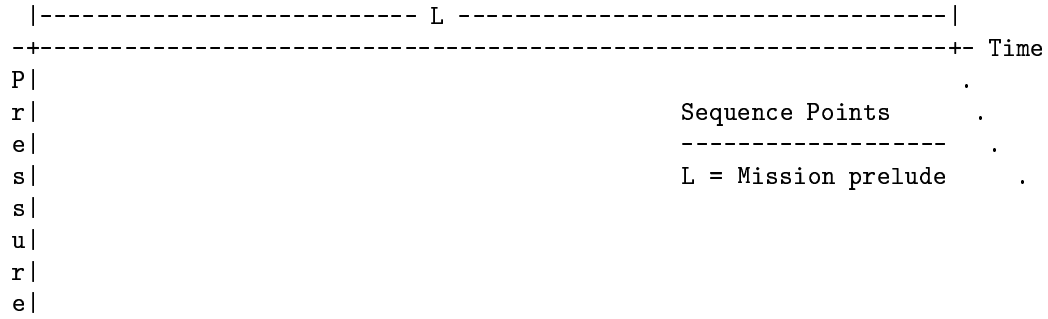


Pressure activation mode is not "on" by default. The user must enable pressure activation mode via the interactive user interface (see Section 3). Enabling the pressure activation mode immediately induces the firmware to perform a self-test of the float.

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).

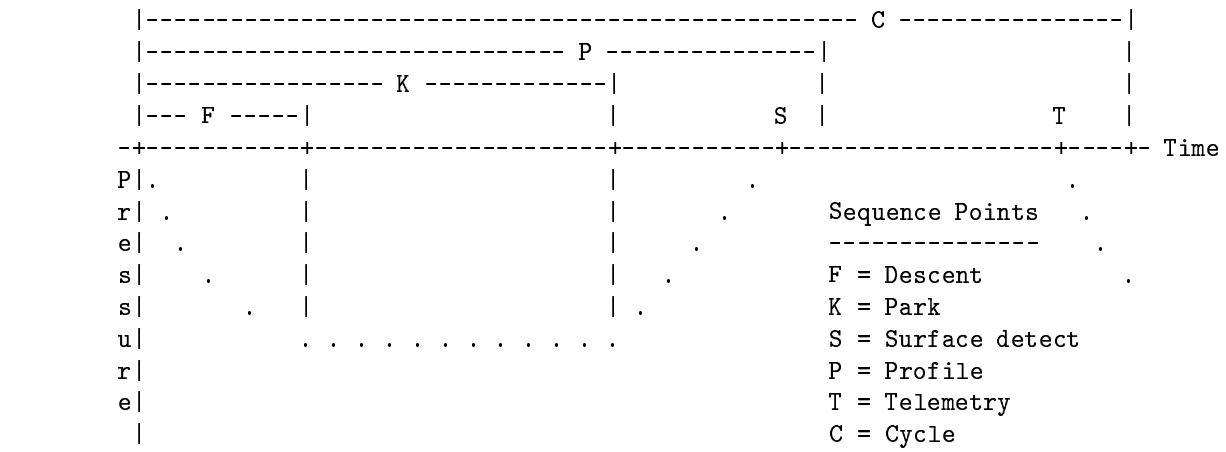


When the mission is launched either manually or else by the pressure activation mechanism, the firmware puts the float into a state of maximum buoyancy by fully extending the buoyancy pump to inflate the oil bladder and then inflating the air bladder.

Special note for under-ice operation: This firmware implements a feature specially designed for under-ice operation (see Section 2.2.3, pg. 7). One element of the ice-ready feature includes the ability to disable (ie., by-pass) the mission prelude so that the float will immediately sink below the surface when deployed. The mission prelude should be disabled if the float is deployed where floating ice might pose a danger to the float. For example, deployments from an ice-breaker should have the mission prelude disabled so that the float will sink out of harm's way. Disabling the mission prelude is compatible with mission activation either manually or else via the pressure-activation feature.

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.



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.

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.

Rafos tracking: The float activates the SeaScan RAFOS receiver to listen for sound pulses emitted by moored acoustic sound sources. Up to six listening windows can be scheduled to begin at user-specified times each day. The user can also specify the duration of these listening windows (in the range 1–60 minutes). A sanity violation will be encountered if the duration is less than 15 minutes. Refer to Section 3 (page 10) or Section 4 (page 19) for details. Refer to Section 6.1.2 for the data format of RAFOS samples.

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 6.1.3 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 6.1.4.

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 6.1.3 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].

[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.

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.

Ice detection: This firmware implements an ice detection/evasion feature designed to enhance float survivability in regions prone to ice cover. The goal is to prevent the float from being crushed between large chunks of floating ice such as that found in fields of large pancake-ice. The strategy is to detect hydrographic features associated with surface ice and then to descend before reaching the surface in order to avoid being crushed. The algorithm attempts to detect large scale ice coverage or smaller scale ice caps based on three criteria:

1. Ice-cover detection criterion. Historical data indicates that the layer of water underneath large regions of ice cover is characterized by a mixed layer of near-freezing water.
   Temperature measurements are made at 2.5 dbar intervals starting at 50 decibars. If, at any pressure less than 30 dbars, the median of the temperature measurements is less than a (user-specified) critical temperature ($T_{critical}$) then the determination is made that the surface is ice covered.
2. Ice-cap detection criterion. Periods when there is a moderately large fraction of open water that contain large blocks of free-floating ice may not be detected by the criterion above and are especially dangerous to floats. If the float reaches the surface in open water then telemetry will happen despite the danger. However, if the float happens to ascend and stay under ice then this criteria is designed to detect this situation, abort the telemetry attempt, and descend from the surface.
   This criterion determines that the float is under an ice cap if all of the following conditions are satisfied:
   - The Iridium LBT has failed to register with the Iridium system. This indicatest that the float does not have a clear view of the sky.
   - The profile's ascent time-out period has expired. This ensures that adequate opportunity has been given for the LBT to connect with the satellites and register with the Iridium system.
   - The pressure is less than 30 decibars. This ensures that the float is near the surface as it would have to be in order to be trapped under ice.
   - The temperature is less than $T_{critical}$. If the other conditions are satisfied then a single temperature measurement is made to determine if the float is within the cold-water halo underneath the ice.

   The last two conditions avoid false-positives that might be caused by factors other than surface ice.
3. Winter/summer criterion. The ice detection/evasion feature can be month-wise enabled or disabled by specifying which months of the year should be considered winter months. Refer to Section 3 (page 12) or Section 4 (page 18) for the method used to specify the months when the ice detection/evasion feature should be enabled.

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.

