Radar Control Server Protocol for User Interface

UI-Server/Radar Control Server Protocol

- as implemented 7/15/2010 dab -

Contents:

- Contents
  - General Discussion
    - Tasks
    - Task Scheduling
    - Global Commands
  - Structures
  - Protocol Discussion
    - Task Definition Operations
    - Tasklist Operations
    - Radar Status Monitor updates

General Discussion

Tasks

Let's define a Task as one complete definition of what the radar, antenna, and processor are doing at any specific time. The Radar Control Server (RCS) maintains the following items for each Task:

- iwrf_scan_segment_t - defines scan pattern
- iwrf_ts_processing_t - defines signal processing
- start_time [optional] - indicates a time_t word when scan should begin (utc)
- repeat_cycle [optional] - if non-zero indicates scan should be restarted after specified seconds.
- priority [optional - set to 0 for sequential preferred scheduling]
- time_last_begun - time_t word when segment was last started

Except as noted, the UI protocol should provide a way for the user to update each of these items for each Task defined. Multiple users should each be kept updated as to the state of the radar and current set of known tasks.

The RCS will maintain (but not archive) information on an arbitrary number of tasks as created by UI "update" commands. Each command requires an identifying Task name which should be the same as the segment_name from the iwrf_scan_segment_t structure. Additionally, there should be a command to delete a Task. Note: the RCS was implemented by a program called syscon, whose source files are maintained in the git cdp.git repository under syscon.

Task Scheduling

In conventional radar operations, task scheduling has been conducted on a primarily sequential manner with the provision of tying certain Tasks to begin and possibly recycle based on the time of day clock. We want to allow a UI to provide this mode of operations. At the same time, we want to allow for priority based scheduling with UI's that support multiple active operators, and automated and/or unattended operational modes. For the initial web-based interface, designed to largely emulate the previous SPOL controller, the Task priorities should all be set to 0. This indicates that scheduling will be done by sequence as defined by a UI specified list of Tasks. When a UI-server connects to the RCS, the RCS will transmit update commands for all known tasks and the tasklist to bring the UI-server in-sync. Subsequently, the RCS will transmit updates to all connected UI-servers whenever there is a change in the tasks, current running task, radar status, etc.

Task List Operations:

- Update current Task list
- Delete a Task <name>
- Update next Task index (one-time over-ride of normal task sequence)
- Set Task index immediate (terminate current task, start new task now)
- Get the currently running Task index and Next Task index
- Update begin_time, repeat cycle for start of task list
- Remove All Tasks (from list)
- Remove one Task from list <index>
- Append Task to list <task name>
- Exchange List Entries <index1> <index2>

Not all UI concepts would need to implement all possible interactions. For example, a UI may allow the user to create more than one Task list. When the user selects one of the lists for use, the UI server would need to upload the desired tasklist to the RCS. Depending on what default action is desired, the UI might then set the 'next Task index' to 0 to assure that the first scan of the new sequence would start when the currently running volume terminates. The tasklist contains a schedule structure which allows a start time and recycle time for the start of the tasklist. This is probably the best way to do list oriented scheduling. The individual tasks also contain a schedule structure which is provided for priority based scheduling. In any case, if the schedule blocks are never set or are cleared to zero's, the tasks will be in a free-running mode.

Global Commands

There are a few cases where we need to set switches/modes that affect all activity.

- Globally enable/disable archiving
• Stop all scans - cancels all timers
• Stop current scan - leave begin timers active
• Report an error
• Get overall status - ?
• Operator/ui_server Authentication ?

These global items are a bit sketchy at this point. We did discuss the desirability of having a way to globally disable all archiving. We may want a similar switch to stop all scanning. Another way to stop scanning would be to delete the tasklist. If one wanted to do say one ppi scan, then stop scanning and wait for a timer to elapse, and then repeat the ppi, you could put an 'idle' mode task (this is one of the scan modes) after the ppi scan. Get overall status - the user will be getting the radar status monitoring packets which will cover perhaps all status monitoring needs, but if needed, we could provide some global status bits. The operator authentication is something I’ve looked at, but am still thinking about. Report an error - The RCS may need to report an error or warning if tasks are not completely defined before use, or if the operator waits until it’s too late to schedule as scan. (Although these could be detected in the UI_server).

