javAPRSSrvr User's Guide
4.3.3b48

Table of Contents

Section 1 - Introduction

javAPRSSrvr was written to provide the amateur radio APRS community a simple, but effective core server for interconnecting IGates and other APRS clients via the Internet. It provides a multi-platform application tuned to present cleanly formatted packets to the APRS-IS network. It does this with effective duplicate checking to reduce bandwidth requirements and reduce loop occurrences present in a loose ad hoc network.

The current APRS-IS network consists of multiple core servers with multiple lower tier servers which may also act as IGates. javAPRSSrvr is designed to act as an APRS-IS server. javAPRSSrvr may be extended by both server and client adjunct software.

It is the author's hope that this software will help to improve the reliability and the utilization of the APRS-IS. Please contact the author at pete@ae5pl.net with any feedback, comments, bugs, or complaints. It is only through this kind of feedback that improvements can be made.

Section 2 - Program Requirements and Description

javAPRSSrvr is designed to run on any OS with any Java Virtual Machine 1.8.0 and later. Due to this newer JVM requirement, OS-specific versions such as Microsoft Java and J# .NET are no longer supported.

javAPRSSrvr is comprised of Java packages and classes which are basically programming components. The main class, javaprssrvr.SrvrMain, is located in javAPRSSrvr.jar. This class is called at startup when using the –jar switch (deprecated) on the command line or when included with the -cp switch or when the javaprssrvr.main module is defined with the -m switch, loads the parameter files and respective classes, and begins execution of the different threads. javAPRSSrvr can be run without any knowledge of Java programming, just like any other command-line application you may use. The primary difference is that you run Java (the Java Virtual Machine or JVM) and tell Java to run javAPRSSrvr. Your operating system thinks it is running Java while Java is running javAPRSSrvr internally. javAPRSSrvr loads all necessary .JAR files and libraries internally. The core javAPRSSrvr .JAR files are javAPRSSrvr.jar, and javAPRSSrvrCore.jar. These .JAR files must be present in the same directory for javAPRSSrvr to run.

javAPRSSrvr considers all connections, upstream and downstream, as client connections. In the case of upstream ("dialing out" to a remote server) connections, all non-duplicate packets not marked as local-only will be passed to the remote server. Any packets received via an upstream connection will be considered "local-only" as well to prevent looping of packets. Upstream connections may be bidirectional or receive-only. If receive-only connections are specified, bidirectional connections are not allowed. This is to prevent a single server putting unnecessary load on remote servers by maintaining multiple connections to various remote servers. Only one bidirectional upstream connection may exist in any server.

A second type of "upstream connection" is the server-to-server "connection". "Connection" is in quotes because a connection in the traditional sense is not created between servers. Instead, the servers send and receive UDP packets in the same fashion as normal upstream connections with the addition that packets injected via local-only ports are also passed (but not packets from other server-to-server connections). Each UDP packet contains one and only one APRS packet. The server-to-server connections are for core server use only and do not serve any useful function for lower tier servers. For server-to-server connections to be properly implemented, all servers in the core group must have a server-to-server connection defined for every other server in the core group.

There are multiple types of connections supported from remote clients. Older APRS server software had predefined port numbers for the various kinds of client ports; javAPRSSrvr eliminated this restriction to better conform to individual system requirements. javAPRSSrvr also modified or removed certain types of client ports to improve APRS-IS integrity. There are also new types of client ports only offered by javAPRSSrvr to better serve the amateur radio community.

Remote clients can connect to javAPRSSrvr via TCP or UDP. A TCP connection can also support a UDP feed from javAPRSSrvr to reduce loading on the server (no TCP overhead). UDP ports are receive-only and can either receive from predefined IP addresses or from any remote client if the UDP packet contains a login line in addition to the APRS packet.