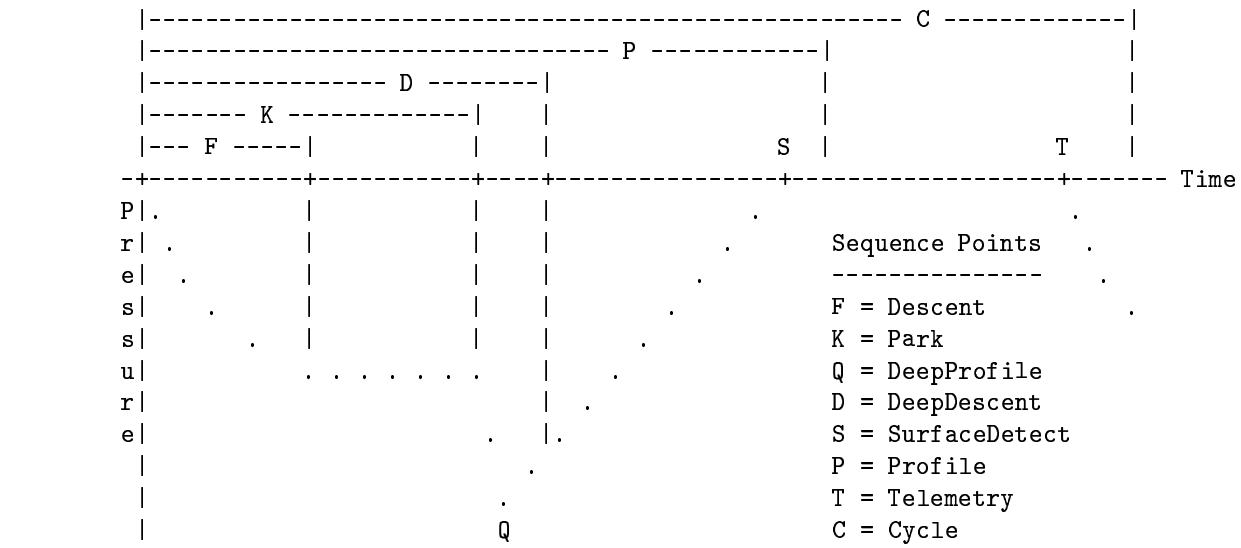
After determining that the float can see the sky then the LBT is shut down and the GPS engine is used to acquire the float's location. After acquiring the fix then the GPS is shut down and the LBT is reregistered with the Iridium system. The float places a call via the Iridium system to the remote host 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.



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.

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 110606  sn 5059
User: f5059
Pwd:  0xafb3
Pri:  AT+CBST=71,0,1;DT0088160000510                Mhp
Alt:  ATDT0012065551234                             Mha
Rafos: 29;0,30,60,90,120,1410             (Minutes) Mtw
 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
   012 Deep-profile piston position. (Counts)       Mbj
   010 Ascent buoyancy nudge. (Counts)              Mbn
   022 Initial buoyancy nudge (Counts)              Mbi
   001 Park-n-profile cycle length.                 Mn
 -1.80 Ice detection: Mixed-layer Tcritical (C)     Mit
 0x807 Ice detection: Winter months [DNOSAJJMAMFJ]  Mib
   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
  0002 DebugBits.                                   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.

Rafos —This configures the listening windows for RAFOS tracking. The syntax for the rafos configurator is: $Span;Start_1,Start_2,Start_3,\ldots,Start_N$ where *Span* represents the duration (minutes) of each listening window and $Start_n$ represents the day-minute (ie., the number of minutes since midnight GMT) when the listening window should begin. The following constraints apply:

- The number of listening windows must be in the closed range [1,6].
- The duration of each window must be in the closed range [1,60] minutes. A sanity violation will be encountered if the duration is less than 15 minutes.
- Each window must start and end within the same day (GMT). The start time must be in the semi-open range [0,1438) while the end time must be in the semi-open range (2,1439] minutes.
- Listening windows may not overlap or be contiguous; there must be at least one minute between the end of one window and the beginning of the next.

Note: To disable RAFOS tracking completely then specify *Span* to be zero followed by the ";" separator. Trailing start times are optional and ignored, if present. For example, the configurator "0;" will disable RAFOS tracking.

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

Special note for under-ice operation: The mission prelude can be disabled (ie., by-passed) by setting the duration of the mission prelude to zero. That is, zero is a special sentinel value that causes the firmware to by-pass the mission prelude and, upon mission activation, immediately initiate the descent phase of the first profile cycle. Disabling the mission prelude will cause the float to immediately sink below the surface at deployment time. The mission prelude should be disabled if floating ice might pose a danger to the float at deployment time. Disabling the mission prelude is compatible with mission activation either manually or else via the pressure-activation feature.

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 5 minutes 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.

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.

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.

Ice detection: Specification of the critical temperature —This parameter determines when the ice detection algorithm initiates ice evasion. Refer to Section 2.2.3, page 7 for a description of the ice detection/evasion feature.

Ice detection: Specification of winter months —A 12-bit integer is used to specify winter months. The bits of the integer represent the months of the year with the least-significant-bit representing January up to the most significant bit which represents December. If the bit is asserted then the corresponding month is specified to be a winter month. Non-winter months should have their corresponding bits set to zero. The ice detection algorithm is applied only during winter months; ice detection is not attempted during non-winter months.

For example, if the argument of this configurator were 0x807 then the winter months are specified to be December, January, February, and March. The ice detection algorithm would be applied only during these winter months. For the non-winter months of April thru November,

the ice detection algorithm would be skipped and the float would ascend to the surface on each profile.

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	$\leq$	Verbosity	$\leq$	5
1 Count	$\leq$	AirBladderMaxP	$\leq$	240 Counts
1 Count	$\leq$	OkVacuumCount	$\leq$	254 Counts
1 Count	$\leq$	PistonBuoyancyNudge	$\leq$	254 Counts
1 Count	$\leq$	DeepProfilePistonPos	$\leq$	254 Counts
1 Count	$\leq$	PistonFullExtension	$\leq$	254 Counts
1 Count	$\leq$	PistonInitialBuoyancyNudge	$\leq$	254 Counts
1 Count	$\leq$	ParkPistonPos	$\leq$	254 Counts
1 Count	$\leq$	PistonStoragePosition	$\leq$	254 Counts
1	$\leq$	PnPCycleLen	$\leq$	254
0 Decibars	$\leq$	ParkPressure	$\leq$	2000 Decibars
0 Decibars	$\leq$	DeepProfilePressure	$\leq$	2000 Decibars
0 Minute	$\leq$	DeepProfileDescentTime	$\leq$	8 Hours
0 Minute	$<$	DownTime	$\leq$	30 Days
1 Minute	$\leq$	AscentTimeOut	$\leq$	10 Hours
0 Minute	$<$	ParkDescentTime	$\leq$	8 Hours
0 Minute	$\leq$	TimePrelude	$\leq$	6 Hours
5 Minutes	$\leq$	TelemetryRetry	$\leq$	6 Hours
0 Minutes	$<$	UpTime	$\leq$	24 Hours
30 Seconds	$\leq$	ConnectTimeOut	$\leq$	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:
$mission.PistonDeepProfilePosition <= mission.PistonParkPosition$

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.

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

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:

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.

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.

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.

FlashFsCreate() —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 rereading 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.

IceMLTCritical(Celsius) —The argument of this configurator specifies the critical mixed-layer-temperature (Celsius) used by the ice detection algorithm. If the median temperature in the layer of water between 30-50 decibars is less than the critical temperature then the likely presence of ice at the surface is indicated. This will cause the ice evasion mechanism to be activated.