Structures

Here are some structures and definitions used in the protocol (These have been entered into CHILL repository: cdp/include/iwrf_user_interface.h):

```c
/************************************************************/
/*\*
* \struct iwrf_ui_task_operations
*  * Main Struct to allow user interface control over radar operations.
*  */
\*************************************************************/

/* all commands from the UI to the RCS consist of the following structure: */
typedef struct iwrf_ui_task_operations
{
    iwrf_packet_info_t packet; /**< packet id = IWRF_UI_OPERATIONS */
    char task_name[IWRF_MAX_SEGMENT_NAME]; /**< req'd for Task Def operations */
    char owner_name[IWRF_UI_MAX_OPERATOR_NAME]; /**< originating operator name */
    iwrf_ui_opcodes_t op_code;
    ui32 padl; /**< structure alignment - */
    iwrf_ui_rcs_payloads_t un;
} iwrf_ui_task_operations_t;

/* the command payload is a union of the following options depending on the op-code in use */
typedef union iwrf_ui_rcs_payloads
{
    iwrf_scan_segment_t scan_segment;
    iwrf_ts_processing_t ts_processing;
    iwrf_ui_schedule_info_t schedule;
    iwrf_ui_name_element_t delete_task;
    iwrf_ui_tasklist_mod_t tasklist_modify;
    iwrf_ui_tasklist_full_t tasklist_full;
    iwrf_ui_global_t global_items;
    iwrf_ui_rcs_status_t rcs_status;
    rsm_pkt_t rsm_pkt;
} iwrf_ui_rcs_payloads_t;

/******************** iwrf_ui specific opcode payloads *******************/
typedef struct iwrf_ui_tasklist_full
{
    si32 num_list_items;
    iwrf_ui_name_element_t tasks[IWRF_UI_MAX_TASKS_PER_TASKLIST];
    iwrf_ui_schedule_info_t tasklist_schedule;
    si32 unused[32];
} iwrf_ui_tasklist_full_t;

/* task names structure */
typedef struct iwrf_ui_name_element
{
    char name[IWRF_MAX_SEGMENT_NAME];
} iwrf_ui_name_element_t;

    /* this struct is used to modify a tasklist in place */
typedef struct iwrf_ui_tasklist_mod
```
typedef struct iwrf_ui_tasklist_mod_t {
  char name[IWRF_MAX_SEGMENT_NAME]; /**< used by APPEND */
  si32 index1; /**< used by REMOVE, EXCHANGE, SET_INDEX op's */
  si32 index2; /**< used by EXCHANGE, GET_CURRENT_INDEX */
} iwrf_ui_tasklist_mod_t;

