RDA 8.02.2 Release Notes (12 Sep 2003)

These release notes cover changes made to the SIGMET Radar Data Acquisition platform, including primarily the RVP8 and RCP8 products. The last release was RDA-8.01.14 generated on 26 August 2003. If you are upgrading from an earlier release, please read those notes also.

Important Upgrade Notes

1. We have added a new configuration directory to RDA which holds GIF background images. The 8.02.2 version of When upgrading RDA, you need to do the following:

   - Add the following line to your ${IRIS_CONFIG}profile file. Place this just after the IRIS_DICTIONARY line:

     ```
     export IRIS_IMAGES="${IRIS_ROOT}/config/images/"
     ```

   - Create the new directory with:

     ```
     $ mkdir /usr/sigmet/config/images
     ```

   - Move GIF files from their previous locations. The background images for **bitex** are stored here now. Previously they were stored in the IRIS_OVERLAY directory. The filenames can be customized, so use the names for your system. Here is an example of the commands to do this:

     ```
     $ cd /usr/sigmet/config/overlay
     $ mv bitex_panel0.gif ../images/
     ```

RVP8 Bug Repairs

1. The documentation of the *frontPanelDisplay()* RVP8/Main function pointer has been improved, and a bug in the "V" command's text representation of the front panel display has been fixed.

2. The *–nofork* command line startup option was not working properly; the RVP8/Main would not complete its initializations after an RVP8/Proc process was manually started. This bug was introduced in V8.01.14.

3. The *rvp8tsSeqNumExpired()* function has been removed from the TimeSeries API because it was both confusing and unnecessary. If you have a trial pulse sequence number and need to check its status please use the following pair of tests:

   ```
   if( rvp8tsSeqNumOkay( TS, iTrialSeqNum ) )
     {
       The sequence number is valid; proceed with processing
     }
   else if( rvp8tsSeqNumOkay( TS, iTrialSeqNum–1 ) )
     {
       Wait a little while; the sequence number will soon be valid
     }
   else
     {
   ```

```
        The sequence number is bad; we're lost, get a new one
    }
```

The example code *rvp8ts_example.c* has been modified to work this way.

4. The *rvp8tsLateness()* function in the TimeSeries API now returns a value of zero (very early) when given a sequence number corresponding to the next pulse that will arrive. Previously the return value was 1.0 for such a pulse because it was not yet valid; but the correct value really should be zero. Many thanks to Darcy Saxion (ROC/RSIS) for pointing out this bug as well as the one described in #3.

5. The TimeSeries API functions that return the number of valid pulses on either side of a given sequence number have been renamed to make them more clear. These functions include the starting pulse in the overall returned count; but their old names, which used the words "After" and "Before", implied that the starting pulse was not included. Thus,

```
rvp8tsNumPulsesAfter()   -->  rvp8tsNumPulsesFwd()
rvp8tsNumPulsesBefore()  -->  rvp8tsNumPulsesRev()
```

RVP8 New Features

1. The *genericBinMoments()* iterator now takes an additional flag argument that gives you greater control over the allocation and scope of intermediate timeseries and spectral data. If you invoke the function with the *GBMF_SAVEIQ* or *GBMF_SAVESPEC* flags, then any timeseries and/or spectral data that are generated during the initial "binmoments" phase will remain available for all subsequent processing phases, as well as for the RVP8/Main threads when the RVP8/Proc computations are finished. You may access these preserved data using the *MMFIQ()* and *MMFSPEC()* addressing macros. There will be somewhat of a memory cache-miss penalty when preserving data in this manner, so only use these features if your algorithms require multiple passes over the timeseries data.

2. The TimeSeries API now keeps track of the auxiliary information (range mask, noise levels, major mode, etc.) that pertain to each group of pulses that were acquired in a mutually consistent manner, i.e., from within the same "ProcSection" of the RVP8. Given a pulse header, *rvp8tsGetPulseInfo()* will return a pointer to this additional information structure.

   This important capability has been missing from the API until now. The example code *rvp8ts_example.c* has also been modified to use it.

