Gateway Users Guide

Kenneth Evans, Jr.

September 2005

Advanced Photon Source
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

Table of Contents

Introduction

Overview

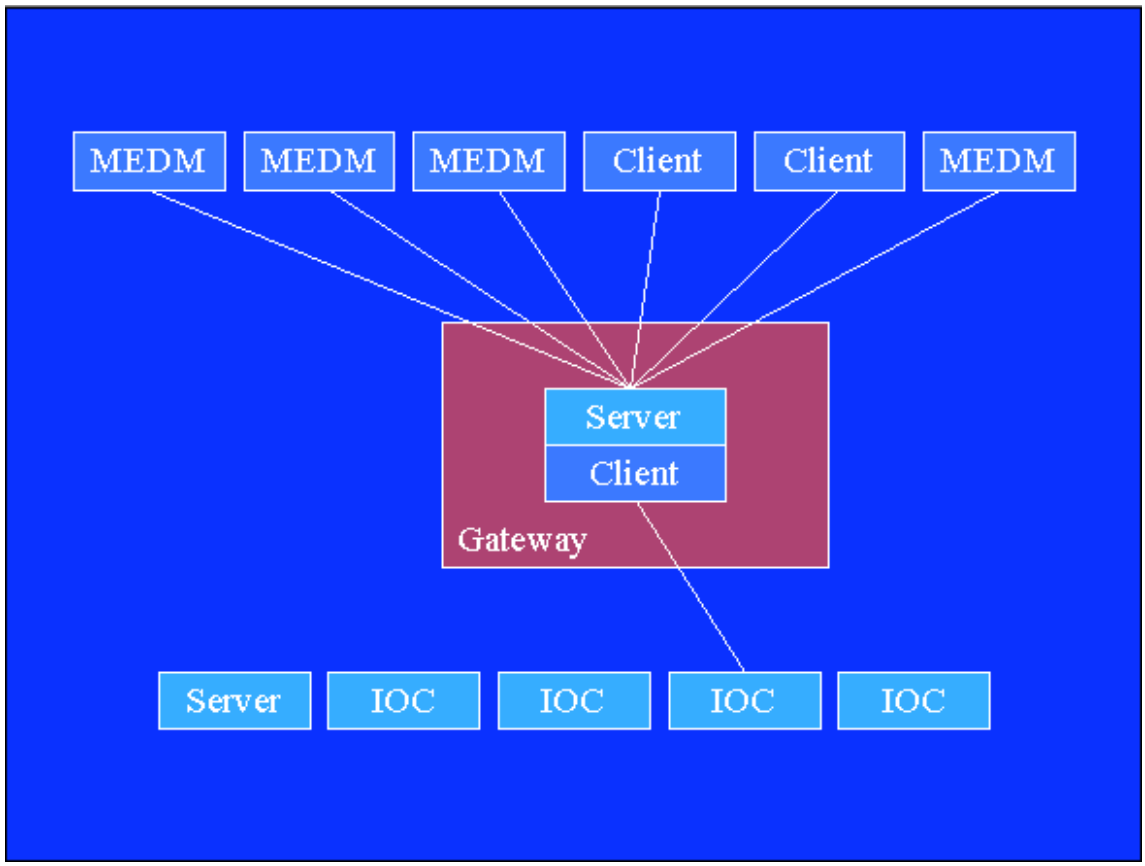
The Gateway is both a Channel Access Server and Channel Access Client that provides a means for many clients to access a process variable while making only one connection to the server that owns the process variable. It also provides additional access security beyond that on the server. It thus protects critical servers while providing possibly restricted access to needed process variables. The Gateway typically runs on a machine with multiple network cards, and the clients and the server may be on different subnets.

Previous versions of the Gateway worked with EPICS 3.13 but required a special version of base to build and

work correctly. The required changes were never incorporated into EPICS base. The latest of these versions is Gateway 1.3. The new version, starting as Gateway 2.0, only works well with EPICS 3.14.6 or later and does not otherwise require a special EPICS base. It also has additional features compared to Gateway 1.3. Note that even if you are using EPICS 3.13 for your other applications, you can still use Gateway 2.x built with EPICS 3.14 as your Gateway.

For this guide we will often refer to the clients as MEDM and the server as an IOC. This is (1) to avoid confusion with the client and server parts of the Gateway and (2) because MEDM and IOCs are the most typical and most used client and servers and most people are familiar with them. Keep in mind, however, that there are many other clients than MEDM and there are other types of servers than an IOC. The Gateway is, in fact, another type of server.

The following is a diagram of several MEDMs connected through the Gateway to a process variable in an IOC. Without the Gateway there would be a connection from each MEDM to the IOC. This uses up extra resources in the IOC and causes additional traffic.



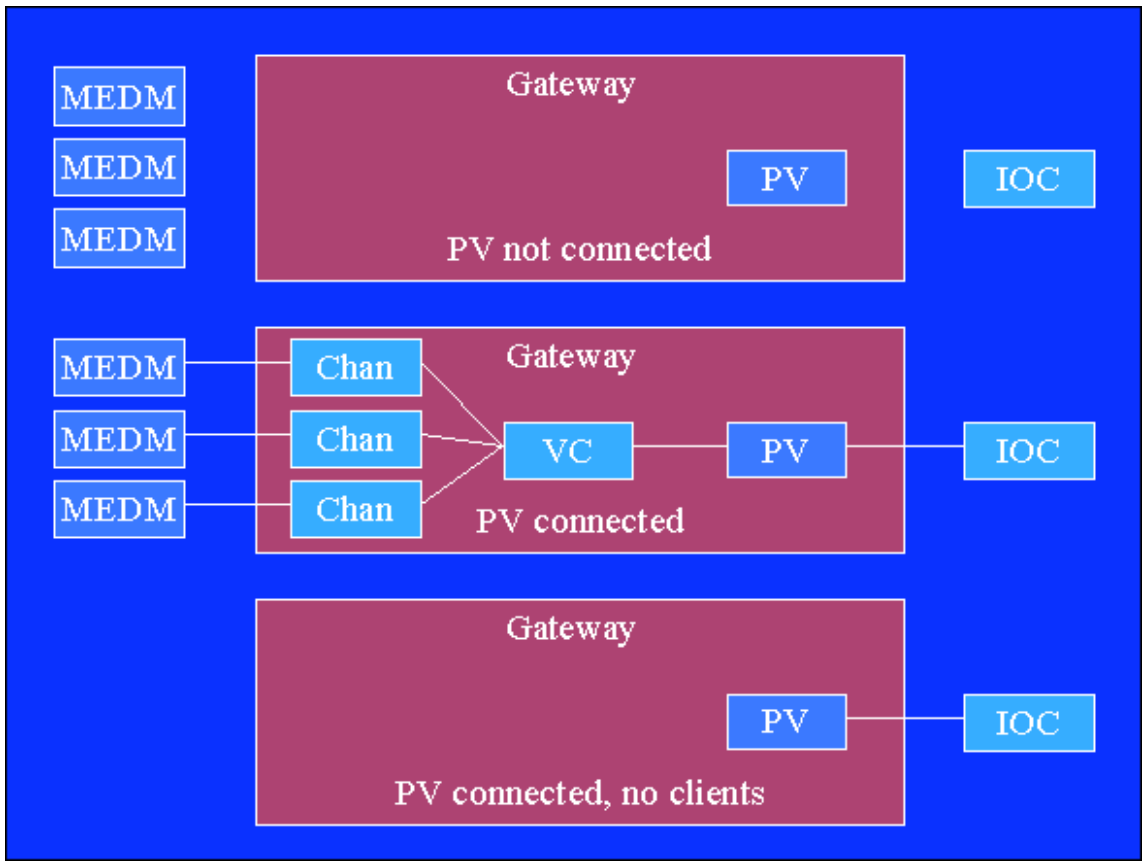
In addition, these MEDMs can be restricted in their access to the process variable. A typical scenario is to allow only read access. Note that the MEDMs see the Gateway as just another server, and the IOCs see the Gateway as just another client. The IOCs could be on a different subnet from the MEDMs, and in fact, the IOCs might not even be visible to the MEDMs without the Gateway. In the case where the IOCs and the MEDMs are on different subnets, the Gateway would be on a machine that has two network interface cards, one to the MEDM subnet and one to the IOC subnet.

At the APS the main Gateway is for people in the office building to connect to the machine network with read access only. There can be many users connected to popular process variables, such as the one for the beam current. There is only the one Gateway connection, however, to the IOC with that process variable. It is little affected by all these users and can go about its primary responsibility of helping keep the accelerator running. This Gateway typically has connections to on the order of 10,000-30,000 process variables in on the order of 300 IOCs. In addition the APS has one Gateway for each experimental team. These teams have read access to machine process variables and read/write access to their own process variables. There are on the order of 60 Gateways running at the APS.

In addition to connections to process variables in IOCs and other servers, the Gateway, as a server, provides its own internal process variables. These are used for diagnostic purposes, to initiate reports, and to control the

Gateway.

Internally the Gateway maintains two objects for each process variable. There is a PV (Process Variable) object that handles the Gateway client connections to the IOC, and there is a VC (Virtual Connection) object that handles the Gateway server connections. Each VC has a list of Chan (Channel) connections to the MEDMs. The PV object implements the client side of the Gateway, and the VC object and its Chans implement the server side. The following picture shows three typical states for these objects.