/* The GET_CURRENT_INDEX command returns the tasklist index to the currently
running scan in index1, and the index of the next scan in index2.
this struct provides scheduling items for either a Tasklist or an individual scan. */
typedef struct iwrf_ui_schedule_info {
  si64 begin_time_utc; /**< 0 to deactivate timer or valid begin time to set timer */
  si32 repeat_cycle_secs; /**< 0 or repeat cycle */
  si32 priority; /**< 0 or priority setting */
  si64 last_run_time_utc; /**< last time run (on get schedule operation) */
} iwrf_ui_schedule_info_t;
#define IWRF_UI_ENABLES_MASTER_RECORD_MASK 1
#define IWRF_UI_ENABLES_IMMEDIATE_SCAN_UPDATE_MASK 2
/* The iwrf_ui_global structure is used to pass info relating to all tasks to system controller. */
typedef struct iwrf_ui_global {
  char operator[IWRF_UI_MAX_OPERATOR_NAME]; /**< set by UI */
  si32 command; /**< global command, set by UI */
  si32 enables; /**< bit-mapped enables, set by UI */
} iwrf_ui_global_t;
/* The iwrf_ui_rcs_status_t is sent by the rcs as the payload of the IWRF_UI_RCS_STATUS command.
It is sent in response to a UI command when a control/command error is detected by the rcs.
In addition, the rcs will send the IWRF_UI_RCS_STATUS command to the UIs (with error=0)
when any of the misc_status bits changes. */
typedef struct iwrf_ui_rcs_status {
  char message[IWRF_UI_MAX_ERROR_MSG]; /**< ascii error message, set by system controller */
  si32 error; /**< error number, set by system controller */
  si32 misc_status; /**< status bits, set by system controller */
  ui32 az_status; /**< az motor status bits, see MS_... in deltatau.h */
  ui32 el_status; /**< el motor status bits, see MS_... in deltatau.h */
} iwrf_ui_rcs_status_t;
typedef enum iwrf_ui_error {
  IWRF_UI_ERROR_DELETE_FAILED = 1, /**< attempt to delete running scan task */
  IWRF_UI_ERROR_TASKLIST_SIZE, /**< illegal number of tasks in tasklist */
  IWRF_UI_ERROR_MISSING_TASK_NAME, /**< task name required, but missing */
  IWRF_UI_ERROR_TASKLIST_INDEX_RANGE, /**< task list index <0 or > max range */
  IWRF_UI_ERROR_TASKLIST_FULL, /**< can not append to tasklist */
  IWRF_UI_ERROR_APPEND_TASK_UNDEF, /**< attempt to append undefined task to tasklist */
  IWRF_UI_ERROR_APPEND_TASK_NAME, /**< no task name in APPEND request */
  IWRF_UI_ERROR_CANT_SCHEDULE, /**< begin time was not 15 seconds into the future */
  IWRF_UI_ERROR_REPEAT_CYCLE_RANGE, /**< schedule repeat cycle <0 or > 3600 secs */
  IWRF_UI_ERROR_TASKLIST_ENTRY_UNDEF, /**< tasklist contains unknown task name */
  IWRF_UI_ERROR_TASKLIST_ENTRY_UNDEF, /**< tasklist entry is not fully defined */
  IWRF_UI_ERROR_TASKLIST_TASK_NOT_IN_TASKLIST, /**< attempt to sched a task not in tasklist */
  IWRF_ANTCON_NOT_CONNECTED, /**< antenna controller not connected */
  IWRF_ANTENA_FAULTED, /**< antenna controller reports fault */
  IWRF_TXCTRL_NOT_CONNECTED, /**< txmit controller not connected */
  IWRF_SCAN_STATE_TIMEOUT, /**< scan state machine time in state limit reached */
  IWRF_SCAN_SEGMENT_LIST_ERROR, /**< scan segment fixed angle list is bad */
} iwrf_ui_error_t;
typedef enum iwrf_ui_opcodes {
  /* Task Definition */
  /* Opcode Payload Flow Dir. Payload */
  IWRF_UI_UPD_SCAN_SEGMENT = 1, /**< UI<>RCS iwrf_scan_segment_t, */
IWRF_UI_UPD_TS_PROCESSING, /**< UI><RCS iwrf_ts_processing_t */
IWRF_UI_UPD_DELETE_TASK, /**< UI><RCS iwrf_ui_name_element_t */
IWRF_UI_UPD_SEGMENT_SCHEDULE, /**< UI><RCS iwrf_ui_schedule_info_t */
IWRF_UI_GO_IDLE, /**< UI><RCS stop scanning (begin SYS_IDLE segment) */
/* last opcode allows for alternative scheduling methods */
/* Tasklist operations */
IWRF_UI_TASKLIST_UPD_LIST, /**< UI<>RCS iwrf_ui_tasklist_full */
IWRF_UI_TASKLIST_GET_UNUSED_LIST, /**< UI<-RCS user query, in response,
RCS sends iwrf_ui_tasklist_full containing unused task names */
IWRF_UI_TASKLIST_GET_CURRENT_INDEX, /**< UI<-RCS user query, in response,
RCS sends current and next task indexes in
iwrf_ui_tasklist_mod->index1&2 */
IWRF_UI_TASKLIST_UPD_NEXT_INDEX, /**< UI<>RCS iwrf_ui_tasklist_mod->index1
indexed task starts when current task ends */
IWRF_UI_TASKLIST_SET_INDEX_IMMEDIATE, /**< UI->RCS iwrf_ui_tasklist_mod->index1
indexed task starts immediately */
IWRF_UI_TASKLIST_UPD_LIST_SCHEDULE, /**< UI<>RCS iwrf_schedule_info_t */
IWRF_UI_TASKLIST_GET_LIST_SCHEDULE, /**< UI->RCS user query, in response,
RCS sends current iwrf_ui_tasklist_full + sched. */
IWRF_UI_TASKLIST_REMOVE_ALL, /**< UI<>RCS none */
IWRF_UI_TASKLIST_REMOVE_ONE, /**< UI<>RCS ...tasklist_mod->index1 */
IWRF_UI_TASKLIST_APPEND, /**< UI<>RCS ...tasklist_mod->name */
IWRF_UI_TASKLIST_EXCHANGE, /**< UI<>RCS ...tasklist_mod->index1&2 */
/* last three are to facilitate shared tasklist management */
/* Misc Global items */
IWRF_UI_UPD_GLOBAL, /**< UI->RCS iwrf_ui_global_t */
IWRF_UI_UPD_RSM_PACKET, /**< UI<->RCS rsm_pkt_t */
IWRF_UI_RCS_STATUS, /**< UI<->RCS iwrf_ui_rcs_status_t */
IWRF_UI_LAST
}

iwrf_ui_opcodes_t;

Some of the opcodes are to support ideas we have been discussing related to a shared operator responsibility. For example, in that environment it would be better to maintain the tasklist incrementally with ops: APPEND, REMOVE_ONE and EXCHANGE, rather than the TASKLIST_SET op.

Protocol Discussion

This protocol is a binary protocol transmitted with native byte-ordering on TCP socket connections between the User-interface (server) and the Radar Control Server(RCS). The RCS listens on a default port of 2515 at the IP address identified as radcon on the radar local network.

