An Extended IBCON for the FS

Mikael Lerner — Onsala Space Observatory — 2013–02–13

Revised 2021–05–31 — adapted for new syntax that allows the selection of the character used as command separator

Summary

Since February 2013 we are using an extended version of IBCON that has been developed locally at Onsala Space Observatory (OSO). In addition to supporting communication with devices via a GPIB-card in the computer, this extended IBCON also supports communication with external devices connected via ethernet and using the TCP/IP protocol — either in the form of true network-based devices or as GPIB-devices that are connected to ethernet via ethernet-to-GPIB converters (e.g. the conversion boxes sold by Prologix). Basically this extension adds the capability to IBCON to talk to a network-based device via standard TCP/IP sockets. We offer this software to other VLBI stations that are interested in switching over from GPIB-cards to network-based communication. Another possibility is that this extension of IBCON could be officially incorporated into the FS.

Introduction

There is an increasing trend that modern hardware devices come directly equipped for communication via ethernet. However, older devices which don't have these capabilities can still be adapted for use on ethernet with suitable GPIB-to-ethernet adapters (such as the *GPIB-Ethernet Controller* sold by *Prologix*). It is thus not necessary to have a GPIB-card equipped computer to communicate with GPIB-based devices.

At Onsala Space Observatory we have thus decided to get rid of the GPIB-card in the FS-computer and instead switch to TCP/IP-based ethernet communication. In our present VLBI-setup, we have two devices which have been connected using the GPIB-card: an *HP 5335A universal counter* used for cable delay measurements and an *HP 53131A universal counter* used for clock measurements. We also have a newer *Agilent 53220A universal frequency counter/timer* which also is going to be used for clock measurements; this latter device comes with a ethernet interface while the former ones use the GPIB-bus. In our new set-up, the two older devices have been equipped with one Prologix GPIB-Ethernet Controller each in order to turn them into independent network-based devices.

To handle the TCP/IP-based communication, we have replaced the standard IBCON program in the FS with a locally developed version that adds the capability to communicate with devices via sockets. The selection of how to communicate with a device is done via the configurations in the *ibad.ctl* control file. This version of IBCON allows you to mix and match between a GPIB-card, a Prologix-box controlling several devices on a common GPIB-bus, multiple Prologix-boxes each controlling a single GPIB-device, devices directly connected to the ethernet or any combination of these options.

Modifications to the FS

Our extended version of IBCON has been designed to fit together with the rest of the FS in such a way that it leaves a minimal impact on the rest of the FS. Apart from the obvious modifications to the IBCON program itself, the only other piece of FS code that needs to be modified is BOSS, which should be modified to allow for longer lines in the *ibad.ctl* configuration file.

For VLBI stations which continue using GPIB-cards, there would be no need for any changes to the *ibad.ctl* or any other file.

For VLBI stations which want to use network-based communication instead of or in addition to the usage of a GPIB-card, the appropriate modifications should be done to *ibad.ctl* and possibly to local procedures. The modified IBCON will not try to access any GPIB-card, if there are no devices registered to use it in *ibad.ctl*.

Implementation

The extended IBCON program consists of the standard FS IBCON program with an extra set of C-procedures added in three new files. The main program file *ibcon.f* has also been modified to include support of the new C-procedures. Of the three new files, one contains the procedures to parse network-based devices from the *ibad.ctl* configuration file, which is read and parsed by IBCON when the FS is started up. The second file contain procedures for communicating with the network-based devices when IBCON is called during FS operations. The third file is a library of socket handling procedures. The socket library was originally developed by Lars Pettersson (OSO), but it has been adapted for this application by Mikael Lerner (also OSO), who also have written the rest of the code.

It is also useful to increase the *ibuf* buffer and *ibadrd* constant in the BOSS program *rdtib.f* to allow for longer lines in the *ibad.ctl* file. This is important since configuration lines for network-based devices can be much longer than for GPIB-based devices.

Although we use the same control file (*ibad.ctl*) to configure network-based devices, it is important to notice that we use the configuration file in a different way for such devices. Traditionally, devices are specified with a two-letter mnemonic and a GPIB-bus address and there may also be GPIB-bus related keywords present in the file — this information is ignored by IBCON when dealing with network-based devices, since we do not use a GPIB-bus then. Instead, we need to add the IP-address, the port number to use as well as the command sequence to use when talking to the device. The inclusion of complete command sequences in *ibad.ctl* is new and the rationale for doing it is described further on. However, it changes the concept of the mnemonics from being an identifier for a device to instead being an identifier for a command or *an action*. The same device can thus have several mnemonics defined in *ibad.ctl* with each one defining a specific action. A device that we want to perform three actions with, for example reset, start a measurement and read out the result would thus have three mnemonics, one for each of these three actions.