In the top state there is no connection to the IOC, either because it is still trying to connect or because the connection has been lost. In that case there is no VC, and the MEDMs see the process variable as not existing. In this case the PV object is destroyed after two minutes unless it connects. The middle state is most typical. There is a connection to the IOC, and each MEDM is connected through the VC. In the lower state the PV is connected to the IOC, but there are no MEDMs interested in the process variable. The PV object and connection to the IOC are maintained for two hours after the last time they are used in case another MEDM or other client wants to use it again. The other client could be a command-line program like caget, that requests the value and then exits. In that case the PV object will still stay around for at least two hours.

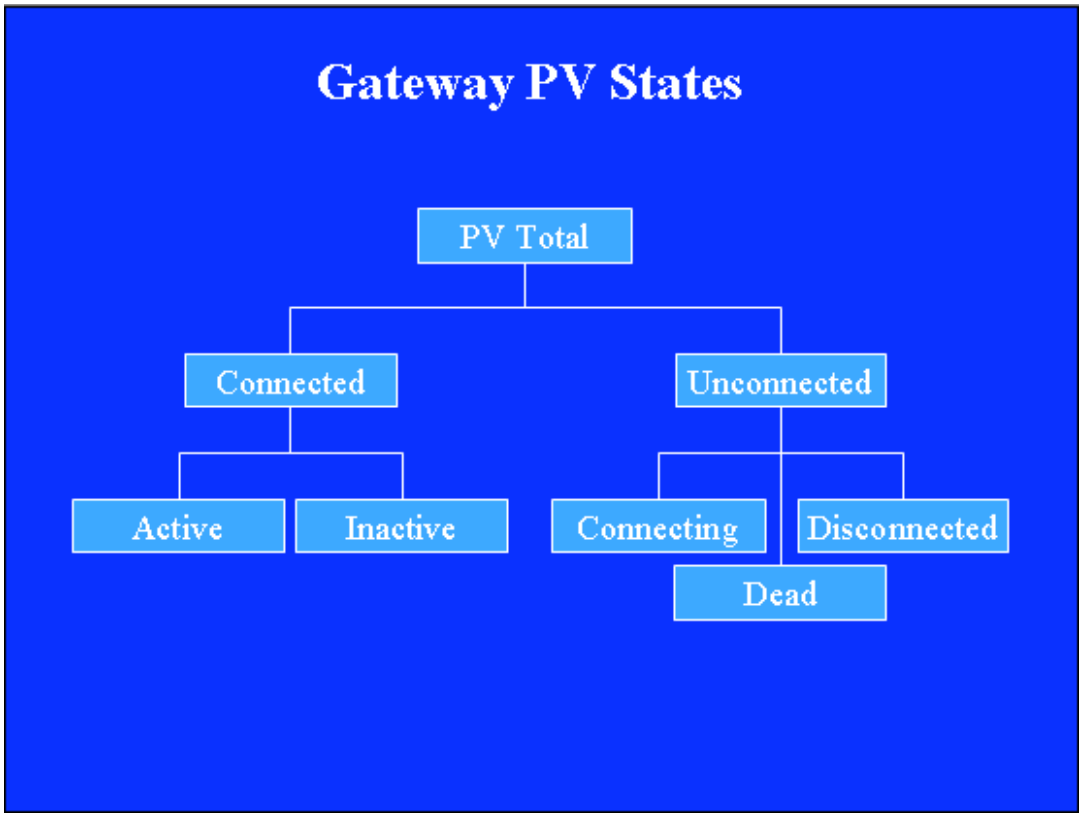
When a Channel Access Client wants to connect to a process variable, it send out a series of search requests, which are UDP packets, on the network. These packets are initially sent at a very fast rate with the time between them doubling at each try until 100 are sent or there is an affirmative reply. Each server that gets such a packet is required to determine if it has the process variable. When the Gateway gets such a request, it creates an unconnected PV object that itself sends a search request to the IOCs. This PV starts out as **Connecting**, and the Gateway responds to the search request as postponed. If it does not connect in a very short time, it becomes **Dead** and the Gateway responds to subsequent search requests as not having it. If it is not found by the Gateway after two minutes, the PV is destroyed. If it becomes connected to an IOC, its new state depends on its old state. If it was **Disconnected** or **Dead**, then it becomes **Inactive** and the next **Exist Test** will return that it was found. If it was **Connecting**, postponed **Exist Tests** are processed, and if a VC is created because an MEDM is still interested, it becomes **Active**. Soon afterward a Chan is created, and the MEDM is connected to the VC and through it to the PV. Once the PV is **Active** or **Inactive**, the Gateway responds to subsequent search requests as having the process variable. If the connection is then lost, the VC and its Chans are destroyed, and the PV becomes **Disconnected**.

Internal process variables in the Gateway give the number of PVs in each state, and can be used to monitor its behavior. In order to interpret the results, it is necessary to have an understanding of these states.

PVs are either **Connected** or **Unconnected**. **Connected** consists of either **Active** or **Inactive**. **Unconnected** consists of **Connecting**, **Dead**, or **Disconnected**. The top state in the picture above is **Unconnected,** but it is not clear whether it is **Connecting**, **Dead**, or **Disconnected**. The middle state is **Connected** and **Active**. The bottom state is **Connected** and **Inactive**.

- total = connected + unconnected
- connected = active + inactive
- unconnected = connecting + dead + disconnected

Saying this another way: the PVs for which the Gateway has an established client connection with an IOC are **Connected**. Otherwise they are **Unconnected**. For the **Connected** ones, if there is a connection to an MEDM, then they are **Active**. Otherwise they are **Inactive**. If they are **Unconnected**, it may because they are **Disconnected** (meaning formerly connected), **Connecting**, or **Dead**.



The Gateway handles most of the information about a process variable that would be needed by most Channel Access Clients, in particular MEDM. An exception is the RTYP field for records in an IOC. PvInfo in MEDM shows this field if directly connected to an IOC, but just shows Not Available if through the Gateway. The field is the record type of the process variable and is little used. It has not been deemed reasonable to add the overhead to the Gateway to make it available. See the discussion under Alarm Handler for more technical information on this issue.

For information on obtaining the Gateway, consult the EPICS Documentation.

History

The Gateway was originally written by Jim Kowalkowski of the APS starting in early 1996. Its basic concept and structure owes to him, but it was never really finished and did not work well or reliably. Janet Anderson did some work on it after Jim left the APS. Kenneth Evans took it over in 1999, and made it work in a usable fashion. This required making several changes to EPICS base, however. These changes were never officially incorporated into EPICS base, and it hence required a special version of base to build the Gateway. Ralph Lange of BESSY also worked on the earlier version and took the Gateway over in 2001, adding regular expressions to the process variable specifications and making it work with the Alarm Handler. Kenneth Evans converted it to EPICS 3.14 in mid-2002, working with Jeff Hill of LANL, the person in charge of Channel Access. New features, including many more internal process variables were added, and it was made to run on Linux and WIN32. There

were problems with the early 3.14 versions of Channel Access, and these were slowly resolved over the period from then until late summer 2003. It now works acceptably with EPICS 3.14.6 or later, uses the standard version of base, uses appreciably less CPU than Gateway 1.3, and has a number of new features as well as some bug fixes.

Starting the Gateway

The Gateway is started by typing the following on a command line:

```
gateway [Options]
```

The options are:

-debug <value>	Enter a value between 0-100. 50 gives lots of information, 1 gives a small amount. For developers.
-pvlist <filename>	Name of the file with all the allowed PVs in it. There is a sample file, gateway.pvlist, in the source distribution and reproduced here. The version in the distribution may be more recent. See Access Security for more information.
-access <filename>	Name of the file with all the EPICS access security rules in it. The syntax for this file is the same as that for EPICS access security. See the Application Developers Guide for more information on this syntax and EPICS access security in general, and see Access Security below for more information.. There is a sample file, gateway.access, in the source distribution and reproduced here. The version in the distribution may be more recent.
-log <filename>	Name of file where all messages from the Gateway go, including stderr and stdout. This file will be automatically renamed if the Gateway is in server mode and it restarts.
-command <filename>	Name of the file where your customized Gateway command file goes. The specified commands in this file are executed when a USR1 signal is sent to the Gateway or the internal process variable, gateway:commandFlag, is set to 1. Lines in the command file may be R1, R2, R3, or AS to run reports 1-3 and reread the access security file, respectively. See Reports below for more information. There is a sample file, gateway.command, in the source distribution and reproduced here. The version in the distribution may be more recent
-putlog <filename>	Name of the file where Gateway put logging goes. Put logging is specified with TRAPWRITE in the access file. See the Application Developers Guide under access security as well as Access Security and Put Logging below for more information. This file will be automatically renamed if the Gateway is in server mode and it restarts.
-report <filename>	Name of the file where reports go. The reports are appended if the file exists when the reports are generated. If not specified, the name is **gateway:report**.
-home <directory>	Directory where the Gateway looks for its input files and writes its output files. Setting this is equivalent to changing to that directory before starting the Gateway. Setting the environment variable GATEWAY_HOME also accomplishes the same result.
-sip <ip-address>	IP address where the Gateway server listens for process variable requests. Sets the environment variable EPICS_CAS_INTF_ADDR.
-ignore <ip-address-list>	List of IP address that the Gateway server ignores. Sets the environment variable EPICS_CAS_IGNORE_ADDR_LIST.
-cip <ip-address-list>	List of IP addresses that the Gateway client uses to find the real process variables. See the Channel Access Reference Manual for more information. Sets the environment variables EPICS_CA_AUTO_LIST=NO and EPICS_CA_ADDR_LIST.
-sport <port>	The port which the Gateway server uses to listen for process variable requests. The default is 5064. Sets the environment variable EPICS_CAS_SERVER_PORT.
-cport <port>	The port which the Gateway client uses to find the real process variables. Sets the environment variable EPICS_CA_SERVER_PORT. Be aware that if you specify -cport and do not specify -sport, then the server port will also be the same as what you specify for -cport. This is a Channel Access feature. To avoid it, specify -sport.
-connect_timeout <sec>	The amount of time in seconds that the Gateway will allow a process variable search to continue before marking the process variable as being **Dead**. The default is 1.