IceMonths() —The argument of this configurator represents a 12-bit integer that specifies winter months. The bits of the integer represent the months of the year with the least-significant-bit representing January up to the most significant bit which represents December. If the bit is asserted then the corresponding month is specified to be a winter month. Non-winter months should have their corresponding bits set to zero. The ice detection algorithm is applied only during winter months; ice detection is not attempted during non-winter months.

For example, if the argument of this configurator were 0x807 then the winter months are specified to be December, January, February, and March. The ice detection algorithm would be applied only during these winter months. For the non-winter months of April thru November, the ice detection algorithm would be skipped and the float would ascend to the surface on each profile.

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.

RafosWindows() —

Rafos —This configures the listening windows for RAFOS tracking. The syntax for the rafos configurator is: *Span;Start*$_1$*,Start*$_2$*,Start*$_3$*,...,Start*$_N$ where *Span* represents the duration (minutes) of each listening window and *Start*$_n$ represents the day-minute (ie., the number of minutes since midnight GMT) when the listening window should begin. The following constraints apply:

- The number of listening windows must be in the closed range [1,6].
- The duration of each window must be in the closed range [1,60] minutes. A sanity violation will be encountered if the duration is less than 15 minutes.
- Each window must start and end within the same day (GMT). The start time must be in the semi-open range [0,1438) while the end time must be in the semi-open range (2,1439] minutes.
- Listening windows may not overlap or be contiguous; there must be at least one minute between the end of one window and the beginning of the next.

Note: To disable RAFOS tracking completely then specify *Span* to be zero followed by the ";" separator. Trailing start times are optional and ignored, if present. For example, the configurator "0;" will disable RAFOS tracking.

TelemetryRetry(Minutes) —This determines the time period between attempts to successfully complete telemetry tasks after each profile. The valid range of the argument is 5 minutes to 6 hours.

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.

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)
ConnectTimeOut(60)
CpActivationP(1000)
DeepProfileDescentTime(300)
DeepProfilePistonPos(16)
DeepProfilePressure(2000)
DownTime(1440)
MaxLogKb(40)
ParkDescentTime(300)
ParkPistonPos(24)
ParkPressure(1000)
PnPCycleLen(1)
TelemetryRetry(15)
```

```
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 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.

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

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. Rafos tracking data: These data are collected by the rafos receiver during the park-phase of the mission cycle and used to implement Rafos tracking.
3. 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.
4. 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.
5. GPS fixes: After the profile is executed and the float reaches the surface, the location of the profile is determined via GPS.
6. 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 park-phase RAFOS-tracking samples.

Up to six listening-windows worth of RAFOS tracking data will be generated as shown below. After the "Rafos:" key, the next three fields are the date & time, the Unix Epoch, and the mission time of the entry of the tracking data into the file. The next two fields are the Unix Epoch when the RAFOS receiver was activated followed by the span (seconds) of the RAFOS listening window. Next are six pairs of correlation-height & time-quantum-index. The correlation height has a range of 0x00-0xfa with higher values indicating higher correlations. The time-quantum index can be used to determine the time when the RAFOS signal was detected. The listening window is quantized into 0.3075s intervals and the time-quantum index indicates which interval is associated with the correlation height.

```
                                                  |---- RAFOS ---|
       |------- date -----|   UnixEpoch   MTime  StartEpoch  Span  |-- Correlation height & time-quantum index --|
Rafos: Jun 30 2006 23:59:11  1151711951   38933  1151710202  1740  2D 0908 2C 0259 2A 08B7 29 0892 28 0792 28 1602
Rafos: Jul 01 2006 00:29:11  1151713751   40733  1151712002  1740  2B 0E62 29 0D25 28 0505 28 0716 28 1111 28 11AA
Rafos: Jul 01 2006 00:59:11  1151715551   42533  1151713802  1740  8E 0423 4D 0E55 34 098F 33 058E 2F 093D 2E 0EDC
Rafos: Jul 01 2006 01:29:11  1151717351   44333  1151715602  1740  5E 05C7 2E 0A34 2B 0FCF 2A 155A 29 1237 28 0122
Rafos: Jul 01 2006 01:59:11  1151719151   46133  1151717402  1740  4E 09CE 2A 058B 29 0E1D 29 0E80 29 0EF3 29 0F9A
Rafos: Jul 01 2006 02:29:11  1151720951   47933  1151719202  1740  64 1318 2C 05BE 2C 011A 2C 14F2 2B 029F 2B 010A
```

6.1.3 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
```

6.1.4 Format for high resolution PTS samples.

For continuously sampled data, 2-decibar bins are used for bin-averaging. These data are encoded as three 16-bit hex integers (PTS) followed by a 16-bit integer that represents the number of samples

in the bin:

```
# May 23 2006 07:32:46 Sbe41cpSerNo[0747] NSample[22445] NBin[879]
00000000000000[2]
13EF0A667C860a
13FD0A667C8503
14120A667C8403
14280A667C8303
143D0A667C8203
14520A667C8103
14660A667C8003
14780A667C7F03
148B0A667C7E03
14A10A667C7D03
[snippage...]
```

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. The special value 0xFF indicates that 255 or more samples were collected into the given bin.

For example, the encoding: 13EF0A667C860A represents a bin with (0x0A=) 10 samples where the mean pressure was 31.8dbars, the mean temperature was 2.662C, and the mean salinity was 31.878PSU. 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 this: 000000000000000000[2] indicates that there were 2 lines with the same encoding....all zeros in this case.

6.1.5 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.
```

6.1.6 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 beyond the scope of this manual.

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 (5046.024.msg) from profile 12 of UW float 5046. 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:    Jun 26 2006 03:10:30  1151291430   14414  991.59  4.0918
ParkPt:    Jun 26 2006 04:10:27  1151295027   18011  987.50  4.0859
ParkPt:    Jun 26 2006 05:10:27  1151298627   21611  989.53  4.0598
ParkPt:    Jun 26 2006 06:10:27  1151302227   25211  988.16  4.0589
[snippage...]
```

```
ParkPt:    Jun 30 2006 13:10:27  1151673027  396011 1005.89  4.0086
ParkPt:    Jun 30 2006 14:10:27  1151676627  399611 1001.43  3.9742
ParkPt:    Jun 30 2006 15:10:27  1151680227  403211 1002.69  3.9785
ParkPt:    Jun 30 2006 16:10:27  1151683827  406811 1009.39  3.9592
$ Profile 5046.012 terminated: Sat Jul  1 05:24:26 2006
$ Discrete samples: 21
$       p        t        s  ofreq
  1006.08   4.0045  34.4462   3988 (Park Sample)
  1948.37   2.1426  34.6166   4686
  1898.02   2.2092  34.6122   4663
  1848.83   2.2576  34.6059   4624
  1799.27   2.3232  34.6036   4606
  1749.56   2.3943  34.5971   4568
  1699.28   2.4802  34.5900   4543
  1649.05   2.5484  34.5831   4498
  1599.70   2.6316  34.5771   4463
  1548.39   2.7273  34.5705   4448
  1499.14   2.8316  34.5623   4401
  1448.57   2.9358  34.5545   4359
  1399.18   3.0138  34.5485   4330
  1349.62   3.1507  34.5393   4291
  1299.39   3.2498  34.5321   4270
  1248.94   3.3528  34.5228   4233
  1199.50   3.4601  34.5166   4198
  1148.61   3.6139  34.5115   4185
  1098.87   3.8241  34.5062   4126
  1049.07   3.8297  34.4713   4085
   997.77   3.9723  34.4435   3972
# Jul 01 2006 05:29:58 Sbe41cpSerNo[1140] NSample[9536] NBin[489]
00000000000000[2]
002B6417888F12
003B6417888F19
00506417888F15
0064641A888F11
0078641B888F0F
008C641B888F10
00A06419888F0E
00B46418888E0F
[snippage...]
256C1024868014
25801021868013
2594101E868013
25A81016868013
25BC1007867F13
25D00FF7867D12
25E40FE5867C13
25F80FE2867C13
260C0FD6867D12
261F0FC3867E12
# GPS fix obtained in 162 seconds.
#         lon     lat mm/dd/yyyy hhmmss nsat
Fix: -158.230  24.147 07/01/2006 054110    7
```