There are several advantages with this system even if it does not conform with the original structure or use of IBCON and *ibad.ctl*. Firstly, it puts the details about the actual command sequences in one place where it is more likely to be seen only by the engineer that installed the device. If we instead would put them in the procedure files (as they traditionally have been done) they would be exposed to whoever is working with those procedures, which could be a scientist who doesn't bother about these kind of details

and not an engineer. It is more logically appealing to move low-level command sequences out of programs and procedure files and instead replace them with single, logical *actions*.

Secondly, by specifying a complete command sequence in the *ibad.ctl*, IBCON can automatically resend the complete sequence in case of a problem, which would not be possible if the commands are given one by one in a procedure.

Thirdly, allowing command sequences in *ibad.ctl* minimizes modifications needed by other parts of the FS. For example, without the command sequences, the built-in CABLE command could not be used with a Prologix-box without a local wrapper that adds the Prologix-specific commands needed. With the current implementation, the CABLE command can be used right out of the box.

Finally — and most importantly — by not programming any specific knowledge about the command sequences into IBCON, we don't need to worry about upgrades to IBCON if there are syntactical changes in the way commands should be processed. If Prologix or another company releases a new GPIB-ethernet controller with a completely different syntax, you don't have to wait for the next release of the FS and hope that it gets support there. You just modify the command sequence in *ibad.ctl* and you are good to go.

The IBCON program is started when the FS is started up. The *ibad.ctl* control file is then parsed. If a GPIB-card is going to be used, it will be configured at this point. Information about network-based devices are stored and is then used when the IBCON program is called to perform communication with them. A socket will then be opened to the specific device and the command sequence will be sent. If a problem occurs during the communication, the socket will be closed and reopened, after which a second attempt to send the command sequence will be made. The socket will always be closed after the communication is finished — this policy can be discussed since an alternative is to open the socket when IBCON is started up and leave the sockets permanently open. The current implementation was, however, dictated by the need at OSO to be able to communicate with the same device from two different FS computers running parallel sessions. (There seems to be no problems having two FS computers trying to access the same device at the same time — the Prologix-box seems to handle the requests sequentially.)

Syntax of the *ibad.ctl* control file

Network-controlled devices use the following syntax in the *ibad.ctl* file:

```
MN=net,IP-address,port,/command1[/command2[/command3...]]
```

where '`MN`' is the two-letter mnemonic used to identify the *action* for the device, "`net`" indicates that this is a network-based device, '`IP-address`' is the IP-address of the device or the Prologix-box given in "`a.b.c.d`" notation, '`port`' is the port number to use (port should be "`1234`" for Prologix-boxes but can be other things for other network-based devices — some network-based GPIB-devices seem to use port "`5025`", for example) and the command sequence is the set of commands to send when communicating with the device, which typically will include Prologix-commands used to set up the Prologix-box as well as the GPIB-command(s) to the device itself. The commands in the command sequence can contain any ASCII characters including spaces and commas except the character used as a *command separator*. You are free to select any character you want as a separator — at OSO we are typically using "`/`" and all examples in this document will be shown with "`/`" as the separator. The first character in the command sequence has

to be the command separator, even if the sequence only consists of one single command, since IBCON needs to now which character you have selected.

Two special commands are recognized: '`##`' means that the GPIB-command passed to IBCON at the time of execution should be inserted at that point in the sequence, while '`$$`' means that the device has generated a reply that should be read at this point in the sequence. There should only be one instance of '`##`' and/or '`$$`' per command sequence, if needed at all. Assuming we have defined the action "`CB`" with the general command sequence "`/##/$$`", we could then send different commands specified at run-time to read out different results, for example:

```
gps-fmout=cb,:read a?
gps-fmout=cb,:read b?
```

The normal use would be to specify fixed command sequences in *ibad.ctl*:

```
CB=net,192.16.6.10,5025,/:READ?/$$/:INIT:CONT ON
```

and use the mnemonic without any argument at run-time:

```
gps-fmout=cb
```

A '`&`' can be appended to any command; this command will then be followed automatically by the '`:SYST:ERR?`' command and any error reported will be written to the log. For example, the command sequence '`/*RST/:INIT:CONT ON&`' will consecutively send the following three commands: '`*RST`', '`:INIT:CONT ON`' and '`:SYST:ERR?`', then wait for the reply and report an error if the reply is not '`+0,"No error"`'. Note that any error reported by the device will only be written to the FS log — IBCON will not act on them or try to resend the command sequence. This behaviour could be discussed, the current implementation was selected to offer a debug possibility for an intermittent communication problem. The '`&`' does not interfere with the '`$$`' command; thus the command sequence '`/:READ?/$$&`' is safe to use.

