PBIO Format Server Administration Guide

Greg Eisenhauer

1 Format server concepts

All PBIO/JIX programs that use `IOContext` values communicate with a third-party *format server*. This format server acts as a repository for the format information that describes JIX-encoded messages and issues the format tokens that are prepended to messages in the JIX scheme. Typical JIX operation is shown in Figure 1. The format server may run anywhere in the reachable network. When encoding messages applications perform the following basic steps: In format registration to obtain the format token:

1. The local format cache is checked to see if the format being registered is already known.[1] If the format is known, the format token is taken from the known format.
2. If there is no current connection to the format server, a TCP/IP connection is established.
3. An marshalled version of the format information, including full field lists and native architecture information, is sent to the format server.
4. The application waits on a `read()` call for the return message from the format server. The return message contains the assigned format token.

[1]I.E. it has already been registered or an identical format is in the cache for other reasons. For formats to be identical all field list entries must be the same, optional information (such as XML representation) must be identical, and the characteristics of the registering architecture (pointer size, byte order, floating point format, etc.) must be the same.

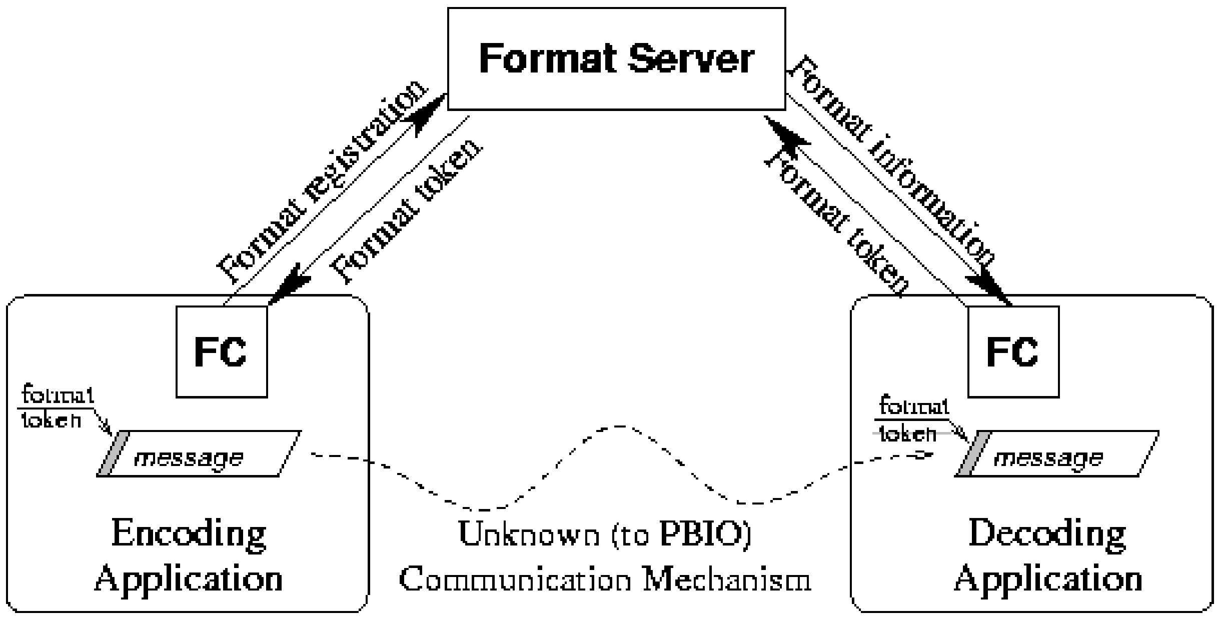

Figure 1: Typical JIX operation.

5. The format information and format token are entered into the local cache.
6. The connection to the format server is left open for possible reuse. The format server may close it asynchronously.

At this point, the format is available for use on the encoding side. During encoding, the format token is associated with encoded data.

In the decoding application format information retrieved from the format server, typically when the incoming buffer is passed to `get_format_IOcontext()` or similar call. This process follows steps similar to those in the encoding application, specifically:

1. The local format cache is checked to see if the format token in the incoming message is already known.[2] If the format token is known, the format information associated with that token is used.
2. If there is no current connection to the format server, a TCP/IP connection is established.
3. The format token is sent to the format server.
4. The application waits on a `read()` call for the return message from the format server. The return message either contains the full format information that was registered by the encoding application, or it indicates that the format token is not known. Among other reasons, the latter situation might result from attempting to decode a corrupt message, the encoding and decoding applications using different format servers, or the format server being restarted since the message was encoded.
5. If format information is retrieved from the format server, the information and format token are entered into the local cache.
6. The connection to the format server is left open for possible reuse. The format server may close it asynchronously.

2 Behavior of the format server

The format server is largely a simple database. It maintains a list of formats (the full description of a format) and format tokens and searches one or the other in response to application requests. The server listens on a well-known internet port (5347) and waits on messages on its open file descriptors with a `select()` call. It allows client applications to maintain connections, but closes them after they have been idle for one hour (`CONN_TIMEOUT_INTERVAL` in `format_server.c`). The format server will also close its oldest connection if it runs out of available file descriptors.