TCP ports are three basic types. The first type is an HTTP receive-only port which accepts POST/PUT requests where the data is a login line followed by the APRS packet. The HTTP port also supports the WebSocket protocol for bidirectional operation emulating the restricted feed TCP port. The second type of TCP port is the "full feed" port. This port sends all non-duplicate packets to the client after the client has logged in. The third type of TCP port is the "restricted feed" port. At a minimum, this port will only send to the client message packets addressed to the client and associated posit (posit from the station sending the message). The "restricted feed" port can also maintain a "recently heard" list to support IGates which gate packets heard on RF the server. This "IGate port" type will also send to the client message packets addressed to "recently heard" stations and associated posits, and the server will send to the client any packets from the "recently heard" stations that were injected directly into APRS-IS.

All non-HTTP TCP "feed" ports may be "send-only" or bidirectional. If the TCP port provides historical packets, it will always be "send-only" to reduce the possibility of looping.

The "restricted feed" ports may also support server adjuncts such as APRSFilter. APRSFilter provides "filters" to add packets which meet either predefined or user-defined criteria to the packets sent to the client. Since this is an additive feature (adds packets to the stream), the full feed ports do not support this feature or any other server adjunct that might be created. No filtering occurs on packets sent to the server from the client.

All operational adjuncts such as email, whois, database, and IGate adjuncts appear as a client attached to javAPRSSrvr. This facilitates multiple adjuncts and port types to run concurrently in a single javAPRSSrvr instance. It also allows each adjunct to specify what type of “feed” it wants to receive and if it wants the “connection” to be bidirectional or not.

javAPRSSrvr has built-in header filtering and certain types of packet filtering. Header filtering analyzes the header and puts all received headers into standard TNC-2 format. Callsigns must be no more than 9 ASCII alphanumeric characters and no less than 3 ASCII alphanumeric characters. All TNC header information such as port number, timestamp, <>, [], and spaces are eliminated from the packet leaving only a clean header for retransmission to clients. The header filtering is used by javAPRSSrvr to accurately do duplicate packet checking based on the Source call, Destination call, and the packet payload. A SSID, if present, must be only 1 or 2 ASCII alphanumeric characters separated from the callsign by a single hyphen and the total length of the callsign, hyphen, and SSID may not exceed 9 characters. Zero (0) is not a valid SSID as it is implied by no SSID being present.

javAPRSSrvr implements the q-construct which is used by javAPRSSrvr, aprsD, and possibly other servers and IGates to perform 2 major functions: reduce/eliminate loops and identify point-of-entry (POE) for packets. The q-construct is always a 3 character sequence in the packet path beginning with a lower-case “q”. The q-construct is NEVER to be seen on RF.

When a station connects to javAPRSSrvr and the connection is accepted, they receive the following comment line:

javAPRSSrvr 4.0.1b01

If the station's connection is rejected, it will receive one of the following comment lines depending on the reason for the rejection:

javAPRSSrvr 4.0.1b01 Port full.
javAPRSSrvr 4.0.1b01 Port unavailable.

The rejection comment line will be sent before disconnection occurs.

When that station logs in, they will receive the following line:

logresp callssid unverified, server userCall

callssid is the logged in call. If the passcode supplied by the client is valid and the port supports receiving packets from the client, unverified is replaced with verified. userCall is the "callssid" for the server.

If there is a server adjunct installed for that port and the login contains a command for the adjunct, the following will be sent by javAPRSSrvr:

logresp callssid unverified, server userCall, adjunct "command param" sa_response

This is the same as before, with the adjunct info appended to the end of the line. "command param" is the command/parameter combination received by javAPRSSrvr. It is enclosed in double quotes to simplify readability. sa_response is the response issued by the adjunct.

Section 3 - Configuration Parameters

The configuration properties reside in properties files for each client adjunct, server adjunct, and port. The main properties file is called javaprssrvr.properties by default. You can use any text file for the main properties file if you pass the name into javAPRSSrvr as a command line parameter.

The property names are not case sensitive but the values can be. Defaults are shown below.

NOTE: UNLESS YOU REQUIRE A SETTING OTHER THAN THE DEFAULT, DO NOT INCLUDE ANY PARAMETERS WITH DEFAULT SETTINGS.