-inactive_timeout <sec>	The amount of time in seconds that the Gateway will hold the real connection to an **Inactive** process variable. The default is 7200 (2 hours).
-dead_timeout <sec>	The amount of time in seconds that the Gateway will continue to look for process variables that are **Connecting** to the IOCs that the Gateway is using. The default is 120 (2 min).
-disconnect_timeout <sec>	The amount of time in seconds that the Gateway will hold **Disconnected** process variables (those that were connected but have been disconnected). The default is 7200 (2 hours).
-reconnect_inhibit <sec>	The minimum amount of time in seconds after the last beacon anomaly sequence before generating a new one. The Gateway generates a beacon anomaly sequence when process variables from the IOCs reconnect and when it rereads access security. Thus causes MEDMs to reissue search requests for unconnected PVs. Used to limit the beacon sequences, and hence the search requests. Search requests last for about 8 min., so it is not necessary to reissue beacon anomalies much more frequently than 8 min. The default is 300 (5 minutes).
-server	Start in server mode. Start a daemon that watches the Gateway and automatically restarts it if it dies. Not available on WIN32. See Server Mode.
-mask <string>	Event mask that is used for connections to the IOCs. Use any combination of v (value), a (alarm), l (log). The default is va (forward value and alarm change events).
-prefix <string>	Set the prefix for the Gateway internal process variables. Defaults to the hostname on which the Gateway is running.
-uid <integer>	Run the Gateway server as this user id number. The Gateway does a setuid(2) to this uid. Not available on WIN32.
-gid <integer>	Run the Gateway server as this group id number. The Gateway does a setgid(2) to this uid. Not available on WIN32.

A typical command line is:

```
% gateway -log gateway.log -cip 164.54.3.255 -sip 164.54.188.65 -server
```

For the command-line options that set environment variables, an alternative would be to set them yourself before starting the Gateway instead of using the option.

With EPICS 3.13, there should be no more than one portable server, such as a Gateway, running on a workstation. The reason is that there is not sufficient information in the beacons to distinguish among servers on the same host. Clients such as MEDM must treat their beacons as one set of beacon signals and will see sum and difference frequencies. They will likely interpret this as an IOC coming up and will continuously reissue their outstanding search requests. As a result, if the Gateway is one of two portable servers on the same host, it will be causing a problem. Two servers on the same host should not be a problem with 3.14 servers. It is not clear what will happen with a mixture of 3.13 and 3.14 servers on the same workstation. That situation should be avoided.

The Gateway makes two files, **gateway.killer** and **gateway.restart**, when it starts. The file **gateway.killer** contains a summary of the settings used, as well as commands to kill the current Gateway, kill the server if in server mode, execute the commands file, and execute the Process Variable Report. It is a valid shell script, and the only line not commented is the one which kills the server in server mode or the current Gateway otherwise. Consequently, it can be run to kill the server in server mode or the current Gateway otherwise. The other command lines can be pasted to a shell to do the other actions. The file **gateway.restart** can be used to kill the current Gateway, whether in server mode or not. These files and features are not available on WIN32.

The Gateway can also create a file named **gateway.reserve**. It does this to get around a problem that a process can only open files with file descriptors (FDs) below a certain limit (256 on Solaris). If there are connections to more than roughly this number of IOCs, not uncommon in a production environment, there is no FD available to reread the access security and command files. The Gateway gets around this by reserving a FD pointing to **gateway.reserve**. Do not delete **gateway.reserve**. Also note that if reading the access security fails for this or any other reason, the access security will go to the default, not what it was before the failure.

The number of threads on Linux is determined by the maximum user space, about half of the 32-bit address

space or about 2 GB. The maximum number of threads is thus 2 GB divided by the stacksize (typically 10240 KB) giving about 204 threads in the typical case. Channel Access uses two threads for each IOC connection, one for receive and one for send. This limits the Gateway from connecting to more than about 100 IOCs. If the stacksize were decreased to say, 4096 KB (ulimit -s 4096), then about 256 IOCs would be allowed. The APS Linux Gateway currently uses this number. Starting with Base 3.14.7, EPICS allocates memory for threads more efficiently, and you should not have to change the stacksize.

Using the Gateway

To use the Gateway as your process variable server, it is usually only necessary to set your EPICS_CA_ADDR_LIST environment variable to the address used for the -sip command-line option when the Gateway was started. If want to use more than one server, add them all to the list for EPICS_CA_ADDR_LIST, separated by spaces. You probably also want to set EPICS_CA_AUTO_ADDR_LIST to NO to prevent also searching the local network for servers. You would certainly do this if there are IOCs on the local network that might have process variables with the same names as ones the Gateway would find. They would then be found from two servers, and your EPICS application would likely complain. If the Gateway is using a non-standard port, you need to set EPICS_CA_SERVER_PORT to the number used with -sport when the Gateway was started. Then just run your EPICS application, for example, MEDM.

To stop the Gateway, use the **gateway.killer** file (described under Starting the Gateway), set the **gateway.quitFlag** or **gateway.quitServer** flag (described under Gateway Process Variables), or kill it in the usual way for your operating system. It may take a while to completely shut down. This is owing to EPICS cleanup processes that run after the program has officially exited. These processes may also print error messages after the Gateway prints its termination message just before calling exit. It is a good idea to check that the Gateway has truly shut down. This behavior may be corrected in future releases of base.

Error messages

The Gateway gets error messages from several places, including internally, the Channel Access Server library, and the Channel Access Client library. These errors typically appear in the log. The Gateway internal messages usually have a timestamp. Messages from the Channel Access Server library are typically sent to the EPICS Errlog facility and are received by the Gateway, often with different lines of the message being received separately. These messages typically do not have a timestamp, and the Gateway puts a line in the log with a timestamp when it thinks the message is complete. This line will say "!!! Errlog message received (message is above)". You must try to decide what lines above this line comprise the message. The problem is compounded by the fact that other kinds of messages may be interspersed between the lines for the Errlog message. The Channel Access Server library can also write messages directly. The Channel Access Client library can use the Errlog facility or send exception messages to the Gateway. The Errlog messages are handled as described. The exception messages are printed with as much additional information as is available and will start with a line with a timestamp saying "gateServer::exCB: Channel Access Exception:". Common exception messages, such as "Virtual circuit unresponsive" and "Virtual circuit disconnect" only print a single, timestamped line without the additional information. In most cases the Gateway prints a line with a timestamp saying how it was terminated. If the process is killed with SIGKILL (-9) or the machine is rebooted, this line has no chance to appear. In all cases the Gateway attempts to timestamp messages as timing is often important in diagnosing problems.

Building the Gateway

The Gateway uses 3.14 makefiles as it can only be built with 3.14. To build it you need to:

1. Obtain base and put it at the same directory level as extensions. Use at least base 3.14.
2. Do a make in base to build base.
3. Obtain extensions/configure.
4. Change extensions/configure/RELEASE so EPICS_BASE points to your base.
5. Do a make in extensions/config.
6. Obtain the GNU Regex extension and put it under extensions/src/gnuregex. (Or provide the GNU Regex routines another way.)
7. Do a make in extensions/src/gnuregex.
8. Obtain the Gateway directory and put it under extensions/src/gateway.
9. Do a make in extensions/src/gateway.

See your version of base for more information on the EPICS build system. The Gateway can also be built with 3.13 makefiles, but you would have to create Makefile and Makefile.Host yourself to do that.

Access Security

The Gateway applies access security in addition to any access security that may be implemented in the IOCs or other servers to which it connects. It supplements but cannot override IOC access security. The access security is specified in two files, **gateway.pvlist** and **gateway.access**, by default. See -pvlist and -access on the command line. The first file, **gateway.pvlist**, specifies what process variable name patterns are allowed and denied, and is described in more detail below. The second, **gateway.access**, specifies the access security rules. These two files are read at startup.

The access security can be changed on demand by setting the internal process variable **gateway:newAsFlag** (see Gateway Process Variables) or by using the command file (see Reports). This causes the two files to be reread and the access security to be changed accordingly. After rereading access security, the Gateway generates a beacon anomaly, which will cause MEDMs to reissue search requests for unfound PVs.

