Annex to Volume 3 - Sample Algorithm Source Code

Part A. Algorithm Structure Templates

These flow charts illustrate the guidance currently provided in this guide. Many legacy algorithms have not been updated to reflect the changes in abort services. NOTE: The logical structure contained in these charts represent just one method of complying with all required structure rules. They are not the only way an algorithm can be structured. The following flow charts are provided:

1. An algorithm reading base data (radial based) and outputting an elevation based product. This chart provides a framework for Sample Algorithm 1 - Digital Reflectivity algorithm, Sample Algorithm 2 - Radial Reflectivity algorithm, and the first task in Sample Algorithm 3 - Multiple Task algorithm.
2. An algorithm reading an elevation based product and outputting a volume based product. This chart provides a framework for the second task in Sample Algorithm 3 - Multiple Task algorithm
3. An algorithm reading base data (radial based) and outputting a volume based product. (No sample algorithm at this time).

Many of the named activities in these flow charts correspond to specific API services. For example, the flow chart for the elevation product can be compared to the source code of the digital reflectivity sample algorithm. Comment labels have been placed in the source code as an aid in relating the source code to the flow chart.

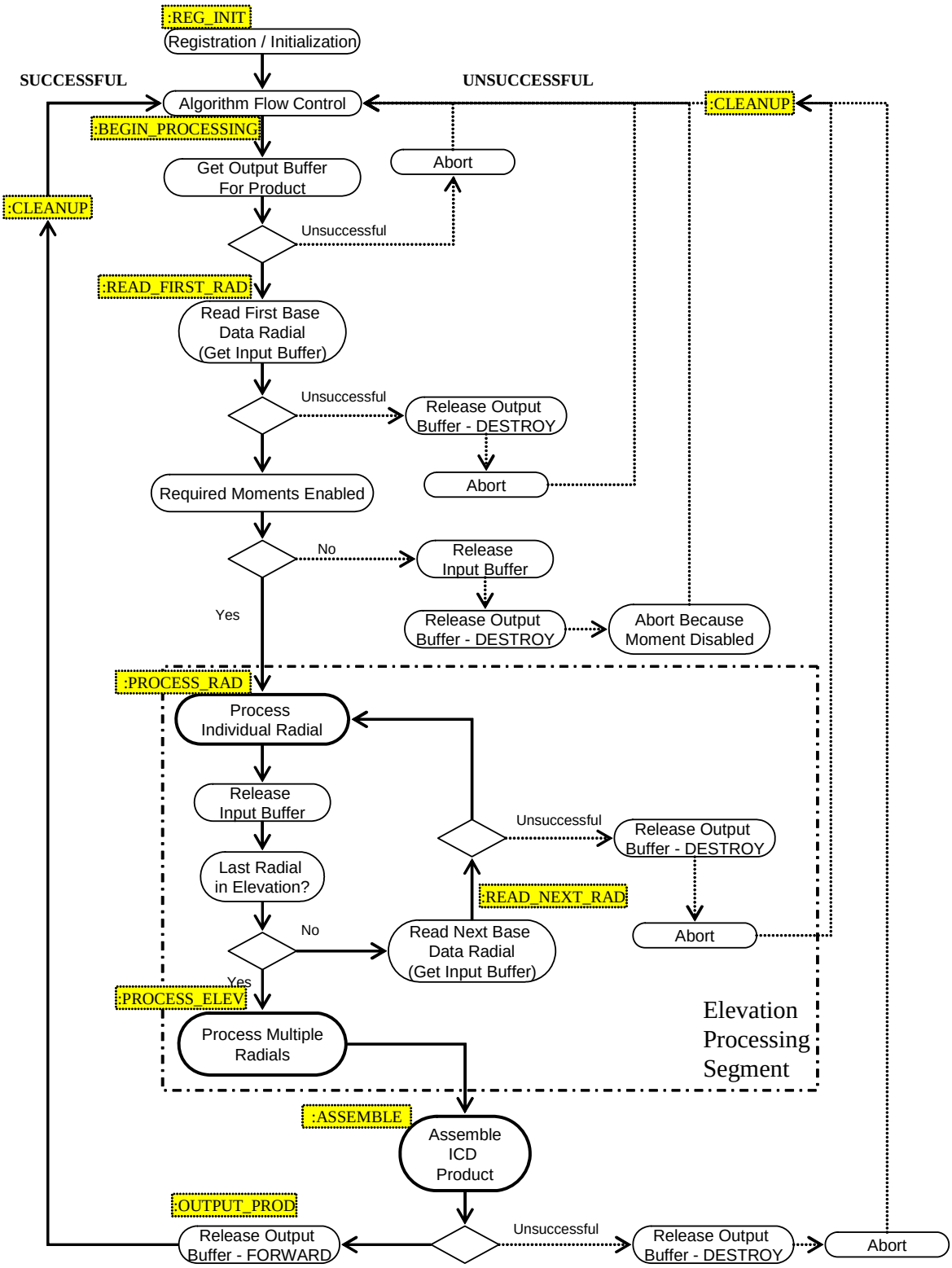
If the sample algorithms are not installed, existing operational algorithms can be used as examples.

Logic Demonstrated	Sample Algorithm	Operational Algorithm
Producing a 256 level elevation product from radial basedata.	Sample 1	cpc007/tsk013/ bref8bit
		cpc007/tsk014/ bvel8bit
		cpc007/tsk015/ superes8bit
Producing a16-level RLE elevation product from radial basedata.	Sample 2	cpc007/tsk001/ basrflct
		cpc007/tsk002/ basvcty
Producing a volume product from elevation basedata.	None	cpc013/tsk003/ epre
A multiple task algorithm with elevation intermediate products and volume final product.	Sample 3	cpc017/tsk009/ tda2d3d cpc018/tsk005/ tdaprod
A multiple task algorithm with final task using the WAIT_ANY form of control loop	Sample 4	cpc004/tsk006/ recclalg cpc004/tsk007/ recclprods
An event driven task.	None	future build

Updated April 2006

One Product Algorithm

Radial_Based Input Elevation_Based Output

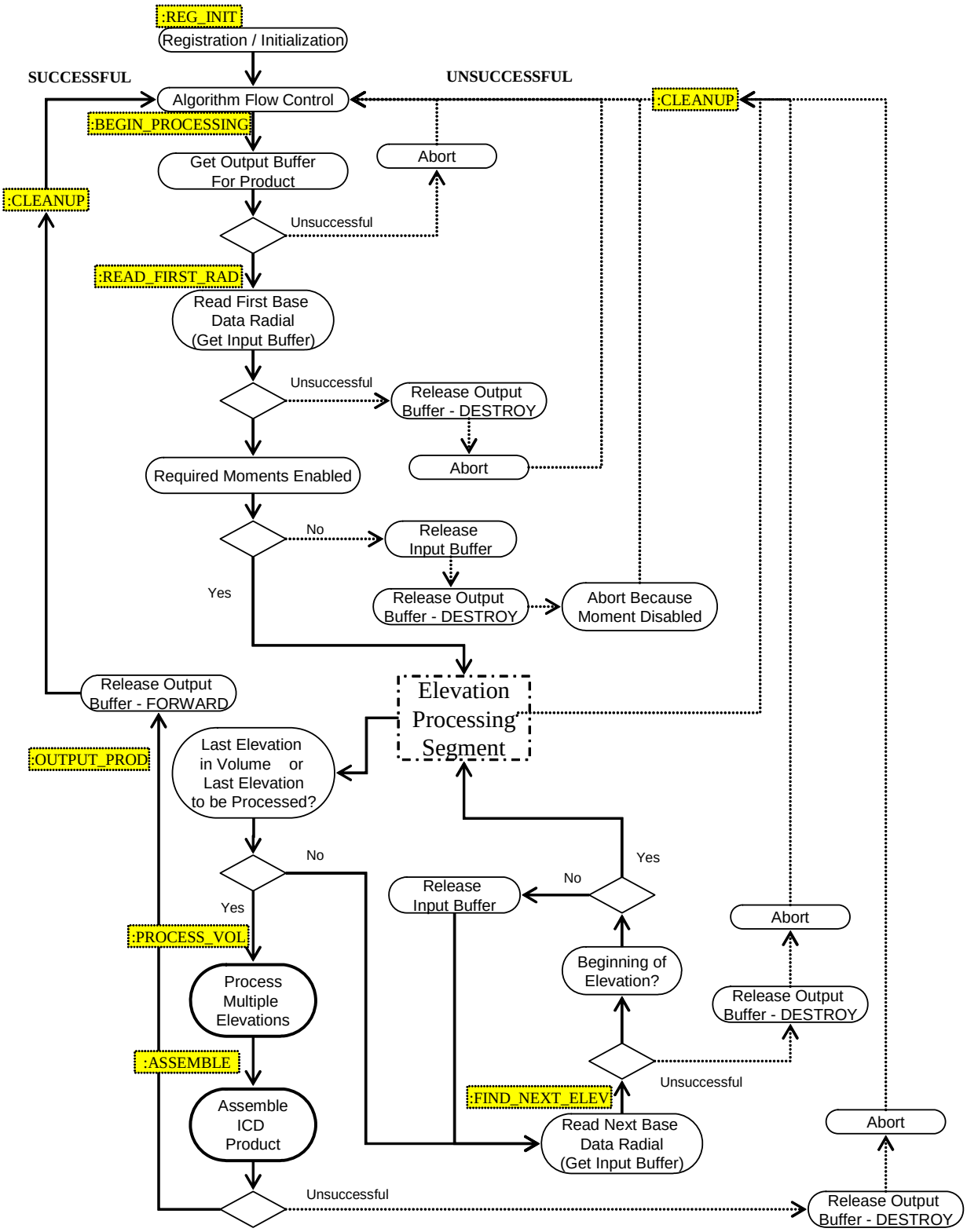


:LABEL included in sample algorithm source code

Updated April 2006

One Product Algorithm

Radial_Based Input Volume_Based Output

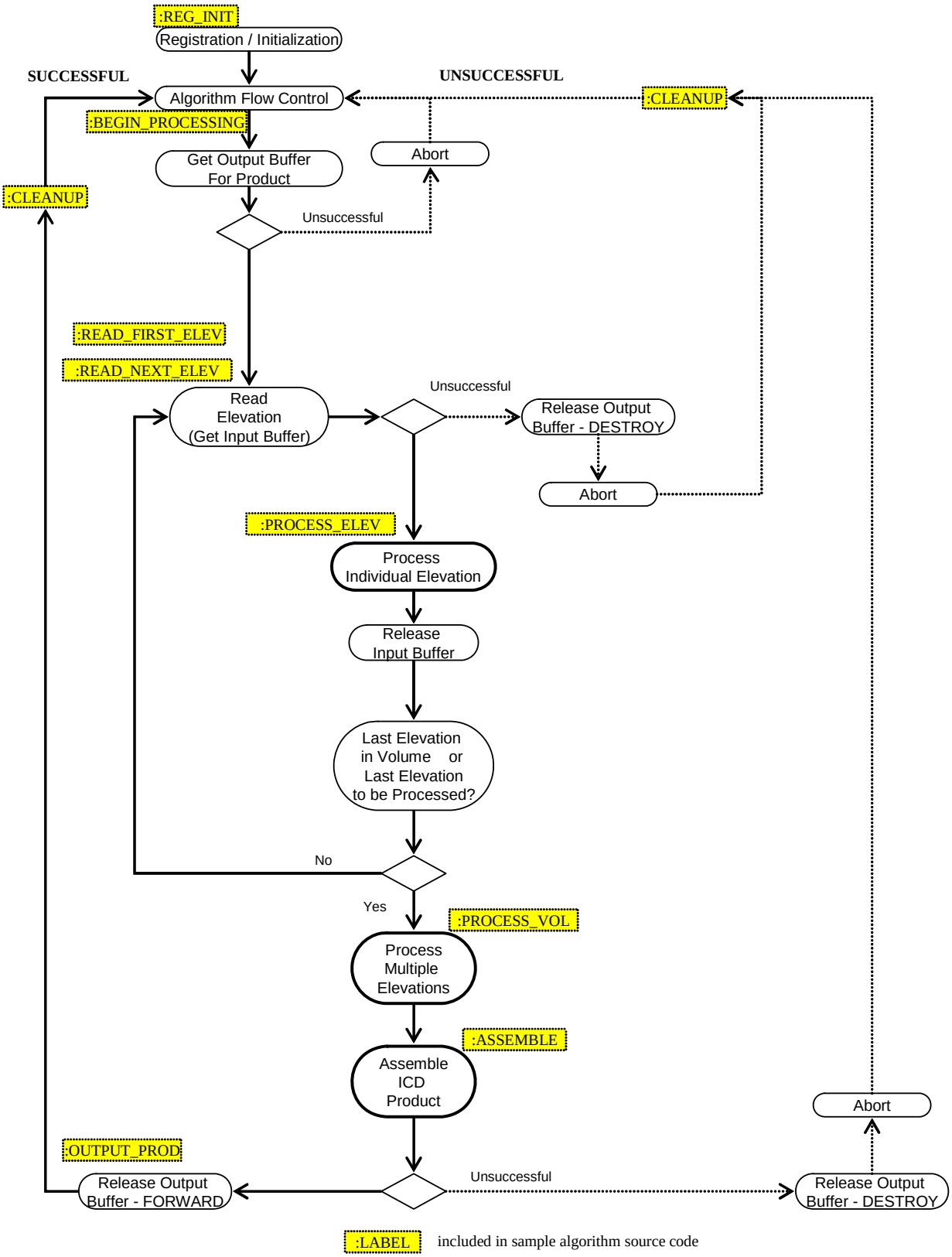


:LABEL included in sample algorithm source code

Updated April 2006

One Product Algorithm

Elevation_Based Input Volume_Based Output



:LABEL included in sample algorithm source code

Part B. Sample Algorithm 1 Main Module

(UPDATED MAY 2009)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample1_digrefl.c

Description:    digital reflectivity is a simple, sample algorithm for
                the ORPG. The algorithm has been implemented using
                ANSI-C interfaces and source code.

                This program can be used as a template to build more
                complex algorithms and includes a significant amount of
                in-line commenting to describe each step.

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3.

                digital_reflectivity ingests radials of base data and
                creates an "ICD-formatted" digital radial data array
                product using reflectivity data.

                This algorithm also includes "TEST" sections not contained
                in operational algorithms.

                Key source files for this algorithm include:

                sample1_digrefl.c       the algorithm driver files
                sample1_digrefl.h

                s2_print_diagnostics.c  source to display message header
                s2_print_diagnostics.h  information (DEBUG output)

                s1_symb_layer.c         source to create the symbology
                s1_symb_layer.h         and data packet layers


Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.11, April 2005   T. Ganger
              Modified to demo new DEA adaptation data using
                 sample_dig.alg as test adaptation data.
```

```
Version 2.0   March 2006   T. Ganger
              Revised to use RPGP_set_packet_16_radial to avoid byte-swap
              Replaced stderr messages outside of test/debug sections with
                   RPGC_log_msg() calls.
              Revised to use current guidance for abort functions and
                   abort reason codes.
              External product_to_disk function added.

Version 2.1   June 2006   T. Ganger
              Revised to use RPGC_reg_io
              Revised to demonstrate the multiple task_name for an executable
                   name feature introduced in Build 9.
              Revised to use the new 'by_name' get_inbuf/get_outbuf functions

Version 2.2   July 2006   T. Ganger
              Modified to use the radial header offset to surveillance
                  data 'ref_offset'.  Tested RPGC_get_surv_data.
              Only malloc space for the header and surveillance data rather
                   than the complete radial
              Replaced WAIT_ALL with WAIT_DRIVING_INPUT.

Version 2.3   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.
              Added note that MAX_BASEDATA_REF_SIZE must be used instead of
                  BASEDATA_REF_SIZE for super resolution elevations.
              RPGC_get_radar_data failed in this algorithm for Build 9.

Version 2.4   June 2007   T. Ganger
              Added and changed in-line comments for clarification

Version 2.5   June 2007   T. Ganger
              For Build 10, removed reading discontinued VCP structure
                  fields.  Several field were temporary (actually spares)
              For Build 10, changed REFL_RAWDATA to RECOMBINED_REFL_RAWDATA

Version 2.6   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated use of defined output buffer ids by using
                  RPGC_get_id_from_name
              Moved API test from main to separate function.
              Corrected the contents of dependent parameter 4 from maximum
                  reflectivity data level to maximum value in dBZ
              Reduced the size of the internal elevation data array by
                  using an array of bytes instead of shorts.
              Modified to handle variable array sizes and specifically to
                  handle super resolution input and produce super resolution
                  output.
              When copying each radial, demonstrated ensuring that data
                  values beyond the last good data bin (if any) are set to '0'.
              Used the new default abort reason code PGM_PROD_NOT_GENERATED.

Version 2.7   June 2008    T. Ganger  (Sample Algorithm ver 1.19)
              Modified to include the new parameter to the function
                  RPGC_is_buffer_from_last_elev.
              Eliminated Red Hat 5 compile warnings.

Version 2.8   Movember 2008    T. Ganger  (Sample Algorithm ver 1.20)
```

```
              Added test to ensure that the algorithm did not attempt to
                  read or copy more bins than are actually in each
                  base data radial.  Depending upon the method used to read
                  the radial, a specific index is tested.

Version 2.9   February 2009    T. Ganger  (Sample Algorithm ver 1.21)
              Modified test section printing out radial information
                  including status to reflect ORDA usually having 720
                  super resolution radials.
              Added test section to print dual pol data parameters.

$Id$
**************************************************************************/

/* see the following include file for descriptions of constants for
for expected ICD block sizes */
#include "sample1_digrefl.h"

/* GLOBAL TEST DEFINES */

int DEBUG=FALSE;           /* test flag: set to true for diag messages */

int TEST_API=FALSE;

int TEST_ADAPT=FALSE;




/* prototype for function which tests several api functions */
void sample1_test_api(Base_data_radial *bdr_ptr, char *output_name, int output_id);


/* variables for new adaptation data scheme */
typedef struct {
  short  my_element_1;
  double my_element_2;
  int    my_element_3;
} sample1_adapt_t;

sample1_adapt_t sample1_adapt;

/* prototype for adaptation callback function for new scheme */
/* OK to use if not registered as a callback */
/* int read_my_adapt();  */

/* REQUIRED for use as a callback */
int read_my_adapt_fx(void *struct_address);


/*************************************************************************/
/*************************************************************************/


/************************************************************************
Description:    main function to drive the digital radial data array
                algorithm
Input:          one radial at a time as defined in the Base_data_radial
                structure (see the basedata.h include file)
Output:         a formatted ICD graphical product which is forwarded to
```

```
                the ORPG for distribution
Returns:        none
Globals:        none
Notes:          1). This algorithm is used to demonstrate and test
                API functions and DEA adaptation data.  It also provides
                an example of basic algorithm structure.
                2). This algorithm provides a demonstration of having
                two different task names for the executable file name.
                The command line arguments determine the task name
                3). Modified to eliminate a hard coded radial size of
                230.
                4).Modified for Super Res data input.

**********************************************************************/

int main(int argc,char* argv[]) {
  /* algorithm variable declarations and definitions ------------------ */
  Base_data_radial
      *basedataPtr=NULL;    /* a pointer to a base data radial structure*/
  Base_data_header *hdr;    /* a pointer to the base data header        */

  int ref_enable;           /* variables: used to hold the status of    */
  int vel_enable;           /* the individual moments of base data.     */
  int spw_enable;           /* tested after reading first radial        */

  int PROCESS=TRUE;         /* Boolean used to control the daemon       */
                            /* portion of the program. Process until set*/
                            /* to false.                                */
  int i;                    /* loop variable                            */

  int rad_idx=0;            /* counter: radial index being processed    */
  short elev_idx=0;         /* variable: used to hold current elev index*/
  short elevation=0;        /* variable: used to hold the value of elev */
  int vol_num=0;            /* current volume number                    */
  int opstatus;             /* variable:  used to hold return values    */
                            /* from API system calls                    */
  char *out_buffer;         /* pointer: access to allocated memory to   */
                            /* hold a completed ICD product             */

  short radial_status=0;    /* variable: status of each radial input    */

  /* array of pointers to radial structures   */
  char *radial[BASEDATA_MAX_SR_RADIALS];

  int bins_to_process = 0;  /* number of data bins to process in radial */
  int bins_to_70k;
  int max_num_bins = BASEDATA_REF_SIZE;  /* 460 */
                            /* could be reduced to limit range of product */
                            /* and is also limited by 70,000 ft MSL       */
  int max_num_radials = BASEDATA_MAX_RADIALS;  /* 400 */
  int new_buffer_size;

  unsigned char *rad_data=NULL;

/* Sample Alg 1.20 - make sure we do not read more than is in a radial */
  int max_bins_to_copy;
  int bins_to_copy;

  short *surv_data=NULL;   /* pointer to the surveillance data in the radial */
  int index_first_bin=999;  /* used with method 2 */
```

```
  int index_last_bin=999;   /* used with method 2 */

  Generic_moment_t gen_moment;  /* used with method 3 */
  /* cannot pass a NULL pointer to RPGC_get_radar_data */
  Generic_moment_t *my_gen_moment = &gen_moment;  /* method 3 */


  int rc = 0;               /* return code from function call           */

/* structure for adaptation data */
  Siteadp_adpt_t site_adapt;


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  */
/*     variables to hold the different data names for the two
 *     defined task names.  See task_attr_table.sample_snippet
 */
  int OUTPUTDATA;
  char INDATA_NAME[65];
  char OUTDATA_NAME[65];
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

  char argument[65];


/* test / debug variable declarations and definitions ------------------*/
  int test_out=FALSE;      /* Boolean to control diagnostic file output */

  int VERBOSE=FALSE;        /* verbose processing output                */

/* end test / debug variables and definitions  ----------------------- */


  hdr = NULL;

  fprintf(stderr, "\nBegin Digital Reflectivity Algorithm in C\n");

  /* LABEL:REG_INIT  Algorithm Registration and Initialization Section  */

  RPGC_init_log_services(argc, argv);


  /* register inputs and outputs based upon the           */
  /* contents of the task_attr_table (TAT).               */
  /* for task sample1_base: input:SR_REFLDATA(78), output:SR_DIGREFLBASE(1990)*/
  /* for task sample1_raw: input:REFL_RAWDATA(54),                            */
  /*                       output:SR_DIGREFLRAW(1995)                         */
  RPGC_reg_io(argc, argv);

  /* register algorithm infrastructure to read the Scan Summary Array   */
  RPGC_reg_scan_summary();


  /* register site adaptation data */
  rc = RPGC_reg_site_info( &site_adapt );
  if ( rc < 0 )
  {
    RPGC_log_msg( GL_INFO,
         "SITE INFO: cannot register adaptation data callback function\n");
  }
```

```
  /* register adaptation data callback (some ported algorithm use old method) */
  /* supported timing types are ADPT_UPDATE_BOV (beginning of volume), */
  /* ADPT_UPDATE_BOE (beginning of elevation), ADPT_UPDATE_ON_CHANGE,  */
  /* ADPT_UPDATE_WITH_CALL, and ADPT_UPDATE_ON_EVENT (not tested)      */

  /* Register algorithm adaptation data */
  rc = RPGC_reg_ade_callback( read_my_adapt_fx,
                              &sample1_adapt,
                              "alg.sample1_dig.",
                              ADPT_UPDATE_BOV );
  if ( rc < 0 )
  {
    RPGC_log_msg( GL_INFO,
         "SAMPLE1: cannot register adaptation data callback function\n");
  }


  /* ORPG task initialization routine. Input parameters argc/argv are   */
  /* not used in this algorithm                                         */
  RPGC_task_init(ELEVATION_BASED, argc, argv);




/*   +++++ demo multiple task names +++++++++++++++++++++++++++++++++   */
    /* This section of code demonstrates having multiple
     * task names with one file name.  This algorithm has different
     * input and output buffers for two different task names
     */

    /* get my task name so I can request the correct buffers     */
        /* because the function RPGC_reg_custom_options() can   */
        /* not be used to get the task name from the -T option,  */
        /* we have to manually parse the command line for "-T"  */
    for( i = 0; i < argc; i++ ) {
        strcpy(argument, argv[i]);
        if(strcmp( argument, "-T") == 0) {
            strcpy(argument, argv[i+1]);
            break;
        }
    } /* end for i < argc */

    /* New 'by_name' functions use the registration name     */
    strcpy(INDATA_NAME, "SR_REFLDATA");
    strcpy(OUTDATA_NAME, "SR_DIGREFLBASE");
    OUTPUTDATA = RPGC_get_id_from_name( "SR_DIGREFLBASE");

    fprintf(stderr, "Sample 1 task_name is '%s'\n", argument);
    if(strcmp(argument, "sample1_base") == 0) {
        strcpy(INDATA_NAME, "SR_REFLDATA");
        strcpy(OUTDATA_NAME, "SR_DIGREFLBASE");
        OUTPUTDATA = RPGC_get_id_from_name( "SR_DIGREFLBASE");
    } else if(strcmp(argument, "sample1_raw") == 0) {
        strcpy(INDATA_NAME, "REFL_RAWDATA");
        strcpy(OUTDATA_NAME, "SR_DIGREFLRAW");
        OUTPUTDATA = RPGC_get_id_from_name( "SR_DIGREFLRAW");
    }
    fprintf(stderr, "-> Input is '%s', output is '%s' \n",
                                INDATA_NAME, OUTDATA_NAME);
```

```
/* +++++ end code section demo multiple task names ++++++++++++++++++++   */


  fprintf(stderr, "-> algorithm initialized and proceeding to loop control\n");