Task Definition Operations

The iwrf_ui_task_operations_t is the key structure used in the UI_server/RCS interactions to manage task definitions as well as execution via the task sequence list (tasklist). The structure begins with the standard iwrf_packet_info header, however, much of the packet info data can be ignored due to the fact that the interactions between user and server are transient and not recorded. At this time, the id, len_bytes, version number and radar_id should contain valid numbers.

Note that in the enum typedef iwrf_ui_opcodes defined above, many of the operations are tagged as updates (UPD). The first part of the comment defines the valid data flow directions. Most of the updates can be sent by either the UI-server or the RCS. However, a few of the opcodes make sense only in one direction.

Typical Data Flow:

- When a UI-server program connects to the RCS, the RCS will send update messages for all tasks and other objects known to the RCS. This will bring the UI-server in sync with the current operational setup.
- When the UI-server needs to update one of the structures, it sends the UPD message to the RCS. The RCS updates it's copy of the structure, and re-transmits the UPD message to all of the UI-server type programs currently connected.
- Potentially, there could be more than one active UI_server type program connected to the RCS. Hence, the UI_server should be prepared to receive unsolicited UPD commands with updated data. The UI_server should update its internal structures references by the unsolicited UPD packets from the RCS.
- During normal operations, the RCS will send updates to UI-servers as the task status changes. For example, when a new task begins, a IWRF_UI_TASKLIST_UPD_CURRENT_INDEX would be sent to indicate the tasklist index for the currently running task (in index1, and the index of the next task in index2).
- Another example would be when a repeating timer restarted the tasklist, the RCS would send a IWRF_UI_TASKLIST_UPD_LIST_SCHEDULE with the next start time filled in.

The iwrf_scan_segment_t and iwrf_ts_processing_t structures have been previously defined in iwrf_data.h. Note that iwrf_scan_segment_t contains the desired fixed angle list in it.

Tasklist Operations
The tasklist is used to begin radar scanning and establish a sequence of scans for the RCS to follow. The UI should first create all tasks needed before referring to them in the tasklist. Tasks are created using the IWRF_UI_UPD_SCAN_SEGMENT and IWRF_UI_UPD_TS_PROCESSING operations. I'd suggest that the UI use the same name for the task and the scan_segment to avoid confusion.

Initially, the tasklist is empty and the radar will not be scanning. If the tasklist is cleared while the radar is scanning, the radar will stop scanning at the end of the current volume. After doing a IWRF_UI_UPD_TASKLIST operation to establish the desired scan sequence, if the radar is not scanning, one would need to do a IWRF_UI_SET_INDEX_IMMEDIATE to start scanning at the desired point in the list, probably the start of list (index=0). When the UI-server issues the IWRF_UI_SET_INDEX_IMMEDIATE, The RCS will start the desired task, and send out IWRF_TASKLIST_UPD_CURRENT_INDEX to all UI-servers to indicate the change of current task. The UI can then maintain a display of the tasklist, with the currently running task highlighted. The UI could also indicate the expected next task. Normally, the next task is just the next task in the tasklist, but if the operator issues a IWRF_UI_TASKLIST_UPD_NEXT_INDEX to do a (one-time) alteration of normal task sequence, the UI could also update its next task indicator.

If the IWRF_UI_UPD_TASKLIST_SCHEDULE operation is not done, (or if the begin_time_utc is set to 0) the tasklist will default to free-running operation. If the user wishes to pause the scanning to stop the antenna and data collection while waiting for either a timer or for the operator to resume scanning, the task list can end with the “idle” task.

Tasks which have been created, but are not in the current tasklist, are kept in the unused task list. The RCS sends out a IWRF_UI_TASKLIST_GET_UNUSED_LIST with the currently unused list whenever that list changes. The UI-server may also request the current list of unused tasks by issuing a IWRF_UI_TASKLIST_GET_UNUSED_LIST operation. Tasks may be deleted by name with the IWRF_UI_DELETE_TASK command, but tasks should be removed from the tasklist before they are deleted.

Radar Status Monitor updates

An additional job for the RCS is to pass selected Radar Status Monitor (RSM) packets to the UI server. When an RSM packet is received, the RCS will embed it into an UI operation packet with the IWRF_UI_UPD_RSM_PACKET op-code and send it to all connected UI-servers. The RSM packets all begin with an rsm_pkt_t header as defined in cdp/rsm/rsm.h. This is followed one of the various payloads depending on the source of the status. It may be desirable for the RCS to limit the update rate on the the antenna angle packets, but most status sources all update packets could be passed along. Each source provides summary info every ~2 seconds and detailed information on request. We could allow the UI to request the detailed information on a sub-system as needed.