Note that if reading the access security fails, the access security will go to the default, not what it was before the failure.

The access security rules specified in **gateway.access** are the same as used for access security in IOCs. In fact, both IOCs and the Gateway use much of the same access-security code. These rules and the concepts can be complicated and will not be reproduced in this document. See the Application Developers Guide for more information. It can be noted that user names are case sensitive and host names are not. WIN32 may use a different case than UNIX for the same account. In this case user names must be entered with both versions. There is a sample file, gateway.access, in the source distribution and reproduced here to get you started. The version in the distribution may be more recent.

The pvlist file is specific to the Gateway. Its function is to specify what process variable name patterns are allowed or denied and to optionally associate patterns with access security groups and security levels in the access file. The patterns are GNU-style regular expressions. (See a UNIX book for more information about regular expressions.) There is a sample file, gateway.pvlist, in the source distribution and reproduced here to get you started. The version in the distribution may be more recent. Directions for this file are in the sample pvlist file as well as here.

The lines in the pvlist file are of the following four forms:

1. EVALUATION ORDER <eval-order>
2. <pv-name-pattern> DENY [FROM] [<host> ...]
3. <pv-name-pattern> ALIAS <real-pv-name> [<asg> [<asl>]]
4. <pv-name-pattern> ALLOW [<asg> [<asl>]]

where:

- <eval-order> is DENY, ALLOW or ALLOW, DENY
- <pv-name-pattern> is a regular expression that matches process variable names.
- <host> is an unqualified host name, e.g. hydra.
- <real-pv-name> is a substitution pattern that specifies the real process variable name. Occurrences of \1 ... \9 in <real-pv-name> are replaced by matched sub-expressions in the <pv-name-pattern>.
- <asg> is an access security group as specified in the access file. [The default group is DEFAULT.]
- <asl> is the access security level (0 or 1). These numbers pertain to the settings for particular fields in a record in an IOC. Permission for level 1 implies permission for level 0. See the Application Developers Guide for more information on access security group and access security level.

DENY FROM is not implemented by default when the Gateway is built. It requires a special switch to be set for the build. It should not be necessary. Use -ignore instead.

EVALUATION ORDER

This will set the evaluation order that is used when a client requests a process variable. Setting this to "DENY, ALLOW" will allow access to a process variable name that matches both a DENY and an ALLOW pattern. "ALLOW, DENY" will make a DENY override an ALLOW for the same variable. (This is the default.)

If DENY FROM is enabled, then matching DENY FROM host-name patterns will always override matching ALLOW process-variable-name patterns.

Stated another way, the Gateway keeps three lists for: DENY FROM (if enabled), DENY, and ALLOW. The DENY FROM list is searched first, and if the host name is found, access is disallowed. If the order is ALLOW, DENY, the DENY list is searched, and if the process variable name is found, access is disallowed. Then the ALLOW list is searched, and if the name is found, access is allowed, otherwise it is disallowed. Disallowed means the Gateway returns that it does not have the process variable when the **Exist Test** is called. In addition, for safety, it will not create a connection to an MEDM.

The lines in the pvlist file are read from bottom to top. The first rule in each list that matches is used, so the most general rules should be placed first.

DENY / DENY FROM

The Gateway will ignore requests for any process variable that matches the DENY pattern, depending on the EVALUATION ORDER and whether there is also an ALLOW rule that matches. This can be used to block the Gateway from responding to groups of process variables. DENY FROM will block the process variables only for the given hosts. This can be useful to prevent loops caused by forwarding to other Gateways. However, the host name lookup for each **Exist Test** can take considerable time. Moreover, the same result can be accomplished via the -ignore command-line option. See the symmetric Gateway configuration below for an example.

ALIAS

This defines a process variable alias and allows it as a pattern for names which the Gateway should serve. For process variable names that match <pv-name-pattern>, the Gateway translates the name into a real process variable name and uses the real name as if it had been the one specified. The <real-pv-name> may contain the special escape sequences \1 ... \9 which will be replaced by the nth subexpression matched. See a UNIX book on regular expressions for more information. There is an example in the sample pvlist file. Access security rules to be used for process variables matched by this pattern may be specified. If not specified, the defaults are the DEFAULT group and level 1. Apart from specifying an alias, this rule is functionally the same as ALLOW. Note that a PV can only belong to one access group and access level. If you specify access to a real PV via several different ALIAS or ALLOW rules that assign different groups or levels to the PV, then which of these groups and levels is used is undefined.

ALLOW

This is used to declare process variable names which the Gateway should serve. Access security rules to be used for process variables matched by this pattern may be specified. If not specified, the defaults are the DEFAULT group and level 1.

Note

1. Commands are not case sensitive.
2. Patterns use GNU-style regular expressions. (See a UNIX book for information on regular expressions.)
3. Any process variable not included in an ALLOW command is not allowed access.
4. If no pvlist file is specified on the command line, a default rule ".* ALLOW" will be created to handle all process variables.
5. The patterns are matched in reverse order; that is, you should always specify general rules before specific rules.

Put Logging

It is possible to log whenever someone writes to a process variable. (A write is sometimes called a "put" and a read is called a "get".) To do this you need to specify -putlog on the command line when the Gateway is started and specify a filename for the put logging. There needs to be an access security group in the **gateway.access** file that has a rule with WRITE and TRAPWRITE specified, for example, "RULE(1,WRITE,TRAPWRITE)". (The security level could also be 0.) Then whenever there is a write (or put) to any process variable in that group, it will be logged in the specified putlog file. As with the log file, this file will be automatically renamed whenever the Gateway restarts. An example may be found in the sample access file, where writes are logged for process variables that belong to the GatewayAdmin group. Using the sample pvlist file, these would be those that fit the pattern "gateway:*Flag".

Beacon Anomalies and Search Requests

A server sends beacons, which are broadcast UDP packets, at regular intervals, EPICS_CA_BEACON_PERIOD, 15 s by default. The clients, such as MEDM, monitor these heartbeats. When there is a beacon anomaly, defined as anything that is different from this regular pattern, the clients reissue search requests for their unconnected PVs.

A client sends search requests when it wants to connect to a process variable. A search request is a series of 100 UDP packets sent at increasingly longer intervals. It takes approximately 8 min. to complete the sequence. In most cases a server responds on the first or second packet; the connection process between the server and the client begins; and the search sequence ends. However for those PVs that are not found, the potential for a connection to happen exists for approximately 8 min., as long as the packets are still going out.

When client gets a hangup message from the server, it closes the TCP connection, the process variable is marked as disconnected (MEDM screens turn white), and search requests are reissued. In 3.13 when a client has no communication from the server for EPICS_CA_CONN_TMO seconds (15 s by default), it sends an "are you there" message to the server. If there is no reply in 5 sec, it closes the TCP connection, the process variable is marked as disconnected, and search requests are reissued, the same as for a hangup. In 3.14, it marks the connection as disconnected, but does not close the TCP connection nor reissue beacons. The client may get a "Virtual circuit unresponsive" message. This is in part to prevent search-request storms under bad network conditions.

When a server, including the Gateway comes up, it intentionally broadcasts an irregular pattern for a short time (a particular form of beacon anomaly) so the clients will know a new server is online and reissue their outstanding search requests in case it might have those PVs. The Gateway also broadcasts this irregular pattern when a PV beomes connected again after having been disconnected and also when access security is reread. This is so the MEDMs will know to try to reconnect.

For security reasons some network administrators will not allow UDP broadcasts to cross subnet boundaries. If this is the case, then when the MEDM is on a different subnet than the Gateway, it will not see beacons at all and will not be able to tell if the Gateway has come up, This is a common case in large installations. Owing to the length of the search request sequence, if a server such as the Gateway goes down and comes back up within approximately 8 min,. the MEDM will reconnect. Otherwise, it will not, unless something else happens to make it reissue its searches. One such thing is to try to connect to a new PV. This causes a client to reissue outstanding searches along with the search for the new one. The latest versions of MEDM and StripTool have a button to try to reconnect. In any event, Gateway administrators should be careful to not have a long delay between killing and restarting a Gateway if possible.

Reports

The Gateway prints three kinds of reports: (1) the Virtual Connection Report, which includes all MEDM or other client connections to all process variables; (2) the Process Variable Report, which includes all process variables grouped by state; and (3) the Access Security Report, which includes the allowed and denied process-variable patterns from the pvlist file, **gateway.pvlist**, by default. The Gateway prints its reports to the report file, **gateway.report**, by default, appending them if the file already exists. These reports can be long.

The reports can be initiated by setting the associated internal process variable, for example, **gateway:report1Flag**, to 1. They can also be initiated by sending a USR1 signal to the Gateway. (Use "kill -USR1 <gateway-pid>".). This causes the command file, **gateway.command**, by default, to be read and the specified reports executed. The command file can be modified to do any combination of the three reports, as well as to reread the access security files. See the -command command-line option for more details and an example. The Process Variable Report can be executed by sending a USR2 signal. (Use "kill -USR2 <gateway-pid>".) The Gateway makes a file, **gateway.killer**, when it starts. The commands and the correct pids are printed in that file, so you can look at it to get the appropriate command lines. Signals and the **gateway.killer** file are not available on WIN32.