  RPGC_log_msg( GL_INFO,
       "-> algorithm initialized and proceeding to loop control\n");


  if(TEST_ADAPT)
  {
      /* New Site Info */
      fprintf(stderr, "->ADAPTATION DATA RDA elevation = %d\n",
                                          site_adapt.rda_elev);
      fprintf(stderr, "->ADAPTATION DATA RPG id = %d\n",
                                        site_adapt.rpg_id);
      fprintf(stderr, "->ADAPTATION DATA RPG name = %s\n",
                                        site_adapt.rpg_name);
  }



  /* while loop that controls how long the task will execute. As long as*/
  /* PROCESS remains true, the task will continue. The task will        */
  /* terminate upon an error condition                                  */
  while(PROCESS) {

    /* system call to indicate a data driven algorithm. block algorithm */
    /* until good data is received the product is requested              */
    RPGC_wait_act(WAIT_DRIVING_INPUT);

  /* LABEL:BEGIN_PROCESSING Released from Algorithm Flow Control Loop   */
  if (VERBOSE)
       fprintf(stderr, "-> algorithm passed loop control and begin processing\n");

  RPGC_log_msg( GL_INFO,
       "-> algorithm passed loop control and begin processing\n");



/* ************ ADAPTATION DATA TEST SECTION *********** */
  /* print out the contents of adaptation data for demonstration purposes */
  if(TEST_ADAPT)
  {
  /* test reading new DEA adaptation data elements */
  /* without the callback registration, would need */
  /* to manually read adaptation data              */
  /*      read_my_adapt() */
  /*      read_my_adapt_fx(&sample1_adapt); */

      fprintf(stderr, "->DEA Test Element 1 Data = %d\n",
                               sample1_adapt.my_element_1);
      fprintf(stderr, "->DEA Test Element 2 Data = %f\n",
                               sample1_adapt.my_element_2);
      fprintf(stderr, "->DEA Test Element 3 Data = %d\n",
                               sample1_adapt.my_element_3);
  }
/* ********** END ADAPTATION DATA TEST SECTION ********** */
```

```
    /* allocate a partition (accessed by the pointer, out_buffer) within the*/
    /* SR_DIGREFLBASE / SR_DIGREFLRAW linear buffer. error return in opstatus */

    out_buffer = (char *)RPGC_get_outbuf_by_name(OUTDATA_NAME, BUFSIZE,
                                                 &opstatus);

    /* check error condition of buffer allocation. abort if abnormal    */
    if(opstatus != NORMAL) {
        RPGC_log_msg( GL_INFO,
             "ERROR: Aborting from RPGC_get_outbuf...opstatus=%d\n", opstatus);
        RPGC_abort();
        continue;
    }

    if (VERBOSE) {
      fprintf(stderr, "-> successfully obtained output buffer\n");
    }
    /* make sure that the buffer space is initialized to zero         */
    clear_buffer(out_buffer);

    /* LABEL:READ_FIRST_RAD Read first radial of the elevation and    */
    /* accomplish the required moments check                          */
    /* ingest one radial of data from the BASEDATA / RAWDATA linear   */
    /* buffer. The data will be accessed via the basedataPtr pointer  */

    if (VERBOSE) {
      fprintf(stderr, "-> reading first radial buffer\n");
    }


    basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name(INDATA_NAME,
                                                            &opstatus);
    hdr = (Base_data_header *) basedataPtr;

    if (VERBOSE) {
       fprintf(stderr, "     opstatus = %d \n", opstatus);
    }
    /* check radial ingest status before continuing                   */
    if(opstatus != NORMAL){
        RPGC_log_msg(GL_INFO, "ERROR: Aborting from RPGC_get_inbuf\n");
        RPGC_rel_outbuf((void*)out_buffer,DESTROY);
        RPGC_abort();
        continue;
    }

    if (VERBOSE) {
      fprintf(stderr, "-> successfully read first radial buffer\n");
    }

    /* test to see if the required moment (reflectivity) is enabled   */
    /* NOTE: the other method, using RPGC_reg_moments, should not be  */
    /*       used in this algorithm because one of the configured     */
    /*       task names uses RAWDATA                                   */
    RPGC_what_moments((Base_data_header*)basedataPtr, &ref_enable,
                                       &vel_enable, &spw_enable);

    if(ref_enable != TRUE) {
        RPGC_log_msg(GL_INFO, "ERROR: Aborting from RPGC_what_moments\n");
```

```
        RPGC_rel_inbuf((void*)basedataPtr);
        RPGC_rel_outbuf((void*)out_buffer, DESTROY);
        RPGC_abort_because(PGM_DISABLED_MOMENT);
        continue;
    }

    if (VERBOSE) {
      fprintf(stderr, "-> required moments enabled\n");
    }


/* ********** API TEST SECTION ******************************** */
/* DEMONSTRATION OF USING VARIOUS HELPER FUNCTIONS this function */
/* an the end of this file demonstrates use of functions that    */
/* that obtain information about the current volume / elevation. */
  if (TEST_API) {
    sample1_test_api( basedataPtr, OUTDATA_NAME, OUTPUTDATA);
  }
/* ********** END API TEST SECTION **************************** */


/* Sample Alg 1.21 */
/* BEGIN TEST SECTION */
/*    test_read_dp_moment_params(basedataPtr); */

/* END TEST SECTION */


    if (VERBOSE) {
      fprintf(stderr, "-> begin ELEVATION PROCESSING SEGMENT\n");
    }
    /* ELEVATION PROCESSING SEGMENT. continue to ingest and process     */
    /* individual base radials until either a failure to read valid     */
    /* input data (a radial in correct sequence) or until after reading */
    /* and processing the last radial in the elevation                  */

    /* get the current elevation angle & index from the incoming data */
    elev_idx = (short)RPGC_get_buffer_elev_index((void *)basedataPtr);

    /* get the current volume number using the following system call  */
    vol_num = RPGC_get_buffer_vol_num((void *)basedataPtr);

    /* if not ingesting base data, RPGCS_get_target_elev_ang must be used */
    elevation = hdr->target_elev;


/* ***************** begin handle Super Res ******************************/

    /* since this algorithm currently only produces high resolution products   */
    /* from elevations having both 0.5 deg radials and 250 m reflectivity      */
    /* we shortcut the algorithm here if the elevations cut is not appropriate */
    if( hdr->azm_reso != 1 || hdr->surv_bin_size != 250 ) {
       if(VERBOSE) fprintf(stderr,"Product Not Generated, not a SR Elevation\n");
       RPGC_log_msg(GL_INFO,
                    "Aborting: cut does not contain required resolution\n");
       RPGC_rel_inbuf((void*)basedataPtr);
       RPGC_rel_outbuf((void*)out_buffer, DESTROY);
       RPGC_abort_because(PGM_PROD_NOT_GENERATED);
       continue;
   }
```

```
    /* NOTE: using hdr->surv_bin_size for this test was chosen over */
    /* using hdr->n_surv_bins.  However, if reading velocity or     */
    /* spectrum width data, hdr->n_dop_bins would be used to set    */
    /* the maximum size of the radial.                              */
    if(hdr->surv_bin_size == 250) {
/*    if(hdr->n_surv_bins > 460) { */
        max_num_bins = MAX_BASEDATA_REF_SIZE;
        if(DEBUG) fprintf(stderr,"Detected Hi Res 250 meter bin \n");
    } else if(hdr->surv_bin_size == 1000) {
/*    } else if(hdr->n_surv_bins <= 460) { */
        max_num_bins = BASEDATA_REF_SIZE;
    } else {
        RPGC_log_msg(GL_INFO, "ERROR: Bad surveillance bin size %d\n",
                                                hdr->surv_bin_size);
        RPGC_rel_inbuf((void*)basedataPtr);
        RPGC_rel_outbuf((void*)out_buffer, DESTROY);
        RPGC_abort_because(PGM_INPUT_DATA_ERROR);
        continue;
    }

    /* limit number of bins to the 70.000 ft MSL ceiling */
    bins_to_70k = RPGC_bins_to_ceiling( basedataPtr, hdr->surv_bin_size );

    if(bins_to_70k < max_num_bins)
        bins_to_process = bins_to_70k;
    else
        bins_to_process = max_num_bins;
    /* Sample Alg 1.20 - */
    /* ensure even number of bins to prevent possible alignment issues */
    if( (bins_to_process % 2) !=0)
        bins_to_process++;

    /* NOTE: when accomplishing numerical calculations including assembly   */
    /*       of the final product arrays, the number of good bins must be   */
    /*       accounted for.  This can be made by                            */
    /*          a. reducing the size of the internal data array processed   */
    /*             (bins_to_process), or by                                 */
    /*          b. ensuring all data values after the last good bin are     */
    /*             set to a value of '0'.                                   */
    /*       This algorithm chooses b and is tested for each radial         */


    if(hdr->azm_reso == 1) {
        max_num_radials = BASEDATA_MAX_SR_RADIALS;
        if(DEBUG) fprintf(stderr, "Detected Super Res Radial (0.5 degree) \n");
    } else if(hdr->azm_reso == 2) {
        max_num_radials = BASEDATA_MAX_RADIALS;
    }


    /* calculate max buffer size and realloc out buffer if required */
    if(max_num_radials == BASEDATA_MAX_SR_RADIALS ||
        max_num_bins == MAX_BASEDATA_REF_SIZE ) { /*need to resize output buffer*/
            new_buffer_size = (BASEDATA_MAX_SR_RADIALS * MAX_BASEDATA_REF_SIZE) +
                              sizeof(Graphic_product) + sizeof(Symbology_block) +
                              sizeof(Packet_16_hdr_t);

        out_buffer = RPGC_realloc_outbuf( (void *)out_buffer, new_buffer_size,
```

```
                                                                    &opstatus );
        if(opstatus != NORMAL){
            RPGC_log_msg(GL_INFO, "ERROR: Unable to realloc out buffer\n");
            RPGC_rel_inbuf((void*)basedataPtr);
            RPGC_rel_outbuf((void*)out_buffer, DESTROY);
            RPGC_abort();
            continue;
        }

    } /* end calculate max size and realloc */

/* ***************** End handle Super Res ******************************/



    while(TRUE) {

      /* LABEL:PROCESS_RAD Here we process each radial individually     */
      /* any processing that can be accomplished without comparison     */
      /* between other radial can be accomplished here                  */

      /* redundant test information for quality control                 */
      if (DEBUG) {
         /* Sample Alg 1.21 - modified test from 362 to 717 for ORDA SR */
         if ( (rad_idx <= 3) || (rad_idx >= 717) ) {
            fprintf(stderr,"  RADIAL CHECK FOR RADIAL %d\n", rad_idx+1);
            fprintf(stderr,"    Azimuth=%f\n",hdr->azimuth);
            fprintf(stderr,"    Start Angle=%hd\n",hdr->start_angle);
            fprintf(stderr,"    Angle Delta=%hd\n",hdr->delta_angle);
            /* Sample Alg 1.21 - read the radial status flag  */
            radial_status = basedataPtr->hdr.status & 0xf;
            fprintf(stderr, "    Radial_Status = %d where\n"
                            "    0=beg_ele, 1=int, 2=end_ele, 3=beg_vol,
4=end_vol,\n"
                            "    8=pseudo_end_ele, 9=pseudo_end_vol\n",
                                                              radial_status);
         } /* end if rad_idx */

      } /* end DEBUG */



      /* for this algorithm...we'll copy the radial header and the      */
      /* reflectivity structure to the temporary storage array. This    */
      /* routine requires several steps. First, allocate enough memory  */
      /* to hold the radial header plus data. Then copy the header and  */
      /* radial data structure portion into the allocated space. The    */
      /* radial can then be accessed via the pointer in the radial array*/

      /* NOTE: IT IS NOT ALWAYS NECESSARY TO COPY THIS DATA.  IF ONLY   */
      /*       PROCESSING A SINGLE RADIAL, THE RADIAL COULD BE          */
      /*       PROCESSED IN THE INPUT BUFFER AND WRITTEN DIRECTLY TO AN */
      /*       OPEN OUTPUT BUFFER. THIS WOULD REDUCE MEMORY USAGE.      */

/* While all of the following work, Method 2 is typical for basic moments */
/* Method 3 must be used for advanced Dual Pol moments                     */
      /* Method 1 */
/*      surv_data = (short *) ((char *) basedataPtr + hdr->ref_offset); */
/*        max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;      */
                          /* '-1' because the indexes are 1-based */
```

```
                        /* we do not assume hdr->surv_range is always 1 */

      /* Method 2 */
      surv_data = (short *) RPGC_get_surv_data( (void *)basedataPtr,
                                  &index_first_bin, &index_last_bin);
      max_bins_to_copy = index_last_bin + 1;
                     /* '+1'because indexes are 0-based */
                     /* we do not assume that index_first_bin is always 0 */

      /* Method 3 */
/*      surv_data = (short *) RPGC_get_radar_data( (void *)basedataPtr, */
/*                                    RPGC_DREF, my_gen_moment );       */
/*        max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;     */

      /* test for NULL moment pointer and 0 offset */
      if(surv_data == NULL) {
         RPGC_log_msg( GL_INFO,"ERROR: NULL radial detected: %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)out_buffer, DESTROY);
         RPGC_abort_because(PGM_INPUT_DATA_ERROR);
         opstatus = TERMINATE;
         break;
      }

      if(hdr->ref_offset == 0) {
         RPGC_log_msg( GL_INFO, "ERROR: reflectivity offset 0: %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)out_buffer, DESTROY);
         RPGC_abort_because(PGM_INPUT_DATA_ERROR);
         opstatus = TERMINATE;
         break;
      }

      /* allocate one radial's worth of memory  */
      /* used an internal radial of bytes rather than shorts */
      radial[rad_idx] = (char *)malloc(sizeof(Base_data_header) +
                                                 (bins_to_process));


      if(radial[rad_idx] == NULL) {
         RPGC_log_msg( GL_INFO, "MALLOC Error: malloc failed on radial %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)out_buffer, DESTROY);
         RPGC_abort_because(PGM_MEM_LOADSHED);
         opstatus = TERMINATE;
         break;
      }


      /* copy the radial header */
      memcpy(radial[rad_idx], basedataPtr, sizeof(Base_data_header));

      /* copy the reflectivity data */
/* Sample Alg 1.20 - only copy the the lessor of (a) number of bins in  */
/*                   the radial or (b) bins_to_process                  */
      if(bins_to_process <= max_bins_to_copy)
         bins_to_copy = bins_to_process;
```

```
else
   bins_to_copy = max_bins_to_copy;

/* Sample Alg 1.21 - added test of bins being copied */
if(DEBUG) {
   /* Sample Alg 1.21 - modified test from 362 to 717 for ORDA SR */
   if ( (rad_idx <= 3) || (rad_idx >= 717) ) {
      fprintf(stderr,"  FOR RADIAL %d\n", rad_idx+1);
      fprintf(stderr,"copying %d bins\n", bins_to_copy);
   }
} /* end DEBUG */

/* when writing the output product, any data bins */
/* beyond max_bins_to_copy are set to '0'         */
rad_data = (unsigned char *) ( radial[rad_idx] + sizeof(Base_data_header) );
for(i = 0; i < bins_to_process; i++)
   /* ensure any data bins beyond the last good bin are set to 0 */
   if( i < bins_to_copy )
       rad_data[i] = (unsigned char) surv_data[i];
   else
       rad_data[i] = (unsigned char) 0;


/* now that the data that we want from the radial is in temporary */
/* storage...release the input buffer                             */
RPGC_rel_inbuf((void*)basedataPtr);

/* increment the radial counter */
rad_idx++;

/* check for radial overflow in the input array  */
/* this should never happen unless the RDA fails */
if(rad_idx > max_num_radials) {
    fprintf(stderr,
         "Count Error: radial index %d exceeded array limits %d\n",
                                              rad_idx, max_num_radials);
    RPGC_log_msg( GL_INFO,
         "Count Error: radial index %d exceeded array limits %d\n",
                                              rad_idx, max_num_radials);
    RPGC_rel_outbuf((void*)out_buffer, DESTROY);
    RPGC_abort_because(PGM_INPUT_DATA_ERROR);
    opstatus = TERMINATE;
    break;
}

/* if end of elevation or volume then exit loop */
/* this is the ACTUAL end of elevation rather than Pseudo end     */
/* NOTE: when reading historical data prior to the ORDA, this     */
/*       results in the possiblilty of an overlapping radial      */
/*       This could be avoided by terminating radial processing   */
/*       at pseudo end of elevation / volume and then just        */
/*       reading  basedata radial messages until actual end.  See */
/*       sample algorithms 2 and 3 for an example of eliminatig   */
/*       the possibility of overlapped radials                    */
if(radial_status == GENDEL || radial_status == GENDVOL) {
    if (VERBOSE)
         fprintf(stderr, "-> End of elevation found\n");
    /* exit with opstatus==NORMAL and no ABORT */
    break;
}
```

```
      /* LABEL:READ_NEXT_RAD Read the next radial of the elevation      */
      /* ingest one radial of data from the BASEDATA / RAWDATA linear   */
      /* buffer. The data will be accessed via the basedataPtr pointer  */



      basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name(INDATA_NAME,
                                                              &opstatus);
      hdr = (Base_data_header *) basedataPtr;

      /* check radial ingest status before continuing                   */
      if(opstatus != NORMAL) {
          RPGC_log_msg( GL_INFO," ERROR reading input buffer\n");
          RPGC_rel_outbuf((void*)out_buffer, DESTROY);
          RPGC_abort();
          break;
      }


    } /* end while TRUE, reading radials */



    if (VERBOSE) {
      fprintf(stderr,"=> out of radial ingest loop...elev count=%d\n",elev_idx);
    }

    /* LABEL:PROCESS_ELEV Here we accomplish any processing of the      */

    if(opstatus == NORMAL) {
        /* elevation as a whole.  Processing that requires comparisons  */
        /* between radials should be accomplished here                  */
        ;
    }
    /* the end of the ELEVATION PROCESSING SEGMENT of the algorithm     */


    /* LABEL:ASSEMBLE The assembly of the ICD formatted product         */
    /* if processing is normal then create the ICD output product       */
    if(opstatus == NORMAL) {
        int result;     /* function return value                        */
        int length=0;   /* accumulated product length                   */
        int max_refl=0; /* maximum reflectivity value                   */


      /* building the ICD formatted output product requires a few steps */
      /* step 1: build the product description block (pdb)              */
      /* step 2: build the symbology block & data layer                 */
      /* step 3: complete missing values in the pdb                     */
      /* step 4: build the message header block (mhb)                   */
      /* step 5: forward the completed product to the system            */

      /* step 1: build the product description block -uses a system call*/
      if (VERBOSE) {
        fprintf(stderr,
           "\nCreating the product description block now: vol=%d\n", vol_num);
      }
      RPGC_prod_desc_block((void*)out_buffer, OUTPUTDATA, vol_num);
```

```
/* step 2: build the symbology layer & digital radial data array. */
/* this routine returns both the overall length (thus far) of the */
/* product and the maximum reflectivity of all radials            */
if (VERBOSE) {
  fprintf(stderr,"begin building the symbology and p16 layers\n");
}

max_refl = build_symbology_layer(out_buffer, radial, rad_idx,
                                             bins_to_process, &length);

/* step 3: finish building the product description block by       */
/* filling in certain values such as elevation index, maximum     */
/* reflectivity, accumulated product length, etc                  */
finish_pdb(out_buffer, elev_idx, elevation, max_refl, length);


/* generate the product message header (use system call) and input*/
/* total accumulated product length minus 120 bytes for the MHB & PDB */
result = RPGC_prod_hdr((void*)out_buffer, OUTPUTDATA, &length);

if (VERBOSE) {
  fprintf(stderr, "-> completed product length=%d\n", length);
}

/*(this routine adds 120 bytes for the MHB & PDB to the      */
/* "length" parameter prior to creating the final header)         */


/* if the creation of the product has been a success              */
if(result == 0) { /*success*/
   /* print product header for testing purposes                    */
   int res;
   if (VERBOSE) {
       fprintf(stderr, "product header creation success...display header\n");
       res = print_message_header(out_buffer);
       res = print_pdb_header(out_buffer);
       fprintf(stderr, "\nprinting of product header complete\n");
   }



    /* diagnostic - interrupts product output and creates a
     * binary output of product to file.
     * this is useful if product problems cause task failure
     */
   if(test_out == TRUE) {

       product_to_disk(out_buffer, length, "sample1_dig", elev_idx);

       RPGC_log_msg( GL_INFO,
          "Interrupted product output for diagnostic output to file.\n");
       RPGC_rel_outbuf( (void*)out_buffer, DESTROY );

   }    /* end test_out TRUE */
   else /* normal product output */

   /* LABEL:OUTPUT_PROD    forward product and close buffer        */
       RPGC_rel_outbuf((void*)out_buffer, FORWARD);
```

```
      } else {  /* product failure (destroy the buffer & contents)        */
         RPGC_log_msg( GL_INFO, "product header creation failure\n");
         RPGC_rel_outbuf((void*)out_buffer, DESTROY);
         RPGC_abort_because(PGM_PROD_NOT_GENERATED);
      } /* end result !=0 */

    }  /* end of if(opstatus==NORMAL) block */


    /* LABEL:CLEANUP   section to reset variables / free memory         */
    /* free each previously allocated radial array                      */
    if (DEBUG) {
        fprintf(stderr,"free radial array (rad_idx=%i)\n",rad_idx);
    }

    for(i = 0; i < rad_idx; i++)
        free(radial[i]);

    rad_idx = 0; /* reset radial counter to 0                             */


    RPGC_log_msg( GL_INFO,"process reset: ready to start new elevation\n");

    } /* end while PROCESS == TRUE */

  fprintf(stderr,"\nProgram Terminated\n");

  return(0);

} /* end main */

/*************************************************************************/
/*************************************************************************/



/*********************************************************************
Description:    clear_buffer: initializes a portion of allocated
                memory to zero
Input:          pointer to the input buffer (already cast to char*)
Output:         none
Returns:        none
Globals:        the constant BUFSIZE is defined in the include file
Notes:          none
*********************************************************************/
void clear_buffer(char *buffer) {
  /* zero out the input buffer */
  int i;

  for(i = 0; i < BUFSIZE; i++)
    buffer[i] = 0;

  return;

}


/*****access function v1**************************************
 * adaptation data access function not used as a callback
```

```
 *      the structure may be a global rather than a parameter
 ************************************************************/
int read_my_adapt()
{

double get_value = 0.0;
int ret = -1;

  ret = RPGC_ade_get_values( "alg.sample1_dig.","db_Element_1", &get_value);
  if(ret == 0)
  {
    sample1_adapt.my_element_1 = (short)get_value;
  }
  else
  {
    sample1_adapt.my_element_1 = TRUE;
    RPGC_log_msg( GL_INFO,
         "my_element_1 DEA value unavailable, using program default\n");
  }


  ret = RPGC_ade_get_values("alg.sample1_dig.", "db_Element_2", &get_value);
  if(ret == 0)
  {
    sample1_adapt.my_element_2 = (float)get_value;
  }
  else
  {
    sample1_adapt.my_element_2 = 10.0;
    RPGC_log_msg( GL_INFO,
         "my_element_2 DEA value unavailable, using program default\n");
  }


  ret = RPGC_ade_get_values("alg.sample1_dig.", "db_Element_3", &get_value);
  if(ret == 0)
  {
    sample1_adapt.my_element_3 = (int)get_value;
  }
  else
  {
    sample1_adapt.my_element_3 = -1;
    RPGC_log_msg( GL_INFO,
         "my_element_3 DEA value unavailable, using program default\n");
  }

  return ret;

} /* end read_my_adapt */

/******access function v2**********************************
 * adaptation data access function used as a callback
 *      the address of the structure must be the parameter
 ************************************************************/
int read_my_adapt_fx( void *struct_address )
{

double get_value = 0.0;
int ret = -1;
```

```
sample1_adapt_t *adapt_ptr = (sample1_adapt_t *)struct_address;

  ret = RPGC_ade_get_values("alg.sample1_dig.", "db_Element_1", &get_value);
  if(ret == 0)
  {
    adapt_ptr->my_element_1 = (short)get_value;
  }
  else
  {
    adapt_ptr->my_element_1 = TRUE;
    RPGC_log_msg( GL_INFO,
         "my_element_1 DEA value unavailable, using program default\n");
  }


  ret = RPGC_ade_get_values("alg.sample1_dig.", "db_Element_2", &get_value);
  if(ret == 0)
  {
    adapt_ptr->my_element_2 = (float)get_value;
  }
  else
  {
    adapt_ptr->my_element_2 = 10.0;
    RPGC_log_msg( GL_INFO,
         "my_element_2 DEA value unavailable, using program default\n");
  }


  ret = RPGC_ade_get_values("alg.sample1_dig.", "db_Element_3", &get_value);
  if(ret == 0)
  {
    adapt_ptr->my_element_3 = (int)get_value;
  }
  else
  {
    adapt_ptr->my_element_3 = -1;
    RPGC_log_msg( GL_INFO,
         "my_element_3 DEA value unavailable, using program default\n");
  }

  return ret;
}



void sample1_test_api(Base_data_radial *bdr_ptr, char *output_name, int output_id)
{
  int test_vol_num;
  int test_elev_ind;
  int test_last_index;
  int test_vcp_num;
  Scan_Summary *test_scan_sum_ptr=NULL;
  int test_target_elev;
  int test_last_elev_index;
  int my_elev_index;
  int last_elev=99;
  int test_wxmode;
  Vcp_struct *test_vcp_ptr=NULL;
  int test_prod_code;
```

```
    if(bdr_ptr == NULL)
        return;

    test_vol_num = RPGC_get_buffer_vol_num((void *)bdr_ptr);
    fprintf(stderr, "API TEST  volume number is %d\n", test_vol_num);

    test_elev_ind = RPGC_get_buffer_elev_index((void *)bdr_ptr);
    fprintf(stderr, "API TEST  elevation index is %d \n", test_elev_ind);

    test_scan_sum_ptr = RPGC_get_scan_summary(test_vol_num);
    if(test_scan_sum_ptr != NULL) {
        fprintf(stderr, "API TEST  using the pointer to scan summary (%d)\n",
                                                    (int)test_scan_sum_ptr);
        fprintf(stderr, "          VCP number: %d\n"
             "          number of rda cuts: %d, number or rpg cuts: %d\n",
             test_scan_sum_ptr->vcp_number, test_scan_sum_ptr->rda_elev_cuts,
             test_scan_sum_ptr->rpg_elev_cuts);
    }

    test_vcp_num = RPGC_get_buffer_vcp_num((void *)bdr_ptr);
    fprintf(stderr,"API TEST  vcp number is %d \n", test_vcp_num);

    test_target_elev = RPGCS_get_target_elev_ang(test_vcp_num, test_elev_ind);
    fprintf(stderr,"API TEST  target elevation is %2.1f \n",
                                              (test_target_elev/10.0));

    test_last_elev_index = RPGCS_get_last_elev_index(test_vcp_num);
    fprintf(stderr, "API TEST  last elevation index is %d \n",
                                      test_last_elev_index);

    last_elev = RPGC_is_buffer_from_last_elev( (void *)bdr_ptr,
                                 &my_elev_index, &test_last_index);
    if( last_elev == 1 )
        fprintf(stderr, "API TEST  buffer (elev %d) is last elev\n"
                        "          last index is: %d\n",
                                    my_elev_index, test_last_index);
    else if( last_elev == 0 )
        fprintf(stderr, "API TEST  buffer (elev %d) is NOT last elev\n"
                        "          last index is: %d\n",
                                      my_elev_index, test_last_index);
    else if( last_elev == -1 )
        fprintf(stderr, "API TEST  error in RPGC_is_buffer_from_last_elev\n");

    test_wxmode = RPGCS_get_wxmode_for_vcp(test_vcp_num);
    fprintf(stderr, "API TEST  weather mode is %d\n", test_wxmode);

    test_vcp_ptr = RPGCS_get_vcp_data( test_vcp_num );
    if(test_vcp_ptr != NULL) {
        fprintf(stderr, "API TEST  using the pointer to vcp data (%d)\n",
                                                      (int)test_vcp_ptr);
        fprintf(stderr ,"          VCP number: %d, number of elevations: %d\n"
         "                         type %d  (2 - const ele, 16 - searchlight.\n",
                 test_vcp_ptr->vcp_num, test_vcp_ptr->n_ele, test_vcp_ptr->type);
    }

    test_prod_code = RPGC_get_code_from_id( output_id );
    fprintf(stderr, "API TEST  product code from  id  is %d\n", test_prod_code);

    test_prod_code = RPGC_get_code_from_name( output_name );
    fprintf(stderr, "API TEST  product code from name is %d\n", test_prod_code);
```

```
}  /* end sample1_test_api() */
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/**********************************************************************
Module:         sample1_digrefl.h

Description:    include file for the digital reflectivity algorithm
                digital_reflectivity.c. This file contains module
                specific include file listings, function prototypes
                and constant definitions.

Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.6,  March 2005   T. Ganger
              Reflect the new size of the base data header (Build 5)
              Modified to use the new product IDs

Version 1.7   March 2006   T. Ganger
              Modified to reflect the new base data message, larger
                   header, new order of moment data, possible
                   larger reflectivity array - Build 8
              External product_to_disk function added.

Version 1.8   June 2006   T. Ganger
              Modified to reflect the multiple outputs for differently
                   named tasks

Version 1.9   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.

Version 2.0   February 2008    T. Ganger    (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated definition of output buffer ids.
              New output buffer size calculated.

Version 2.1   February 2009    T. Ganger    (Sample Algorithm ver 1.21)
              Added function to print dual pol data parameters.



$Id$
**********************************************************************/
#ifndef _DIGITAL_REFLECTIVITY_H_
#define _DIGITAL_REFLECTIVITY_H_
```

```
/* system includes ------------------------------------------------- */


/* ORPG   includes ------------------------------------------------- */
#include <rpg_globals.h>

/*** the following are included in rpg_globals.h */
/***                                              */
/***  #include <stdio.h>                          */
/***  #include <stdlib.h>                         */
/***  #include <time.h>                           */
/***  #include <ctype.h>                          */
/***  #include <string.h>                         */
/***                                              */
/***  #include <a309.h>                           */
/***  #include <basedata.h>                       */
/***  #include <basedata_elev.h>                  */
/***  #include <rpg_port.h>                       */
/***  #include <prod_gen_msg.h>                   */
/***  #include <prod_request.h>                   */
/***  #include <prod_gen_msg.h>                   */
/***  #include <rpg_vcp.h>                        */
/***  #include <orpg.h>                           */
/***  #include <mrpg.h>                           */
/************************************************/

#include <rpgc.h>
#include "rpgcs.h"

#include <packet_16.h>

/* include files for adaptation data demo */
#include <siteadp.h>


/* ---------------------------------------------------------------- */
/* the structure of a base data radial message, which is ingested by the
   algorithm, is defined in  basedata.h  */

/*
 * The order and size of the base data data arrays are no longer fixed.
 *
 * The offsets in the base data header or the API access function must
 * be used to read the basic moments: reflectivity, velocity, and
 * spectrum width.
 *
 * Fields in the base data header are used to determine array size.
 */


/* The contents of Base_data_header is defined in basedata.h  */
/* ---------------------------------------------------------------- */


/* local  includes ------------------------------------------------- */
/* #include "s1_symb_layer.h" */


/* ORPG constants not defined in include files ----------------------- */
```

```
/* Algorithm specific constants/definitions --------------------------- */

/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define SR_DIGREFLBASE 1990  */
/* #define SR_DIGREFLRAW 1995   */


/* the estimated buffer size for non Super Res data (in bytes):
 * (Pre ICD header     96 not included)
 * Msg Hdr/PDB        120
 * Symb Block          10
 * Layer Header         6
 * Packet Header       14
 * Packet Data     171022 ((460 bins+6 header bytes) x 367 radials)
 * TOTAL           171172
 */

/* to allow for 400 radials, the following can be used if not in a       */
/* Super Res elevation, the output buffer must be reallocated otherwise. */
#define BUFSIZE 184250

/* the algorithm will re-allocate the output buffer for larger size if    */
/* the basedata radial has increase surveillance resolution or 0.5 degree */
/* radials. For reference, the following buffer size are calculated needed.*/
/*    1472250 for 800 radials and 1840 bins (full super resolution)       */
/*     736250 for 400 radials and 1840 bins                               */
/*     368250 for 800 radials and  460 bins                               */
/*     184250 for 400 radials and  460 bins (original resolution )        */


/* function prototypes ------------------------------------------------ */
void clear_buffer(char*);

extern short build_symbology_layer(char* buffer, char **radial,
                           int rad_count, int bin_count, int* length);

extern void finish_pdb(char* buffer, short elev_ind, short elevation,
                                        short max_refl, int prod_len);

extern int print_message_header(char* buffer);

extern int print_pdb_header(char* buffer);

extern void product_to_disk(char * output_buf, int outlen, char *pname,
                           short ele_idx);

/* Sample Alg 1.21 */
extern void test_read_dp_moment_params(Base_data_radial *radialPtr);

#endif
```

Part C. Sample Algorithm 2 Main Module

(UPDATED MAY 2009)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample2_radrefl.c

Description:    sample2_radrefl.c is a sample algorithm for the ORPG.
                The algorithm has been implemented using ANSI-C
                interfaces and source code and takes advantage of existing
                ORPG system services.

                This program can be used as a template to build more
                complex (integrated) algorithms and includes a
                significant amount of in-line commenting to describe
                each step.

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3.

                sample2_radrefl.c ingests radials of base data and creates
                an "ICD-formatted" 16-level run length encoded product
                using reflectivity data.

                Key source files for this algorithm include:

                sample2_radrefl.c        the algorithm driver files
                sample2_radrefl.h

                s2_print_diagnostics.c   source to display message &
                s2_print_diagnostics.h   header information

                s2_symb_layer.c          source to create the symbology
                s2_symb_layer.h          and data packet layers

                Run Length Encoding Routines ------------------------

                This algorithm now uses the new run length encode
                function provided by the Build 9 ORPG.

-----------------------------------------------------------------------
Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org
```

```
Version 1.9,  June 2004    T. Ganger
              Modified to reflect the new size of the base data header.

Version 2.0   March 2006   T. Ganger
              Replaced stderr messages outside of test/debug sections with
                   RPGC_log_msg() calls.
              Revised to use current guidance for abort functions and
                   abort reason codes.

Version 2.1   June 2006   T. Ganger
              Revised to use RPGC_reg_io
              Revised to use the new 'by_name' get_inbuf/get_outbuf functions

Version 2.2   July 2006   T. Ganger
              Modified to use the radial header offset to surveillance
                  data 'ref_offset'.  Tested RPGC_get_surv_data.
              Only malloc space for the header and surveillance data rather
                   than the complete radial
              Modified to use RPGC_run_length_encode rather than local
                  function.
              Replaced WAIT_ALL with WAIT_DRIVING_INPUT.
              In symb_layer.c: Used the radial header fields for index to
                  first good bins and number of bins rather than just assuming
                  the usual values.

Version 2.3   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.
              Added note that MAX_BASEDATA_REF_SIZE must be used instead of
                  BASEDATA_REF_SIZE for super resolution elevations.
              RPGC_get_radar_data failed in this algorithm for Build 9.

Version 2.4   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated use of defined buffer id's by using
                  RPGC_get_id_from_name
              Changed number of bins from 230 to 460 maximum and used a
                  variable to hold the value rather than a constant.
              Used globally defined constants for maximum number of bins
                  and maximum number of radials.
              Reduced size of the internal data array if needed to account
                  for the last "good" bin in the input array.
              Used the new default abort reason code PGM_PROD_NOT_GENERATED.
              Passed calibration constant as a single float rather than
                  two shorts.

Version 2.5   Movember 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Added test to ensure that the algorithm did not attempt to
                  read or copy more bins than are actually in each
                  base data radial.  Depending upon the method used to read
                  the radial, a specific index is tested.
              Eliminated overlapping radials from the output product by
                  terminating processing at pseudo end of elev / volume and
                  reading until actual end of elev / volume.

Version 2.6   February 2009    T. Ganger  (Sample Algorithm ver 1.21)
              Changed input data from REFLDATA to SAMPLE2IN in order to
                  facilitate modification to tast_attr_table to switch
```

```
                between REFLDATA(79) and DUALPOL_REFLDATA(307).
            If ingesting DUALPOL_REFLDATA, increased max number of bins,
                increased max number of radials, and increased the size
                of the output buffer allocated. In Build 12 the max
                number of radials will not have to be increased.
            Modified test section printing out radial information
                including status to reflect ORDA usually having 360
                radials.
            Added test section to print dual pol data parameters.

$Id$
**********************************************************************/


/* see the following include file for descriptions of constants for
for expected ICD block sizes */
#include "sample2_radrefl.h"


/**********************************************************************
Description:    main function to drive the sample radial data array
                algorithm
Input:          one radial at a time as defined in the Base_data_radial
                structure (see basedata.h)
Output:         a formatted ICD graphical product which is forwarded to
                the ORPG for distribution
Returns:        none
Globals:        none
Notes:          1). This sample algorithm is a demonstration of producing
                a run length encoded (RLE) product from base data.
                2). This algorithm provides the capability to set the
                desired product range up to 460 bins.  This limit is
                related to registration for a non super resolution data
                type : BASEDATA, RECOMBINED_RAWDATA, etc.
                3). The program takes no command line arguments.

**********************************************************************/
int main(int argc,char* argv[]) {
  /* variable declarations and definitions ---------------------------- */
  Base_data_radial
      *basedataPtr=NULL;    /* a pointer to a base data radial structure*/
  Base_data_header *hdr;    /* a pointer to the base data header        */

  int ref_enable;           /* variables: used to hold the status of    */
  int vel_enable;           /* the individual moments of base data.     */
  int spw_enable;           /* tested after reading first radial        */

  int PROCESS=TRUE;         /* Boolean used to control the daemon       */
                            /* portion of the program. Process until set*/
                            /* to false.                                */
  int i;                    /* loop variable                            */

  int rad_idx=0;            /* counter: radial index being processed    */
  int last_rad=0;           /* Sample Alg 1.21 */
  short elev_idx=0;         /* variable: used to hold current elev index*/
  short elevation=0;        /* variable: used to hold the value of elev */
  int vol_num=0;            /* current volume number                    */
  int opstatus;             /* variable:  used to hold return values    */
                            /* from system calls                        */
  short *buffer;            /* pointer: access to allocated memory to   */
```

```
                            /* hold a completed ICD product             */
  short radial_status=0;    /* variable: status of each radial input    */
  /* Sample Alg 1.21 - increased to an array of 800 pointers for test   */
  char *radial[BASEDATA_MAX_SR_RADIALS]; /* 800 */
                            /* WARNING Must be increased from 400 to    */
                            /* 800 if registered for an SR_ data type   */
  float cal_const;          /* calibration constant (used for pdp)      */

  int output_id, input_id;

  /* dynamically determine number of bins to process */
  int bins_to_process = 0;  /* number of data bins to process in radial */
  int bins_to_70k;
  int max_num_bins = BASEDATA_REF_SIZE;  /* 460 */
                            /* could be reduced to limit range of product   */
                            /* and is subsequently limited by 70,000 ft MSL */
  int max_num_radials = BASEDATA_MAX_RADIALS;  /* 400 */

/* Sample Alg 1.20 - make sure we do not read more than is in a radial */
  int max_bins_to_copy;
  int bins_to_copy;
  short *short_data=NULL;

  short *surv_data=NULL;   /* pointer to the surveillance data in the radial */
  int index_first_bin=999;  /* used with method 2             */
  int index_last_bin=999;   /* used with method 2             */

  Generic_moment_t gen_moment;  /* used with method 3         */
  /* if passed as NULL causes a malloc by RPGC_get_radar_data */
  Generic_moment_t *my_gen_moment = &gen_moment;  /* method 3 */


/* test / debug variable declarations and definitions -------------------*/
  int TEST=FALSE;           /* test flag: set to true for diag messages */

  int VERBOSE=FALSE;        /* verbose processing output                */


  hdr = NULL;

  fprintf(stderr, "\nBegin Radial Reflectivity Algorithm in C\n");

  /* LABEL:REG_INIT  Algorithm Registration and Initialization Section  */

  RPGC_init_log_services(argc, argv);


  /* register inputs and outputs based upon the           */
  /* contents of the task_attr_table (TAT).               */
  /* Sample Alg 1.21 */
  /* input: either REFLDATA(79), or DUALPOL_REFLDATA(307)  */
  /*        based upon definition of SAMPLE2IN            */
  /* output:RADREFL(1991)                                 */
  RPGC_reg_io(argc, argv);

/* register algorithm infrastructure to read the Scan Summary Array   */
  RPGC_reg_scan_summary();

  /* register adaptation data and read current data into structures    */
```

```
  /* ORPG task initialization routine. Input parameters argc/argv are   */
  /* not used in this algorithm                                         */
  RPGC_task_init(ELEVATION_BASED, argc, argv);

  fprintf(stderr, "-> algorithm initialized and proceeding to loop control\n");

  RPGC_log_msg( GL_INFO,
       "-> algorithm initialized and proceeding to loop control\n");


  /* Sample Alg 1.21 */
  input_id = RPGC_get_id_from_name( "SAMPLE2IN");


  /* while loop that controls how long the task will execute. As long as*/
  /* PROCESS remains true, the task will continue. The task will        */
  /* terminate upon an error condition                                  */
  while(PROCESS) {

    /* system call to indicate a data driven algorithm. block algorithm */
    /* until good data is received and the product is requested         */
    RPGC_wait_act(WAIT_DRIVING_INPUT);

   /* LABEL:BEGIN_PROCESSING Released from Algorithm Flow Control Loop   */
   if (VERBOSE) fprintf(stderr,
        "-> algorithm passed loop control and begin processing\n");

   RPGC_log_msg( GL_INFO,
        "-> algorithm passed loop control and begin processing\n");

    /* allocate a partition (accessed by the pointer, buffer) within the*/
    /* RADREFL linear buffer. error returns in opstatus                 */

    /* Sample Alg 1.21 - With the option to register DUALPOL_REFLDATA in */
    /*       instead of REFLDATA, the maximum number fo bins could be    */
    /*       1840 instead of 460                                         */
    /*       DUALPOL_REFLDATA will have an azimuth resolution of 1.0 deg */
    /*       for Build 12, however in Build 11 we get 0.5 deg            */
    if(input_id == 307) {
        buffer = (short*)RPGC_get_outbuf_by_name("RADREFL", SR_BUFSIZE, &opstatus);
        max_num_bins = MAX_BASEDATA_REF_SIZE; /* 1840 */
        /* the following will not be required with the final version of */
        /* dpprep output.  The azimuth will be recombined to 400 max    */
        max_num_radials = 800;
        if(VERBOSE)
            fprintf(stderr,"Setting num radials to %d and radial size to %d\n",
                                                 max_num_radials, max_num_bins);
    } else {
        buffer = (short*)RPGC_get_outbuf_by_name("RADREFL", BUFSIZE, &opstatus);
        max_num_bins = BASEDATA_REF_SIZE; /* 460 */
        if(VERBOSE)
            fprintf(stderr,"Using Legacy num radials and radial size\n");

    }



    /* check error condition of buffer allocation. abort if abnormal    */
    if(opstatus != NORMAL) {
```

```
        RPGC_log_msg( GL_INFO,
            "ERROR: Aborting from RPGC_get_outbuf...opstatus=%d\n",
                                                        opstatus);
        RPGC_abort();
        continue;
        }
    if (VERBOSE) {
      fprintf(stderr, "-> successfully obtained output buffer\n");
    }
    /* make sure that the buffer space is initialized to zero           */
    clear_buffer((char*)buffer, BUFSIZE);

   /* LABEL:READ_FIRST_RAD Read first radial of the elevation and      */
   /* accomplish the required moments check                            */
    /* ingest one radial of data from the BASEDATA linear buffer. The */
    /* data will be accessed via the basedataPtr pointer               */

    if (VERBOSE) {
      fprintf(stderr, "-> reading first radial buffer\n");
    }
    /* Sample Alg 1.21 */
    basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("SAMPLE2IN",
                                                            &opstatus);

    hdr = (Base_data_header *) basedataPtr;

    if (VERBOSE) {
        fprintf(stderr, "     opstatus = %d \n", opstatus);
    }

    /* check radial ingest status before continuing                    */
    if(opstatus != NORMAL){
        RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_get_inbuf\n");
        RPGC_rel_outbuf((void*)buffer, DESTROY);
        RPGC_abort();
        continue;
    }

    if (VERBOSE) {
        fprintf(stderr, "-> successfully read first radial buffer\n");
    }

    /* test to see if the required moment (reflectivity) is enabled   */
    RPGC_what_moments((Base_data_header*)basedataPtr, &ref_enable,
                                        &vel_enable, &spw_enable);

    if(ref_enable != TRUE){
        RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_what_moments\n");
        RPGC_rel_inbuf((void*)basedataPtr);
        RPGC_rel_outbuf((void*)buffer, DESTROY);
        RPGC_abort_because(PGM_DISABLED_MOMENT);
        continue;
    }

    if (VERBOSE) {
      fprintf(stderr, "-> required moments enabled\n");
    }

/* Sample Alg 1.21 */
/* BEGIN TEST SECTION */
```

```
/*    if(input_id == 307) {                          */
/*        test_read_dp_moment_params(basedataPtr);  */
/*    }                                              */
/* END TEST SECTION */

    if (VERBOSE) {
      fprintf(stderr, "-> begin ELEVATION PROCESSING SEGMENT\n");
    }
    /* ELEVATION PROCESSING SEGMENT. continue to ingest and process     */
    /* individual base radials until either a failure to read valid     */
    /* input data (a radial in correct sequence) or until after reading */
    /* and processing the last radial in the elevation                  */

    /* get the current elevation & index from the incoming data       */
    elev_idx = (short)RPGC_get_buffer_elev_index( (void *)basedataPtr );

    /* get the current volume number using the following system call  */
    vol_num = RPGC_get_buffer_vol_num( (void *)basedataPtr );

    /* if not ingesting base data, RPGCS_get_target_elev_ang must be used */
    elevation = basedataPtr->hdr.target_elev;  /* target elevation */

    /* determine the number of bins to process here      */
    /* limit number of bins to the 70.000 ft MSL ceiling */
    bins_to_70k = RPGC_bins_to_ceiling( basedataPtr, hdr->surv_bin_size );



    if(bins_to_70k < max_num_bins)
        bins_to_process = bins_to_70k;
    else
        bins_to_process = max_num_bins;
    /* Sample Alg 1.20 - */
    /* ensure even number of bins to prevent possible alignment issues */
    if( (bins_to_process % 2) !=0)
        bins_to_process++;

    /* NOTE: when accomplishing numerical calculations including assembly   */
    /*       of the final product arrays, the number of good bins must be   */
    /*       accounted for.  This can be made by                            */
    /*          a. reducing the size of the internal data array processed   */
    /*             (bins_to_process), or by                                 */
    /*          b. ensuring all data values after the last good bin are     */
    /*             set to a value of '0'.                                   */
    /*       This algorithm chooses b and is tested for each radial         */



    while(TRUE) {

      /* LABEL:PROCESS_RAD Here we process each radial individually     */
      /* any processing that can be accomplished without comparison     */
      /* between other radial can be accomplished here                  */


      cal_const = basedataPtr->hdr.calib_const;       /* calibration const*/

/* DEBUG */
/*fprintf(stderr,"DEBUG at top of radial processing loop\n"); */
```

```
      /* for this algorithm...we'll copy the radial header and the      */
      /* REFLECTIVITY structure to the temporary storage array. This    */
      /* effectively accumulates radials into an entire elevation.  The */
      /* routine requires several steps. First, allocate enough memory  */
      /* to hold the radial header plus data. Then copy the header and  */
      /* radial data structure portion into the allocated space. The    */
      /* radial can then be accessed via the pointer in the radial array*/

      /* NOTE: IT IS NOT ALWAYS NECESSARY TO COPY THIS DATA.  IF ONLY   */
      /*       PROCESSING A SINGLE RADIAL, THE RADIAL COULD BE          */
      /*       PROCESSED IN THE INPUT BUFFER AND WRITTEN DIRECTLY TO AN */
      /*       OPEN OUTPUT BUFFER. THIS WOULD REDUCE MEMORY USAGE.      */


/* While all of the following work, Method 2 is typical for basic moments */
/* Method 3 must be used for advanced Dual Pol moments                    */
      /* Method 1 */
/*      surv_data = (short *) ((char *) basedataPtr + hdr->ref_offset);  */
/*        max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;      */
                          /* '-1' because the indexes are 1-based */
                          /* we do not assume hdr->surv_range is always 1 */

      /* Method 2 */
      surv_data = (short *) RPGC_get_surv_data( (void *)basedataPtr,
                                  &index_first_bin, &index_last_bin);
      max_bins_to_copy = index_last_bin + 1;
                     /* '+1'because indexes are 0-based */
                     /* we do not assume that index_first_bin is always 0 */

      /* Method 3 */
/*      surv_data = (short *) RPGC_get_radar_data( (void *)basedataPtr,  */
/*                                    RPGC_DREF, my_gen_moment );         */
/*        max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;      */

      /* test for NULL moment pointer and 0 offset */
      if(surv_data == NULL) {
         RPGC_log_msg( GL_INFO, "ERROR: NULL radial detected: %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)buffer, DESTROY);
         RPGC_abort_because(PGM_INPUT_DATA_ERROR);
         opstatus = TERMINATE;
         break;
      }

      if(hdr->ref_offset == 0) {
         RPGC_log_msg( GL_INFO, "ERROR: reflectivity offset 0: %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)buffer, DESTROY);
         RPGC_abort_because(PGM_INPUT_DATA_ERROR);
         opstatus = TERMINATE;
         break;
      }

      /* allocate one redials worth of memory                          */
      radial[rad_idx] = (char *)malloc(sizeof(Base_data_header) +
                                                  (bins_to_process * 2) );
      /* NOTE: If registered for SR_ data and reading from an SR   */
```

```
      /*       elevation, must use MAX_BASEDATA_REF_SIZE           */


      if(radial[rad_idx] == NULL) {
         RPGC_log_msg( GL_INFO, "MALLOC Error: malloc failed on radial %d\n",
            rad_idx);
         RPGC_rel_inbuf((void*)basedataPtr);
         RPGC_rel_outbuf((void*)buffer, DESTROY);
         RPGC_abort_because(PGM_MEM_LOADSHED);
         opstatus = TERMINATE;
         break;
      }

      /* copy the radial header first                                    */
      memcpy(radial[rad_idx], basedataPtr, sizeof(Base_data_header));

      /* copy the reflectivity data second                               */
      /* Sample Alg 1.20 - only copy the the lessor of (a) number of bins in  */
      /*                   the radial or (b) bins_to_process                  */
      if(bins_to_process <= max_bins_to_copy)
         bins_to_copy = bins_to_process;
      else
         bins_to_copy = max_bins_to_copy;
      /* when writing the output product, any data bins */
      /* beyond max_bins_to_copy are set to '0'         */
      /* Sample Alg 1.20 - used bins_to_copy instead of bins_to_process */
      memcpy(radial[rad_idx] + sizeof(Base_data_header),
                      (char *)surv_data, (bins_to_copy * 2));

      short_data = (short *) ( radial[rad_idx] + sizeof(Base_data_header) );
      for(i = 0; i < bins_to_process; i++)
         /* ensure any data bins beyond the last good bin are set to 0 */
         if( i >= bins_to_copy )
             short_data[i] = (short) 0;

             /* NOTE: If registered for SR_ data and reading from an SR   */
             /*       elevation, must use MAX_BASEDATA_REF_SIZE           */


      /* test section: this optional block has been used to double check*/
      /* the integrity of the new radial memory block                    */
      if(TEST) {
         /* Sample Alg 1.21 - modified test from 363 to 357 for ORDA */
         if ( (rad_idx <= 3) || (rad_idx >= 357) ) {
            Base_data_header *hdr=(Base_data_header*)radial[rad_idx];
            fprintf(stderr,"  RADIAL CHECK FOR RADIAL %d\n", rad_idx+1);
            fprintf(stderr,"    Azimuth=%f\n",hdr->azimuth);
            fprintf(stderr,"    Start Angle=%hd\n",hdr->start_angle);
            fprintf(stderr,"    Angle Delta=%hd\n",hdr->delta_angle);
            fprintf(stderr,"    calib const=%f\n",cal_const);
            /* read the radial status flag  */
            radial_status = basedataPtr->hdr.status & 0xf;
            fprintf(stderr, "    Radial_Status = %d where\n"
                            "    0=beg_ele, 1=int, 2=end_ele, 3=beg_vol,
4=end_vol,\n"
                            "    8=pseudo_end_ele, 9=pseudo_end_vol\n",
                                                                radial_status);
         } /* end if rad_idx */

      } /* end if TEST */
```

```
      /* now that the data that we want from the radial is in temporary */
      /* storage...release the input buffer                             */
      RPGC_rel_inbuf((void*)basedataPtr);

      /* increment the radial counter                                   */
      rad_idx++;

/* DEBUG */
/*fprintf(stderr,"DEBUG finished copying to radial[%d]\n", rad_idx-1); */


      /* check for radial overflow in the input array                   */
      /* this should never happen if we define max_num_radials correctly*/
      /* and the ORPG has not been changed (pbd algorithm)              */
      if(rad_idx > max_num_radials) {
          RPGC_log_msg( GL_INFO,
               "Count Error: radial index %d exceeded array limits %d\n",
                                                  rad_idx, max_num_radials);
          RPGC_rel_outbuf((void*)buffer, DESTROY);
          RPGC_abort_because(PGM_INPUT_DATA_ERROR);
          opstatus = TERMINATE;
          break;
      }

      /* if end of elevation or volume then exit loop.                 */
      /* Sample Alg 1.20 - modified to exit radial processing at       */
      /*                   pseudo end rather than actual end in        */
      /*                   order to avoid overlapping radials if       */
      /*                   ingesting historical data before ORDA       */
      /*      This requires subsequently reading to the actual end     */
      /*      of elevation                                             */
      if(radial_status == GENDEL || radial_status == GENDVOL ||
         radial_status == PGENDEL || radial_status == PGENDVOL ) {
          if (VERBOSE)
               fprintf(stderr,"-> End of elevation found\n");
          /* exit with opstatus==NORMAL and no ABORT */
          break;
      }

      /* LABEL:READ_NEXT_RAD Read the next radial of the elevation      */
      /* ingest one radial of data from the BASEDATA linear buffer. The */
      /* data will be accessed via the basedataPtr pointer              */


      /* Sample Alg 1.21 */
      basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("SAMPLE2IN",
                                                              &opstatus);

      hdr = (Base_data_header *) basedataPtr;

      /* check radial ingest status before continuing                   */
      /* check radial ingest status before continuing                   */
      if(opstatus != NORMAL){
          RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_get_inbuf\n");
          RPGC_rel_outbuf((void*)buffer, DESTROY);
          RPGC_abort();
          break;
      }
```

```
    } /* end while TRUE, reading radials */

    /* Sample Alg 1.21 */
    last_rad = rad_idx;

    if (VERBOSE) {
      fprintf(stderr, "=> out of radial ingest loop...radial  = %d\n", rad_idx+1);
      fprintf(stderr, "   Radial Status = %d  where\n"
                      "    0=beg_ele, 1=int, 2=end_ele, 3=beg_vol, 4=end_vol,\n"
                      "    8=pseudo_end_ele, 9=pseudo_end_vol\n",
                                                          radial_status);
    }

    /* LABEL:PROCESS_ELEV Here we accomplish any processing of the      */
    if(opstatus == NORMAL) {

         /* Sample Alg 1.20 - may have exited on pseudo end, now must read */
         /*                   until the actual end of elevation / vol       */
         while(TRUE) { /* read remaining radials if any */
            if( radial_status != GENDEL && radial_status != GENDVOL ) {
               if (TEST)
                  fprintf(stderr, "=> exited on pseudo, looking for actual end\n");
               /* Sample Alg 1.21 */
               basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("SAMPLE2IN",
                                                                  &opstatus);
               if(opstatus != NORMAL){
                  RPGC_log_msg( GL_INFO,
                    "ERROR: Aborting from RPGC_get_inbuf, opstatus=%d\n",
                                                              opstatus);
                  RPGC_rel_outbuf((void*)buffer, DESTROY);
                  RPGC_abort();
                  break;
               }
               hdr = (Base_data_header *) basedataPtr;
               radial_status = basedataPtr->hdr.status & 0xf;
               RPGC_rel_inbuf((void*)basedataPtr);
               rad_idx++; /* Sample Alg 1.21 */

            } else { /* we found actual end of elevation / volume */
                if (TEST)
                   fprintf(stderr,
                           "=> finished finding actual end, radial = %d\n",
rad_idx+1);
                   fprintf(stderr,
                           "   Radial Status = %d  where\n"
                           "    0=beg_ele, 1=int, 2=end_ele, 3=beg_vol,
4=end_vol,\n"
                           "    8=pseudo_end_ele, 9=pseudo_end_vol\n",
                                                          radial_status);
               /* exit with opstatus==NORMAL and no ABORT */
               break;
            }

          } /* end while */


        /* elevation as a whole.  Processing that requires comparisons      */
        /* between radials should be accomplished here                      */
        ;
```

```
    }
    /* the end of the ELEVATION PROCESSING SEGMENT of the algorithm     */



    /* LABEL:ASSEMBLE The assembly of the ICD formatted product         */
    /* if processing is normal then create the ICD output product       */
    if(opstatus == NORMAL) {
        int result;     /* function return value                        */
        int length=0;   /* accumulated product length                   */
        int max_refl=0; /* maximum reflectivity value                   */


      /* building the ICD formatted output product requires a few steps */
      /* step 1: build the product description block (pdb)              */
      /* step 2: build the symbology block & data layer                 */
      /* step 3: complete missing values in the pdb                     */
      /* step 4: build the message header block (mhb)                   */
      /* step 5: forward the completed product to the system            */

      output_id = RPGC_get_id_from_name( "RADREFL");

      /* step 1: build the product description block -uses a system call*/
      if (VERBOSE) {
        fprintf(stderr, "\nCreating the product description block now: vol=%d\n",
                                                                    vol_num);
      }
      RPGC_prod_desc_block((void*)buffer, output_id, vol_num);

      /* for testing diagnostics show the header information now        */
/*      if(TEST) {
 *         fprintf(stderr,"\n==> OUTPUT INITIAL MHB AND PDB\n");
 *         result=print_message_header((char*)buffer);
 *         result=print_pdb_header((char*)buffer);
 *      }
 */
      /* step 2: build the symbology layer & RLE radial data packet.    */
      /* this routine returns both the overall length (thus far) of the */
      /* product and the maximum reflectivity of all radials            */
      if (VERBOSE) {
        fprintf(stderr,"begin building the symbology and AF1F layers\n");
      }

      /* Sample Alg 1.21 - changed rad_idx to last_rad */
      max_refl = build_symbology_layer(buffer, radial, last_rad,
                                             bins_to_process, &length);

      if (VERBOSE) {
        fprintf(stderr, "finish building the symbology and AF1F layers\n");
      }
      /* step 3: finish building the product description block by       */
      /* filling in certain values such as elevation index, maximum     */
      /* reflectivity, accumulated product length, etc                  */
      /* passing calibration constant as a single float */
      finish_pdb(buffer, elev_idx, elevation, cal_const, max_refl, length);

      /* generate the product message header (use system call) and input*/
      /* total accumulated product length minus 120 bytes for the MHB & PDB */
      result = RPGC_prod_hdr((void*)buffer, output_id, &length);
```

```
      if (VERBOSE) {
        fprintf(stderr,"-> completed product length=%d\n",length);
      }
      /*(this routine adds 120 bytes for the MHB & PDB to the      */
      /* "length" parameter prior to creating the final header)         */

      /* if the creation of the product has been a success              */
      if(result == 0) { /*success!!!*/
        int res;
        if(VERBOSE) {
           fprintf(stderr, "product header creation success...display header\n");
           res = print_message_header((char*)buffer);
           res = print_pdb_header((char*)buffer);
           fprintf(stderr, "\nprinting of product header complete\n");
        }


        /* LABEL:OUTPUT_PROD   forward product and close buffer         */
        RPGC_rel_outbuf((void*)buffer, FORWARD);

      } else {  /* product failure!  (destroy the buffer & contents)      */
        RPGC_log_msg( GL_INFO, "product header creation failure. buffer abort\n");
        RPGC_rel_outbuf((void*)buffer, DESTROY);
        RPGC_abort_because(PGM_PROD_NOT_GENERATED);
      } /* end result !=0 */

    }  /* end of if(opstatus==NORMAL) block */


    /* LABEL:CLEANUP   section to reset variables / free memory         */
    /* free each previously allocated radial array                      */

    /* Sample Alg 1.21 - changed rad_idx to last_rad */
    if(TEST) fprintf(stderr, "free radial array (last_rad=%i)\n", last_rad);
    for(i = 0; i < last_rad; i++)
        free(radial[i]);

    rad_idx = 0; /* reset radial counter to 0                                */
    last_rad = 0;

    RPGC_log_msg( GL_INFO,"process reset: ready to start new elevation\n");

    } /* end while PROCESS == TRUE */

   fprintf(stderr, "\nRadial Reflectivity Program Complete\n");

   return(0);

} /* end main */



/************************************************************************
Description:    clear_buffer: initializes a portion of allocated
                memory to zero
Input:          pointer to the input buffer (already cast to char*)
Output:         none
Returns:        none
Globals:        the constant BUFSIZE is defined in the include file
```

```
Notes:          none
*********************************************************************/
void clear_buffer(char *buffer,int value) {
  /* zero out the input buffer */
  int i;

  for(i = 0; i < value; i++)
    buffer[i] = 0;

  return;

}
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/**********************************************************************
Module:         sample2_radrefl.h

Description:    include file for the 16-level reflectivity algorithm
                sample2_radrefl.c. This file contains module specific
                         include file listings, function prototypes and
                         constant definitions.
----------------------------------------------------------------------
          Modified to reflect the new size of the base data header

----------------------------------------------------------------------
Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.5,  April 2005   T. Ganger
              Reflect the new size of the base data header (Build 5)
              Modified to use the new product IDs

Version 1.6   March 2006   T. Ganger
              Modified to reflect the new base data message, larger
                   header, new order of moment data, possible
                   larger reflectivity array - Build 8

Version 1.7   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.

Version 1.8   February 2008    T. Ganger    (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated definition of output buffer ids.
              Changed reflectivity product from maximum 230 bins to a
                  maximum 460 bins.
              Used globally defined constants for maximum number of bins
                  and maximum number of radials.
              Passed calibration constant as a float.

Version 1.9   February 2009    T. Ganger    (Sample Algorithm ver 1.21)
              Added function to print dual pol data parameters.
              Added max size estimate for SR output data.

$Id$
**********************************************************************/
#ifndef _RAD_REFL_H_
```

```
#define _RAD_REFL_H_

/* system includes ------------------------------------------------- */


/* ORPG   includes ------------------------------------------------- */
#include <rpg_globals.h>

/*** the following are included in rpg_globals.h */
/***                                             */
/***  #include <stdio.h>                         */
/***  #include <stdlib.h>                        */
/***  #include <time.h>                          */
/***  #include <ctype.h>                         */
/***  #include <string.h>                        */
/***                                             */
/***  #include <a309.h>                          */
/***  #include <basedata.h>                      */
/***  #include <basedata_elev.h>                 */
/***  #include <rpg_port.h>                      */
/***  #include <prod_gen_msg.h>                  */
/***  #include <prod_request.h>                  */
/***  #include <prod_gen_msg.h>                  */
/***  #include <rpg_vcp.h>                       */
/***  #include <orpg.h>                          */
/***  #include <mrpg.h>                          */
/************************************************/

#include <rpgc.h>
#include "rpgcs.h"


/* ---------------------------------------------------------------------- */
/* the structure of a base data radial message, which is ingested by the*/
/* algorithm, is defined in  basedata.h                                   */

/*
 * The order and size of the base data data arrays are no longer fixed.
 *
 * The offsets in the base data header or the API access function must
 * be used to read the basic moments: reflectivity, velocity, and
 * spectrum width.
 *
 * Fields in the base data header are used to determine array size.
 */

/* The contents of Base_data_header is defined in basedata.h  */
/* ---------------------------------------------------------------------- */



/* local  includes ------------------------------------------------- */


/* ORPG constants not defined in include files ----------------------- */

/* number of bins per radial (BPR) (can be set to 230 up_to 460)       */
/* if using the new SR_ data types, this could be set to 1840          */
/* this is used to set the value of a single    */
/*             variable in the main module      */
```

```
#define PROD_RANGE_NUM_BINS 460  /* WARNING - This must correspond to   */
                     /* the value set for BUFSIZE below                */



/* Algorithm specific constants/definitions --------------------------- */

/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define RADREFL 1991 */  /* ORPG Linear Buffer Code, normally */
                            /* defined in a309.h , must match    */
                            /* the product_tables configuration  */
                            /* file entry for this product       */



#define BUFSIZE  92000      /* estimated output size for ICD product    */
                            /* run length encoded products will vary    */
                            /* in size, this must be larger than any    */
                            /* product produced by the algorithm        */

/* The following is estimated mas size for SR_ data of at least 720     */
/* radials and 1840 bins                                                */
#define SR_BUFSIZE 735000

/* function prototypes ------------------------------------------------*/
void clear_buffer(char*,int);

extern short build_symbology_layer(short* buffer,char **radial,int rad_count,
                                                  int bin_count, int* length);

extern void finish_pdb(short* buffer, short elev_ind, short elevation,
                       float calib_const, short max_refl, int prod_len);

extern int print_message_header(char* buffer);

extern int print_pdb_header(char* buffer);

/* Sample Alg 1.21 */
extern void test_read_dp_moment_params(Base_data_radial *radialPtr);

#endif
```

