

ISC (aka Interlock Status Controller) commands sent by CONTROL:

(Initialize commands)

echo off

sstatus

istatus

pstatus

(sent once a minute)

temp

(sent when needed)

calibrate sky

calibrate hot

(would send 8 channels of A/C converter values, but none are connected)

analog

These commands are sent to the ISC over a serial port and should be terminated with a `\r` character.

The CONTROL program code declares a time out if no response is received within 5 seconds of issuing these commands. Since the CONTROL program can't be blocked waiting for the response, it's possible for several responses to be pending. The count of pending responses and the corresponding commands are stored in variables `num_isc_pend` and `pending()` (If this was done instead in a program dedicated to reading out the ISC, it could wait up to 5 seconds for the response and not issue any other commands until a response was received. This would simplify the code and make the `pending()` array unnecessary.)

The routine that reads the input should accumulate characters as they come in, stripping off any leading `\r` and `\n` characters. When it sees a `\r\n` sequence it should call the `intrepret` function below with a string that is the characters received up to but not including the `\r\n`. Since the response may be several lines long, the result may not be stored completely until the explanation point character is seen.

The array `status_var()` stores the character strings that are reported for the values of the various ISC variables. The names the ISC uses for the variables are stored in array `name_stat()`.

The value strings that could be reported for the ISC variables are stored in array `name_state()`. Numerical values are also assigned to each of these value strings. The numerical state of the ISC variables is stored in the array `isc_status()`. The array elements of `isc_status` are each equivalenced to simple variables with suggestive names. For example the "calibrate" variable (which corresponds to the position of the calibration vane) might be reported to have the value, "at hot". This string would be stored in the corresponding `status_var()` array element. The corresponding `isc_status()` element (which is equivalenced to the variable, `isc_vane`) would be given the numerical value,

AT_HOT. Thus some code that wanted to test the position of the vane could simply test

```
if (isc_vane == AT_HOT)
```

The `isc_status()` array only corresponds to the first 10 ISC variables, which are the only ones that can have one of the `name_state()` values. The `interpret` function translates the `status_var()` strings into numerical values for these other ISC variables, too:

Array `shelter_status(4)` holds the 4 shelter status bytes.

Array `vane_temp(6)` holds the 6 vane temperatures reported.

Array `isc_volts(8)` holds the 8 analog input values

Array `interlock(32)` holds the status of the various interlocks (0 or 1).

The `istatus` command's response is the names of those interlocks that are on. (The names are given at the end of the `name_stat()` array, beginning with "E STOP CONSOL".) The ones that are off are not mentioned. So the first thing the `interpret` function does is to set the `interlock()` array elements to -1. Then as it finds the names of the interlocks, it sets the corresponding array elements to 1. When the exclamation point that marks the end of the response is read, the array elements with the value -1 are set to 0. (So a routine interpreting the `interlock()` array in global memory should take 0 to mean Off, 1 to mean On, and -1 to be Undefined.)