```
Apf9iFwRev=051905
ActiveBallastAdjustments=2
AirBladderPressure=126
AirPumpAmps=77
AirPumpVolts=191
BuoyancyPumpOnTime=1937
BuoyancyPumpAmps=213
BuoyancyPumpVolts=172
CurrentPistonPosition=216
DeepProfilePistonPosition=21
GpsFixTime=162
FloatId=5046
ParkDescentPCnt=5
ParkDescentP[0]=5
ParkDescentP[1]=47
ParkDescentP[2]=80
ParkDescentP[3]=96
ParkDescentP[4]=99
ParkPistonPosition=76
ParkObs={ 1006.1dbar,    4.005C,  34.4462PSU,  3988}
ProfileId=012
ObsIndex=20
QuiescentAmps=7
QuiescentVolts=196
RtcSkew=7
Sbe41cpAmps=37
Sbe41cpVolts=184
Sbe41cpStatus=0x0000
status=0x0001
SurfacePistonPosition=194
SurfacePressure=0.13
Vacuum=78

<EOT>
```

B Sample: Decoded and processed data.

The following is an example of decoded and processed data for profile 12 of UW float 5048. The blocks with park-phase PT data and high resolution PTS data have been snipped to save space.

```
$  APEX-Seabird (051906) Iridium Message Parser & Calibration Applicator [SwiftWare]
$  $Revision: 1.1 $  $Date: 2006/11/03 19:08:57 $
$  Cmd Line: /www/argo/bin/ApexSbeIr051906-parser if=5046.012.msg of=5046.012.edf
$
$ E    lat     lon       date   time  zbot  zmax sh co           stnid      n
$ H  24.15 -158.23 07/01/2006  5.686     *  1948  *  *        5046.012    508
$
$  Processed engineering data:
$  BatteryVoltage=15.7V  VoltCount=196
$  AirBladderPressure=7.1"Hg  AirBladderPressureCount=126
```

```
$  Vacuum=-7.0"Hg  VacuumCount=78
$  BottomPistonPosition=21
$  IAirPump=295mA  AmpCount=77
$  IHighPressurePump=844mA  AmpCount=213
$  IQuiescent=13.0mA  AmpCount=7
$  ISbe41cp=134mA  AmpCount=37
$  VAirPump=15.3V  VoltCount=191
$  VHighPressurePump=13.8V  VoltCount=172
$  VLoad=13.8V  VoltCount=172
$  VQuiescent=15.7V  VoltCount=196
$  VSbe41cp=14.7V  VoltCount=184
$  PumpMotorSeconds=1937 sec
$  DeepProfilePistonPosition=21
$  ParkPistonPosition=76
$  SurfacePistonPosition=194
$  SurfacePressure=0 dbar
$  ProfileTermination=0x0001 : Deep profile.
$  Sbe41cpStatus=0x0000
$
$  Raw engineering meta data:
$  ActiveBallastAdjustments=2
$  AirBladderPressure=126
$  AirPumpAmps=77
$  AirPumpVolts=191
$  Apf9iFwRev=051906
$  BuoyancyPumpAmps=213
$  BuoyancyPumpOnTime=1937
$  BuoyancyPumpVolts=172
$  CurrentPistonPosition=216
$  DeepProfilePistonPosition=21
$  FloatId=5046
$  GpsFixTime=162
$  ObsIndex=20
$  ParkDescentPCnt=5
$  ParkDescentP[0]=5
$  ParkDescentP[1]=47
$  ParkDescentP[2]=80
$  ParkDescentP[3]=96
$  ParkDescentP[4]=99
$  ParkObs={ 1006.1dbar,    4.005C,  34.4462PSU,  3988}
$  ParkPistonPosition=76
$  ProfileId=012
$  QuiescentAmps=7
$  QuiescentVolts=196
$  RtcSkew=7
$  Sbe41cpAmps=37
$  Sbe41cpStatus=0x0000
$  Sbe41cpVolts=184
$  SurfacePistonPosition=194
$  SurfacePressure=0.13
$  Vacuum=78
$  status=0x0001
```

```
$
$  NFix=1   //      lon     lat Julian-sec       date   hour nsat FixTime
$  Fix(First): -158.230  24.147 1151761270 07-01-2006  5.686    7     162
$ F %6.1f %6.3f %6.3f %6.3f %6.3f %5.1f %4u
$ T    p       t       s   theta   sigma  speed    n
     4.3  25.623  34.959  25.622  23.122   11.1   18
     5.9  25.623  34.959  25.622  23.122    8.0   25
     8.0  25.623  34.959  25.621  23.122    9.5   21
    10.0  25.626  34.959  25.624  23.121   11.8   17
    12.0  25.627  34.959  25.624  23.121   13.3   15
    14.0  25.627  34.959  25.624  23.121   12.5   16
    16.0  25.625  34.959  25.621  23.122   14.3   14
    18.0  25.624  34.958  25.620  23.121   13.3   15
    20.0  25.623  34.958  25.619  23.122   11.8   17
[snippage...]
   956.0   4.137  34.432   4.064  27.328   10.5   19
   958.0   4.132  34.432   4.059  27.328   10.0   20
   960.0   4.129  34.432   4.056  27.329   10.5   19
   962.0   4.126  34.432   4.053  27.329   10.5   19
   964.0   4.118  34.432   4.045  27.330   10.5   19
   966.0   4.103  34.431   4.029  27.331   10.5   19
   968.0   4.087  34.429   4.013  27.331   11.1   18
   970.0   4.069  34.428   3.995  27.332   10.5   19
   972.0   4.066  34.428   3.992  27.332   10.5   19
   974.0   4.054  34.429   3.980  27.334   11.1   18
   975.9   4.035  34.430   3.961  27.337   11.1   18
   997.8   3.972  34.444   3.897  27.354      *    1
  1006.1   4.004  34.446   3.928  27.353      *    1
  1049.1   3.830  34.471   3.751  27.391      *    1
  1098.9   3.824  34.506   3.742  27.420      *    1
  1148.6   3.614  34.512   3.529  27.445      *    1
[snippage...]
  1799.3   2.323  34.604   2.199  27.638      *    1
  1848.8   2.258  34.606   2.130  27.646      *    1
  1898.0   2.209  34.612   2.078  27.655      *    1
  1948.4   2.143  34.617   2.008  27.664      *    1
$
$  Park-phase PT time-series: 110 samples
$                  UnixEpoch  MTime       p       t  |-------- Date --------|
$  ParkPt [001]:  1151291430  14414   991.6  4.0918  Sun Jun 25 20:10:30 2006
$  ParkPt [002]:  1151295027  18011   987.5  4.0859  Sun Jun 25 21:10:27 2006
$  ParkPt [003]:  1151298627  21611   989.5  4.0598  Sun Jun 25 22:10:27 2006
$  ParkPt [004]:  1151302227  25211   988.2  4.0589  Sun Jun 25 23:10:27 2006
$  ParkPt [005]:  1151305827  28811  1002.8  4.0130  Mon Jun 26 00:10:27 2006
[snippage...]
$  ParkPt [108]:  1151676627 399611  1001.4  3.9742  Fri Jun 30 07:10:27 2006
$  ParkPt [109]:  1151680227 403211  1002.7  3.9785  Fri Jun 30 08:10:27 2006
$  ParkPt [110]:  1151683827 406811  1009.4  3.9592  Fri Jun 30 09:10:27 2006
```

C Sample: Iridium engineering log file.

The following is an example engineering log file (5048.012.log) from profile 24 of UW Iridium float 5048. You will note that the log file starts with entries of the telemetry of profile 11 and then continues with entries of the execution of profile 12. 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 11 is included in the engineering logs of profile cycle 12.