Part D. Sample Algorithm 3 Tasks 1 & 2 Main Module

SAMPLE 3 TASK 1
(UPDATED MAY 2009)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample3_t1.c

Description:    sample3_t1.c is part of a chain-algorithm demonstration.
                The first task reads radial data and produces elevation
                data.  The second task reads elevation data and produces
                volume data.

                THE PURPOSE OF THIS SAMPLE ALGORITHM IS TO DEMONSTRATE
                A MULTIPLE TASK ALGORITHM.  NO USEFUL SCIENCE IS
                ACCOMPLISHED BY THE SECOND TASK

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3.

                sample3_t1.c reads each radial and places the base data
                header and reflectivity moment into a container. When
                the elevation completes a product is output. This structure
                is an early demonstration of a polar array intermediate
                product and will be replaced in the future with the generic
                radial component.

                Key source files for this algorithm include:

                sample3_t1.c         program source
                sample3_t1.h         main include file

                s3_t1_prod_struct.h  definition of s1_t1_intermed_prod_hdr

Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.4,  May 2004     T. Ganger
              Linux update

Version 1.5   March 2006   T. Ganger
```

```
              Replaced stderr messages outside of test/debug sections with
                   RPGC_log_msg() calls.
              Revised to use current guidance for abort functions and
                   abort reason codes.
              Simplified reading short data into byte arrays

Version 1.6   June 2006   T. Ganger
              Revised to use RPGC_reg_io
              Revised to use the new 'by_name' get_inbuf/get_outbuf functions

Version 1.7   July 2006   T. Ganger
              Modified to use the radial header offset to surveillance
                  data 'ref_offset'.  Tested RPGC_get_surv_data.
              Only malloc space for the header and surveillance data rather
                   than the complete radial
              Replaced WAIT_ALL with WAIT_DRIVING_INPUT.

Version 1.8   March 2007  T. Ganger
              Included test code that aborts product output on designated
                  elevations in order to test this task's output as an
                  optional input to sample 2 task 1 (tsk005).
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.
              Added note that MAX_BASEDATA_REF_SIZE must be used instead of
                  BASEDATA_REF_SIZE for super resolution elevations.
              Removed the non-standard test for radial limit.  This would
                  have to be a dynamic test with SR_ data types.
              RPGC_get_radar_data failed in this algorithm for Build 9.

Version 1.9   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated use of defined output buffer ids by using
                  RPGC_get_id_from_name
              Modified to handle variable array sizes and changed product
                  range from 230 km to 460 km.
              When copying each radial, demonstrated ensuring that data
                  values beyond the last good data bin (if any) are set to '0'.


Version 2.0   November 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Updated description of algorithm task and main function.
                  Added comments to clarify processing.
              Added test to ensure that the algorithm did not attempt to
                  read or copy more bins than are actually in each
                  base data radial.  Depending upon the method used to read
                  the radial, a specific index is tested.
              Fixed a problem where the elevations after the first had some
                  blank radials.
              Eliminated overlapping radials from the output product by
                  terminating processing at pseudo end of elev / volume and
                  reading until actual end of elev / volume.
              Used a new structure to write the output intermediate product.

Version 2.1   March 2009    T. Ganger  (Sample Algorithm ver 1.21)
              Modified test section printing out radial information
                  including status to reflect ORDA usually having 360
                  radials.
```

```
$Id$
**********************************************************************/

/* see the following include file for descriptions of constants and
for expected ICD block sizes */
#include "sample3_t1.h"


/**********************************************************************
Description:    main function to drive the chain-algorithm demonstration
                (first task out of two).
Input:          basedata radial as defined in the Base_data_radial
                structure (see the basedata.h include file)
Output:         an intermediate product formatted via a header structure in
                s3_t1_prod_struct.h with radial data in an array of bytes
                (unsigned char).
Returns:        none
Globals:        none
Notes:          1). This sample algorithm task is a demonstration of
                    producing an elevation product from radial data.  A
                    copy of all radials is made into an elevation
                    structure to facilitate processing multiple radials.
                    Memory could be saved if all of the processing is on
                    a radial by radial basis.
                2). This algorithm can set different radial sizes not to
                    exceed 460 bins.  Since the maximum number of radials
                    is hard coded to 400 (and a static array used), this
                    algorithm cannot use any of the super resolution
                    data types.  Hence 460 is the maximum number of bins.
                3). The program takes no command line arguments.

**********************************************************************/
int main(int argc,char* argv[]) {
   Base_data_radial
       *basedataPtr=NULL;    /* pointer to a base data radial structure   */
   Base_data_header *hdr;    /* pointer to the base data header structure */

   int *out_buffer;          /* pointer: access to allocated memory       */
                             /* which holds a completed output product    */

   /* A limit of 400 radials precludes this algorithm from */
   /* using one of the super resolution data types         */
   char *radial[BASEDATA_MAX_RADIALS];
                             /* array of pointers to radial structures    */
                             /* used to contain an entire elevation       */
                             /* WARNING Must be increased from 400 to     */
                             /* 800 if registered for an SR_ data type    */

   int ref_enable;           /* variables: used to hold the status of     */
   int vel_enable;           /* the individual moments of base data.      */
   int spw_enable;           /* tested after reading the first radial     */

   int PROCESS=TRUE;         /* Boolean used to control the daemon        */
                             /* portion of the program. Process until set */
                             /* to FALSE.                                 */

   int TEST=FALSE;            /* Boolean to allow diagnostic output        */
/*   int TEST=TRUE; */
```

```
   int VERBOSE=FALSE;       /* Boolean for verbose output                */
/*   int VERBOSE=TRUE; */

   int i;                    /* loop variable                             */
   int rad_idx=0;            /* counter: radial index being processed    */
   int last_rad=0;           /* Sample Alg 1.21 */
   short elev_idx=0;         /* variable: holds current elevation index   */
   short target_elev=0;      /* short: target elevation times 10          */
   int vcp_num=0;            /* variable: holds vcp number                */
   int loop_exit_stat=0;     /* variable: holds exit status from loop     */
   int opstatus;             /* variable: holds function return values    */
   short radial_status=0;      /* variable: status of each radial input     */

  int output_id;


  int bins_to_process = 0;  /* number of data bins to process in radial */
  int bins_to_70k;
  int max_num_bins = BASEDATA_REF_SIZE;  /* 460 */
                           /* could be reduced to limit range of product */
                           /* and is also limited by 70,000 ft MSL       */

/* Sample Alg 1.20 - make sure we do not read more than is in a radial */
  int max_bins_to_copy;
  int bins_to_copy;

/*  int new_buffer_size; */


  short *surv_data;   /* pointer to the surveillance data in the radial */
  int index_first_bin=999;  /* used with method 2 */
  int index_last_bin=999;   /* used with method 2 */

  Generic_moment_t gen_moment;  /* used with method 3 */
  /* if passed as NULL causes a malloc by RPGC_get_radar_data */
  Generic_moment_t *my_gen_moment = &gen_moment;  /* method 3 */

   /* special test flag to abort this task in order to test use of this
    * product as an optional input to sample 4 task 1                    */
   int ABORT_FOR_OPT_SAMPLE_4=FALSE;
/*   int ABORT_FOR_OPT_SAMPLE_4=TRUE; */

   /* ------------------------------------------------------------------ */

   hdr = NULL;

   fprintf(stderr, "\nBegin Sample 3 Task 1 Algorithm\n");

   /* LABEL:REG_INIT  Algorithm Registration and Initialization Section   */

   /* register inputs and outputs based upon the            */
   /* contents of the task_attr_table (TAT).                */
   /* input:REFLDATA(79), output:SAMPLE3_IP(1999)           */
   RPGC_reg_io(argc, argv);

   /* register algorithm infrastructure to read the Scan Summary Array    */
   RPGC_reg_scan_summary();

   /* ORPG task initialization routine. Input parameters argc/argv are    */
   /* not used in this algorithm                                          */
```

```
   RPGC_task_init(ELEVATION_BASED, argc, argv);

   /* output message to both the task output file and the task log file */
   fprintf(stderr,
      "-> algorithm initialized and proceeding to loop control\n");
   RPGC_log_msg( GL_INFO,
      "-> algorithm initialized and proceeding to loop control\n");


   /* while loop that controls how long the task will execute. As long as */
   /* PROCESS remains TRUE, the task will continue. The task will         */
   /* terminate upon an error condition                                   */
   while(PROCESS) {

      /* system call to indicate a data driven algorithm. block algorithm */
      /* until good data is received the product is requested             */
      RPGC_wait_act(WAIT_DRIVING_INPUT);

      /* LABEL:BEGIN_PROCESSING Released from Algorithm Flow Control Loop */
      if (VERBOSE) fprintf(stderr,
          "\n-> sample3_t1 algorithm passed ACL and began processing\n");
      RPGC_log_msg( GL_INFO,
          "-> sample3_t1 algorithm passed ACL and began processing\n");

      /* allocate a partition (accessed by the pointer, buffer) within the*/
      /* SAMPLE3_IP linear buffer. error return in opstatus              */

      out_buffer = (int*)RPGC_get_outbuf_by_name("SAMPLE3_IP", BUFFSIZE,
&opstatus);

      /* check error condition from buffer allocation. abort if abnormal  */
      if(opstatus != NORMAL) {
         RPGC_log_msg( GL_INFO,
            "ERROR: Aborting from RPGC_get_outbuf...opstatus=%d\n", opstatus);
         RPGC_abort();
         continue;
      }

      if (VERBOSE) {
        fprintf(stderr, "-> successfully obtained output buffer size %d\n",
                                                              BUFFSIZE);
      }
      /* make sure that the buffer space is initialized to zero           */
      clear_buffer((char*)out_buffer);

      /* LABEL:READ_FIRST_RAD Read first radial of the elevation and      */
      /* accomplish the required moments check.                           */
      /* ingest one radial of data from the BASEDATA linear buffer. The   */
      /* data will be accessed via the basedataPtr pointer                */
      if (VERBOSE) {
        fprintf(stderr, "sample3_t1 -> reading first radial buffer\n");
      }

      basedataPtr =
          (Base_data_radial*)RPGC_get_inbuf_by_name("REFLDATA", &opstatus);

      hdr = (Base_data_header *) basedataPtr;

      /* check radial ingest status before continuing                     */
      if(opstatus != NORMAL){
```

```
      RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_get_inbuf\n");
      RPGC_rel_outbuf((void*)out_buffer, DESTROY);
      RPGC_abort();
      continue;
   }

   if (VERBOSE) {
     fprintf(stderr,
             "sample3_t1 -> successfully read first radial buffer\n");
   }

   /* test to see if the required moment (reflectivity) is enabled     */
   RPGC_what_moments((Base_data_header*)basedataPtr, &ref_enable,
                                      &vel_enable, &spw_enable);

   if(ref_enable != TRUE){
      RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_what_moments\n");
      RPGC_rel_inbuf((void*)basedataPtr);
      RPGC_rel_outbuf((void*)out_buffer, DESTROY);
      RPGC_abort_because(PGM_DISABLED_MOMENT);
      continue;
   }

   if (VERBOSE) {
     fprintf(stderr, "sample3_t1 -> required moments enabled\n");
   }


   /* ELEVATION PROCESSING SEGMENT. continue to ingest and process     */
   /* individual base radials until either a failure to read valid     */
   /* input data (a radial in correct sequence) or until after reading */
   /* and processing the last radial in the elevation.                 */
   /* Set Exit Status to NORMAL */