List parameters (L) may be defined on the property line or may be defined in a text file with the suffix .lst. If defined on the line, each entry is separated by a semicolon. If defined in a file, each entry is put on a separate line in the .lst file and the file name is the property value. Do not put blank lines in the file. For instance, this could be a definition for ListProperty (example only):

ListProperty=first.aprs.net:1313;second.aprs.net:1313

Or you could have the following 2 lines in a file named hubs.lst:

first.aprs.net:1313
second.aprs.net:1313

with ListProperty=hubs.lst

Properties preceded by a (M) are unchangeable and should not be included in your properties files. They are included in the descriptions below to indicate what common properties are available vs. those that have been forcibly overridden.

javAPRSSrvr sets the default time zone to UTC and the default locale to US English. This ensures proper operation regardless of country it is run in. Because of this, all time values are assumed to be UTC unless otherwise indicated and all number formats use the period (.) as a decimal point and zero (0) as the zero character. The starting time zone is recorded in the Java system properties as orig.timezone so clients (such as WxNowGen) can access the local time zone for conversions.

Server-wide Properties (javaprssrvr.properties)

ServerCall=

This is the callsign-SSID used for logging into other servers and identifying itself in the q algorithm.

stderr=err.%g.log

This is the path of the file where all error messages are sent. The log utilizes the Java Logger utilities which include a rotating to a new file once MaxLogSize is reached. The number of log files created is controlled by MaxLogs. All log files will have .N replacing %g where N is the log number. Zero is always the active log. The active default log would be err.0.log More information of formatting the file name can be found at https://docs.oracle.com/javase/8/docs/api/java/util/logging/FileHandler.html

MaxLogSize=20000

Maximum log file size before rotating.

MaxLogs=3

Maximum number of log files to create.

.level=INFO

Sets logging level for application. Set to ALL to see all exceptions including trace messages.

GCTimer=60

This sets when memory cleanup should occur (in seconds).
It defaults to 60 seconds as this seems to be a good number to prevent the garbage collector from taking too many CPU cycles at one time. A setting of zero or less will cause no GC timer to be set leaving memory “garbage collection” to the JVM.

DupeMaxPriority=false

If set to true, the DupeProcessing thread is given maximum priority. This can be used in large server environments to give priority to the packet processing thread over client threads.

MaxBufferSize=512

This sets the maximum packet size that will be considered valid.

ServerDest=APJS40

This is the “TOCALL” used when sending message responses. This should not be changed.

NetworkID=TCPIP

Used to identify direct attached validated clients.

UnverifiedNetworkID=TCPXX

Used to identify direct attached non-validated clients.

qNetwork=A

APRS-IS is the “A” q network.

AllowedQNetworks=A

If set, this is a string of capital letters between A and Z designating allowed q networks. For instance, it could be ACF.

AlwaysqTrace=

(L)These callsign-SSIDs as origin callsigns will cause the q algorithm to alter the q construct to a qAI construct. This causes tracing via the q construct from this server on. This should only be used in rare instances where a consistent APRS-IS loop is suspected.

BlockedCalls=

(L)These callsigns (SSIDs must not be included in list) are blocked from logging in and all packets from these callsigns are dropped. Always included in the list are SERVER, NOCALL, N0CALL, USERLIST, JAVAMSG, JAVATITLE, JAVATITL2, and KIPSS. The callsigns are case-insensitive.

BlockedStations=

(L)These callsigns-SSIDs are blocked from logging in and all packets from these stations are dropped. This is more selective than BlockedCalls to prevent problem stations from participating in the network. In essence, ServerCall is always on this list.

BlockedToCalls=

(L)These callsigns (SSIDs must not be included in list) found as the TOCALL will drop the packet. The callsigns are case-insensitive.

BlockedPrefixes=

(L)These prefixes are used to filter out non-amateur radio origin callsigns. The prefixes are case-insensitive. Prefixes can be 4 to 8 characters and must NOT be used to filter out legitimate amateur radio stations.