Server Mode

The Gateway can be operated in server mode by specifying -server on the command line. In this mode a server process is started that in turn starts the regular Gateway process and then watches it and automatically restarts it if it dies. Using this mode the Gateway can recover from crashes. The running Gateway can effectively be

reset by killing it via the **gateway:quitFlag** process variable, by running **gateway.restart**, or by killing the process in some other way. When any of these methods is used or the Gateway crashes, a new instance of the Gateway is started. The server process itself can be killed by setting the **gateway:quitServerFlag**, by running **gateway.killer**, or by otherwise killing the server process. Then no more regular Gateway processes will be automatically started.

In order to avoid runaway creation of processes, there can only be ten restarts in any ten-minute interval. If the number of restarts in this interval exceeds this limit, the server will be stopped and no more processes will be created.

This feature is not available on WIN32.

Gateway Process Variables

The Gateway publishes several process variables, allowing you to control it and monitor it. By default these have a prefix, which is the name of the host machine, but this can be changed by -prefix in the command-line options. Here, we will use the prefix "gateway". See the Introduction for a more detailed description of the states and concepts.

The Gateway internal process variables are not record based as in an IOC. They do have a DESC field, however. This avoids delays when using PvInfo in MEDM and allows StripTool to put a description in the legend. In older versions of the Gateway, the internal process variables used a dot as a separater rather than a colon. When the DESC field was added, they were changed to use a colon. This makes them more consistent with records in IOCs, and allows MEDM and StripTool to properly determine the name for the DESC field. The sample GATEWAY.pvlist file shows how to set aliases if you want to use the old names. The old names will cause MEDM and StripTool to not correctly form the name for the DESC field, however. This will cause delays for PvInfo in MEDM, and StripTool will not display the descriptions in the legend.

gateway:vctotal

This is the total number of VC objects and should be the same as **gateway:active, except for short, transient situations.**

gateway:pvtotal

This is the number of PV objects.

gateway:connected

This is the number of connected PV objects, that is, process variables for which the Gateway has a working connection to the IOC. The connections may be **Active** or **Inactive**. In earlier versions of the Gateway, this was named **gateway:alive**.

gateway:active

This is the number of **Active** connections. The connection is **Active** if an MEDM is attached to the Gateway and using the process variable

gateway:inactive

This is the number of **Inactive** connections. The connection is **Inactive** if no MEDM connected to the Gateway is using it.

gateway:unconnected

This is the number of PV objects which are **Connecting**, **Dead**, or **Disconnected**. There is no working connection to an IOC but the PV object is still around. There is no VC object.

gateway:connecting

This is the number of PV objects which are **Connnecting**, that is, which have just been created and are trying to connect to an IOC.

gateway:disconnected

This is the number of PV objects which are **Disconnected**, that is, which were formerly connected but have lost the connection.

gateway:dead

This is the number of PV objects which are **Dead**, that is, which did not connect and are scheduled for removal.

gateway:fd

This is the number of file descriptors in use by the gateway. This process variable will only be available if registering file descriptors is enabled in the Gateway when it is built. Not registering them appears to significantly decrease the CPU usage, so this process variable will probably not be available. It requires a special switch to be set when the Gateway is built.

gateway:clientEventRate

This is the rate in Hz at which client events are happening. Client events include such things as attempted read and writes, connection changes, value changes, and access rights changes. They relate to communication from the IOC.

gateway:clientPostRate

This is the rate in Hz at which events are posted from the VC object to CAS and hence to the MEDMs. It is independent of the number of MEDMs (providing there is at least one). It tends to be similar to the clientEventRate; but only value changes cause post events. The other events do not contribute.

In earlier versions of the Gateway this was named **gateway:postEventRate**.

gateway:existTestRate

This is the rate in Hz at which the Gateway receives search requests. For each search request, it has to do an **Exist Test** to see if it has the process variable or not. If it is not already connected to the process variable, it creates a PV object which itself initiates a series of search requests to the IOCs. See the Introduction for more information on search requests. **Exist Tests** put a load on the Gateway. For process variables that do not exist, there can be many unsuccessful **Exist Tests**. See CaSnooper for a tool to monitor **Exist Tests**.

gateway:loopRate

This is the rate in Hz at which the Gateway executes its main loop. The main loop consists of a call to CAS followed by a call to CAC followed by Gateway housekeeping routines. The call to CAS lasts for 10 ms unless there is activity on a file descriptor (in which case it returns early) or if it takes longer than that to process its callbacks (in which case it returns late). The call to CAC takes as long as it takes to do the work required. The Gateway housekeeping typically uses comparatively little time. As the Gateway becomes loaded down, the calls to CAS take longer and so do the calls to CAC until eventually all the available CPU time is used. As the load increases, the loopRate tends to decrease. However, no load can have a lower loop rate than with some load because the call to CAS waits the whole 10 ms. It is a rule of thumb that CAC should be called at least every 100 ms. By examining the loopRate you can tell if this is happening. That is, it should remain above 10 Hz.

gateway:cpuFract

This is the fraction of the available CPU time that is being used by the Gateway; that is, the CPU time divided by the elapsed time. It is a good monitor of the load on the Gateway. On WIN32 the cpuFract is always 1 and not useful. This is owing to their implementation of the clock() function. On Linux, threads are separate processes, and the CPU time for them is not included, only that for the main process. This is because clock() on Linux returns only the CPU time for the process that called it. If there is more than one processor, the cpuFract could exceed unity. Note that Top on Linux shows the CPU time divided by the number of processors and optionally lists all threads separately. Thus, on Linux, the Gateway cpuFract should agree with the Top value for the main Gateway process multiplied by the number of processors. You can use Top to monitor the CPU time for all the Gateway processes if you need that.

gateway:load

This is the load average for the machine on which the Gateway is running. The load average is the number of processes in the system run queue averaged over one minute. It is implemented by the getloadavg() function,

and you can look at the man page for more information. It is the same as the process variable with the same name in IOCs. It is another way to monitor the load. On WIN32 it is always zero and not useful.

gateway:serverEventRate

This is the rate in Hz at which CAS processes events. It tends to be the clientEventRate multiplied by the number of MEDMs looking at the process variable.

gateway:serverPostRate

This is the rate in Hz at which events are posted to CAS. If the Gateway is keeping up, this will be very close to the serverEventRate. If not, it will be higher and all posted events will not have been processed.

gateway:commandFlag

Writing a 1 to this process variable causes the command file to be processed. The process variable is reset to 0 after the Command File is processed. See the description of command-line arguments and the Reports section for a description of the command file.

gateway:report1Flag

Writing a 1 to this process variable causes Report 1 to be appended to the report file. Report 1 is the VC Report.The process variable is reset to 0 afterward.

gateway:report2Flag

Writing a 1 to this process variable causes Report 2 to be appended to the report file. Report 2 is the PV Report.The process variable is reset to 0 afterward.

gateway:report3Flag

Writing a 1 to this process variable causes Report 3 to be appended to the report file. Report 3 is the Access Security Report. The process variable is reset to 0 afterward.

gateway:newAsFlag

Writing a 1 to this process variable causes the access security files, by default **gateway.access** and **gateway.pvlist**, to be reread. The process variable is reset to 0 afterward. There is currently a problem on some platforms that files can only be opened using file descriptors 0 to 255. If there are many CA socket connections, there may not be a file descriptor in this range available. In that case the access security file will not be read and an error will appear in the log. There is a workaround in the Gateway for this problem for Solaris.

gateway:quitFlag

Writing a 1 to this process variable causes the Gateway process to exit. If the Gateway is in server mode, it will be restarted as a new process. In that case it functions more as a reset.

gateway:quitServerFlag

Writing a 1 to this process variable causes the Gateway server process to end, also ending the current gateway. No more Gateways will be started by the current server. If the Gateway is not in server mode, it has the same effect as **gateway:quitFlag**.

