

Download DABBEL Changes

Problem analysis

Mon, Apr 10, 2000

Changes made by the Acnet database entry program are sent to the front-end nodes so that the changes can be reflected in a front-end database. This note discusses a problem that may occur when a device is deleted by changing its Acnet source node.

Server logic

It is common to use a server node when communicating with Acnet devices. Each device has an associated source node that is specified during DABBEL device input. When the post-processing of the DABBEL audit file sends database change commands to the front-end that is the source node, that front-end act as a server node and decide where to send the database data. It learns that from an SDR entry that includes the real target node and target analog channel number in that node. The server front-end then forwards the command to the real target node, which makes the changes locally, returning any error status to the server node, which then returns proper status to the host. If there is no response coming back to the server node, then the server node returns a time-out status to the host.

Switch to new source node

What happens when the source node is modified by DABBEL input? In this case, the device will have to be deleted from the original front-end database and added to the new front-end database. But there are more than two front-ends involved in this operation, because there is a target front-end as well. If the target node is the same, but only the source node changes, what then?

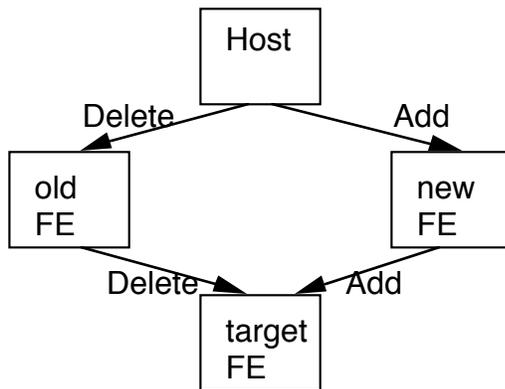
A queue is maintained of commands to be sent to each front end that can be a source node. This means that one front-end that may be down will not impede the updating of other front end databases. When the front-end comes back up, the next post-DABBEL downloading will presumably bring that front-end back up-to-date with database changes. (It won't matter that the front end was not up-to-date while it was down, because it wasn't running, anyway.)

The problem

A potential problem arises when only changing the source node for a device. A delete command will be placed into the old front-end's queue, and an add command will be placed into the new front-end's queue. But since the same target node is used in both cases—we are assuming that the target node was not changed—there is an implicit sequence that is required to make it all work.

It is necessary to first delete the device as seen via the old front-end before adding the device to the new front end. This is because the device is not in either front-end, really; it is actually in the target node that is reachable via both new and old front end. If the add to the new front-end, operating via an independent queue, should be processed before the delete of the old front end, the real database information will first be modified in the target node by way of the new front end; then it will be deleted in the (same) target node by way of the old front-end.

Although the order is important between processing of the two queues, it is counter-intuitive when one does not recognize the existence of the common target node. Here is a diagram that indicates the flow of downloaded information:



When a Delete command is sent from a host, the front-end receiving it checks to see whether the device name is present in its own local database. If it is, then it processes the command itself, and the device is deleted. If it is not found in its local database, it forwards the command to the "broadcast" node, which is normally a multicast destination that should reach all nodes of the same project—the project with which that front-end is associated.

When the forwarded command is received by a node, it of course searches its database to see whether the device is present. If it is found, the node processes the command and returns status to the node who forwarded it. If it is not found, *and the command was sent via multicast*, then it does nothing. If a Delete command is sent from a host to a node that cannot find the device name in its own database, and the forwarded command does not reach any node that can find it within its own database, then no response is returned, and the server node will return a time-out error status. This behavior is quite normal, because of the possibility that the target node is down. The local databases are distributed; each front end does not know within itself of the existence of devices that reside in other front-ends. It also does not know of the existence of devices that were *formerly* a part of its own database.