BlockedCallsList=

This should be the filename of a text file that can be changed while javAPRSSrvr is running. The file, if it exists and it is ok if it doesn't, consists of a single callsign per line. Callsigns are added/deleted from the dynamic blocked calls list as they are added and deleted from this file. This should be used SPARINGLY or not at all. This does not alter the BlockedCalls list. Callsigns are case-insensitive. If includes path, must be in same folder as BlockedStationsList and BlockedToCallsList if they are used.

BlockedStationsList=

This should be the filename of a text file that can be changed while javAPRSSrvr is running. The file, if it exists and it is ok if it doesn't, consists of a single station per line. Stations are added/deleted from the dynamic blocked stations list as they are added and deleted from this file. This should be used SPARINGLY or not at all. This does not alter the BlockedStations list. If includes path, must be in same folder as Blocked…List files if they are used.

BlockedToCallsList=

This should be the filename of a text file that can be changed while javAPRSSrvr is running. The file, if it exists and it is ok if it doesn't, consists of a single callsign per line. Callsigns are added/deleted from the dynamic blocked TOCALLs list as they are added and deleted from this file. This should be used SPARINGLY or not at all. This does not alter the BlockedToCalls list. Callsigns are case-insensitive. If includes path, must be in same folder as Blocked…List files if they are used.

BlockedPrefixesList=

This should be the filename of a text file that can be changed while javAPRSSrvr is running. The file, if it exists and it is ok if it doesn't, consists of a single prefix per line. Prefixes are added/deleted from the dynamic blocked prefix list as they are added and deleted from this file. This should be used SPARINGLY or not at all. This does not alter the BlockedPrefixes list. Prefixes are case-insensitive. If includes path, must be in same folder as Blocked…List files if they are used.

PassUnverifiedPackets=false

If true, packets from unverified remote (via other server's) client will be passed. If false, UnverifiedPositsWx is set to false as no packets from any unverified client will be passed. **Must be left at false for APRS-IS.**

UnverifiedPositsWx=false

If true, weather and position packets only will be accepted from unverified logins. If false, no packets will be accepted from unverified logins. **Must be left at false for APRS-IS.** May be set to true with PassUnverifiedPackets for CWOP.

ValidateCallsigns=false

If true, logins from non-amateur radio callsigns will not be validated.

DupeCheckDisable=false

Disable duplicate checking for **standalone IGate configuration**. **Do NOT disable on open access servers!**

LoginTimeout=30

How long to wait (seconds) on a TCP connection for a login sequence before disconnecting.

History=30

Time (minutes) to hold non-message packets for distribution via history TCP ports. If set to -1, no history will be maintained.

MaxTotalConnections=50

Maximum number of concurrent remote inbound connections allowed on this server. Currently used for TCP ports.

ServerAdjuncts=

(Depricated)(L)List of server adjunct jar files. Server adjuncts are loaded using ServiceLoader and get their properties from javaprssrvr.properties. For instance, if ServerAdjuncts=APRSFilter.jar, the FilterAPRS class uses properties prefixed with FilterAPRS. as its properties.

LogPorts=

(L)List of log port property files.

SerialPorts=

(L)List of serial interface property files.

Clients=

(L)List of client property files (client adjuncts such as APRSIGate).

ListenerPorts=

(L)List of listener port property files (TCP & UDP listener ports).

UpstreamServers=

(L)List of upstream dialer property files. Class= is not needed in the upstream server files as all use the UpstreamDialer class.

Subclass Properties (common)

These properties are used to tell javAPRSSrvr what class to load and where to find the class and (if necessary) binary files for that class.

Class=

(Depricated) This property must be defined in each properties file except javaprssrvr.properties. It defines the class that will be created using this property file.

ClassPath=

(Depricated)(L)This property is required for any class that is not in the core group of .JAR files. This is either the .JAR file name(s) or the directory(ies) that holds the .CLASS files required.