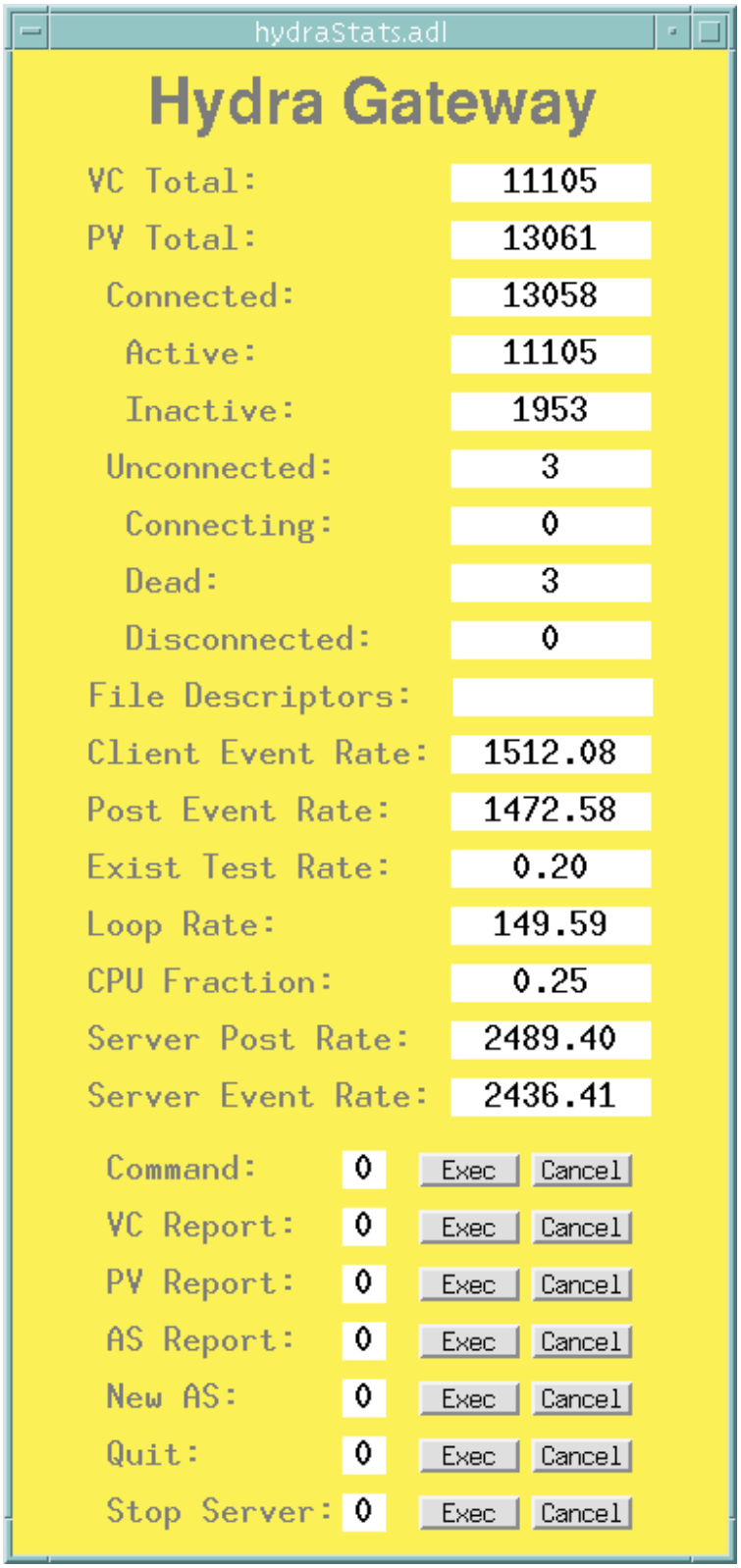
Alarm Handler

The Alarm Handler works through the Gateway. The following discussion is for those who understand Channel Access in more depth: For most process variables the Gateway can get all of its information in a single DBR_TIME_xxx structure, for example, DBR_TIME_DOUBLE, where xxx is the native type of the process variable. These structures are defined in db_access.h. The Gateway establishes an event handler for this structure and gets notified when it changes and modifies it when it needs to be changed. There is only one structure per process variable. This structure does not contain all the information needed by the Alarm Handler, so the Gateway need to keep track of another structure, DBR_STSACK_STRING. This is only done when needed. A related issue is that the Gateway does not handle the RTYP field in a process variable from an IOC. To do this it

would have to handle DBR_CLASS_NAME. Each structure the Gateway needs to handle adds to its overhead.

GUI Interface

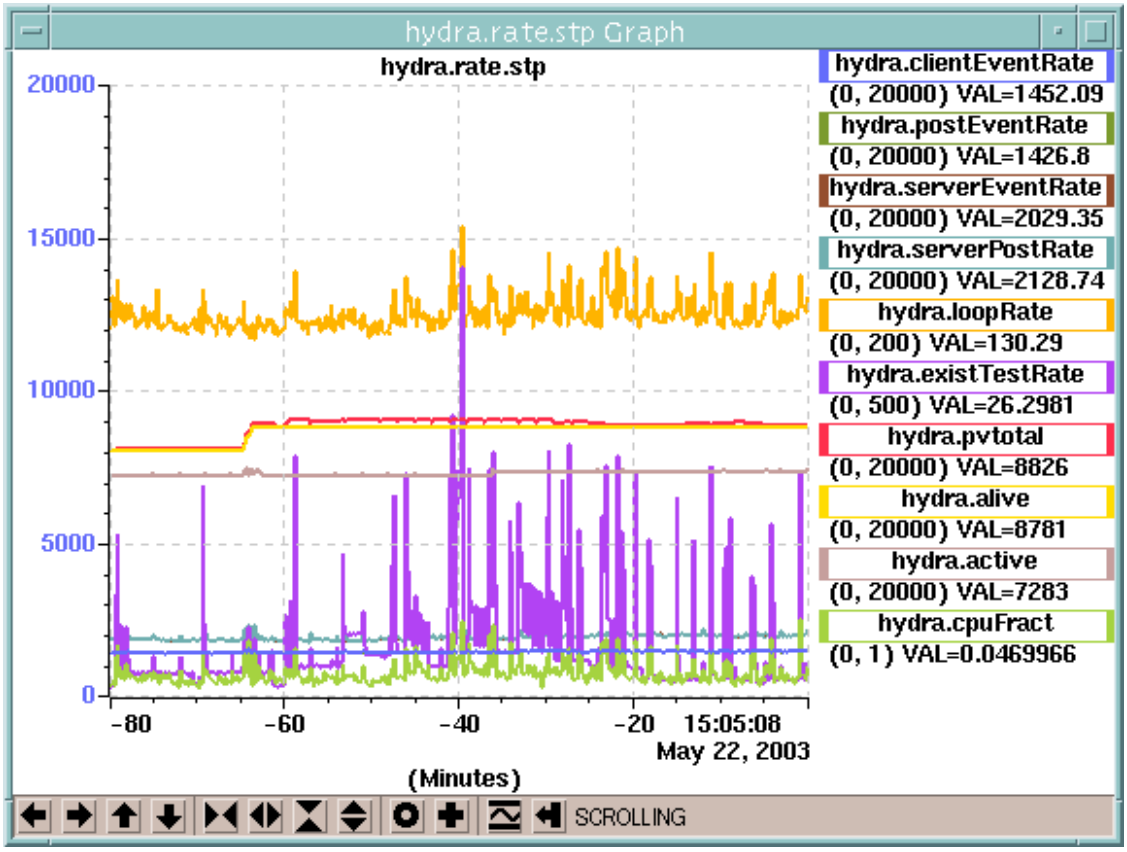
Owing to its published internal process variables, you can monitor and control the Gateway via MEDM or another tool. You can set the writable internal process variables (if you have access) and cause the Gateway to print reports, reread the access security file, or quit. This is an example MEDM screen:



You could also add a Shell Command button to run a script to start the Gateway. The MEDM ADL file for this

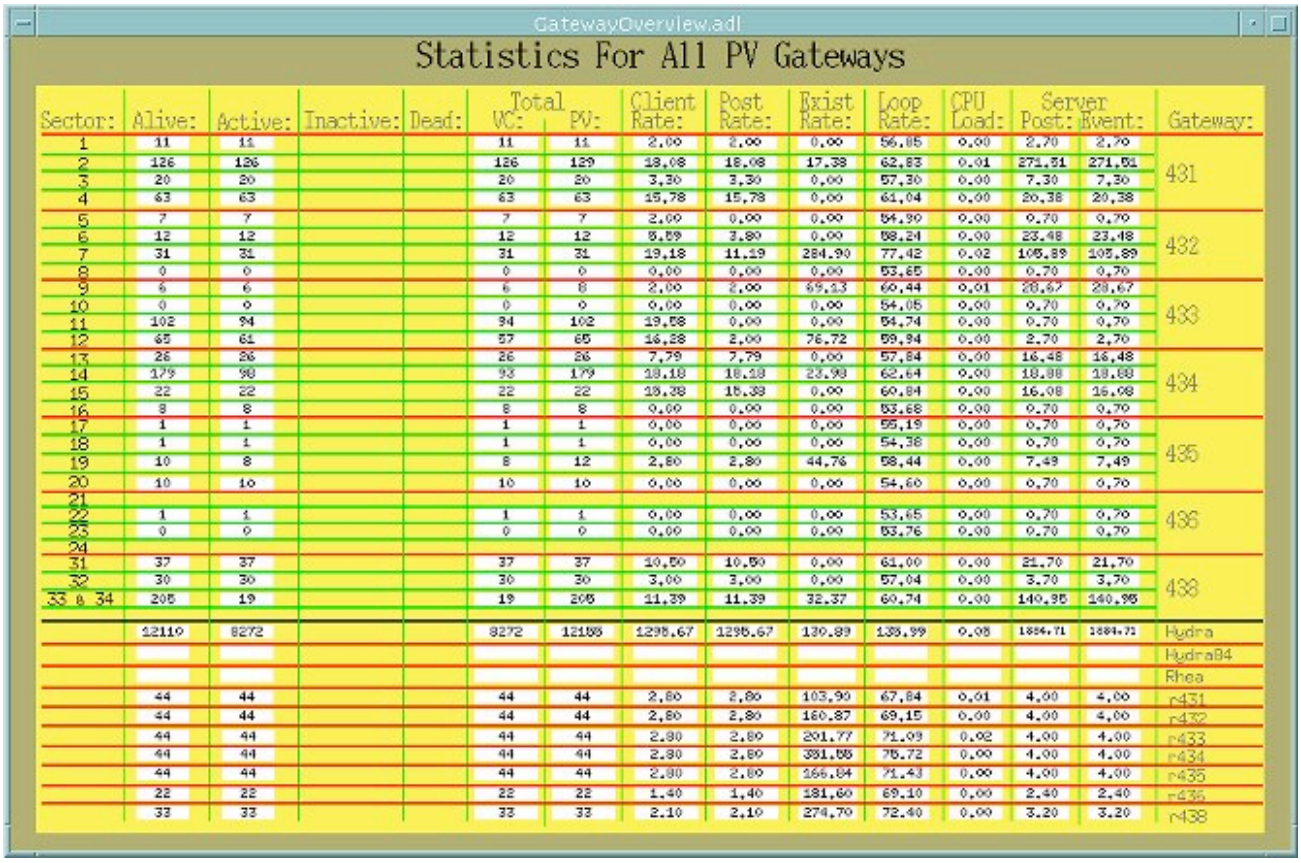
screen is included in the Gateway distribution as hydraStats.adl. Typically the Exec buttons operate so fast that you will not see the value change from 0, and the Cancel button is of little use. However, the chain of events is that the Exec button sets the process variable to 1, the Gateway notices this during the cleanup phase of the main loop, the appropriate action is executed, and the process variable is reset to zero.