```
(Jun 25 2006 22:55:36,  443840 sec) TelemetryInit()      Profile 11. (Apf9i FwRev: 051906)
(Jun 25 2006 23:03:07,  444291 sec) AirSystem()          Battery [187cnt, 14.5V]  Current [86cnt, 34.7mA]  Barometer [126cnt, 6.5"Hg]  Run-Time [61s]
(Jun 25 2006 23:03:32,  444316 sec) GpsServices()        GPS almanac is current.
(Jun 25 2006 23:03:32,  444316 sec) GpsServices()        Initiating GPS fix acquisition.
(Jun 25 2006 23:03:48,  444332 sec) gga()                $GPGGA,230558,2359.5884,N,15847.6288,W,0,00,,,M,,M,,*41
(Jun 25 2006 23:03:58,  444342 sec) gga()                $GPGGA,230608,2359.5884,N,15847.6288,W,0,00,,,M,,M,,*47
(Jun 25 2006 23:04:08,  444352 sec) gga()                $GPGGA,230618,2359.5884,N,15847.6288,W,0,00,,,M,,M,,*46
(Jun 25 2006 23:04:18,  444362 sec) gga()                $GPGGA,230628,2406.2507,N,15838.8831,W,0,00,,,M,,M,,*47
(Jun 25 2006 23:04:43,  444387 sec) gga()                $GPGGA,230410,2406.2506,N,15838.8835,W,1,07,1.3,,M,-0.7,M,,*65
(Jun 25 2006 23:04:43,  444387 sec) GpsServices()        Profile 11 GPS fix obtained in 71 seconds.
(Jun 25 2006 23:04:43,  444387 sec) GpsServices()                 lon     lat mm/dd/yyyy hhmmss nsat
(Jun 25 2006 23:04:43,  444387 sec) GpsServices()        Fix: -158.648  24.104 06/25/2006 230410    7
(Jun 25 2006 23:05:10,  444414 sec) gga()                $GPGGA,230440,2406.2501,N,15838.8891,W,1,08,1.0,,M,-0.7,M,,*65
(Jun 25 2006 23:05:16,  444420 sec) gga()                $GPGGA,230510,2406.2486,N,15838.8949,W,1,08,1.1,,M,-0.7,M,,*6A
(Jun 25 2006 23:05:16,  444420 sec) GpsServices()        APF9 RTC skew (6s) OK.
(Jun 25 2006 23:05:16,  444420 sec) LogNmeaSentences()   ,,M,9.4,M*0B
(Jun 25 2006 23:05:16,  444420 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Jun 25 2006 23:05:16,  444420 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Jun 25 2006 23:05:25,  444430 sec) LogNmeaSentences()   $GPRMC,230520,A,2406.2486,N,15838.8974,W,001.9,293.5,250606,010.2,E*6E
(Jun 25 2006 23:05:26,  444430 sec) LogNmeaSentences()   $GPGGA,230520,2406.2486,N,15838.8974,W,1,08,1.0,,M,-0.7,M,,*66
(Jun 25 2006 23:05:27,  444431 sec) LogNmeaSentences()   $GPGSA,M,2,02,04,08,,13,,,20,23,24,27,,1.4,1.0,*19
(Jun 25 2006 23:05:27,  444431 sec) LogNmeaSentences()   $GPGSV,3,3,11,23,29,049,43,24,55,228,50,27,75,239,47*4E
(Jun 25 2006 23:05:27,  444431 sec) LogNmeaSentences()   $PGRME,4.2,M,,M,9.4,M*0B
(Jun 25 2006 23:05:28,  444432 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Jun 25 2006 23:05:28,  444432 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Jun 25 2006 23:05:35,  444440 sec) LogNmeaSentences()   $GPRMC,230530,A,2406.2482,N,15838.8990,W,001.5,262.4,250606,010.2,E*62
(Jun 25 2006 23:05:36,  444440 sec) LogNmeaSentences()   $GPGGA,230530,2406.2482,N,15838.8990,W,1,08,1.0,,M,-0.7,M,,*69
(Jun 25 2006 23:05:37,  444441 sec) LogNmeaSentences()   $GPGSA,M,2,02,04,08,,13,,,20,23,24,27,,1.4,1.0,*19
(Jun 25 2006 23:05:37,  444441 sec) LogNmeaSentences()   $GPGSV,3,1,11,02,22,309,45,04,48,277,48,08,42,218,47,11,00,158,00*7E
(Jun 25 2006 23:05:37,  444441 sec) LogNmeaSentences()   $PGRME,4.2,M,,M,9.4,M*0B
(Jun 25 2006 23:05:38,  444442 sec) LogNmeaSentences()   $PGRMB,0.0,200,,,,K,,N,N*31
(Jun 25 2006 23:05:38,  444442 sec) LogNmeaSentences()   $PGRMM,WGS 84*06
(Jun 25 2006 23:05:45,  444450 sec) LogNmeaSentences()   $GPRMC,230540,A,2406.2481,N,15838.9009,W,001.8,267.1,250606,010.2,E*63
(Jun 25 2006 23:05:46,  444450 sec) LogNmeaSentences()   $GPGGA,230540,2406.2481,N,15838.9009,W,1,08,1.0,,M,-0.7,M,,*65
(Jun 25 2006 23:05:47,  444451 sec) LogNmeaSentences()   $GPGSA,M,2,02,04,08,,13,,,20,23,24,27,,1.4,1.0,*19
(Jun 25 2006 23:05:47,  444451 sec) LogNmeaSentences()   $GPGSV,3,2,11,13,47,010,48,16,03,044,37,17,00,219,00,20,35,114,47*7B
(Jun 25 2006 23:05:47,  444452 sec) LogNmeaSentences()   $PGRME,4.2,M,,M,9.4,M*0B