When the format server accepts a new connection, the connection is optionally verified against a *format service domain*. This is a simple security mechanism that allows a format server to limit its service to a subset of the Internet. The format service domain value is a colon-separated list of Internet domain names (such as “gatech.edu:gt.atl.ga.us”). This value is set at built time with the –with-format-server=value argument to configure, or at format server run-time via the `FORMAT_SERVICE_DOMAIN` environment variable. At client connection time, the format server will perform a reverse DNS hostname lookup on the connecting IP address. If the reverse DNS lookup is successful and the host falls within one of the service domains, the format server allows unconditional use. Otherwise the connecting IP address is added to a list and use is allowed only for a limited period. After that, connections from that IP address will be immediately closed. This domain validation can be disabled by defining `FORMAT_SERVICE_DOMAIN` as an empty string.

When registering previously unknown formats, the format server must assign a new format token. In the current implementation, format tokens are 10 bytes long and have the following format:

[2] I.E. a message of that format has been seen before or an identical format is in the cache for other reasons. In this case only the format tokens of known formats are compared.

```
typedef struct {
    unsigned char version;
    unsigned char salt;
    unsigned INT2 port;
    unsigned INT4 IP_addr;
    unsigned INT2 format_identifier;
} version_1_format_ID;
```

The 'version' field is the version of the format token and is '1' for this format. 'salt' is a random value determined at the time of format server startup. It is present simply to aid in the detection of improper or old format tokens. 'port' is the IP port that the format server is listening on (currently always 5347). 'IP_addr' is the IPv4 address of the machine upon which the format server is running. 'format_identifier' is a unique value assigned to each format registered with a particular server. That format_identifier is 16-bits limits the format server to 65536 distinct formats. The 'port' and 'IP_addr' values were added to the format token in anticipation of supporting multiple interacting format servers. That support has not yet been implemented, so these values are not "operational" in the sense that they cause PBIO/JIX to seek out the appropriate format server. Instead, the location of the format server is specified at build time for the PBIO/JIX library (via the –with-format-server=value argument to configure). That value can be overridden at run-time by specifying the `FORMAT_SERVER_HOST` environment variable. There is currently no mechanism for customizing the port that the format server is expected to listen upon.

During normal operation, the format server logs a minimum amount of information to a text file located at "/tmp/format_server_log".

3 Running the format server

The format server is designed so that it can be easily run automatically in the background by systems like the Unix cron facility. To this end, it is capable of automatically backgrounding itself and disassociating from the controlling terminal. When it is run, it first carefully checks to make sure that there is not already a format server running on its current machine at its assigned port. If there is, it exits immediately. If there is not it forks a copy of itself into the background. Most sites run the format server from cron every 10 minutes to protect against crashes or reboots.

The format server accepts the following command line arguments:

-no_fork causes the server to run in the foreground instead of forking into the background. This is primarily used for debugging and also turns out detailed debugging output.

-quiet suppresses normal startup printouts. This is useful when running the format server from cron.

-restart causes the format server to reload itself with format information stored in a checkpoint/restart file. The file's location is currently hardcoded to be "/tmp/FS_restart_file". (Triggering a checkpoint is discussed in the next section.)

4 Interacting with the format server

The format server has some capacity for remote management. The principal interface to this is the `format_cmd` program. format_cmd can be used to check to see if a format server is operational, to dump statistics about the format server's use, and to cause a checkpoint or restart operation, as specified below:

ping The command line "format_cmd ping" will result in output of the form "Format server marquesas.cc.gatech.edu is responsive." if the contacted format server is alive and responding. Otherwise the response will be of the form "Failed to contact format server on host marquesas.cc.gatech.edu".

checkpoint The command line "format_cmd checkpoint" will cause the format server to checkpoint its current list of formats and format tokens to the file "/tmp/FS_restart_file".

restart The command line “format_cmd restart” will cause the format server to read the format restart file located at “/tmp/FS_restart_file”. The format information there will be added to any formats that are currently known. Since this is a state-modifying command, this command is only accepted from clients on the same host as the format server.

stats The command line “format_cmd stats” will extract statistical information from the format server and dump it to the standard output. The information below is a sample of such output:

```
latte% format_cmd stats
Statistics for PBIO/JIX format server running on marquesas.cc.gatech.edu
    Server Up Since : Mon Sep 30 12:37:54 2002
    Known Format Count : 305
    Test Char Count : 31290
    Format Registration Count : 16920
    Format Fetch Count : 15402
    Client Count (since initiation) : 7140
    Clients Rejected : 98
    Current Client Count : 1
    Stats for client :
        Client Hostname  : latte.cc.gatech.edu
        Byte order different : false
        Provisional Use : false
        Server Protocol Version  : 2
        Connected Since : Mon Oct 14 15:47:46 200
        Bytes received from client : 1
        Bytes sent to client : 0
        Number of Formats Registered : 0
        Number of Formats Fetched : 0
```

5 Environment variables

Previous sections have discussed the `FORMAT_SERVER_HOST` environment variable that specifies the location of the format server and the `FORMAT_SERVICE_DOMAIN` variable that controls the internet domain freely serviced by the format server. In addition, the following environment variables impact PBIO/JIX operation in some way.

FORMAT_SERVER_PWD - When this variable is set to any value, the format server will substitute the current working directory for ’/tmp’ as the location of the format server restart file and the format server log.

FORMAT_SERVER_VERBOSE - When this variable is set for *client* applications (not the format server), those applications will generate verbose messages describing their interactions with the format server.

PBIO_FIXED_FORMAT_IDS - When this variable is set for *client* applications, they will request and use version-0-style 4-byte format tokens. The code that supports older-generation format tokens is no longer exercised during testing, so this option is deprecated.