It is convenient to monitor and possibly record the state of the Gateway using StripTool. You could, of course, use any other EPICS monitoring tool of your choice. This is an example of using StripTool:



The StripTool configuration file for this image is included in the Gateway distribution as hydra.rate.stp. Note that for the example shown, the cpuFract is low, the loop rate is high, and the postRate's and eventRate's are nearly equal, indicating the Gateway is healthy.

This is a screen that shows the status of all of the APS Gateways:


GatewayOverview.adl

Statistics For All PV Gateways

Sector:	Alive:	Active:	Inactive:	Dead:	Total VC:	Total PV:	Client Rate:	Post Rate:	Exist Rate:	Loop Rate:	CPU Load:	Server Post:	Server Event:	Gateway:
1	11	11			11	11	2.00	2.00	0.00	56.85	0.00	2.70	2.70	
2	126	126			126	129	18.08	18.08	17.38	62.83	0.01	271.51	271.51	431
3	20	20			20	20	3.30	3.30	0.00	57.30	0.00	7.30	7.30	
4	63	63			63	63	15.78	15.78	0.00	61.04	0.00	20.38	20.38	
5	7	7			7	7	2.00	0.00	0.00	54.90	0.00	0.70	0.70	
6	12	12			12	12	5.59	3.80	0.00	58.24	0.00	23.48	23.48	432
7	31	31			31	31	19.18	11.19	284.90	77.42	0.02	105.89	105.89	
8	0	0			0	0	0.00	0.00	0.00	53.85	0.00	0.70	0.70	
9	6	6			6	8	2.00	2.00	69.13	60.44	0.01	28.67	28.67	
10	0	0			0	0	0.00	0.00	0.00	54.05	0.00	0.70	0.70	433
11	102	94			94	102	19.58	0.00	0.00	54.74	0.00	0.70	0.70	
12	65	61			57	65	16.28	2.00	76.72	59.94	0.00	2.70	2.70	
13	26	26			26	26	7.79	7.79	0.00	57.84	0.00	16.48	16.48	
14	179	90			93	179	18.18	18.18	23.98	62.64	0.00	18.88	18.88	434
15	22	22			22	22	18.38	18.38	0.00	60.84	0.00	16.08	16.08	
16	8	8			8	8	0.00	0.00	0.00	53.68	0.00	0.70	0.70	
17	1	1			1	1	0.00	0.00	0.00	55.19	0.00	0.70	0.70	
18	1	1			1	1	0.00	0.00	0.00	54.38	0.00	0.70	0.70	435
19	10	8			8	12	2.80	2.80	44.76	58.44	0.00	7.49	7.49	
20	10	10			10	10	0.00	0.00	0.00	54.60	0.00	0.70	0.70	
21														
22	1	1			1	1	0.00	0.00	0.00	53.65	0.00	0.70	0.70	436
23	0	0			0	0	0.00	0.00	0.00	53.76	0.00	0.70	0.70	
24														
31	37	37			37	37	10.50	10.50	0.00	61.00	0.00	21.70	21.70	
32	30	30			30	30	3.00	3.00	0.00	57.04	0.00	3.70	3.70	438
33 & 34	205	19			19	205	11.39	11.39	32.37	60.74	0.00	140.35	140.35	
	12110	8272			8272	12188	1298.67	1298.67	130.89	135.99	0.08	1884.71	1884.71	Hydra
														Hydra84
														Rhea
	44	44			44	44	2.80	2.80	103.90	67.84	0.01	4.00	4.00	r431
	44	44			44	44	2.80	2.80	160.87	69.15	0.00	4.00	4.00	r432
	44	44			44	44	2.80	2.80	201.77	71.09	0.02	4.00	4.00	r433
	44	44			44	44	2.80	2.80	381.55	75.72	0.00	4.00	4.00	r434
	44	44			44	44	2.80	2.80	186.84	71.43	0.00	4.00	4.00	r435
	22	22			22	22	1.40	1.40	181.60	69.10	0.00	2.40	2.40	r436
	33	33			33	33	2.10	2.10	274.70	72.40	0.00	3.20	3.20	r438

(Click here for full-size image)

The values for inactive and dead are not being used because these Gateways are mostly using Gateway 1.3, and those process variables did not exist for Gateway 1.3. This Gateway is running on the machine Hydra. The values for Hydra84 and Rhea are white because these Gateways are not visible from Hydra. Similarly, the Gateways running on Rhea and Hydra84 do not see the other two. The Gateways labeled r431 through r438 are reverse Gateways that allow the internal process variables for all the Gateways 1-34 to be seen on one MEDM screen. See below for a discussion of Reverse Gateways.

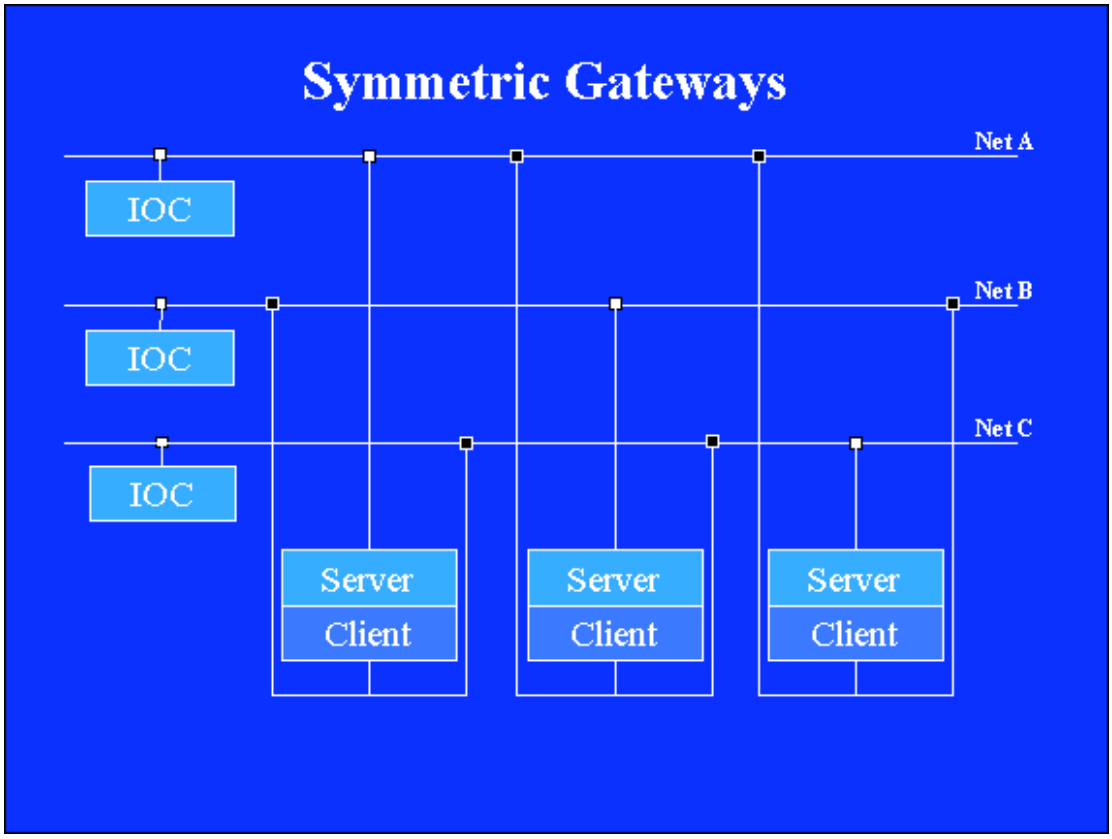
The existTestRate for the reverse Gateways, whose server side is on the machine network, is similar to that seen by all IOCs and servers on the machine network and is used for monitoring channel access activity on that network.

Gateway Configurations

Symmetric Gateways

The following shows a configuration in which each of three networks has a Gateway that finds process variables in IOCs on the other two networks. This is a configuration used at BESSY.