(Jun 25 2006 23:05:48,  444452 sec) LogNmeaSentences()   $PGRMT,GPS 15-L Ver. 2.70,P,P,R,R,P,,19,R,*23
(Jun 25 2006 23:05:48,  444452 sec) GpsServices()        GPS services complete.
(Jun 25 2006 23:06:12,  444476 sec) CLogin()             Connecting to primary host.
(Jun 25 2006 23:06:29,  444493 sec) CLogin()             Connection 1 established in 17 seconds.
(Jun 25 2006 23:06:42,  444506 sec) login()              Login successful.
(Jun 25 2006 23:06:44,  444508 sec) CLogin()             Logged in to host.  [Login required 15 seconds]
(Jun 25 2006 23:06:46,  444511 sec) RxConfig()           Downloading "mission.cfg" from host.
(Jun 25 2006 23:06:47,  444511 sec) Rx()                 Initiating transfer. [0x43]
(Jun 25 2006 23:06:51,  444515 sec) Rx()                 Pad character [0x1a] found in ascii mode - truncating packet.
(Jun 25 2006 23:06:55,  444519 sec) Rx()                 Received EOT - transfer complete. [1 packets, 1024 bytes, 8 sec, 128.0 bps]
(Jun 25 2006 23:06:55,  444519 sec) RxConfig()           Download successful.
(Jun 25 2006 23:06:55,  444519 sec) WriteVitals()        Writing vitals to "5046.011.msg".
(Jun 25 2006 23:06:56,  444521 sec) UpLoadFile()         Uploading "5046.011.msg" to host as "5046.011.msg".
(Jun 25 2006 23:07:00,  444524 sec) Tx()                 CRC negotiation successful. [16-bit CRC]
(Jun 25 2006 23:08:25,  444609 sec) Tx()                 Transmission completed successfully [18 packets, 18106 bytes, 85 sec, 216.8 bps]
(Jun 25 2006 23:08:25,  444609 sec) UpLoadFile()         Upload successful.
(Jun 25 2006 23:08:26,  444610 sec) UpLoadFile()         Uploading "5046.011.log" to host as "5046.011.log".
(Jun 25 2006 23:08:30,  444614 sec) Tx()                 CRC negotiation successful. [16-bit CRC]
(Jun 25 2006 23:09:56,  444700 sec) Tx()                 Transmission completed successfully [18 packets, 17643 bytes, 86 sec, 214.3 bps]
(Jun 25 2006 23:09:56,  444700 sec) UpLoadFile()         Upload successful.
(Jun 25 2006 23:09:56,  444700 sec) UpLoad()             Files successfully uploaded: 2
(Jun 25 2006 23:09:56,  444700 sec) UpLoad()             Upload complete.
(Jun 25 2006 23:09:57,  444701 sec) logout()             Log-out successful.
(Jun 25 2006 23:09:58,  444703 sec) Telemetry()          Telemetry cycle complete: PrfId=11 ConnectionAttempts=1  Connections=1
(Jun 25 2006 23:09:59,  444703 sec) TelemetryTerminate() Parsing new mission configuration.
(Jun 25 2006 23:09:59,  444703 sec) configure()          Parsing configurators in "mission.cfg".
(Jun 25 2006 23:10:04,  444709 sec) configure()             TelemetryRetry(15)          [0x7BBC] [TelemetryRetry(15)].
(Jun 25 2006 23:10:10,  444714 sec) configure()             PnPCycleLen(3)              [0xB1B2] [PnpCycleLength(3)].
(Jun 25 2006 23:10:16,  444720 sec) configure()             DownTime(7200)              [0xBC7F] [DownTime(7200)].
(Jun 25 2006 23:10:16,  444720 sec) configure()          Configuration CRCs and syntax OK.
(Jun 25 2006 23:10:16,  444720 sec) ConfigSupervisor()   Configuration accepted.
(Jun 25 2006 23:10:16,  444720 sec) TelemetryTerminate() Reconditioning the file system.
(Jun 25 2006 23:10:16,       1 sec) DescentInit()        Deep profile 12 initiated at mission-time 444720sec.
(Jun 25 2006 23:10:19,       3 sec) DescentInit()        Surface pressure: 0.1dbars.
(Jun 25 2006 23:10:24,       8 sec) PistonMoveAbsWTO()    210->076 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191 190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 1
(Jun 25 2006 23:21:17,     661 sec) Descent()            Pressure: 53.2
(Jun 26 2006 00:10:20,    3604 sec) Descent()            Pressure: 465.3
(Jun 26 2006 01:10:20,    7204 sec) Descent()            Pressure: 802.5
(Jun 26 2006 02:10:20,   10804 sec) Descent()            Pressure: 961.9
(Jun 26 2006 03:10:20,   14404 sec) Descent()            Pressure: 991.5
```