   /* get the current elevation angle & index from the incoming data*/
   elev_idx = (short)RPGC_get_buffer_elev_index((void *)basedataPtr);

   vcp_num = RPGC_get_buffer_vcp_num((void*)basedataPtr);
   target_elev = (short)RPGCS_get_target_elev_ang(vcp_num, elev_idx);


   /* limit number of bins to the 70.000 ft MSL ceiling */
   bins_to_70k = RPGC_bins_to_ceiling( basedataPtr, hdr->surv_bin_size );

   if(bins_to_70k < max_num_bins)
       bins_to_process = bins_to_70k;
   else
       bins_to_process = max_num_bins;   /* 460 */
   /* Sample Alg 1.20 - ensure even number of bins for alignment of BDH */
   if( (bins_to_process % 2) !=0)
       bins_to_process++;

   /* NOTE: when accomplishing numerical calculations including assembly   */
   /*       of the final product arrays, the number of good bins must be   */
   /*       accounted for.  This can be made by                            */
   /*          a. reducing the size of the internal data array processed   */
   /*             (bins_to_process), or by                                 */
   /*          b. ensuring all data values after the last good bin are     */
   /*             set to a value of '0'.                                   */
   /*       This algorithm chooses b and is tested for each radial         */
```

```
      rad_idx = 0;
      loop_exit_stat = 0;

      /* ---------------------------------------------------------------- */

      while(TRUE) {

         /* LABEL:PROCESS_RAD Here we process each radial individually.   */
         /* Any processing that can be accomplished without comparison    */
         /* between other radial can be accomplished here.                */


/* DEBUG */
 /* fprintf(stderr,"\nDEBUG at top of radial processing loop\n"); */

         /* ===== BEGIN SPECIAL TEST CODE ================================== */
         /*     THIS CODE BLOCK ABORTS THIS TASK IN ORDER TO TEST THE USE      */
         /*     OF THIS PRODUCT AS AN OPTIONAL INPUT FOR SAMPLE 4 TASK 1.      */
         if(ABORT_FOR_OPT_SAMPLE_4 == TRUE) {
            if(elev_idx == 2) {
                RPGC_log_msg( GL_INFO,
                    "Test Induced Abort at elevation %d\n", elev_idx);
                if(VERBOSE)
                   fprintf(stderr, "Induced Abort at elevation %d\n", elev_idx);
                RPGC_rel_inbuf((void*)basedataPtr);
                RPGC_rel_outbuf((void*)out_buffer, DESTROY);
                RPGC_abort_because(PGM_INPUT_DATA_ERROR);
                loop_exit_stat = TERMINATE;
                break;
            } else {
                if(VERBOSE)
                  if(rad_idx==1)
                    fprintf(stderr, "Abort not induced at elev %d\n", elev_idx);
            }
         }
         /* ===== END SPECIAL TEST CODE ================================== */


/* While all of the following work, Method 2 is typical for basic moments */
/* Method 3 must be used for advanced Dual Pol moments                     */
         /* Method 1 */
/*         surv_data = (short *) ((char *) basedataPtr + hdr->ref_offset); */
/*         max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;      */
                        /* '-1' because the indexes are 1-based */
                        /* we do not assume hdr->surv_range is always 1 */
         /* Method 2 */
         surv_data = (short *) RPGC_get_surv_data( (void *)basedataPtr,
                                  &index_first_bin, &index_last_bin);
         max_bins_to_copy = index_last_bin + 1;
                        /* '+1'because indexes are 0-based */
                        /* we do not assume that index_first_bin is always 0 */

         /* Method 3 */
/*         surv_data = (short *) RPGC_get_radar_data( (void *)basedataPtr, */
/*                                     RPGC_DREF, my_gen_moment );         */
/*         max_bins_to_copy = hdr->surv_range + hdr->n_surv_bins - 1;      */

         /* test for NULL moment pointer (Methods 2 & 3) and for 0 offset */
         if(surv_data == NULL) {
```

```
            RPGC_log_msg( GL_INFO, "ERROR: NULL radial detected: %d\n",
               rad_idx);
            RPGC_rel_inbuf((void*)basedataPtr);
            RPGC_rel_outbuf((void*)out_buffer, DESTROY);
            RPGC_abort_because(PGM_INPUT_DATA_ERROR);
            loop_exit_stat = TERMINATE;
            break;
         }

         if(hdr->ref_offset == 0) {
            RPGC_log_msg( GL_INFO, "ERROR: reflectivity offset 0: %d\n",
               rad_idx);
            RPGC_rel_inbuf((void*)basedataPtr);
            RPGC_rel_outbuf((void*)out_buffer, DESTROY);
            RPGC_abort_because(PGM_INPUT_DATA_ERROR);
            loop_exit_stat = TERMINATE;
            break;
         }


         /* allocate one radial's worth of memory                          */
         radial[rad_idx]=
             (char *)malloc( sizeof(Base_data_header) + (bins_to_process * 2) );


         if(radial[rad_idx] == NULL) {
            RPGC_log_msg( GL_INFO,
                "MALLOC Error: malloc failed on radial %d\n", rad_idx);
            RPGC_rel_inbuf((void*)basedataPtr);
            RPGC_rel_outbuf((void*)out_buffer, DESTROY);
            RPGC_abort_because(PGM_MEM_LOADSHED);
            loop_exit_stat = TERMINATE;
            break;
         }

         /* copy the radial header */
         memcpy(radial[rad_idx], basedataPtr, sizeof(Base_data_header));

         /* copy the reflectivity data */
/* Sample Alg 1.20 - only copy the the lessor of (a) number of bins in  */
/*                   the radial or (b) bins_to_process                  */
         if(bins_to_process <= max_bins_to_copy)
            bins_to_copy = bins_to_process;
         else
            bins_to_copy = max_bins_to_copy;
         /* later when writing the output product, any data bins */
         /* beyond max_bins_to_copy are set to '0'               */

/* DEBUG T1-a */

         memcpy(radial[rad_idx] + sizeof(Base_data_header),
                  (char *)surv_data, bins_to_copy * 2);


         /* read the radial status flag                                    */
         radial_status = basedataPtr->hdr.status & 0xf;

         /* now that the data that we want from the radial is in temporary*/
         /* storage...release the input buffer                             */
         RPGC_rel_inbuf((void*)basedataPtr);
```

```
         /* increment the radial counter                                    */
         rad_idx++;

/* DEBUG T1-b */

         /* if end of elevation or volume then exit loop.                   */
         /* Sample Alg 1.20 - modified to exit radial processing at         */
         /*                   pseudo end rather than actual end in          */
         /*                   order to avoid overlapping radials if         */
         /*                   ingesting historical data before ORDA         */
         /*      This requires subsequently reading to the actual end       */
         /*      of elevation                                               */
         if( radial_status == GENDEL || radial_status == GENDVOL ||
             radial_status == PGENDEL || radial_status == PGENDVOL ) {
            if(TEST) fprintf(stderr,
                        "-> End of elevation found, loop exit status=%d\n",
                         loop_exit_stat);
            /* exit with loop_exit_stat==NORMAL and no ABORT */
            break;
         }

         /* LABEL:READ_NEXT_RAD Read the next radial of the elevation.    */
         /* ingest one radial of data from the BASEDATA linear buffer. The*/
         /* data will be accessed via the basedataPtr pointer             */


         basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("REFLDATA",
                                                                 &opstatus);

         hdr = (Base_data_header *) basedataPtr;

         /* check radial ingest status before continuing                  */
         if(opstatus != NORMAL){
            loop_exit_stat = opstatus;
            RPGC_log_msg( GL_INFO,
               "ERROR: Aborting from RPGC_get_inbuf, loop exit status=%d\n",
                                                          loop_exit_stat);
            RPGC_rel_outbuf((void*)out_buffer, DESTROY);
            RPGC_abort();
            break;
         }

      } /* end while TRUE, read radials */

      /* ------------------------------------------------------------------ */

      /* Sample Alg 1.21 */
      last_rad = rad_idx;


      if (VERBOSE) {
        fprintf(stderr, "=> out of radial ingest loop...elev index=%d\n",
                                                             elev_idx);
      }

      /* LABEL:PROCESS_ELEV Here we accomplish any processing of the      */

      if(loop_exit_stat == NORMAL) {
```

```
         /* Sample Alg 1.20 - may have exited on pseudo end, now must read */
         /*                   until the actual end of elevation / vol       */
         while(TRUE) { /* read remaining radials if any */
            if( radial_status != GENDEL && radial_status != GENDVOL ) {

               basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("REFLDATA",
                                                                       &opstatus);
               if(opstatus != NORMAL){
                  loop_exit_stat = opstatus;
                  RPGC_log_msg( GL_INFO,
                    "ERROR: Aborting from RPGC_get_inbuf, loop exit status=%d\n",
                                                                 loop_exit_stat);
                  RPGC_rel_outbuf((void*)out_buffer, DESTROY);
                  RPGC_abort();
                  break;
               }
               hdr = (Base_data_header *) basedataPtr;
               radial_status = basedataPtr->hdr.status & 0xf;
               RPGC_rel_inbuf((void*)basedataPtr);
               rad_idx++; /* Sample Alg 1.21 */

            } else { /* we found actual end of elevation / volume */
               if (TEST) {
                  fprintf(stderr,
                          "=> finished finding actual end, radial = %d\n",
rad_idx+1);
                  fprintf(stderr,
                          "   Radial Status = %d  where\n"
                          "    0=beg_ele, 1=int, 2=end_ele, 3=beg_vol,
4=end_vol,\n"
                          "    8=pseudo_end_ele, 9=pseudo_end_vol\n",
                                                    radial_status);
               } /* end TEST */

               /* exit with loop_exit_stat==NORMAL and no ABORT */
               break;
            }

         } /* end while */

         /* elevation as a whole.  Processing that requires comparisons   */
         /* between radials should be accomplished here                   */
         /* Note: in this algorithm we have not yet ensured that the      */
         /* the data values beyond the last good bin for each radial have */
         /* have been set to '0'                                          */
      } /* end if loop_exit_stat == NORMAL */

      /* the end of the ELEVATION PROCESSING SEGMENT of the algorithm     */


      /* LABEL:ASSEMBLE The assembly of the intermediate product.         */
      /* product conforms to the structure defined in s3_t1_prod_struct.h */
      if(loop_exit_stat == NORMAL) {
         int length=0;   /* accumulated product length                    */
         /* Sample Alg 1.20 - replace CVG_radial with s3_t1_intermed_prod_hdr */
         s3_t1_intermed_prod_hdr *out_hdr = (s3_t1_intermed_prod_hdr *)out_buffer;

         Base_data_header *bdh=NULL;  /* base data header struct          */

         unsigned char *char_buf; /* used to access individual raw data bins  */
```

```
short *short_data; /* pointer to short to read base data levels  */
                   /* from a base data radial                    */
int offset;     /* offset into various memory buffers            */
/* Sample Alg 1.20 */
int rad_beg;    /* holds offset to beginning of current radial   */
int r, b;       /* loop variables for radial and bin             */
FILE *outfile;  /* file pointer used for diagnostic output       */
char fname[30]; /* variable used to hold output file name        */
int test_out=FALSE; /* Boolean used to control diagnostic output */

if (VERBOSE) {
  fprintf(stderr, "=> loop exit status is normal\n");
}

/* load the beginning portion of the s1_t1_prod_radial structure */

output_id = RPGC_get_id_from_name( "SAMPLE3_IP");
/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
out_hdr->linearbuffer_id = output_id;   /* output LB ID          */

/* limited to 460 bins because radial limit precludes */
/* using a super resolution type                      */
/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
out_hdr->num_range_bins = bins_to_process;

if(TEST) fprintf(stderr,
        "setting last elev flag: out_hdr->flag=%d\n",out_hdr->flag);

/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
/* Sample Alg 1.21 - changed rad_idx to last_rad */
out_hdr->num_radials = last_rad;          /* #radials read this elev*/
out_hdr->num_bytes_per_bin = 1;         /* 1 byte/bin 256 levels */

/* first fill in the angle structs by extracting the start angle */
/* & delta angle information for the base data radials that we've*/
/* read so far - each set of angle info is stored contiguously   */
/* Sample Alg 1.20 - changed CVG_RADIAL_HEADER_OFFSET (6) */
/*                   to S3_T1_DATA_OFFSET (5)             */
offset = S3_T1_DATA_OFFSET;   /* offset=5 4-byte integers */
/* load start angle data                                      */
/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
for(r = 0; r < out_hdr->num_radials; ++r) {
   bdh = (Base_data_header *)(radial[r]);
   out_buffer[offset++] = bdh->start_angle;
}

/* load angle delta data          */
/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
for(r = 0; r < out_hdr->num_radials; ++r) {
   bdh = (Base_data_header *)(radial[r]);
   out_buffer[offset++] = bdh->delta_angle;
}

/* now, fill in the actual data - one byte at a time (offset=8)  */
/* here, char_buf points to the beginning of the data array      */
char_buf = (unsigned char *)(&(out_buffer[offset]));

offset = 0;  /* set offset to zero */

/* Sample Alg 1.20 - replaced raw_rad with out_hdr */
```

```
         for(r = 0; r < out_hdr->num_radials; ++r) {
            bdh = (Base_data_header *)(radial[r]);
            /* reading basedata values as shorts solves Endian issues     */
            short_data = (short *) &*(radial[r]+sizeof(Base_data_header));
/* DEBUG T1-1*/
            rad_beg = offset; /* used in debug printout */

            for(b = 0; b < bins_to_process; ++b) {
               /* ensure any data bins beyond the last good bin are set to 0 */
               if( b < bins_to_copy )
                  char_buf[offset++] = (unsigned char) short_data[b];
               else
                  char_buf[offset++] = (unsigned char) 0;

            } /* end for j<bins_to_process */
/* DEBUG T1-2 */
         } /* end for r< num_radials */

         if (VERBOSE) {
           fprintf(stderr, "=> finished loading all bins into char array\n");
         }

         /* recalculate offset value                                          */
         /* Sample Alg 1.20 - changed raw_rad to out_hdr and  */
         /*     CVG_RADIAL_HEADER_OFFSET to S3_T1_DATA_OFFSET */
         offset = S3_T1_DATA_OFFSET*sizeof(int) +
                   2 * out_hdr->num_radials*sizeof(int) +
                    bins_to_process * out_hdr->num_radials * sizeof(char);

         if (TEST) {
            /* diagnostics to test integrity of a radial of data           */
            bdh = (Base_data_header*)(radial[0]);
            fprintf(stderr,
               "bdh test: vcp=%hd target elev=%hd start angle=%hd delta=%hd\n",
              bdh->vcp_num, bdh->target_elev, bdh->start_angle, bdh->delta_angle);
            fprintf(stderr,
               "bdh test: msg len=%hd msg type=%hd spot blank range=%hd\n",
                         bdh->msg_len, bdh->msg_type, bdh->spot_blank_flag);
         }

         /* copy a basedata HEADER into the output product for use in */
         /* sample3_t2                                                */
         /* here, char_buf points to the beginning of the output buffer     */
         /* NOTE: alignment assured by having number of bins always even   */
         char_buf = (unsigned char*)out_buffer;
         memcpy(char_buf+offset, bdh, sizeof(Base_data_header));

         /* calculate the final overall length of the output product   */
         /* Sample Alg 1.20 - changed raw_rad to out_hdr and 6 to 5 */
         length = 5 * sizeof(int) + 2 * out_hdr->num_radials * sizeof(int) +
            bins_to_process * out_hdr->num_radials * sizeof(char) + 1 * sizeof(int)
            + sizeof(Base_data_header);

         if (VERBOSE) {
           fprintf(stderr, "->final length of product=%d\n",length);
           fprintf(stderr, "  number of radials is %d, bins processed is %d\n",
                                       out_hdr->num_radials, bins_to_process);
         }

         /* forward constructed data structure to intermediate LB.        */
```

```
         /* if diagnostic output is desired...store data now              */

         if(test_out == TRUE) {
            sprintf(fname, "sample3_t1out.%d", elev_idx);
            if((outfile = fopen(fname, "w")) != NULL) {
               /* copy the data out of the buffer in memory to a file     */
               fwrite(out_buffer, length, 1, outfile);
               fclose(outfile);
            }
         }

         /* LABEL:OUTPUT_PROD    forward product and close buffer         */
         RPGC_rel_outbuf((void*)out_buffer, FORWARD);

      }    /* end of if(opstatus==NORMAL) block                          */

      /* LABEL:CLEANUP   section to reset variables / free memory         */
      /* free each previously allocated radial array                      */
      if (VERBOSE) {
        /* Sample Alg 1.21 - changed rad_idx to last_rad */
        fprintf(stderr, "free radial array (last_rad=%i)\n", last_rad);
      }

      /* Sample Alg 1.21 - changed rad_idx to last_rad */
      for(i = 0; i < last_rad; i++)
         free(radial[i]);

      rad_idx = 0; /* reset radial counter to 0                                */
      last_rad = 0;

      if(VERBOSE)
         fprintf(stderr, "Elev Complete:  Elev Index=%d  Target Elev=%d\n",
                                                  elev_idx,target_elev);

   }    /* end while PROCESS == TRUE                                        */

   fprintf(stderr,"\nSample 3 Task 1 Program Terminated\n");

   return(0);

} /* end of main ------------------------------------------------- */



/************************************************************************
Description:    clear_buffer: initializes a portion of allocated
                memory to zero
Input:          pointer to the input buffer (already cast to char*)
Output:         none
Returns:        none
Globals:        the constant BUFFSIZE is defined in the include file
Notes:          none
************************************************************************/
void clear_buffer(char *buffer) {
   /* zero out the input buffer */
   int i;

   for(i = 0; i < BUFFSIZE; i++)
      buffer[i] = 0;
```

```
   return;

}
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/**************************************************************************
Module:         sample3_t1.h

Description:    include file for the sample3_t1 demonstration program.
--------------------------------------------------------------------------
          Modified to reflect the new size of the base data header

--------------------------------------------------------------------------
Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.3,  April 2005   T. Ganger
              Modified to use the new product IDS

Version 1.4   March 2006   T. Ganger
              Modified to reflect the new base data message, larger
                   header, new order or moment data, possible
                   larger reflectivity array - Build 8

Version 1.5   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.

Version 1.6   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Eliminated definition of output buffer ids.
              New output buffer sizes calculated.

Version 2.0   November 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Included new structure for intermediate product output.

$Id$
**************************************************************************/
#ifndef _SAMPLE3_T1_H_
#define _SAMPLE3_T1_H_

/* system includes ------------------------------------------------- */
#include <stdio.h>

/* ORPG   includes ------------------------------------------------- */
#include <rpg_globals.h>

/*** the following are included in rpg_globals.h */
/***                                             */
/***  #include <stdio.h>                         */
```

```
/***  #include <stdlib.h>                          */
/***  #include <time.h>                            */
/***  #include <ctype.h>                           */
/***  #include <string.h>                          */
/***                                               */
/***  #include <a309.h>                            */
/***  #include <basedata.h>                        */
/***  #include <basedata_elev.h>                   */
/***  #include <rpg_port.h>                        */
/***  #include <prod_gen_msg.h>                    */
/***  #include <prod_request.h>                    */
/***  #include <prod_gen_msg.h>                    */
/***  #include <rpg_vcp.h>                         */
/***  #include <orpg.h>                            */
/***  #include <mrpg.h>                            */
/************************************************/

/* required by rpgc after API enhancement patch */
#include <gen_stat_msg.h>

#include <rpgc.h>
#include <rpgcs.h>

/* ------------------------------------------------------------------- */
/* the structure of a base data radial message, which is ingested by the
   algorithm, is defined in  basedata.h  */

/*
 * The order and size of the base data arrays are no longer fixed.
 *
 * The offsets in the base data header or the API access function must
 * be used to read the basic moments: reflectivity, velocity, and
 * spectrum width.
 *
 * Fields in the base data header are used to determine array size.
 */

/* The contents of Base_data_header is also defined in basedata.h  */
/* ------------------------------------------------------------------- */



/* Local includes ----------------------------------------------------- */
/* #include "s3_cvg_struct.h" */
#include "s3_t1_prod_struct.h"

/* Algorithm specific constants/definitions --------------------------- */


/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define SAMPLE3_IP 1999  */


/* the estimated buffer size consists of the following data (in bytes):
 * (Pre ICD header     96 not included)
 * Header fields       24
 * Start Angles      1468 (4 x 367 radials)
 * Delta Angles      1468 (4 x 367 radials)
```

```
 * Base_data_header   200 (current size)
 * Radial Data Ptr      4
 * SUB TOTAL         3164
 * Radial Data      84410 (230 bins x 367 radials)
 * TOTAL            87574 (230 bins x 367 radials)
 * OR
 * Packet Data     168820 (460 bins x 367 radials)
 * TOTAL           171984
 */

/* to allow for 400 radials, the following can be used if NOT using      */
/* one of the super resolution data types.                                */
#define BUFFSIZE 187164

/* the algorithm would need to calculate a different buffer size to       */
/* allocate if using one of the super resolution data types. For example  */
/*    1475164 for 800 radials and 1840 bins (full super resolution)       */
/*     739164 for 400 radials and 1840 bins                               */
/*     371164 for 800 radials and  460 bins                               */
/*     187164 for 400 radials and  460 bins (original resolution )        */

/* function prototypes ------------------------------------------------ */
void clear_buffer(char*);