Different commands can be sent to the same device by giving them different mnemonics, and for network-controlled devices the mnemonics should be seen as symbols for different *actions* instead of different devices.

Examples of usage

At Onsala Space Observatory, we use two HP-devices for CABLE and CLOCK measurements, both being controlled by one individual Prologix-box each. The *ibad.ctl* file looks like this:

```
CA=net,192.16.6.15,1234,/++auto 0/++addr 2/++read_tmo_ms 1000/++read 10/$$
CB=net,192.16.6.16,1234,/++addr 3/++auto 1/:READ?/$$/++auto 0/:INIT:CONT ON
```

The first unit (an *HP 5335A*) has a nasty habit of sending messages as soon as we tell the Prologix-box to address it with the '`++addr 2`' command. The limited memory of the Prologix-box will then quickly fill up and the box will hang. It is thus of utter importance that the '`++auto 0`' command is sent **before** the addressing command to make sure that the Prologix-box will ignore the output from the device. We then set up a time-out with '`++read_tmo_ms 1000`' and ask the Prologix-box to grab one of the messages with '`++read 10`'. Once we have the message, we use the special command '`$$`' to tell IBCON to pick up the message from the Prologix-box and pass it on to the calling program.

The second unit (an *HP 53131A*) is more well-behaved so we can use a more standard way of talking to it. We start by selecting the address of the device with '`++addr 3`' and then we use '`++auto 1`' to tell the Prologix-box that we are going to send a command which will produce a response from the device that we want to obtain. We then send the read command to the unit with '`:READ?`' and tell IBCON to pick up the answer with '`$$`'. At this point we are not yet finished since the read command switched the *HP 53131A* from making continuous measurements to just make a measurement when told, so we want to restore the continuous measurements. We do that by first telling the Prologix-box that we now will send a command that won't produce any response ('`++auto 0`') before we send the actual reconfiguration command to the device ('`:INIT:CONT ON`').

We also have another device (an Agilent 53220A) that can be used for CLOCK measurements. It uses a similar syntax but it is connected directly to the ethernet. We can control that one with the following *ibad.ctl* line which includes some error checking:

```
CX=net,192.16.6.17,5025,/:INIT:CONT OFF&/:READ?/$$/:INIT:CONT ON&
```

This device can be used with a syntax that is very similar to the one used for the *HP 53131A* without the Prologix-box specific commands. However, a difference seems to be that it is important to switch from continuous measurements to single measurements before doing one, and we thus have to start with the '`:INIT:CONT OFF&`' command. The "`&`" at the end will make IBCON send a '`:SYST:ERR?`' command and report any error to the log — any error message will, however, not stop IBCON from continuing, so the read command is then going to be sent as in the previous example. The '`:INIT:CONT ON&`' again uses the "`&`" to get a warning in case of errors.

The following example shows four mnemonics to talk to the same device connected with a Prologix-box: the first one is a standard read request, the second one is a reset command and the last two ones are for general commands and questions, respectively.

```
CB=net,192.16.6.16,1234,/++addr 3/++auto 1/:READ?/$$/++auto 0/:INIT:CONT ON
CC=net,192.16.6.16,1234,/++addr 3/++auto 0/*RST
CD=net,192.16.6.16,1234,/++addr 3/++auto 0/##
CE=net,192.16.6.16,1234,/++addr 3/++auto 1/##/$$
```

At OSO, we earlier used the following station procedure for clock measurements:

```
define clock 00000000000x
gps-fmout=cb,:read?
gps-fmout=cb
gps-fmout=cb,:init:cont on
enddef
```

This procedure is now replaced with the following one, relying on the command sequence shown above which is programmed into *ibad.ctl*:

```
define clock 00000000000x
gps-fmout=cb
enddef
```

Note that the old procedure actually still could be used with the new *ibad.ctl*. It would, however, give us three consecutive measurements, since we will get a measurement each time the mnemonic "`cb`" is called, regardless of what kind of argument we pass along (since we defined "`CB`" to use a fix command sequence and ignore any run-time arguments).

If there really is a need to specify the command sequence command by command, this is also possible (but less elegant) and with the above definitions we could use the following procedure:

```
define clock 00000000000x
gps-fmout=ce,:read?
gps-fmout=cd,:init:cont on
enddef
```

In this case the sending of a read command and the actual read-out will still be an atomic process.

Conclusion

An extended version of IBCON is now in use at Onsala Space Observatory. Based upon the standard IBCON in the FS, the extended version also allows communication with network-based devices either connected directly to ethernet or connected via GPIB-to-ethernet converters (for example Prologix-converters). This version is available for other VLBI stations which want to use network-based communication and could also be included in a future release of the FS, since it won't require any modifications for VLBI stations which continue to communicate via a GPIB-card.