```
(Jun 26 2006 03:10:20,   14404 sec) ParkInit()
(Jun 26 2006 06:10:27,   25211 sec) Park()                ParkPOutOfBand[-3, 988.2 dbars]: retract piston.
(Jun 26 2006 06:10:27,   25212 sec) PistonMoveAbsWTO()     076->075 075 [4sec, 14.8Volts, 0.129Amps, CPT:647sec]
(Jun 28 2006 03:10:27,  187211 sec) Park()                ParkPOutOfBand[3, 1010.8 dbars]: extend piston.
(Jun 28 2006 03:10:27,  187212 sec) PistonMoveAbsWTO()     075->076 076 [1sec, 14.3Volts, 0.544Amps, CPT:648sec]
(Jun 30 2006 17:10:30,  410414 sec) CtdPower()            [ 1006.09,  4.0043, 34.4461]
(Jun 30 2006 17:10:30,  410414 sec) CtdPower()            CTD Power consumption [184VCnt 37ACnt]: 14.263Volts * 0.149Amps = 2.13Watts.
(Jun 30 2006 17:10:30,  410415 sec) ParkTerminate()       Piston Position:76 Vacuum:78 Vq:196 Aq:7 Vsbe:184 Asbe:37
(Jun 30 2006 17:10:49,  410433 sec) ParkTerminate()       PTS: 1006.1dbars 4.0045C 34.4462PSU
(Jun 30 2006 17:10:49,  410433 sec) GoDeepInit()          Moving piston.
(Jun 30 2006 17:10:49,  410434 sec) PistonMoveAbsWTO()     076->022 075 074 073 072 071 070 069 068 067 066 065 064 063 062 061 060 059 058 057 056 055 054 053 052 051 050 049 048 047 046 045 044 043 042 0
(Jun 30 2006 23:10:20,  432004 sec) ProfileInit()         PrfId:012  Pressure:1991.8dbar  pTable[1]:1950dbar
(Jun 30 2006 23:10:23,  432007 sec) PistonMoveAbsWTO()     022->044 023 024 025 026 027 [30sec, 13.9Volts, 0.850Amps, CPT:929sec]
(Jun 30 2006 23:11:03,  432047 sec) PistonMoveAbsWTO()     027->044 028 029 030 031 032 [30sec, 13.6Volts, 0.790Amps, CPT:959sec]
(Jun 30 2006 23:11:43,  432087 sec) PistonMoveAbsWTO()     032->044 033 034 035 036 037 [30sec, 13.4Volts, 0.842Amps, CPT:989sec]
(Jun 30 2006 23:12:23,  432127 sec) PistonMoveAbsWTO()     037->044 038 039 040 041 [30sec, 13.3Volts, 0.822Amps, CPT:1019sec]
(Jun 30 2006 23:13:03,  432167 sec) PistonMoveAbsWTO()     042->044 043 044 [13sec, 13.3Volts, 0.858Amps, CPT:1032sec]
(Jun 30 2006 23:17:47,  432452 sec) Profile()             Sample 0 initiated at 1950.3dbars for bin 1 [1950dbars].  PTS: 1948.4dbars 2.1426C 34.6166PSU
(Jun 30 2006 23:25:09,  432894 sec) Profile()             Sample 1 initiated at 1899.9dbars for bin 2 [1900dbars].  PTS: 1898.0dbars 2.2092C 34.6122PSU
(Jun 30 2006 23:32:57,  433362 sec) Profile()             Sample 2 initiated at 1850.5dbars for bin 3 [1850dbars].  PTS: 1848.8dbars 2.2576C 34.6059PSU
(Jun 30 2006 23:41:45,  433890 sec) Profile()             Sample 3 initiated at 1800.8dbars for bin 4 [1800dbars].  PTS: 1799.3dbars 2.3232C 34.6036PSU
(Jun 30 2006 23:51:53,  434498 sec) Profile()             Sample 4 initiated at 1750.8dbars for bin 5 [1750dbars].  PTS: 1749.6dbars 2.3943C 34.5971PSU
(Jun 30 2006 23:55:46,  434731 sec) AscentControlAgent() Bouyancy nudge to 54 (v=0.072dbar/sec).
(Jun 30 2006 23:55:47,  434731 sec) PistonMoveAbsWTO()     044->054 045 046 047 048 [30sec, 13.7Volts, 0.745Amps, CPT:1062sec]
(Jun 30 2006 23:56:27,  434771 sec) PistonMoveAbsWTO()     049->054 050 051 052 053 054 [30sec, 13.6Volts, 0.729Amps, CPT:1092sec]
(Jul 01 2006 00:01:31,  435076 sec) Profile()             Sample 5 initiated at 1700.9dbars for bin 6 [1700dbars].  PTS: 1699.3dbars 2.4802C 34.5900PSU
(Jul 01 2006 00:10:25,  435610 sec) Profile()             Sample 6 initiated at 1650.4dbars for bin 7 [1650dbars].  PTS: 1649.0dbars 2.5484C 34.5831PSU
(Jul 01 2006 00:19:52,  436177 sec) Profile()             Sample 7 initiated at 1601.0dbars for bin 8 [1600dbars].  PTS: 1599.7dbars 2.6316C 34.5771PSU
(Jul 01 2006 00:26:08,  436553 sec) AscentControlAgent() Bouyancy nudge to 64 (v=0.079dbar/sec).
(Jul 01 2006 00:26:09,  436553 sec) PistonMoveAbsWTO()     054->064 055 056 057 058 059 [30sec, 13.8Volts, 0.725Amps, CPT:1122sec]
(Jul 01 2006 00:26:49,  436593 sec) PistonMoveAbsWTO()     059->064 060 061 062 063 064 [29sec, 13.6Volts, 0.669Amps, CPT:1151sec]
(Jul 01 2006 00:29:39,  436764 sec) Profile()             Sample 8 initiated at 1550.1dbars for bin 9 [1550dbars].  PTS: 1548.4dbars 2.7273C 34.5705PSU
(Jul 01 2006 00:37:36,  437241 sec) Profile()             Sample 9 initiated at 1500.7dbars for bin 10 [1500dbars].  PTS: 1499.1dbars 2.8316C 34.5623PSU
(Jul 01 2006 00:46:10,  437755 sec) Profile()             Sample 10 initiated at 1450.1dbars for bin 11 [1450dbars].  PTS: 1448.6dbars 2.9358C 34.5545PSU
(Jul 01 2006 00:54:57,  438282 sec) Profile()             Sample 11 initiated at 1400.6dbars for bin 12 [1400dbars].  PTS: 1399.2dbars 3.0138C 34.5485PSU
(Jul 01 2006 01:04:43,  438867 sec) Profile()             Sample 12 initiated at 1350.8dbars for bin 13 [1350dbars].  PTS: 1349.6dbars 3.1507C 34.5393PSU
(Jul 01 2006 01:06:36,  438981 sec) AscentControlAgent() Bouyancy nudge to 74 (v=0.079dbar/sec).
(Jul 01 2006 01:06:37,  438981 sec) PistonMoveAbsWTO()     064->074 065 066 067 068 069 [30sec, 13.9Volts, 0.616Amps, CPT:1181sec]
(Jul 01 2006 01:07:17,  439021 sec) PistonMoveAbsWTO()     069->074 070 071 072 073 074 [27sec, 13.7Volts, 0.584Amps, CPT:1208sec]
(Jul 01 2006 01:13:07,  439372 sec) Profile()             Sample 13 initiated at 1301.0dbars for bin 14 [1300dbars].  PTS: 1299.4dbars 3.2498C 34.5321PSU
(Jul 01 2006 01:21:50,  439894 sec) Profile()             Sample 14 initiated at 1250.3dbars for bin 15 [1250dbars].  PTS: 1248.9dbars 3.3528C 34.5228PSU
(Jul 01 2006 01:31:45,  440490 sec) Profile()             Sample 15 initiated at 1200.6dbars for bin 16 [1200dbars].  PTS: 1199.5dbars 3.4601C 34.5166PSU
(Jul 01 2006 01:31:58,  440503 sec) AscentControlAgent() Bouyancy nudge to 84 (v=0.079dbar/sec).