#endif
```

SAMPLE 3 TASK 2
(UPDATED NOVEMBER 2008)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/*****************************************************************
Module:         sample3_t2.c

Description:    sample3_t2.c is part of a chain-algorithm demonstration.
                The first task reads radial data and produces elevation
                data.  The second task reads elevation data and produces
                volume data.

                THE PURPOSE OF THIS SAMPLE ALGORITHM IS TO DEMONSTRATE
                A MULTIPLE TASK ALGORITHM.  NO USEFUL SCIENCE IS
                ACCOMPLISHED BY THE SECOND TASK

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3, with one exception.
                Though the algorithm has been tested for memory leaks,
                logic to handle a failure of "malloc" in this file is
                incomplete.  Even when the failure is tested for, the
                cleanup process is not adjusted correctly.


                sample3_t2.c, the second part of the chain algorithm,
                accepts the intermediate elevation data and allocates
                memory to contain each elevation. When the volume is
                complete or the elevation specified via an adaptable
                parameter is received, the task generates a 460 KM
                digital reflectivity CD-compliant final product.  The
                adaptable parameter can be changed from the HCI.

                Even though multiple elevations of data are stored, no
                volume processing is actually performed.  The algorithm
                simply selects an elevation (based upon the adaptation
                data) to convert to a final product.

                Previously this task was artificially limited to process
                no more than the first 5 elevations of a volume coverage
                pattern (VCP). This artificial limitation has been
                eliminated.  The task terminates processing at the last
                elevation in the VCP or the elevation stated in the
                adaptation data, which ever occurs first.


                Key source files for this algorithm include:

                sample3_t2.c           program source
```

```
                sample3_t2.h           main include file

                s3_t1_prod_struct.h    definition of s3_t1_intermed_prod_hdr

                s3_symb_layer.c        creates the ICD compliant product
                s3_symb_layer.h        local include file

                s3_print_diagnostics.c used for diagnostic output
                s3_print_diagnostics.h local include file

Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.4,  April 2005   T. Ganger
              Modified to use the new adaptation data (DEA) using
                   sample3_t2.alg as source of data

Version 1.5   March 2006   T. Ganger
              Replaced stderr messages outside of test/debug sections with
                   RPGC_log_msg() calls.
              Revised to use current guidance for abort functions and
                   abort reason codes.

Version 1.6   June 2006   T. Ganger
              Revised to use RPGC_reg_io
              Revised to use the new 'by_name' get_inbuf/get_outbuf functions

Version 1.7   March 2007   T. Ganger
              BUGFIX. Task crashed with abnormal inner loop exit (get_inbuf
                  or other failure) because excessive number of elevation
                  data was freed.

Version 1.8   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated use of defined output buffer ids by using
                  RPGC_get_id_from_name
              Noted setting number of bins per radial at 230 is artificial
                  and dependent upon using recombined data.
              Corrected the contents of dependent parameter 4 from maximum
                  reflectivity data level to maximum value in dBZ
              Provided the number of bins in each array to the subroutine
                  responsible to build the symbology block. Will permit
                  dynamically changing the array size in the future.
              Used the new default abort reason code PGM_PROD_NOT_GENERATED.

Version 2.0   November 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Updated description of algorithm task and main function.
                  Added comments to clarify processing.
              Used RPGC_is_buffer_from_last_elev to determine when the
                  last elevation produced by the first task is received
                  rather than the Legacy method of using the flag in the
                  elevation input product.
              Eliminated artificial limit on number of elevations to read.
              Used a new structure to read the input intermediate product.

$Id$
**********************************************************************/
```

```
/* see the following include file for descriptions of constants and
for expected ICD block sizes */
#include "sample3_t2.h"


int TEST=FALSE;           /* Boolean to control diagnostic output       */
                          /* set to TRUE to increase number of messages */
/* int VERBOSE=FALSE; */
 int VERBOSE=TRUE;

/* file scope variables for adaptation data value */
  int input_elev;


/* prototype for adaptation data access function for DEA                  */
/*     this function cannot be registered as a callback because a         */
/*     parameter providing the address of the c structure is not provided */
int read_sample3_t2_adapt();


/*************************************************************************
Description:    main function to drive the chain-algorithm demonstration
                (second task out of two).
Input:          receive an elevation intermediate product via a structure
                in s3_t1_prod_struct.h with radial data in an array of
                bytes (unsigned char) produced by sample3_t1.
Output:         a digital reflectivity final product associated with the
                elevation index adaptable parameter
Returns:        none
Globals:        none
Notes:          1). This sample algorithm task reads and stores multiple
                elevations of an intermediate product.  No volume
                processing is performed other than to select an elevation
                to convert to a final product.
                2). This algorithm can set different radial sizes not to
                exceed 460 bins.  Since the maximum number of radials is
                hard coded to 400 (and a static array used), this
                algorithm cannot use any of the super resolution data
                types.  Hence 460 is the maximum number of bins.
                3). The program takes no command line arguments.


*************************************************************************/
int main(int argc,char* argv[]) {
   /* algorithm variable declarations and definitions ------------------  */
   int PROCESS=TRUE;         /* Boolean used to control the processing    */
                             /* portion of the program. Process until set */
                             /* to FALSE.                                 */
   int i;                    /* loop variables                            */
   int loop_exit_stat=0;    /* variable: holds exit status from loop      */
   int opstatus;            /* variable:  used to hold API return values  */

   char *buffer;            /* pointer to contain a complete ICD product  */
   char *inbuf;             /* pointer to intermediate input buffer       */
   /* Sample Alg 1.20 - changed CVG_radial* to unsigned char* and         */
   /*                   rad_data[] to input_prod_data[]                   */
   unsigned char *input_prod_data[MAX_ELEVS+1]; /* container to hold      */
                            /* intermediate products for processing      */
```

```
   int result=0;           /* variable: holds function results          */
   /* Sample Alg 1.20 */
   int last_elev=FALSE;    /* Boolean used to hold the status of function*/
                           /* RPGC_is_buffer_from_last_elev             */
   int elev_ind;           /* output of RPGC_is_buffer_from_last_elev   */
   int last_index;         /* out out of RPGC_is_buffer_from_last_elev   */

   int elev_count=1;       /* elevation counter through the loop        */

   /* ---------------------------------------------------------------- */

   fprintf(stderr,"\nBegin Sample 3 Task 2 Algorithm\n");

   /* LABEL:REG_INIT  Algorithm Registration and Initialization Section  */

   /* register inputs and outputs based upon the           */
   /* contents of the task_attr_table (TAT).               */
   /* input:SAMPLE3_IP(1999), output:SAMPLE3_FP(1992)      */
   RPGC_reg_io(argc, argv);


   /* register algorithm infrastructure to read the Scan Summary Array   */
   RPGC_reg_scan_summary();


  /* the adaptation data read function cannot be registered as a
   * callback function because it does not have the needed parameters
   */


   /* ORPG task initialization routine. Input parameters argc/argv are   */
   /* not used in this algorithm                                         */
   RPGC_task_init(VOLUME_BASED, argc, argv);

   fprintf(stderr, "-> sample3_t2 algorithm initialized\n");

   RPGC_log_msg( GL_INFO,
      "-> sample3_t2 algorithm initialized\n");

   /* while loop that controls how long the task will execute. As long as */
   /* PROCESS remains TRUE, the task will continue.                      */
   while(PROCESS) {
      /* Boolean used to control diagnostic output */
      int test_out=FALSE;

      /* system call to indicate a data driven algorithm. block algorithm */
      /* until good data is received the product is requested             */
      RPGC_wait_act(WAIT_ALL);

      /* LABEL:BEGIN_PROCESSING Released from Algorithm Flow Control Loop */
      if (VERBOSE)
           fprintf(stderr,
                "-> sample3_t2 passed wait act loop control. count=%d\n",
                                                          elev_count);
      RPGC_log_msg( GL_INFO,
           "-> sample3_t2 passed wait act loop control. count=%d\n",
                                                          elev_count);

      /* open output buffer                                             */
      buffer = (char*)RPGC_get_outbuf_by_name("SAMPLE3_FP", BUFSIZE, &opstatus);
```

```
      if(opstatus != NORMAL) {
         RPGC_log_msg( GL_INFO,
            "ERROR: Sample 3 Task 2 Aborting, RPGC_get_outbuf..opstatus=%d\n",
                                                                  opstatus);
         RPGC_abort();
         continue;
      }

      if (VERBOSE) {
        fprintf(stderr,"-> sample3_t2 successfully obtained output buffer\n");
      }

      clear_buffer(buffer);

    /* reading new DEA adaptation data elements manually because   */
    /* the access function does not include the required parameter */
    /* in order to be registered as a callback function            */
    read_sample3_t2_adapt();

    if (VERBOSE) {
      fprintf(stderr, "-> sample3_t2 - adaptation elev_index selection = %d\n",
                                                                  input_elev);
    }

      /* Set Exit Status to NORMAL */
      loop_exit_stat = 0;

      /* PROCESS ELEVATION LOOP                                  */
      while(TRUE) {

         /* Sample Alg 1.20 - replaced CVG_radial with s3_t1_intermed_prod_hdr */
         s3_t1_intermed_prod_hdr *ptr=NULL;  /* pointer to input product header */
         int length=0;          /* holds calculated length of product     */

         /* LABEL:PROCESS_ELEV Here we process each elevation individually.*/
         /* obtain an input buffer from intermediate buffer */
         if (VERBOSE) {
           fprintf(stderr, "Sample 3 Task 2 - Waiting for Inbuf - Elev=%d\n",
                                                                 elev_count);
         }

         inbuf = (char *)RPGC_get_inbuf_by_name("SAMPLE3_IP", &opstatus);

         if (VERBOSE) {
           fprintf(stderr, "-> sample3_t2 get inbuf opstatus = %d \n", opstatus);
         }

         if(opstatus != NORMAL){
            loop_exit_stat=opstatus;
            RPGC_log_msg( GL_INFO,"ERROR: Aborting from RPGC_get_inbuf\n");
            RPGC_rel_outbuf((void*)buffer, DESTROY);
            RPGC_abort();
            break;
         }

         if (VERBOSE) {
           fprintf(stderr,
                     "-> sample3_t2 successfully read intermediate product\n");
         }
```

```
 /* NOTE: This algorithm originally always used 230 as the number of  */
 /*           bins per radial.  This task can now read intermediate   */
 /*           products having different sizes.                        */
 /*        This provides flexibility in independently modifying the   */
 /*           first task of this sample algorithm                     */

/* Sample Alg 1.20 - replaced CVG_radial with s3_t1_intermed_prod_hdr */
ptr = (s3_t1_intermed_prod_hdr*)inbuf;

/* allocate memory to hold current intermediate product          */
/* Sample Alg 1.20 - changed 6 to 5 */
length = (5 * sizeof(int)) + (2 * ptr->num_radials * sizeof(int)) +
         (ptr->num_range_bins * ptr->num_radials * sizeof(char))
                                          + sizeof(Base_data_header);
if (VERBOSE) {
  fprintf(stderr, "-> sample3_t2 allocation size = %d\n", length);
  }

/* allocate memory for incoming intermediate product             */
/* Sample Alg 1.20 - replaced CVG_radial* with unsigned char* */
input_prod_data[elev_count] = (unsigned char*)malloc((size_t)length);

if(input_prod_data[elev_count] == NULL) {
   RPGC_log_msg( GL_ERROR,
         "MALLOC Error: malloc failed on input_prod_data[%d]\n",
                elev_count);
   RPGC_rel_inbuf((void*)inbuf);
   RPGC_rel_outbuf((void*)buffer, DESTROY);
   RPGC_abort_because(PGM_MEM_LOADSHED);
   loop_exit_stat = TERMINATE;   /* SHOULD THIS BE NO_DATA?  */
   break;
}

if (VERBOSE) {
  fprintf(stderr,
          "-> sample3_t2 memory allocated for input_prod_data[%d]\n",
               elev_count);
}

/* copy input buffer to allocated memory */
memcpy(input_prod_data[elev_count], inbuf, length);
if (VERBOSE) {
  fprintf(stderr, "-> sample3_t2 data copied to temporary storage\n");
}


/* get pointers to the data set                                  */
/* Sample Alg 1.20 - replaced CVG_radial with s3_t1_intermed_prod_hdr  */
ptr = (s3_t1_intermed_prod_hdr*)input_prod_data[elev_count];

/* show integrity of the data set (optional)                     */
if(TEST) {
   fprintf(stderr, "-> sample3_t2: test output num_range_bins=   %d\n",
                                                 ptr->num_range_bins);
   fprintf(stderr, "-> sample3_t2: test output num_radials=      %d\n",
                                                    ptr->num_radials);
   fprintf(stderr, "-> sample3_t2: test output num_bytes_per_bin=%d\n",
                                               ptr->num_bytes_per_bin);
}
```

```
      /* diagnostic output of the data to disk (optional)  */
      if(test_out == TRUE) {
         char *c_ptr;
         c_ptr = (char *)input_prod_data[elev_count];
         product_to_disk(c_ptr, length, "sample3_fp", elev_count);
      }

      /* LABEL: LAST_ELEV Two tests to determine if we are done           */

      /* For Test 1. Is this the last elevation produced by the first task?*/
      /* Originally we used a flag sent in the intermediate product. Now   */
      /* Now use the recommended API function to determine the last    */
      /* elevation received by the RPG. If the last elevation, then break  */
      /* out of the inner loop with a NORMAL exit loop status             */

      last_elev = RPGC_is_buffer_from_last_elev( (void*)inbuf,
                                                 &elev_ind, &last_index );

      /* now that we've read everything in that we want to  */
      RPGC_rel_inbuf((void*)inbuf);
      if (VERBOSE)
        fprintf(stderr, "-> sample3_t2 released input buffer. count=%d\n",
                                                       elev_count);

      /* Test 1. no more data to receive */
      if( (elev_ind == last_index) || (last_elev == -1) ) {
         /* either the last elev or function failed */
         if(TEST) fprintf(stderr, "-> sample3_t2 RECEIVED LAST ELEVn");
         break;
      }

      /* Test 2. Is this the elevation selected by adaptation data to */
      /* be used to construct the product? This produces the product  */
      /* as soon as possible.                                         */
      if(elev_count == input_elev) {
         if(TEST) fprintf(stderr, "-> sample3_t2 SELECTED ELEVATION\n");
         break;
      }

      /* increment elev count */
      elev_count++;

   } /* -------- end of process elevation loop  ----------- */

   if (VERBOSE) {
     fprintf(stderr,
            "-> sample3_t2 out of Elevation Process loop. Elev cnt=%d\n",
            elev_count);
   }

   /* LABEL: OUTPUT_PROD  */

   if(loop_exit_stat == NORMAL) {  /* successfully received input data */

      /* create output product-dependent on the adaptation elevation index*/
      if(input_elev > 0 && input_elev <= last_index) {
         /* convert product to the s3_t2_internal_rad struct & create output*/
         result = translate_product(input_prod_data[input_elev], buffer);
```

```
            if (VERBOSE) {
              fprintf(stderr,
                   "-> sample3_t2 returned from translate product. result=%d\n",
                                                                        result);
            }

         } else  { /* adaptation data out of range, use first elevation */
            if (VERBOSE) {
              fprintf(stderr,
                     "-> Adaptation Data out of range, first elevation used.\n");
            }
            /* convert product to the s3_t2_internal_rad struct & create output */
            result = translate_product(input_prod_data[1], buffer);

            if (VERBOSE) {
              fprintf(stderr,
                     "-> sample3_t2 returned from translate product. result=%d\n",
                                                                          result);
            }

         } /* end else out of range */

      }  /* end of if(loop_exit_stat==NORMAL) block      */


      /* If product successfully output and not last elevation */
      /* abort the remaining volume scan; the alternative is   */
      /* continue reading elevations until end of volume       */
      if( (loop_exit_stat==0) && (result==0)  /* successful product */
          && (elev_count < last_index)) {     /* not last elevation */

         RPGC_abort_remaining_volscan();

         if (VERBOSE) fprintf(stderr,
                   "aborting remaining volume, result=%d, last_elev=%d\n",
                                                  result, last_elev);
      }

      /* LABEL_CLEANUP  */

      /*  with abnormal loop exit, correct elevation count for free(). */
      if(loop_exit_stat != 0)
         elev_count--;

      /* clean up memory after product assembly                          */
      if (VERBOSE) {
        fprintf(stderr, "-> sample3_t2 freeing memory, elev_count is %d\n",
                                                              elev_count);
      }

      for(i = 1; i <= elev_count; i++) {
         free(input_prod_data[i]);
      }

      /* reset elevation counter                                         */
      elev_count = 1;

      if (VERBOSE) fprintf(stderr, "-> sample3_t2 reset for new volume\n");

   } /* end of Main PROCESS loop */
```

```
   fprintf(stderr, "\nsample3_t2 Program Terminated\n");

   return(0);

} /* end main */




/*************************************************************************
Description:    clear_buffer: initializes a portion of allocated
                memory to zero
Input:          pointer to the input buffer (already cast to char*)
Output:         none
Returns:        none
Globals:        the constant BUFSIZE is defined in the include file
Notes:          none
*************************************************************************/
void clear_buffer(char *buffer) {
  /* zero out the input buffer */
  int i;

  for(i = 0; i < BUFSIZE; i++)
    buffer[i]=0;

  return;
  }



/*************************************************************************
Description:    translate_product takes the raw intermediate product and
                places the data into a s3_t2_internal_rad structure
Input:          unsigned char *in_data_ptr - a pointer to the raw
                                             intermediate product
                char *buffer - a pointer to the output buffer
Output:         none
Returns:        returns -1 if an error, otherwise returns 0
Globals:        none
Notes:          none
*************************************************************************/
/* Sample Alg 1.20 replaced CVG_radial* with unsigned char *        */
int translate_product(unsigned char *in_data_ptr, char *buffer) {

   /* Sample Alg 1.20 - replaced CVG_radial with s3_t2_internal_rad */
   s3_t2_internal_rad *rad_data=NULL; /* pointer to an internal structure */
   int size_in_bytes;         /* temporary size storage variable          */
   int result2;               /* variable: holds function results         */
   int r, b;                  /* loop variables for radial and bin        */
   int rad_count;             /* hold count of number of radials          */


   if (VERBOSE) {
     fprintf(stderr, "->sample3_t2 inside translate_product\n");
   }
```

```
   /* allocate memory for data transfer and assign data access pointers   */
   /* Sample Alg 1.20 - replaced CVG_radial with s3_t2_internal_rad */
   rad_data = (s3_t2_internal_rad*)malloc(sizeof(s3_t2_internal_rad));

   if(rad_data == NULL) {
      RPGC_log_msg( GL_ERROR,
          "MALLOC Error: malloc failed on rad_data\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
   }

   /* save the first 5 values (all 4 byte integers) in the input product  */
   /* Sample Alg 1.20 - changed 6 to 5 */
   size_in_bytes = sizeof(int) * 5;
   memcpy(rad_data, in_data_ptr, (size_t) size_in_bytes);
   in_data_ptr += size_in_bytes;

   rad_count = rad_data->num_radials;  /* bins_to_process from task 1 */

   if(TEST) {
      fprintf(stderr, "-> sample3_t2 read linear buffer id=%d\n",
            rad_data->linearbuffer_id);
      fprintf(stderr, "-> sample3_t2 read last elevation flag=%d\n",
            rad_data->flag);
   }

   /* now copy over the start angle data (all 4 byte integers)            */
   size_in_bytes = sizeof(int) * rad_data->num_radials;
   rad_data->start_angle = (int*)malloc( (size_t) (size_in_bytes) );

   if(rad_data->start_angle == NULL) {
      RPGC_log_msg( GL_ERROR,
         "MALLOC Error: malloc failed on rad_data->start_angle\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
   }

   memcpy(rad_data->start_angle, in_data_ptr, (size_t) size_in_bytes);
   in_data_ptr += size_in_bytes;

   if(TEST) {
     fprintf(stderr, "-> sample3_t2 start angle first radial=%.1f\n",
                                    (rad_data->start_angle[0]/10.0));
     fprintf(stderr, "-> sample3_t2 start angle last radial=%.1f\n",
               (rad_data->start_angle[rad_data->num_radials-1]/10.0));
   }

   /* now copy over the angle delta data  (all 4 byte integers)           */
   rad_data->angle_delta = (int*)malloc((size_t)size_in_bytes);

   if(rad_data->angle_delta == NULL) {
      RPGC_log_msg( GL_ERROR,
         "MALLOC Error: malloc failed on rad_data->angle_delta\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
      }
   memcpy(rad_data->angle_delta, in_data_ptr, size_in_bytes);
```

```
   in_data_ptr += size_in_bytes;

   /* now, copy over the bin data -- 1 byte per bin in the input product  */
   /*                     but 4 bytes per bin in the internal array       */
   rad_data->radial_data = malloc(sizeof(int *) * rad_data->num_radials);

   if(rad_data->radial_data == NULL) {
      RPGC_log_msg( GL_ERROR,
         "MALLOC Error: malloc failed on rad_data->radial_data\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
      }

   /* allocate bins for each radial, 4 bytes per bin in internal array    */
   for(r = 0; r < rad_data->num_radials; r++) {
      rad_data->radial_data[r] = (int*)malloc(sizeof(int) *
                                          rad_data->num_range_bins);
      if(rad_data->radial_data[r] == NULL) {
         RPGC_log_msg( GL_ERROR,
             "MALLOC Error: malloc failed on rad_data->radial_data #%d\n",r);
         RPGC_rel_outbuf((void*)buffer, DESTROY);
         RPGC_abort_because(PGM_MEM_LOADSHED);
         return(-1);
      }
   }
   /* copy the bin data now                                               */
   /* NOTE: the number of range bins is based upon the calculated number  */
   /*       of bins to process.  If this is greater than the number of    */
   /*       good bins, the data have been padded with '0'.                */
   for(r = 0; r < rad_data->num_radials; r++) {
/* DEBUG T2-1 */
      /* the input array (in_data_ptr) is an array of unsigned char    */
      /* the internal array (rad_data->radial_data) is an array of int    */
      for(b = 0; b < rad_data->num_range_bins; b++)
         rad_data->radial_data[r][b] = (int) *(in_data_ptr++);
/* DEBUG T2-2 */
   } /* end for r < number of radials */

   /* now copy over the base data header                                  */
   rad_data->bdh = (Base_data_header*)malloc(sizeof(Base_data_header));

   if(rad_data->bdh == NULL) {
      RPGC_log_msg( GL_ERROR, "MALLOC Error: malloc failed on rad_data->bdh\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
   }

   memcpy(rad_data->bdh, in_data_ptr, sizeof(Base_data_header));

   if (VERBOSE) {
     fprintf(stderr, "sample3_t2 copy complete: num radials = %d\n",
                                           rad_data->num_radials);
     fprintf(stderr, "sample3_t2 elevation number=%d   target_elev=%.1f\n",
                 rad_data->bdh->elev_num, rad_data->bdh->target_elev/10.0);
   }

   /* LABEL: ASSEMBLE - assemble the final ICD product now                */
   result2 = assemble_product(buffer, rad_data->num_range_bins, rad_data);
```

```
   /* clean up memory after product assembly                                */
   if (VERBOSE) {
     fprintf(stderr,
            "-> sample3_t2 freeing local memory rad_count=%d\n",rad_count);
   }

   free(rad_data->start_angle);
   free(rad_data->angle_delta);
   for(r = 0; r < rad_count; r++)
      free(rad_data->radial_data[r]);
   free(rad_data->radial_data);
   free(rad_data->bdh);
   free(rad_data);

   if (VERBOSE) {
     fprintf(stderr,"-> sample3_t2 memory clear complete\n");
   }

   /* Were we successful in assembling the product?     */
   if(result2 == 0)
      return(0);  /* success */
   else
      return(-1); /* failure */

}  /* end translate_product() */



/*************************************************************************
Description:    assemble_product
Input:          s3_t2_internal_rad *rad_data - pointer to internal structure
                int number_bins

                char *buffer - buffer where ICD product will be constructed
Output:         none
Returns:        returns -1 if an error, otherwise returns 0
Globals:        none
Notes:          none
*************************************************************************/
/* Sample Alg 1.20 - replaced CVG_radial with s3_t2_internal_rad */
int assemble_product(char *buffer, int number_bins, s3_t2_internal_rad *rad_data)
{
   int result3;     /* function return value                              */
   int length=0;   /* accumulated product length                         */
   int vol_num=0;  /* current volume number                              */
   int max_refl=0; /* maximum reflectivity value                         */

   int output_id;


   /* get the current volume number from the base data header            */
   /* that has been passed as part of the intermediate product           */
   vol_num = rad_data->bdh->volume_scan_num;

   /* building the ICD formatted output product requires a few steps     */
   /* step 1: build the product description block (pdb)                  */
   /* step 2: build the symbology block & data layer                     */
   /* step 3: complete missing values in the pdb                         */
   /* step 4: build the message header block (mhb)                       */
```

```
   /* step 5: forward the completed product to the system                */

   output_id = RPGC_get_id_from_name( "SAMPLE3_FP");

   /* step 1: build the product description block -uses a system call*/
   if (VERBOSE) {
     fprintf(stderr,
          "\nsample3_t2: Creating the product description block now: vol=%d\n",
                                                                    vol_num);
   }

   RPGC_prod_desc_block((void*)buffer, output_id, vol_num);

   /* for testing show the header information now                        */
   if (TEST) {
      fprintf(stderr, "\n==> OUTPUT INITIAL MSG HEADER AND PROD DESC BLOCK\n");
      print_message_header(buffer);
      print_pdb_header(buffer);
   }

   /* step 2: build the symbology layer & digital radial data array.     */
   /* this routine returns both the overall length (thus far) of the     */
   /* product and the maximum reflectivity of all radials                */
   if (VERBOSE) {
     fprintf(stderr, "sample3_t2: begin building the symbology and p16 layers\n");
   }

   max_refl = build_symbology_layer(buffer, rad_data , rad_data->num_radials,
                                    number_bins,  &length);
/* build_symbology_layer returns -1 on failure to malloc */
   if(max_refl == -1) {
      RPGC_log_msg( GL_ERROR,
         "MALLOC Error: malloc failed on rad_data->radial_data\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_MEM_LOADSHED);
      return(-1);
   }


   /* step 3: finish building the product description block by           */
   /* filling in certain values such as elevation index, maximum         */
   /* reflectivity, accumulated product length, etc                      */

   /* get elevation index and actual elevation using base data header    */
   /* that has been passed as part of the intermediate product           */
   if (VERBOSE) {
     fprintf(stderr, "sample3_t2: elev_ind=%hd  elevation=%hd\n",
           rad_data->bdh->rpg_elev_ind,rad_data->bdh->target_elev);
   }

   finish_pdb(buffer, rad_data->bdh->rpg_elev_ind,
             rad_data->bdh->target_elev, max_refl, length);
   /* alternatively, we could use the new API functions to accomplish this*/
   /* which would eliminate the need for passing the base data header    */
   /* for example, we would have obtained the elevation index after      */
   /* reading the intermediate product:                                  */
   /*          elev_idx=(short)RPGC_get_buffer_elev_index((void *)inbuf); */
   /* and use the following to get target elevation:                     */
   /*     vcp_num=RPGC_get_buffer_vcp_num((void*)inbuf);                 */
   /*     target_elev=(short)RPGCS_get_target_elev_ang(vcp_num,elev_idx); */
```

```
   /* generate the product message header (use system call) and input     */
   /* total accumulated product length minus 120 bytes                    */
   result3 = RPGC_prod_hdr((void*)buffer, output_id, &length);

   if (VERBOSE) {
     fprintf(stderr, "-> sample3_t2 completed product length=%d\n", length);
   }
   if (TEST) {
      fprintf(stderr, "\n==> after prod_hdr\n");
      print_message_header(buffer);
      print_pdb_header(buffer);
   }

   /*(this routine adds the length of the product header to the           */
   /* "length" parameter prior to creating the final header)              */

   /* if the creation of the product has been a success                   */
   if(result3 == 0) { /*success*/
      /* print product header for testing purposes                        */
      int res;

      if(TEST) {
         fprintf(stderr,
         "sample3_t2: product header creation success...PRINT header\n");
         res=print_message_header(buffer);
         res=print_pdb_header(buffer);
         fprintf(stderr, "\nsample3_t2: printing of header complete\n");
      }

      /* LABEL:OUTPUT_PROD    forward product and close buffer            */
      RPGC_rel_outbuf((void*)buffer, FORWARD);
      return(0);

   } else {  /* product failure (destroy the buffer & contents)            */
      RPGC_log_msg( GL_INFO, "sample3_t2: product header creation failure\n");
      RPGC_rel_outbuf((void*)buffer, DESTROY);
      RPGC_abort_because(PGM_PROD_NOT_GENERATED);
      return(-1);
   }

}  /* end of assemble product */



/***********************************
 * adaptation data read function
 ***********************************/
int read_sample3_t2_adapt()
{

double get_value = 0.0;
int ret = -1;


  ret = RPGC_ade_get_values("alg.sample3_t2.", "elev_select", &get_value);
  if(ret == 0) {
    input_elev = (int)get_value;
```

```
  } else {
    input_elev = 1;
    RPGC_log_msg( GL_INFO,
         "input_elev DEA value unavailable, using program default\n");
  }

  return ret;

}
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/**************************************************************************
Module:         sample3_t2.h

Description:    include file for the sample3_t2 demonstration program.

Authors:        Andy Stern, Software Engineer, Noblis Inc.
                    astern@noblis.org
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org

Version 1.3,  April 2005   T. Ganger
              Modified to use the new product IDs

Version 1.4   March 2007   T. Ganger
              Added warning that the standard 400 radial limit must be
                  increased to 800 radials if registering for the
                  new SR_ data types.
              Changed NUM_ELEV from 6 to 5.  This is a hard coded limit
                  of max number of elevations to process.  Without
                  testing for last elevations (number in current VCP),
                  the limit should not exceed the smallest VCP.

Version 1.5   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated definition of output buffer ids.
              New output buffer size calculated.

Version 1.6   November 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Eliminated artificial limit on number of elevations to read.
              Included new structure for intermediate product input and
                  internal data array.

$Id$
**************************************************************************/
#ifndef _SAMPLE3_T2_H_
#define _SAMPLE3_T2_H_

/* system includes ------------------------------------------------- */
#include <stdio.h>
#include <math.h>

/* ORPG   includes ------------------------------------------------- */
#include "rpg_globals.h"

/*** the following are included in rpg_globals.h */
/***                                              */
```

```
/***  #include <stdio.h>                          */
/***  #include <stdlib.h>                         */
/***  #include <time.h>                           */
/***  #include <ctype.h>                          */
/***  #include <string.h>                         */
/***                                              */
/***  #include <a309.h>                           */
/***  #include <basedata.h>                       */
/***  #include <basedata_elev.h>                  */
/***  #include <rpg_port.h>                       */
/***  #include <prod_gen_msg.h>                   */
/***  #include <prod_request.h>                   */
/***  #include <prod_gen_msg.h>                   */
/***  #include <rpg_vcp.h>                        */
/***  #include <orpg.h>                           */
/***  #include <mrpg.h>                           */
/************************************************/

/* required by rpgc after API enhancement patch */
#include <gen_stat_msg.h>

#include <rpgc.h>
#include "rpgcs.h"

/* The contents of Base_data_header is defined in basedata.h  */


/* Local Includes ------------------------------------------------------ */
/* #include "s3f_cvg_struct.h" */
#include "s3_t2_prod_struct.h"

/* Algorithm specific constants/definitions --------------------------- */

/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define SAMPLE3_FP 1992  */    /* Output Linear Buffer ID Code       */
/* #define SAMPLE3_IP 1999  */    /* Input Linear Buffer ID Code        */


/* the estimated buffer size for non Super Res data (in bytes):
 * (Pre ICD header     96 not included)
 * Msg Hdr/PDB        120
 * Symb Block          10
 * Layer Header         6
 * Packet Header       14
 * Packet Data     171022 ((460 bins+6 header bytes) x 367 radials)
 * TOTAL           171172
 */

/* to allow for 400 radials, the following can be used if not in a       */
/* Super Res elevation, the output buffer must be reallocated otherwise. */
#define BUFSIZE 184250

/* the algorithm will re-allocate the output buffer for larger size if    */
/* the basedata radial has increase surveillance resolution or 0.5 degree */
/* radials. For reference, the following buffer size are calculated needed.*/
/*    1472250 for 800 radials and 1840 bins (full super resolution)       */
/*     736250 for 400 radials and 1840 bins                               */
/*     368250 for 800 radials and  460 bins                               */
```

```
/*     184250 for 400 radials and  460 bins (original resolution )       */


/* Sample Alg 1.20 - eliminated artificial limit on number of elevations to read */
/*                   replaced with MAX_ELEVS defined in basedata.h               */
/* #define NUM_ELEV 5  */


/* function prototypes ------------------------------------------------ */
void clear_buffer(char*);

/* Sample Alg 1.20 - replaced CVG_radial* with unsigned char* */
int translate_product(unsigned char *ptr,char *buffer);

/* Sample Alg 1.20 - replaced CVG_radial with s3_t2_internal_rad */
int assemble_product(char *buffer, int number_of_bins, s3_t2_internal_rad
*rad_data);

extern int print_message_header(char* buffer);

extern int print_pdb_header(char* buffer);

/* Sample Alg 1.20 - replaced CVG_radial with s3_t2_internal_rad */
extern short build_symbology_layer(char* buffer, s3_t2_internal_rad *rad_ptr,
                                   int rad_count, int bin_count, int* length);

extern void finish_pdb(char* buffer, short elev_index, short target_elev,
                       short max_refl, int prod_len);

extern void product_to_disk(char * output_buf, int outlen, char *pname,
                                   short ele_idx);


#endif
```

Part E. Sample Algorithm 4 Tasks 1 & 2 Main Module

SAMPLE 4 TASK 1
(UPDATED MAY 2009)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample4_t1.c

Description:    sample4_t1.c is part of a chain-algorithm demonstration
                algorithm for the ORPG. The overall intent of the
                algorithm is to demonstrate a two task algorithm that
                produces multiple intermediate products and multiple
                final products.