3. The *rvp8PulseHdr*, *rvp8PulseHdrRx*, and *rvp8PulseInfo* structures have been redefined as "public" structures, i.e., they will remain as constant as possible in all future releases. The new structures contain internal pads, roughly doubling the structure size, which permit adding new elements without disturbing old ones. The idea is that these public structures can be included directly in user's saved data without having to worry about them changing in the future.

4. The major mode function pointers in *rvp8main.h* and *rvp8proc.h* now document which ones take on default meanings when supplied as NULL (there are only two or three that behave this way).

5. The windowed power spectra that are computed within the RVP8 can now be rescaled according to the total power that was present in the original timeseries data segment that produced them. This prevents the increased variance that the windowing step would otherwise impose on the power estimates that are computed from the spectra. The element *struct dftConf.fReNormExp* controls the "strength" of renormalization; a value of 0.0 does not change the spectra at all, and 1.0 causes exact agreement with the original timeseries power.

6. The power spectra that are output by the RVP8 are no longer scaled to keep the peak height constant for a given pure tone regardless of the type of window being used. This behavior was inherited from the RVP5/6/7, and has always been confusing. The **ascope** utility has also been changed to know that the RVP8 now outputs true spectral terms.

7. A new class of spectral clutter filters are now available in the RVP8. These Gaussian Model Adaptive Processing (GMAP) filters are the most subtle and robust filters that we have ever produced, and we're hopeful that they will meet or exceed the NEXRAD Technical Requirements for legacy clutter filter performance. Reflectivity biases are easily kept to within ±1dB for Clutter-to-Signal ratios up to 40dB, SNR of +10dB at a spectral width of 4m/sec (±25m/sec Nyquist), and all possible velocities.

8. The RVP8/Main, RVP8/Proc and RCP8 are no longer built under a shared library model. This model was originally chosen because it seemed to provide the most flexibility for replacing individual pieces of each product. However, the difficulties with specifying dynamic library paths in a secure manner, along with the limitations on runtime profiling of shared objects, have led us back to a statically linked model. All of the original goals for flexibility are still available to RDA developers simply by rebuilding the OPEN trees each time.

Licensing Issues

1. The Intel Integrated Performance Primitives (IPP) library is used by the RVP8, and thus, the IPP runtime libraries are bundled into each RDA release. SIGMET has purchased a single-user developer's license from Intel which allows one engineer to develop code which links to the IPP library, and then to distribute an unlimited number of copies of that code in executable form. Our license does not, however, allow for any additional programmers to develop their own code as an extension of SIGMET's license.

   Essentially, the IPP license is a per-developer license. There are no restrictions or royalties on the distribution of compiled binaries, but each additional developer must be licensed at a cost of approximately $250/year. Some relevant text is included below from the Intel FAQ *http://www.intel.com/software/products/ipp/ipp30/faq_lic.htm*. The bottom line is that our community of RDA developers must individually license themselves with Intel in order to write new code that uses the IPP library. Perhaps there might be special licensing terms already in place allowing Government developers to use the Intel IPP.

   - ***What are the redistributable files?***
     In general, the redistributable files include the linkable files (.DLL and .LIB files

for Windows*, .SO files for Linux*) including the Runtime Installer. With your purchase of the Intel IPP product (and updates through the support service subscription), the redist.txt file outlines the list of files that can be redistributed.

- ***Do I need to buy an the Intel IPP license for each copy of our software that we sell?***
  No. there is no per copy royalty fee. Please check the Intel IPP end user license agreement for more details.
- ***How many copies of my company's application can redistribute the Intel IPP library files?***
  You may redistribute an unlimited number of copies of the files that are found in the directories defined in the Redistributables section of the end–user license agreement.
- ***Are there royalty fees in using the Intel IPP?***
  No, there is no royalty fee for redistributing the Intel IPP libraries with your software. By licensing the Intel IPP product for your developers, you have redistribution rights to distribute the Intel IPP library files with your software for an unlimited number of copies. For more information please refer to the end user license agreement.
- ***How many copies of the Intel IPP product do I need to secure for my project team or company?***
  The number of copies of the Intel IPP product needed is determined by the number of developers who are writing code compiling and testing using the Intel IPP API as well as the number of build machines involved in compiling and linking, thereby needing the full development tools file set of Intel IPP product.