APS runControl Library

Claude Saunders, Michael Borland

Table of Contents

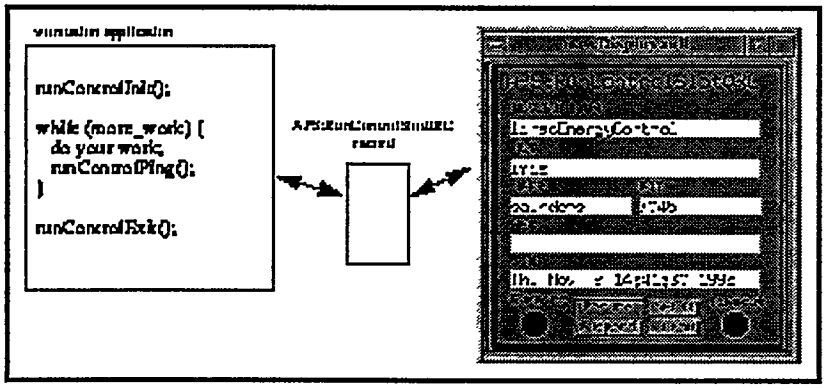
Operations Analysis Group
Accelerator Systems Division
Argonne National Laboratory
Oct. 25, 1995

1. Introduction

This document serves as a User's Manual and Reference for the runControl library.

This library is designed to be used by closed–loop EPICS control applications which are generally run in the background on the controls workstations. It permits an application to "register" itself with an EPICS record, thereby preventing additional instances of the same application from being run. In addition, the executing application may in turn be suspended or aborted via an MEDM control screen or other standard channel access client.

The process is roughly as follows:



This library is "cooperative" in nature, in that each application must select and use a unique application description string, and act on the library return codes properly.

2. Sample Application

The runControl library is a set of three C functions for communicating with an EPICS runcontrol record. Here is a simple program depicting proper use of the library.

```
#include <stdio.h>
#include <string.h>
#include <cadef.h>
#include <ezca.h>
#include <libruncontrol.h>
main(int argc, char *argv[])
{
  char pv[255];
  char handle[255];
  int status;
  if (argc < 2) {
     fprintf(stderr,"usage: %s <description-string>\n\n",argv[0]);
     exit(1);
  }

  /* NULL means the next available runcontrol record should be used.
     argv[1] is the unique application description string.
     A ping timeout of 2000 milliseconds will be used.
     The handle for subsequent calls is returned in the last argument.
  */
  status = runControlInit(NULL, argv[1], 2000.0, handle);
  if (status != RUNCONTROL_OK) {
    fprintf(stderr,"ERROR initializing run control\n");
    exit(1);
  }
  while (1) {
    /* Note: application may suspend inside the runControlPing() call */
    status = runControlPing(handle);
    switch (status) {
    case RUNCONTROL_ABORT:
      fprintf(stderr,"Application aborted\n");
      exit(1);
      break;
    case RUNCONTROL_TIMEOUT:
      fprintf(stderr,"Application timed out\n");
      exit(1);
      break;
    case RUNCONTROL_OK:
      /* do your application work here, but don't take more than 2000 ms */
      sleep(1);
      /* Send some useful status message and set alarm severity if you wish. */
      status = runControlLogMessage(handle, "informative message", NO_ALARM);
      if (status != RUNCONTROL_OK) {
         fprintf(stderr,"Unable to write status message and alarm severity\n");
         exit(1);
```

```
      }
      break;
    case RUNCONTROL_ERROR:
      fprintf(stderr,"Communications error with runcontrol record\n");
      exit(1);
      break;
    default:
      fprintf(stderr,"Unknown error code\n");
      break;
    }
  }
  /* When loop above is done, exit gracefully. May have to do this as part
     of a kill signal handler if that is how you stop your application.
  */
  status = runControlExit(handle);
  if (status != RUNCONTROL_OK) {
    fprintf(stderr,"ERROR during exit run control\n");
    exit(1);
  }
  exit(0);
}
```

3. Compiling

The runcontrol library may be accessed directly in a compilation with:

–I/usr/local/oag/apps/include

–L/usr/local/oag/apps/lib/sun4 –lrunControl

Note that you must also link in ezca and ca (which is likely the case already).

4. Library Reference

#include <libruncontrol.h>

runControlInit

int runControlInit(char *pv, char *desc, float timeout, char *handle);

Grab control of the specified EPICS runcontrol record, and load it with various application information, such as process–id, hostname, username, and start time. A specific record may be given via the pv argument, or NULL may be given and the next free runcontrol record will be found for you.

- **pv** – ptr to null terminated string containing pv name, not to exceed 39 characters, or NULL if you want library to find next free runcontrol record for you.
- **desc** – ptr to null terminated string (<= 39 chars) which identifies your application.

- **timeout** – timeout interval for ping in milliseconds. Your application must call runControlPing periodically within this interval, or the runcontrol record will timeout, and your application will have to exit.
- **handle** – ptr to zero length, preallocated string of 255 bytes. Function will copy in an identifier which is to be used in the other library calls.

Returns:

- **RUNCONTROL_OK** – application is registered, proceed
- **RUNCONTROL_DENIED** – another application (or application instance) is using the same runcontrol record, or has the same description string.
- **RUNCONTROL_ERROR** – unable to communicate with runcontrol record

runControlPing

int runControlPing(char *handle);

Notifies runcontrol record that you application is still alive. Also provides a means for runcontrol record to suspend you, or request that you abort. The return codes from this call must be checked and acted on properly.

- **handle** – ptr to string initialized by runControlInit() call.

Returns:

- **RUNCONTROL_OK** – all is well, continue
- **RUNCONTROL_ABORT** – your application should clean up and exit
- **RUNCONTROL_TIMEOUT** – your application didn't ping the record within the timeout interval and should clean up and exit.
- **RUNCONTROL_ERROR** – unable to communicate with record, you should attempt another runControlInit(), or exit.

runControlExit

int runControlExit(char *handle);

Release control of the runcontrol record.

- **handle** – ptr to string initialized by runControlInit() call.

Returns:

- **RUNCONTROL_OK** – all is well
- **RUNCONTROL_ERROR** – unable to communicate with record

runControlLogMessage

int runControlLogMessage(char *handle, char *message, short severity);

Log a message and new alarm severity to the runcontrol record. The runcontrol record will enter given alarm state.

- **handle** – ptr to string initialized by runControlInit() call.
- **message** – ptr to null terminated string (<= 39 chars) with status message.
- **severity** – NO_ALARM, MINOR_ALARM, MAJOR_ALARM, INVALID_ALARM

Returns:

- **RUNCONTROL_OK** – all is well
- **RUNCONTROL_ERROR** – unable to communicate with record