                THE PURPOSE OF THIS SAMPLE ALGORITHM TASK IS TO DEMONSTRATE
                AN ALGORITHM USING THE 'WAIT_ALL' FORM OF THE CONTROL
                LOOP AND HAVING MORE THAN ONE OUTPUT.  Constructions of a
                simple intermediate product is accomplished.  No useful
                science is demonstrated.

                This algorithm also demonstrates a task reading an elevation
                input and a radial input.  The radial base data is not
                actually used, the task simply reads until the actual end
                of elevation.

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3.

                sample4_t1.c determines which (of two) elevation based
                intermediate products are requested creates the products
                requested.  These products contain a base data header and
                a short text message.

                Key source files for this algorithm include:

                sample4_t1.c         program source
                sample4_t1.h         main include file



Authors:        Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org
```

```
Version 1.2   March 2006   T. Ganger
              Revised to use current guidance for abort functions and
                   abort reason codes.
              Modified to demonstrate output buffer reallocation and to
                   produce a small intermediate product message rather
                   than an empty intermediate product message.

Version 1.3   June 2006   T. Ganger
              Revised to use RPGC_reg_io
              Revised to use the new 'by_name' get_inbuf/get_outbuf
                   functions

Version 1.4   August 2006   T. Ganger
              Modified to use the radial header offset to surveillance
                  data 'ref_offset'. Tested RPGC_get_surv_data.
              Replaced WAIT_ALL with WAIT_DRIVING_INPUT
              Deleted first argument from realloc_outbuf

Version 1.5   March 2007  T. Ganger
              Demonstrated the prod_attr_table method of declaring input
                  optional.
              Demonstrated RPGC_get_radar_data and RPGCS_radar_data_conversion
                  used with basic moment R. RPGC_get_radar_data is not reliable
                  in Build 9, wait for Build 10.
              Demonstrated combining radial and elevation inputs.

Version 1.6   June 2007  T. Ganger
              Modified for change in parameter type for RPGC_get_radar_data.

Version 1.8   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.

Version 2.0   November 2008    T. Ganger  (Sample Algorithm ver 1.20)
              Modified to provide a correct example of an algorithm having more
                  than out output with a WAIT_ALL loop having a driving input.
                  The algorithm may produce only one or all of the products
                  depending upon the requests for the outputs. Previously this
                  task had an undocumented assumption that both output
                  intermediate products were always requested (scheduled). Uses
                  RPGC_check_data_by_name() before acquiring the output buffer
                  and creating the product.  Uses RPGC_abort_dataname_because()
                  intead of RPGC_abort() where appropriate.  In subsequent
                  processing, including release of output buffers, tested each
                  output buffer for NULL before execution.

Version 2.1   March 2009    T. Ganger  (Sample Algorithm ver 1.21)
              Minor admin changes to improve clarity.

$Id$
**********************************************************************/

#include "sample4_t1.h"



/**********************************************************************/
int main(int argc,char* argv[]) {

   Base_data_radial
       *basedataPtr=NULL;    /* pointer to a base data radial structure  */
```

```
   Base_data_header
       *bdh=NULL;            /* base data header struct                  */

   char *inbuf=NULL;         /* pointer to optional input buffer         */

   char *buffer1=NULL;       /* pointer: access to allocated memory      */
   char *buffer2=NULL;       /* pointer: access to allocated memory      */
                             /* which holds a completed ICD product      */

   char *new_buffer1=NULL;   /* pointer to reallocated output product    */
   char *new_buffer2=NULL;   /* memory buffer                            */

   int bufsize;              /* size of product buffer                   */

   int ref_enable;           /* variables: used to hold the status of    */
   int vel_enable;           /* the individual moments of base data.     */
   int spw_enable;           /* tested after reading the first radial    */

   int PROCESS=TRUE;         /* Boolean used to control the daemon       */
                             /* portion of the program. Process until set*/
                             /* to FALSE.                                */
   int opstatus, status_opt; /* variables: hold function return values   */
   short radial_status;      /* variable: status of each radial input    */

   int opt_avail;            /* hold status of reading optional input    */

   /* data to be included in the product */
   char text1[] = "SAMPLE ALGORITHM 4 Product 1 (Intermediate)";
   char text2[] = "SAMPLE ALGORITHM 4 Product 2 (Intermediate)";



/* following support demonstration of multiple methods of reading base data */
   /*short *surv_data=NULL;  */ /* pointer to the surveillance data in the radial*/
   /*int index_first_bin=999;*/  /* used with method 2 */
   /*int index_last_bin=999; */  /* used with method 2 */
   int rad_idx;
   short *surv_data3=NULL; /* pointer to the surveillance data in the radial */


Generic_moment_t gen_moment;  /* used with method 3 */
/* cannot pass a NULL pointer to RPGC_get_radar_data */
Generic_moment_t *my_gen_moment = &gen_moment; /* method 3 */

float *surv_values=NULL;
int decode_res;

   int VERBOSE=FALSE;
/*   int VERBOSE=TRUE; */


   /* ---------------------------------------------------------------- */

   fprintf(stderr,"\nBegin Sample 4 Task 1 Algorithm\n");

   /* register inputs and outputs based upon the                       */
   /* contents of the task_attr_table (TAT).                           */
   /* input:REFLDATA(79), output:SAMPLE4_IP1(1998)  SAMPLE4_IP2(1997) */
   /* we also have a test optional input SAMPLE3_IP(1999)              */
```

```
    RPGC_reg_io(argc,argv);


    /* ------------------------------------------------ */
    /* the original method for designating an optional input is    */
    /* using the following function. Now we use the opt_prods_list */
    /* attribute in the product attribute table to designate an    */
    /* the optional input.          This function can be used to   */
    /* modify the default 5 second delay to an 8 second delay.     */
    RPGC_in_opt_by_name("SAMPLE3_IP", 8);


   /* Register algorithm infrastructure to read the Scan Summary Array   */
   RPGC_reg_scan_summary();

   /* ORPG task initialization routine. Input parameters argc/argv are   */
   /* not used in this algorithm                                         */
   RPGC_task_init(ELEVATION_BASED, argc, argv);

   /* output message to both the task output file and the task log file */
   fprintf(stderr,
      "-> algorithm initialized and proceeding to loop control\n");
   RPGC_log_msg( GL_INFO,
      "-> algorithm initialized and proceeding to loop control\n");


   /* while loop that controls how long the task will execute. As long as*/
   /* PROCESS remains TRUE, the task will continue. The task will        */
   /* terminate upon an error condition                                  */
   while(PROCESS) {

      /* system call to indicate a data driven algorithm. block algorithm*/
      /* until good data is received at & least one product is requested */
      RPGC_wait_act(WAIT_DRIVING_INPUT);

      if (VERBOSE) fprintf(stderr,
          "-> sample4_t1 algorithm passed ACL and began processing\n");
      RPGC_log_msg( GL_INFO,
          "-> sample4_t1 algorithm passed ACL and began processing\n");


      /* initial size of output product buffer                          */
      bufsize = (int) sizeof(Base_data_header);

      /* Sample Alg version 1.20 */
      /* Check to see if there is a request for each output datatype. */
      /* If so, allocate memory (accessed by the pointer, buffer) in  */
      /* which the output is constructed. error return in opstatus    */
      buffer1 = NULL;
      buffer2 = NULL;

      if( RPGC_check_data_by_name( "SAMPLE4_IP1" ) == NORMAL ) {
          buffer1 = (char *)RPGC_get_outbuf_by_name("SAMPLE4_IP1", bufsize,
                                                    &opstatus);
          /* check error condition from buffer allocation. abort if abnormal */
          if(opstatus != NORMAL) {
             RPGC_log_msg( GL_INFO,
                "ERROR: Aborting from RPGC_get_outbuf...opstatus=%d\n", opstatus);
             RPGC_abort_dataname_because( "SAMPLE4_IP1", opstatus );
             /* note: buffer1 is NULL in this case */
```

```
        }
    }

    if( RPGC_check_data_by_name( "SAMPLE4_IP2" ) == NORMAL ) {

        buffer2 = (char *)RPGC_get_outbuf_by_name("SAMPLE4_IP2", bufsize,
                                                  &opstatus);
        /* check error condition from buffer allocation. abort if abnormal */
        if(opstatus != NORMAL) {
           RPGC_log_msg( GL_INFO,
              "ERROR: Aborting from RPGC_get_outbuf...opstatus=%d\n", opstatus);
           RPGC_rel_outbuf((void *)buffer1, DESTROY);
           RPGC_abort_dataname_because( "SAMPLE4_IP2", opstatus );
           /* note: buffer2 is NULL in this case */
        }
    }

    /* Sample Alg version 1.20 */
    /* all needed output buffers can not be obtained, already aborted above */
    if( (buffer1 == NULL)  &&  (buffer2 == NULL) )
        continue;

    /* LABEL:READ_FIRST_RAD Read first radial of the elevation and     */
    /* accomplish the required moments check.                          */
    /* ingest one radial of data from the BASEDATA linear buffer. The  */
    /* data will be accessed via the basedataPtr pointer               */

    basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("REFLDATA",
                                                    &opstatus);
    bdh = (Base_data_header *) basedataPtr;

    /* check radial ingest status before continuing                    */
    if(opstatus != NORMAL){
       RPGC_log_msg( GL_INFO,"ERROR: Aborting from RPGC_get_inbuf\n");
       /* Sample Alg version 1.20 */
       if(buffer1 != NULL) RPGC_rel_outbuf((void*)buffer1, DESTROY);
       if(buffer2 != NULL) RPGC_rel_outbuf((void*)buffer2, DESTROY);
       RPGC_abort(); /* abort all requested prods */
       continue;
    }

    /* test to see if the required moment (reflectivity) is enabled    */
    RPGC_what_moments(bdh,&ref_enable,
                      &vel_enable,&spw_enable);

    if(ref_enable != TRUE){
       RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_what_moments\n");
       RPGC_rel_inbuf((void*)basedataPtr);
       /* Sample Alg version 1.20 */
       if(buffer1 != NULL) RPGC_rel_outbuf((void*)buffer1, DESTROY);
       if(buffer2 != NULL) RPGC_rel_outbuf((void*)buffer2, DESTROY);
       RPGC_abort_because(PGM_DISABLED_MOMENT); /* abort all requested prods */
       continue;
    }

    /* NORMALLY AN ELEVATION INPUT CAN BE READ IMMEDIATELY AFTER READING */
    /* THE FIRST RADIAL INPUT.  HOWEVER, SINCE THIS ELEVATION INPUT      */
    /* IS DESIGNATED OPTIONAL, IT MUST BE READ AFTER ALL RADIAL INPUTS   */
    /* ARE READ                                                          */
```

```
      opt_avail = FALSE;  /* flag to signal later processing of input, */
                          /* if available                              */

      rad_idx=0;

      while(TRUE) {

      /* RADIAL PROCESSING SEGMENT. continue to ingest and process       */
      /* individual base radials until either a failure to read valid    */
      /* input data (a radial in correct sequence) or until after reading*/
      /* and processing the last radial in the elevation.                */

         /* read the radial status flag to determine end of elevation    */
         radial_status=basedataPtr->hdr.status & 0xf;


         /* Radial processing here, This algorithm simply copies the     */
         /* base data header into the product buffer.                    */

         /* While all of the following work, Method 2 is typical for basic */
         /* moments, Method 3 must be used for advanced Dual Pol moments   */
         /* Method 1 */
         /* surv_data = (short *) ((char *) basedataPtr + bdh->ref_offset); */

         /* Method 2 */
         /* surv_data = (short *) RPGC_get_surv_data( (void *)basedataPtr, */
         /*                           &index_first_bin, &index_last_bin);   */

         /* Method 3 */
         surv_data3 = (short *) RPGC_get_radar_data( (void *)basedataPtr,
                                   RPGC_DREF, my_gen_moment );


         /* Sample Alg 1.18 - test for NULL moment pointer and 0 offset */

         if(surv_data3 == NULL) {
            RPGC_log_msg( GL_INFO, "ERROR: NULL radial detected: %d\n",
                                                                 rad_idx);
            RPGC_rel_inbuf((void*)basedataPtr);
            /* Sample Alg version 1.20 */
            if(buffer1 != NULL) RPGC_rel_outbuf((void*)buffer1, DESTROY);
            if(buffer2 != NULL) RPGC_rel_outbuf((void*)buffer2, DESTROY);
            RPGC_abort_because(PGM_INPUT_DATA_ERROR); /* abort all requested prods
*/
            opstatus = TERMINATE;
            break;
         }

         if(bdh->ref_offset == 0) {
            RPGC_log_msg( GL_INFO,"ERROR: reflectivity offset 0: %d\n",
               rad_idx);
            RPGC_rel_inbuf((void*)basedataPtr);
            /* Sample Alg version 1.20 */
            if(buffer1 != NULL) RPGC_rel_outbuf((void*)buffer1, DESTROY);
            if(buffer2 != NULL) RPGC_rel_outbuf((void*)buffer2, DESTROY);
            RPGC_abort_because(PGM_INPUT_DATA_ERROR); /* abort all requested prods
*/
            opstatus = TERMINATE;
            break;
         }
```

```
        /* though not used, we demonstrate decoding data into dBZ values */
        decode_res = RPGCS_radar_data_conversion( (void *)surv_data3,
                                    my_gen_moment, -999.0, -888.0,
                                                    &surv_values);

        if(decode_res == -1) { /* decode error, abort processing ************** */

              /* if the products actually used the data, would abort here */

        } else { /* PROCESS RADIAL DATA AND COPY ***************************** */

           /* if radail data were actually used, the radial data would be */
           /* copied and processed as required                             */
           /* this algorithm just includes debug print outs                */

           /* BEGIN DEBUG ------------------------------------------------- */
           /* if( (rad_idx==0) || (rad_idx==100) || (rad_idx==360) ) {      */
           /* fprintf(stderr,                                              */
           /*   "DEBUG RADIAL %d: range to first bin is %d,  num bins is %d\n", */
           /*                         rad_idx, bdh->surv_range - 1,        */
           /*                         bdh->n_surv_bins );                  */
           /* fprintf(stderr,                                              */
           /*   "DEBUG funct parameters    first bin is %d,  last bin is %d\n", */
           /*                         index_first_bin, index_last_bin);    */
           /* }                                                            */
           /*                                                              */
           /* if( ((rad_idx<=2) || (rad_idx==100)) && (surv_data!=NULL) ) {  */
           /* fprintf(stderr,                                              */
           /*              "Radial %3d: 1   2   3   4   5   6   7   8   9  10"  */
           /*              "  11  12  13  14  15  16  17  18  19  20\n",
rad_idx);*/
           /* fprintf(stderr,                                              */
           /*              "          %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d"  */
           /*              " %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d \n",  */
           /*    surv_data[0], surv_data[1], surv_data[2], surv_data[3],   */
           /*    surv_data[4], surv_data[5], surv_data[6], surv_data[7],   */
           /*    surv_data[8], surv_data[9], surv_data[10], surv_data[11],  */
           /*    surv_data[12], surv_data[13], surv_data[14], surv_data[15],  */
           /*    surv_data[16], surv_data[17], surv_data[18], surv_data[19]);  */
           /* }                                                            */
           /*                                                              */
           /*                                                              */
           /* if( (rad_idx==0) || (rad_idx==100) || (rad_idx==360) ) {      */
           /* fprintf(stderr,                                              */
           /*     "DEBUG first gate range is %d, number of gates is %d\n",  */
           /*      my_gen_moment->first_gate_range, my_gen_moment->no_of_gates);*/
           /* }                                                            */
           /*                                                              */
           /* if( ((rad_idx<=2) || (rad_idx==100)) && (surv_data3!=NULL) ) {  */
           /* fprintf(stderr,                                              */
           /*              "Radial %3d: 1   2   3   4   5   6   7   8   9  10"  */
           /*              "  11  12  13  14  15  16  17  18  19  20\n",
rad_idx);*/
           /* fprintf(stderr,                                              */
           /*              "          %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d"  */
           /*              " %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d \n",  */
           /*   surv_data3[0], surv_data3[1], surv_data3[2], surv_data3[3],  */
           /*   surv_data3[4], surv_data3[5], surv_data3[6], surv_data3[7],  */
```

```
            /*   surv_data3[8], surv_data3[9], surv_data3[10], surv_data3[11],   */
            /*   surv_data3[12], surv_data3[13], surv_data3[14], surv_data3[15], */
            /*   surv_data3[16], surv_data3[17], surv_data3[18], surv_data3[19]);*/
            /* }                                                                 */
            /*                                                                   */
            /* if( ((rad_idx<=2) || (rad_idx==100)) && (decode_res!=-1) ) {      */
            /* fprintf(stderr,"Decoded Values:\n");                              */
            /* fprintf(stderr,                                                   */
            /*  "  %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f\n"*/
            /*  "  %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f
%3.2f\n",*/
            /*   surv_values[0], surv_values[1], surv_values[2], surv_values[3], */
            /*   surv_values[4], surv_values[5], surv_values[6], surv_values[7], */
            /*   surv_values[8], surv_values[9], surv_values[10],
surv_values[11],*/
            /*   surv_values[12],surv_values[13],surv_values[14],surv_values[15],*/
            /*
surv_values[16],surv_values[17],surv_values[18],surv_values[19]);*/
            /* }                                                                 */
            /* END DEBUG ---------------------------------------------------- */

            if( surv_values != NULL) {
               free(surv_values);
               surv_values = NULL;
            }

         } /* end PROCESS RADIAL DATA AND COPY ******************************** */



         /* if end of elevation or volume process radial then exit loop. */
         /* this is the ACTUAL end of elevation rather than Pseudo end   */
         if(radial_status == GENDEL || radial_status == GENDVOL) {

            /* Sample Alg version 1.20 - only process requested product*/
            if(buffer1 != NULL)
                memcpy((void*)buffer1,(void*)basedataPtr,sizeof(Base_data_header));
            if(buffer2 != NULL)
                memcpy((void*)buffer2,(void*)basedataPtr,sizeof(Base_data_header));

            /* now that the data that we want from the radial is in      */
            /* the product...release the input buffer                    */
            RPGC_rel_inbuf((void*)basedataPtr);



           /* DEMONSTRATION begin read optional elevation input *********** */
           /* AFTER LAST RADIAL INPUT DEMO READING ELEVATION INPUT          */

           /* WITH NO TIME DELAY, THIS OPTIONAL PRODUCT MAY NOT BE READ.    */
           /* needed to add a sleep delay if using the product_attr_table   */
           /* method no delay is used and this elevation product is never   */
           /* actually read                                                 */

           inbuf = (char *)RPGC_get_inbuf_by_name("SAMPLE3_IP", &status_opt);

           if(status_opt != NORMAL){
              /* optional input not available */
              if (VERBOSE)
                 fprintf(stderr,
```

```
                    "-> sample4_t1 optional elev input not available (%d)\n",
                                                                    status_opt);
              /* remaining processing should not use optional input */
              opt_avail = FALSE;

            } else {
              if (VERBOSE)
                  fprintf(stderr,
                      "-> sample4_t1 optional elev input successfully read\n");
              opt_avail = TRUE;
            }
            /* DEMONSTRATION end read optional elevation product************ */


             /* exit inner loop with opstatus==NORMAL and no ABORT */
             break;


         } else { /* not last radial, continue in the inner loop */
            /* this demonstration does nothing until the last radial     */
            /* release the input buffer                                  */
            RPGC_rel_inbuf((void*)basedataPtr);

         }

         /* LABEL:READ_NEXT_RAD Read the next radial of the elevation.   */
         /* ingest one radial of data from the BASEDATA linear buffer.   */
         /* The data will be accessed via the basedataPtr pointer        */

         basedataPtr = (Base_data_radial*)RPGC_get_inbuf_by_name("REFLDATA",
                                                          &opstatus);
         bdh = (Base_data_header *) basedataPtr;

         /* check radial ingest status before continuing                 */
         if(opstatus != NORMAL) {
            RPGC_log_msg( GL_INFO, "ERROR: Aborting from RPGC_get_inbuf\n");
            /* Sample Alg version 1.20 */
            if(buffer1 != NULL) RPGC_rel_outbuf((void*)buffer1, DESTROY);
            if(buffer2 != NULL) RPGC_rel_outbuf((void*)buffer2, DESTROY);
            RPGC_abort();   /* aborted all requested products */
            break;  /* break out of inner loop */
         }
         rad_idx++;

      } /* end while TRUE, process elevation */


      if(VERBOSE) {
          if(opstatus == NORMAL)
               fprintf(stderr, "DEBUG out of read loop, opstatus NORMAL.\n");
          else
              fprintf(stderr, "DEBUG out of read loop, opstatus %d.\n", opstatus);
      }

      /* LABEL:ASSEMBLE The assembly of the intermediate product         */
      if(opstatus == NORMAL) {

         /* Elevation processing here. */

         /* could use optional input here, released after use  */
```

```
        if(opt_avail == TRUE) {
           /* simulate use of optional input if available */

           RPGC_rel_inbuf((void*)inbuf);
        }


        /* The reallocation of the output buffer size is demonstrated.  */

        bufsize = bufsize + 50;

        /* Sample Alg version 1.20 only process if requested   */
        /* PROCESS PRODUCT 1 ******************************** */
        if(buffer1 != NULL) {

            new_buffer1 = RPGC_realloc_outbuf( (void*)buffer1,
                                                  bufsize, &opstatus );

            if(opstatus != NORMAL) { /* abort product 1 */
               RPGC_log_msg( GL_INFO,
                  "ERROR: Aborting from RPGC_realloc_outbuf...opstatus=%d\n",
                  opstatus);
               RPGC_rel_outbuf((void *)buffer1, DESTROY);
               /* Sample Alg version 1.20 */
               RPGC_abort_dataname_because( "SAMPLE4_IP1", opstatus );

            } else { /* output product 1 */

            memcpy( (void*)new_buffer1+sizeof(Base_data_header),
                  (void*)text1, strlen(text1)+1);

               /* LABEL:OUTPUT_PROD    forward product and close buffer  */
               RPGC_rel_outbuf((void*)new_buffer1, FORWARD);

           } /* end output product 1 */

        } /* END PROCESS PRODUCT 1 ************************** */


        /* Sample Alg version 1.20 only process if requested   */
        /* PROCESS PRODUCT 2 ******************************** */
        if(buffer2 != NULL) {

            new_buffer2 = RPGC_realloc_outbuf( (void*)buffer2,
                                                  bufsize, &opstatus );

            if(opstatus != NORMAL) { /* abort product 2 */
               RPGC_log_msg( GL_INFO,
                  "ERROR: Aborting from RPGC_realloc_outbuf...opstatus=%d\n",
                  opstatus);
               RPGC_rel_outbuf((void *)buffer2, DESTROY);
               /* Sample Alg version 1.20 */
               RPGC_abort_dataname_because( "SAMPLE4_IP2", opstatus );

            } else { /* output product 2 */

               memcpy( (void*)new_buffer2+sizeof(Base_data_header),
                  (void*)text2, strlen(text2)+1);

               /* LABEL:OUTPUT_PROD    forward product and close buffer  */
```

```
                RPGC_rel_outbuf((void*)new_buffer2, FORWARD);

             } /* end output product 2 */

          } /* END PROCESS PRODUCT 2 ************************** */


       }  /* end of if(opstatus==NORMAL) block                          */


       if(opt_avail == TRUE) {
          RPGC_rel_inbuf((void*)inbuf);
       }



    } /* end while PROCESS == TRUE                                      */

    return 0;

} /* end of main ------------------------------------------------------ */
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample4_t1.h

Authors:        Steve Smith, Software Engineer, NWS ROC
                    steve.smith@noaa.nssl.gov
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org
                Version 1.0, January 2002

Version 1.1   April 2005   T. Ganger
              Modified to use new product IDs

Version 1.2   March 2006   T. Ganger
              Eliminated constant output buffer size.