(Jul 01 2006 01:31:59,  440503 sec) PistonMoveAbsWTO()     074->084 075 076 077 078 079 [30sec, 13.9Volts, 0.588Amps, CPT:1238sec]
(Jul 01 2006 01:32:39,  440543 sec) PistonMoveAbsWTO()     079->084 080 081 082 083 084 [27sec, 13.7Volts, 0.564Amps, CPT:1265sec]
(Jul 01 2006 01:39:58,  440982 sec) Profile()             Sample 16 initiated at 1150.2dbars for bin 17 [1150dbars].  PTS: 1148.6dbars 3.6139C 34.5115PSU
(Jul 01 2006 01:48:03,  441467 sec) Profile()             Sample 17 initiated at 1100.2dbars for bin 18 [1100dbars].  PTS: 1098.9dbars 3.8241C 34.5062PSU
(Jul 01 2006 01:56:55,  441999 sec) Profile()             Sample 18 initiated at 1050.2dbars for bin 19 [1050dbars].  PTS: 1049.1dbars 3.8297C 34.4713PSU
(Jul 01 2006 02:07:14,  442619 sec) AscentControlAgent() Bouyancy nudge to 94 (v=0.070dbar/sec).
(Jul 01 2006 02:07:15,  442619 sec) PistonMoveAbsWTO()     084->094 085 086 087 088 089 [30sec, 14.0Volts, 0.516Amps, CPT:1295sec]
(Jul 01 2006 02:08:13,  442677 sec) Profile()             Sample 19 initiated at 998.9dbars for bin 20 [1000dbars].  PTS: 997.8dbars 3.9723C 34.4435PSU
(Jul 01 2006 02:08:14,  442678 sec) PistonMoveAbsWTO()     089->094 090 091 092 093 094 [25sec, 13.8Volts, 0.520Amps, CPT:1320sec]
(Jul 01 2006 02:11:43,  442887 sec) Sbe41cpStartCP()      Continuous profile started.
(Jul 01 2006 02:37:15,  444420 sec) AscentControlAgent() Bouyancy nudge to 104 (v=0.076dbar/sec).
(Jul 01 2006 02:37:16,  444420 sec) PistonMoveAbsWTO()     094->104 095 096 097 098 099 [30sec, 13.9Volts, 0.467Amps, CPT:1350sec]
(Jul 01 2006 02:37:55,  444460 sec) PistonMoveAbsWTO()     099->104 100 101 102 103 104 [25sec, 13.7Volts, 0.479Amps, CPT:1375sec]
(Jul 01 2006 03:03:51,  446016 sec) AscentControlAgent() Bouyancy nudge to 114 (v=0.076dbar/sec).
(Jul 01 2006 03:03:52,  446016 sec) PistonMoveAbsWTO()     104->114 105 106 107 108 109 [30sec, 13.9Volts, 0.427Amps, CPT:1405sec]
(Jul 01 2006 03:04:31,  446056 sec) PistonMoveAbsWTO()     109->114 110 111 112 113 114 [23sec, 13.8Volts, 0.431Amps, CPT:1428sec]
(Jul 01 2006 03:25:20,  447304 sec) AscentControlAgent() Bouyancy nudge to 124 (v=0.076dbar/sec).
(Jul 01 2006 03:25:21,  447305 sec) PistonMoveAbsWTO()     114->124 115 116 117 118 119 [30sec, 14.0Volts, 0.379Amps, CPT:1458sec]
(Jul 01 2006 03:26:01,  447346 sec) PistonMoveAbsWTO()     119->124 120 121 122 123 124 [22sec, 13.9Volts, 0.387Amps, CPT:1480sec]
(Jul 01 2006 03:46:48,  448592 sec) AscentControlAgent() Bouyancy nudge to 134 (v=0.080dbar/sec).
(Jul 01 2006 03:46:48,  448593 sec) PistonMoveAbsWTO()     124->134 125 126 127 128 129 [30sec, 14.0Volts, 0.347Amps, CPT:1510sec]
(Jul 01 2006 03:47:29,  448633 sec) PistonMoveAbsWTO()     129->134 130 131 132 133 134 [23sec, 14.0Volts, 0.351Amps, CPT:1533sec]
(Jul 01 2006 04:08:15,  449880 sec) AscentControlAgent() Bouyancy nudge to 144 (v=0.073dbar/sec).
(Jul 01 2006 04:08:16,  449880 sec) PistonMoveAbsWTO()     134->144 135 136 137 138 139 [30sec, 14.1Volts, 0.310Amps, CPT:1563sec]
(Jul 01 2006 04:08:55,  449920 sec) PistonMoveAbsWTO()     140->144 141 142 143 144 [21sec, 14.0Volts, 0.306Amps, CPT:1584sec]
(Jul 01 2006 04:29:42,  451166 sec) AscentControlAgent() Bouyancy nudge to 154 (v=0.077dbar/sec).
(Jul 01 2006 04:29:42,  451167 sec) PistonMoveAbsWTO()     144->154 145 146 147 148 149 150 [30sec, 14.2Volts, 0.266Amps, CPT:1614sec]
(Jul 01 2006 04:30:23,  451207 sec) PistonMoveAbsWTO()     150->154 151 152 153 154 [20sec, 14.1Volts, 0.270Amps, CPT:1634sec]
(Jul 01 2006 04:51:08,  452453 sec) AscentControlAgent() Bouyancy nudge to 164 (v=0.066dbar/sec).
(Jul 01 2006 04:51:09,  452453 sec) PistonMoveAbsWTO()     154->164 155 156 157 158 159 160 [30sec, 14.3Volts, 0.242Amps, CPT:1664sec]
(Jul 01 2006 04:51:48,  452492 sec) PistonMoveAbsWTO()     160->164 161 162 163 164 [19sec, 14.2Volts, 0.242Amps, CPT:1683sec]
(Jul 01 2006 05:02:20,  453125 sec) AscentControlAgent() Bouyancy nudge to 174 (v=0.068dbar/sec).
(Jul 01 2006 05:02:21,  453125 sec) PistonMoveAbsWTO()     164->174 165 166 167 168 169 170 [30sec, 14.3Volts, 0.230Amps, CPT:1713sec]
(Jul 01 2006 05:03:00,  453164 sec) PistonMoveAbsWTO()     170->174 171 172 173 174 [20sec, 14.3Volts, 0.226Amps, CPT:1733sec]
(Jul 01 2006 05:13:32,  453797 sec) AscentControlAgent() Bouyancy nudge to 184 (v=0.053dbar/sec).
(Jul 01 2006 05:13:33,  453797 sec) PistonMoveAbsWTO()     174->184 175 176 177 178 179 180 [30sec, 14.4Volts, 0.218Amps, CPT:1763sec]
(Jul 01 2006 05:14:12,  453837 sec) PistonMoveAbsWTO()     180->184 181 182 183 184 [19sec, 14.3Volts, 0.214Amps, CPT:1782sec]
(Jul 01 2006 05:18:32,  454097 sec) AscentControlAgent() Bouyancy nudge to 194 (v=0.053dbar/sec).
(Jul 01 2006 05:18:33,  454097 sec) PistonMoveAbsWTO()     184->194 185 186 187 188 189 190 [30sec, 14.3Volts, 0.210Amps, CPT:1812sec]
(Jul 01 2006 05:19:12,  454137 sec) PistonMoveAbsWTO()     190->194 191 192 193 194 [18sec, 14.3Volts, 0.210Amps, CPT:1830sec]
(Jul 01 2006 05:22:34,  454338 sec) SurfaceDetect()       SurfacePressure:0.1dbars Pressure:3.9dbars PistonPosition:194
(Jul 01 2006 05:22:35,  454339 sec) Sbe41cpStopCP()       Continuous profile stopped.
(Jul 01 2006 05:22:39,  454343 sec) PistonMoveAbsWTO()     194->216 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 [107sec, 14.3Volts, 0.177Amps, CPT:1937sec]
(Jul 01 2006 05:29:19,  454744 sec) Sbe41cpBinAverage()  Finished averaging 9536 samples in 282 seconds.
(Jul 01 2006 05:29:30,  454754 sec) Sbe41cpUploadCP()     Sbe41cpSerNo[1140] NSample[9536] NBin[489]
(Jul 01 2006 05:30:38,  454822 sec) Sbe41cpUploadCP()     Continuous profile uploaded [489 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;
}
```