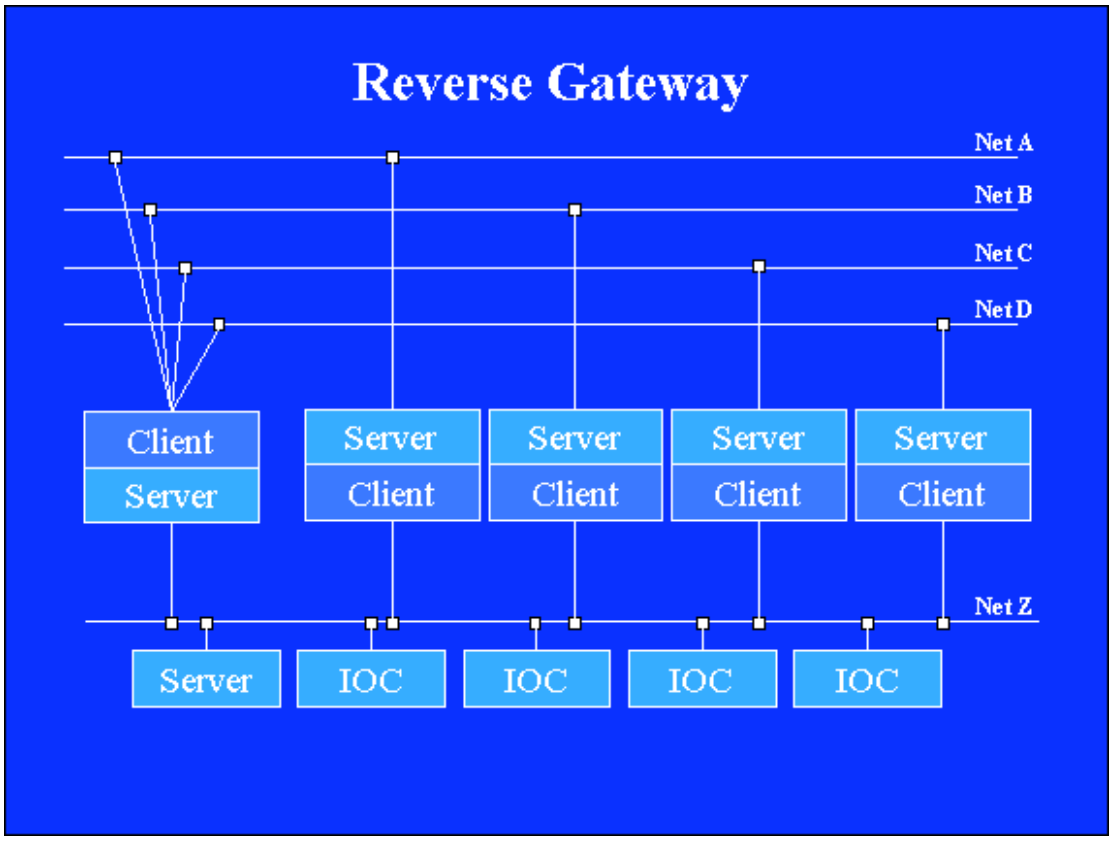
However, note that if you are on say, Net A, and using the Gateway whose server is on Net A, then the client side of this Gateway sees the server side of the other two Gateways. However, these Gateways each see servers on Net A and consequently see the Net A Gateway. Therefore, the Net A Gateway can see process variables on IOCs on Net B and Net C directly and also through the other two servers. To prevent this loop, each Gateway must be configured to not allow process variables from the other two Gateways. This can be done through the -ignore command-line option.



Reverse Gateway

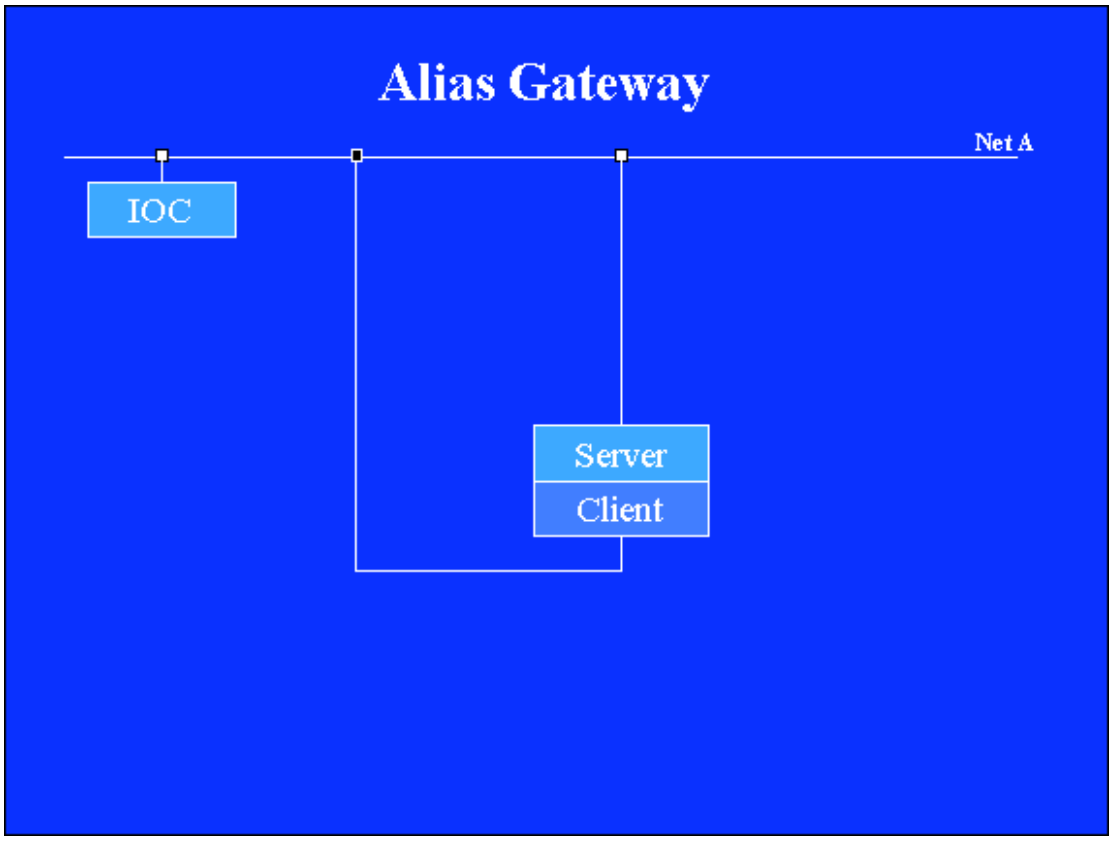
The following configuration shows a reverse Gateway. In this configuration the four networks, Net A, Net B, Net C, and Net D, each has a Gateway that gets process variables from Net Z. This is a configuration used at Argonne. Net Z is the machine network, and the other networks belong to individual experimental teams. These teams have read access to the machine network and whatever access they want to the IOCs on their own network. Argonne at present has eight of these configurations, most with four Gateways plus a reverse Gateway. Owing to the reverse Gateways, the internal process variables for all these Gateways can be seen on one MEDM screen, as in the example above, through a Gateway that sees process variables on the machine network. One does not need (and typically does not have) access to each of these private networks.

To prevent looping, the reverse Gateway only allows the internal process variables from the other four Gateways and itself, and each of the other Gateways denies the internal process variables from the other three. As a result these four Gateways do not see internal process variables from the other three. They do see the ones from other configurations of four. This limitation could be overcome by using one reverse Gateway for every regular Gateway. Note that another Gateway that is not on a network with the client side of any of the reverse Gateways does not need to deny anything and can see all the internal process variables from any of the Gateways in any of these configurations (as in the example above).



Alias Gateway

It is possible to use the Gateway just to provide aliases for process variables, and the following configuration shows an alias Gateway. Only one subnet is needed. If an MEDM asks for the alias name, it is found through the Gateway. If it uses the real name, it is found through the IOC. The two names must be different to avoid finding the same process variable in two different servers.



Channel Access

Channel Access [CA] is the part of EPICS that governs communication between servers and clients. The two major parts are the **Channel Access Server** [CAS] and **Channel Access Client** [CAC, sometimes also just CA since there was originally no CAS]. A CAC, such as MEDM, uses functions in the CAC library for communication with EPICS and provides its own specific functionality on top of that. A server uses the CAS library for communication with EPICS and also provides its own specific functionality. A program that uses CAS is called a **Server Tool** to distinguish it from the CAS library itself. Servers that use CAS are called **Portable Servers**. The Gateway is a ServerTool, a Portable Server, a Channel Access Server, and also a Channel Access Client. There are also **Soft IOC**s, which, like Portable Servers, run on platforms such as Solaris, WIN32, and Linux. These do not currently use CAS and are not Portable Servers.

You can find out more information by looking at the EPICS Channel Access Reference Manual and the Application Developers Guide. There is a version included with each EPICS release. They can be found by starting at IOC Software in the EPICS Documentation and following links to the release and point release of the desired EPICS base.

Acknowledgements

Jeff Hill of Los Alamos National Laboratory, the person responsible for Channel Access, was of great help in developing the Gateway throughout its entire development.

Copyright