Version 1.3   March 2007   T. Ganger
              Removed buffer name defines since by_name functions are used.

Version 1.4   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Eliminated definition of output buffer ids.

$Id$
************************************************************************/

#ifndef _SAMPLE4_T1_H_
#define _SAMPLE4_T1_H_

#include <unistd.h>

/* ORPG   includes ------------------------------------------------*/
#include <rpg_globals.h>

/*** the following are included in rpg_globals.h */
/***                                             */
/***  #include <stdio.h>                         */
/***  #include <stdlib.h>                        */
/***  #include <time.h>                          */
/***  #include <ctype.h>                         */
/***  #include <string.h>                        */
/***                                             */
/***  #include <a309.h>                          */
/***  #include <basedata.h>                      */
/***  #include <basedata_elev.h>                 */
/***  #include <rpg_port.h>                      */
/***  #include <prod_gen_msg.h>                  */
/***  #include <prod_request.h>                  */
/***  #include <prod_gen_msg.h>                  */
/***  #include <rpg_vcp.h>                       */
```

```
/***  #include <orpg.h>                              */
/***  #include <mrpg.h>                              */
/************************************************/


#include <basedata.h>

/* required by rpgc after API enhancement patch */
#include <gen_stat_msg.h>

#include <rpgc.h>
#include "rpgcs.h"




/* local defines -----------------------------------------------------*/


/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define SAMPLE4_IP1        1998  */
/* #define SAMPLE4_IP2        1997  */



#endif
```

SAMPLE 4 TASK 2
(UPDATED NOVEMBER 2008)

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/**************************************************************************
Module:         sample4_t2.c

Description:    sample4_t2.c is part of a chain-algorithm demonstration
                algorithm for the ORPG. The overall intent of the
                algorithm is to demonstrate a two task algorithm that
                produces multiple intermediate products and multiple
                final products.

                THE PURPOSE OF THIS SAMPLE ALGORITHM TASK IS TO DEMONSTRATE
                AN ALGORITHM USING THE 'WAIT_ANY' FORM OF THE CONTROL
                LOOP AND HAVING MORE THAN ONE OUTPUT.  Constructions of a
                simple product using a single text data packet is
                accomplished.  No useful science is demonstrated.

                The structure of this algorithm complies with the guidance
                provided in the CODE Guide Vol 3.

                sample4_t2.c reads the available intermediate product,
                determines which product was read and if the corresponding
                final product was requested.  The final product is a basic
                geographic product containing trivial text data.


                Key source files for this algorithm include:

                sample4_t2.c         program source
                sample4_t2.h         main include file



Authors:        Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org


Version 2.0   April 2006   T. Ganger
              Revised to use current guidance for abort functions and
                   abort reason codes.
              Rewritten to create a simple graphic product containing
                   a single text packet rather than a stand-alone
                   tabular alphanumeric product (SATAP)

Version 2.1   June 2006    T. Ganger
              Revised to use RPGC_reg_io
              Revised to use the new 'by_name' get_outbuf, check_data,
```

```
                  and abort_dataname functions

Version 2.2   August 2007  T. Ganger
              Separated first layer header from the symbology block
                 structure for clarity
              Added defined offsets

Version 2.3   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated use of defined output buffer ids by using
                  RPGC_get_id_from_name and by testing the registration
                  name for the output rather than the id.

Version 2.4   November 2008   T. Ganger   (Sample Algorithm ver 1.20)
              Clarified overall description of task.

$Id$
**************************************************************************/


#include "sample4_t2.h"


/* SET TO GET VARIOUS DIAGNOSTIC OUTPUTS */
int TEST = FALSE;
int VERBOSE = FALSE;
int test_out=FALSE; /* Boolean to control diagnostic product file output */



/* Function prototypes.------------------------------------------- */

static int Process_input( char *inbuf, char *out_data_name,
                                       int output_size );

void build_data_layer(char *outbuffer, char *text_data, int *length);

void finish_prod_desc_block(char *outbuffer, short elev_index,
                            short elevation, int prod_len);

void clear_buffer(char *buffer, int bufsize);



/**************************************************************************/
/* Main routine for sample4_t2 test program.                              */
int main( int argc, char *argv[] ){

   /* algorithm variable declarations and definitions  */
   int opstatus;            /* variable:  used to hold API return values */
   char *inbuf;             /* pointer to intermediate input buffer      */

   /* keep until Build 9 changes completed */
   int in_datatype;         /* prod ID returned from RPGC_get_inbuf_any  */

   int output_size;         /* output product size, in bytes             */

   char OUTDATA_NAME[65];
```

```
/*---------------------------------------------------------------------*/

   /* Initialize log-error services. */
   RPGC_init_log_services( argc, argv );


   /* register inputs and outputs based upon the           */
   /* contents of the task_attr_table (TAT).               */
   /* input:  SAMPLE4_IP1(1998)  SAMPLE4_IP2(1997)         */
   /* output: SAMPLE4_FP1(1993)  SAMPLE4_FP2(1994)         */
   RPGC_reg_io(argc, argv);


   /* Specify task timing. */
   RPGC_task_init( ELEVATION_BASED, argc, argv );

   fprintf(stderr, "Sample4_t2 initialization complete. \n");
   RPGC_log_msg( GL_INFO, "Sample4_t2 initialization complete. \n");


   /* Main processing loop ... */
   while(1) {

      /* Wait for any of the inputs to become available. */
      RPGC_wait_for_any_data( WAIT_ANY_INPUT );

      if(VERBOSE)
          fprintf(stderr, "Sample4_t2 released for processing \n");
      RPGC_log_msg( GL_INFO, "Sample4_t2 released for processing \n");

      /* One of the inputs is available, get the input and process
         accordingly. */
      in_datatype = 0;
      opstatus = NORMAL;

      /* WISH LIST: WOULD BE NICE TO HAVE a get_inbuf_by_name_any or */
      /*            a get_name_from_id function                     */
      /* Instead, we compare the data_type to the value returned by  */
      /* RPGC_get_id_from_name                                       */

      inbuf = (char *) RPGC_get_inbuf_any( &in_datatype, &opstatus );

       /* For SAMPLE4_IP1 input, do the following algorithm processing.*/
      if( in_datatype == RPGC_get_id_from_name( "SAMPLE4_IP1") ) {

         RPGC_log_msg( GL_INFO,
                     "RPGC_get_inbuf_any returned SAMPLE4_IP1\n" );
         strcpy(OUTDATA_NAME, "SAMPLE4_FP1");
         output_size = SAMPLE4_FP_SIZE;

        /* For SAMPLE3_IP2 input, do the following algorithm processing.*/
      } else if( in_datatype == RPGC_get_id_from_name( "SAMPLE4_IP2") ) {

         RPGC_log_msg( GL_INFO,
                    "RPGC_get_inbuf_any returned SAMPLE4_IP2\n" );
         strcpy(OUTDATA_NAME, "SAMPLE4_FP2");
         output_size = SAMPLE4_FP_SIZE;

      } else { /* Can this actually happen? */
```

```
         RPGC_log_msg( GL_INFO, "Unknown Datatype Received %d\n",
                       in_datatype );
         if( opstatus == NORMAL )
             RPGC_rel_inbuf( inbuf );
         RPGC_abort();
         continue;
      }

      /* If non-NORMAL status returned, abort processing. */
      if( opstatus != NORMAL ){
         RPGC_log_msg( GL_INFO,
                      "RPGC_get_inbuf_any Returned Bad Status (%d)\n",
                       opstatus );
         if( opstatus == TERMINATE ) /* IS THIS NECESSARY?  */
            RPGC_abort();
         else
            RPGC_abort_dataname_because( OUTDATA_NAME, opstatus );
         continue;
      }


      /* Process the input data  .... generate the output data. */
      Process_input( inbuf, OUTDATA_NAME, output_size );

   } /* End of "while" loop. */


} /* End of main */




/**************************************************************************

   Description:
      Processes the input buffer and generates an output buffer based on
      outstanding request.

   Inputs:
      inbuf - pointer to input buffer
      out_data_name - output data name associated with input datatype
      output_size - size, in bytes, of output data name

   Outputs:
      None

   Returns:
      -1 on error, 0 otherwise.

**************************************************************************/
static int Process_input( char *inbuf, char *out_data_name,
                          int output_size ){

   char *outbuf;            /* pointer to contain a SATAP product*/
   int opstatus;            /* status returned from RPGC_get_outbuf call */

   Base_data_header *bdh;  /* the first base data header */

   char data_string[SAMPLE4_IP_SIZE - 24]; /* intermediate product data   */
```

```
   int ret_val;

   int prod_id;


   /* Check to see if there is a request for output datatype. */
   if( RPGC_check_data_by_name( out_data_name ) != NORMAL ) { /* no request */

      RPGC_log_msg( GL_INFO, "No Requests For %s Found\n", out_data_name  );
      RPGC_rel_inbuf( inbuf );

      return(-1);

   } else { /* there is a request for the product out_datatype */

      /* Get output buffer associated with this input. */
      outbuf = (char*) RPGC_get_outbuf_by_name(out_data_name, output_size,
                                                              &opstatus);

      if( opstatus != NORMAL ) {
         RPGC_log_msg( GL_INFO,
                     "RPGC_get_outbuf for %s Returned Bad Status (%d)\n",
                     out_data_name , opstatus);
         RPGC_abort_dataname_because( out_data_name, opstatus );
         RPGC_rel_inbuf( inbuf );

         return(-1);

      }  else { /* opstatus NORMAL, create the product */

         int vol_num, outlen = 0;


         clear_buffer(outbuf, output_size);

         /* If there is processing of the input and building of the
            output, do it here. */
         vol_num = RPGC_get_buffer_vol_num( inbuf );

         prod_id = RPGC_get_id_from_name( out_data_name );

         /* step 1: build the product description block -uses a system call*/
         RPGC_prod_desc_block( outbuf, prod_id, vol_num );


         /* step 2: build the symbology layer & and packet 1 data.      */

         /* get the character text from the intermediate product */
         strcpy( data_string, inbuf + sizeof(Base_data_header) );

         /* this routine returns both the overall length (thus far) of the      */
         /* product and the maximum reflectivity of all radials                 */
         build_data_layer(outbuf, data_string, &outlen);



         /* step 3: finish building the product description block by            */

         /* using elevation index and actual elevation using base data header   */
         /* that has been passed as part of the intermediate product            */
```

```
         bdh = (Base_data_header *) inbuf;

         finish_prod_desc_block(outbuf, bdh->rpg_elev_ind,
                                        bdh->target_elev, outlen);
         /* if we did not pass the base data header in the intermediate product */
         /* helper functions could be used to obtain elevation index and angle  */



         /* step 4: build the message header block (mhb)                         */
         ret_val = RPGC_prod_hdr( outbuf, prod_id, &outlen );

        if (TEST) {
           fprintf(stderr, "\n==> after prod_hdr\n");
           print_message_header(outbuf);
           print_pdb_header(outbuf);
        }


         if(ret_val != 0) {
            RPGC_log_msg( GL_INFO, "product header creation failure\n");
            RPGC_rel_outbuf((void*)outbuf, DESTROY);
            RPGC_rel_inbuf( inbuf );
            RPGC_abort_because(PGM_PROD_NOT_GENERATED);
            return(-1);
         }


         /* diagnostic - interrupts product output and creates a
          * binary output of product to file.
          * this is useful if product problems cause task failure
          */
         if(test_out == TRUE) {

            product_to_disk(outbuf, outlen, out_data_name, bdh->rpg_elev_ind);

            RPGC_log_msg( GL_INFO,
               "Interrupted product output for diagnostic output to file.\n");
            RPGC_rel_outbuf( (void*)outbuf, DESTROY );

         } else {

            /* step 5: forward the completed product to the system    */

            RPGC_rel_outbuf( (void*)outbuf, FORWARD );

         }


         /* Release the input buffer, then wait for any data. */
         RPGC_rel_inbuf( inbuf );

         return(0);

      } /* end else opstatus NORMAL */


   } /* end else there is a request for out_ datatype */
```

```
} /* End of Product_processing() */



/*************************************************************************/
void build_data_layer(char *outbuffer, char *text_data, int *length) {


  /* buffer entry points (offsets) ----------------------------------------
    bytes shorts
       0     0    message header block offset (120 bytes or 60 shorts)
     120    60    symbology block offset      ( 10 bytes or  5 shorts)
     130    65    first layer header offset   (  6 bytes or  3 shorts)
     136    68    packet 1 header offset      (  8 bytes or  4 shorts)
     144    72    packet 1 character data offset

  constant SYMB_OFFSET = 120
  constant SYMB_SIZE = 10
  constant LYR_HDR_SIZE = 6
  constant PKT_HDR_SIZE = 8
  ------------------------------------------------------------------------*/


  /* variable declarations and definitions ---------------------------- */
  /* added Sample Alg 1.17 */
  sym_block_hdr sym;      /* symbology block header struct              */
  layer_hdr lyr;          /* symb layer header struct                   */

  pkt_1_hdr pkt;          /* packet 1 header sruct                      */

  unsigned int block_len;
  short num_chars;
  int i;

  char added_text[] = " (Final Product)";
  char output_text[ SAMPLE4_FP_SIZE -
                    (SYMB_OFFSET+SYMB_SIZE+LYR_HDR_SIZE+PKT_HDR_SIZE) ];
  char a_space[] = " ";

  if (VERBOSE) {
    fprintf(stderr, "building symbology block now\n");
  }


   if(TEST) {
     fprintf(stderr, "Input Character Data is:\n'");
     for(i = 0; i < strlen(text_data); i++)
         fprintf(stderr, "%c", text_data[i]);
     fprintf(stderr, "'\n");
   }


  /* PRODUCT DATA PROCESSING - modify the input text for output */
  /*-----------------------------------------------------------------------*/

  /* combine the first 28 characters from input text with added text */
  if(strlen(text_data) >= 28)
      num_chars = 28;
  else
```

```
     num_chars = strlen(text_data);

  memcpy(output_text, text_data, num_chars);
  memcpy(output_text+num_chars, added_text, strlen(added_text));

  /* total text length is input text plus added text */
  num_chars += (short) strlen(added_text);

  if(TEST)
     fprintf(stderr, "Length of output text is %d characters.\n", num_chars);

  /* packet code 1 requires an even number of characters */
  /* if odd, num_chars is incremented and a space is appended to output */
  if( (num_chars % 2) != 0 ) {
       num_chars++;
       memcpy(output_text+num_chars, a_space, 1);
       if(TEST)
          fprintf(stderr, "Padded output text with a space\n.");
  }



  /* set symbology block header and first layer length */
  /*-------------------------------------------------------------------*/
  sym.divider = (short)-1;      /* block divider (constant value)         */
  sym.block_id = (short)1;      /* block ID=1 (constant value)            */

  /* block length: symb & layer header + packet 1 hdr + num characters  */
  block_len = (unsigned int)(SYMB_SIZE + LYR_HDR_SIZE + PKT_HDR_SIZE + num_chars);
  RPGC_set_product_int( (void *) &sym.block_length, block_len );

  sym.num_layers = (short)1;    /* number of data layers included         */

  lyr.divider2 = (short)-1;     /* block divider (constant value)         */

  /* first layer length: block_length - sym block length                */
  RPGC_set_product_int( (void *) &lyr.layer_length,
              (unsigned int)(block_len - SYMB_SIZE - LYR_HDR_SIZE) );


  /* set packet 1 header */
  /*-------------------------------------------------------------------*/
  pkt.code = (short)1;
  /* length of data is num_chars + 4bytes for the I and J position */
  pkt.num_bytes = num_chars+4;
  pkt.pos_i = (short)270;
  pkt.pos_j = (short)270;


  /*copy symbology header and layer 1 header to product                 */
  /*-------------------------------------------------------------------*/
  /* Sample Alg 1.17 */
  /* OLD: previously the structure containing the symb header and the
   *      the layer header could be copied as a block
   */
  /* NEW: the separate symb header and layer header structures have
   *      alignment issues and some data fields must be copied separately
   */
  memcpy(outbuffer + SYMB_OFFSET, &sym, SYMB_SIZE);
  memcpy(outbuffer + SYMB_OFFSET + SYMB_SIZE, &lyr.divider2, 2);
```

```
  memcpy(outbuffer + SYMB_OFFSET + SYMB_SIZE + 2, &lyr.layer_length, 4);


  /*  copy packet 1 header and character data to product                */
  /*-------------------------------------------------------------------*/
  memcpy(outbuffer + SYMB_OFFSET + SYMB_SIZE + LYR_HDR_SIZE,
                                              &pkt, PKT_HDR_SIZE);

  /* the packet 1 character data does NOT include the terminating Null */
  memcpy(outbuffer + SYMB_OFFSET + SYMB_SIZE + LYR_HDR_SIZE + PKT_HDR_SIZE,
                                                  output_text, num_chars);

  /* return the total length of the product minus the Graphic_product   */
  /* structure length which is added in the RPGC_prod_hdr function       */
  *length = SYMB_SIZE + LYR_HDR_SIZE + PKT_HDR_SIZE + num_chars;


  if(TEST) {

     unsigned short *short_ptr;

     /* print contents of symbology and first layer header for diagnostic */
     print_symbology_header( (outbuffer + SYMB_OFFSET) );


     /* print contents of data packet 1 for diagnostic */
     short_ptr = (unsigned short *)&pkt;

     fprintf(stderr,"\n");
     fprintf(stderr,"Packet 1.code is      %d(%d)\n",pkt.code,
                          short_ptr[0]);
     fprintf(stderr,"Packet 1.num_bytes is %d(%d)\n",pkt.num_bytes,
                          short_ptr[1]);
     fprintf(stderr,"Packet 1.pos_i is     %d\n",pkt.pos_i);
     fprintf(stderr,"Packet 1.pos_j is     %d\n",pkt.pos_j);
     fprintf(stderr,"Input String Length is %d, Characters in Product is %d\n",
                                strlen(text_data), num_chars);
     fprintf(stderr,"Output Character Data is:\n'");
     for(i = 0; i < num_chars; i++)
         fprintf(stderr, "%c", output_text[i]);
     fprintf(stderr,"'\n");

  }


}



/*********************************************************************/
void finish_prod_desc_block(char *outbuffer, short elev_index,
                            short elevation, int prod_len) {

  /* complete entering values into the product description block (pdb)
     Enter: threshold levels, product dependent parameters, version and
     block offsets for symbology, graphic and tabular attribute.       */

  /* cast the msg header and pdb block to the pointer hdr              */
  Graphic_product* hdr=(Graphic_product*)outbuffer;
```

```
  /* enter the data threshold values                                    */
  hdr->level_1 = (short)0;
  hdr->level_2 = (short)0;
  hdr->level_3 = (short)0;
  hdr->level_4 = (short)0;
  hdr->level_5 = (short)0;
  hdr->level_6 = (short)0;
  hdr->level_7 = (short)0;
  hdr->level_8 = (short)0;
  hdr->level_9 = (short)0;
  hdr->level_10 = (short)0;
  hdr->level_11 = (short)0;
  hdr->level_12 = (short)0;
  hdr->level_13 = (short)0;
  hdr->level_14 = (short)0;
  hdr->level_15 = (short)0;
  hdr->level_16 = (short)0;

  /* product dependent parameters
   * these will follow the base products where
   * halfword 30 (pdp3)=elev angle
   */
  hdr->param_3 = (short)elevation;


  /* number of blocks in product = 3                                    */
  hdr->n_blocks=(short)3;

  /* message length                                                     */
  /* BUILD 6 LINUX change                                               */
  RPGC_set_product_int( (void *) &hdr->msg_len,
                        (unsigned int) prod_len );

  /* ICD block offsets                                                  */

  RPGC_set_product_int( (void *) &hdr->sym_off, 60 );
  RPGC_set_product_int( (void *) &hdr->gra_off, 0 );
  RPGC_set_product_int( (void *) &hdr->tab_off, 0 );

  /* elevation index                                                    */
  hdr->elev_ind=(short)elev_index;

  return;

} /* end finish_prod_desc_block */



/************************************************************************/
void clear_buffer(char *buffer, int bufsize) {
  /* zero out the input buffer */
  int i;

  for(i = 0; i < bufsize; i++)
    buffer[i] = 0;

  return;

}
```

```
/*
 * RCS info
 * $Author$
 * $Locker$
 * $Date$
 * $Id$
 * $Revision$
 * $State$
 */

/************************************************************************
Module:         sample4_t2.h

Authors:        Steve Smith, Software Engineer, NWS ROC
                    steve.smith@noaa.nssl.gov
                Tom Ganger, Systems Engineer,  Noblis Inc.
                    tganger@noblis.org
                Version 1.0, January 2002

Version 1.1   April 2005   T. Ganger
              Modified to use new product ID's

Version 1.2   April 2006   T. Ganger
              Added a packet 1 header structure and a symbology block
                   header structure and included product.h.
              Added two new external print diagnostic functions.

Version 1.3   August 2007  T. Ganger
              Separated first layer header from the symbology block
                 structure for clarity
              Added defined offsets

Version 1.4   February 2008    T. Ganger  (Sample Algorithm ver 1.18)
              Replaced C++ style comments using '//' for ANSI compliance.
              Created unique file names with respect to other algorithms.
              Eliminated definition of output buffer ids.

$Id$
************************************************************************/

#ifndef _SAMPLE4_T2_H_
#define _SAMPLE4_T2_H_

/* ORPG   includes ------------------------------------------------------*/
#include <rpg_globals.h>

/*** the following are included in rpg_globals.h */
/***                                             */
/***  #include <stdio.h>                         */
/***  #include <stdlib.h>                        */
/***  #include <time.h>                          */
/***  #include <ctype.h>                         */
/***  #include <string.h>                        */
/***                                             */
/***  #include <a309.h>                          */
/***  #include <basedata.h>                      */
/***  #include <basedata_elev.h>                 */
/***  #include <rpg_port.h>                      */
```

```
/***  #include <prod_gen_msg.h>                  */
/***  #include <prod_request.h>                  */
/***  #include <prod_gen_msg.h>                  */
/***  #include <rpg_vcp.h>                       */
/***  #include <orpg.h>                          */
/***  #include <mrpg.h>                          */
/***********************************************/

/* required by rpgc after API enhancement patch */
#include <gen_stat_msg.h>
#include <product.h>

#include <rpgc.h>
#include "rpgcs.h"



/* local defines -----------------------------------------------------*/

/* the buffer id numbers are not predefined when using the new 'by_name'*/
/* functions, in the past the following would be defined in a309.h      */
/* when an algorithm was integrated into the operational system         */
/* #define SAMPLE4_IP1                 1998  */
/* #define SAMPLE4_IP2                 1997  */
/* #define SAMPLE4_FP1                 1993  */
/* #define SAMPLE4_FP2                 1994  */


/* estimated size for the products */
#define SAMPLE4_FP_SIZE           1000
#define SAMPLE4_IP_SIZE           1000



/* added Sample Alg 1.17 */
/* in bytes */
#define SYMB_OFFSET 120
#define SYMB_SIZE 10
#define LYR_HDR_SIZE 6
#define PKT_HDR_SIZE 8


/* symbology header block structure (header portion)  per ICD format    */
typedef struct {
  short divider;
  short block_id;
  int block_length;
  short num_layers;
/*  short divider2;  */
/*  int layer_length;*/
  } sym_block_hdr;

/* added Sample Alg 1.17 */
/* symbology data layer header                                          */
typedef struct {
  short divider2;    /* layer divider (always -1)                      */
  int layer_length;  /* length of first (upcoming) data layer in bytes  */
  } layer_hdr;
```

```
/* data packet 1 header structure (minus character data)                */
typedef struct
{
   short  code;         /* Packet code                                  */
   short  num_bytes;    /* Byte length not including self or packet code */
   short  pos_i;        /* I starting coordinate                        */
   short  pos_j;        /* J starting coordinate                        */

} pkt_1_hdr;



extern int print_message_header(char* buffer);

extern int print_pdb_header(char* buffer);

extern int print_symbology_header(char *c_ptr);

extern void product_to_disk(char * output_buf, int outlen, char *pname,
                            short ele_idx);

#endif
```