Packages=

(Depricated)(L)This property can be used to specify third-party packages required by an internal client or serial interface. For example, the JDBC driver for DBGate would be specified here. Only specify the file name prefix; do not include version number or ".jar". javAPRSSrvr will chose the latest release based on file name (if it has a version number in the name).

LibraryPath=

(L)This property is required for any class that must load an external binary. This is the path of the directory(ies) that holds the external binary(ies).

Log Port Properties

Log output is a tab delimited line of text as follows:

#1 #2 #3 #4 #5
#1: < or > indicating outbound or inbound respectively
#2: Long value – Java timestamp (milliseconds since midnight, January 1, 1970 UTC)
#3: Login – callsign-SSID of the client
#4: Client-specific information – This is created by the specific client module.
#5: Packet contents – This is the raw packet or TNC formatted packet if available.

Class=

Set to appropriate log port type. Log port class name is one of the following (prior compatibility maintained with full class name definitions):

- **StandardLogPort** (logs all non-error packets)
- **ErrorLogPort** (all error packets)
- **DupeLogPort** (all duplicate packets)
- **ConnectLogPort** (all remote connections)

AllowIP=

(L)IPv4 and IPv6 addresses allowed to connect to this log port. Connections are only allowed from listed IP addresses.

NICAddress=

Local network interface address if system is multihomed.

NICPort=-1

Local TCP port to listen on.

Upstream Dialer Properties

These properties define the “dialing out” to remote servers to receive and send packets to a higher level in the APRS-IS hierarchy. If write-only desired, see Send-only Upstream Properties

NICAddress=

Local network interface address if system is multihomed.

BufferSize=512

Size in bytes of maximum packet length.

UDPPort=-1

UDP port to use if UDP feed from remote server desired. This port may be shared with other Upstream Dialers.

UpstreamServers=

(L)Upstream server address:port list. Address may be any type of IP address including a DNS name. Port is the remote port number to connect to. To connect to a WebSocket server (see HTTP TCP Port), use a WebSocket URI (ws://FQDN[:port]/ or wss://FQDN[:port]/). The old style ws:FQDN[:port] is still supported but cannot be used to go to subfolders or implement secure connections.

Timeout=30

Timeout (seconds) of inactivity before disconnecting.

DataTimeoutCheck=(see description)

When true, every TCP line received will cause a check to see when the last valid packet was received (could be via UDP). If the time is greater than Timeout seconds, the upstream connection will be closed. The default is true if there is a UDPPort > 0 and javAPRSSrvr was successful in opening the UDP port. The default is false if data is only expected via TCP. This allows filtered feeds to be provided using TCP without fear of connection loss due to low data throughput yet ensuring full UDP feeds are being received properly.

Bidirectional=false

If true, log in with valid passcode and pass packets to remote server. Only one upstream connection is allowed if bidirectional=true.

ServerCommand=

Any text will be added to the upstream login to facilitate using limited feeds from the upstream server.

LogErrors=false

Log discarded received packets to rcvpacketerrN.log file (N is auto-generated).

Base64Encode=false

Login line is sent using Base64 encoding. Do not use if upstream server(s) is not javAPRSSrvr.

Listener Port Properties (common)

The port configuration properties are found in the Client Properties. These properties define the local port that is listened to and its capabilities. Supported classes are (prior compatibility with full class names the Class property is maintained):

- **SrvrSrvrPort**
- **StatusListenerPort**
- **TCPListenerPort**
- **UDPListenerPort.**

PortType=

Replaces the deprecated Class to definitively designate what type of port listener is desired. (case-insensitive)

- **SrvrSrvrPort -> Srvr-Srvr**
- **StatusListenerPort -> Status**
- **TCPListenerPort -> TCP, HTTP, HTTPS**
- **UDPListenerPort -> UDP**

NICAddress=

Local network interface address if system is multihomed.

NICPort=-1

Local port to listen on.

Description=

If present, this description and port will be included in the general XML status page. This can be used to selectively include/exclude ports from the general XML status page.

Client Properties (common)

These properties are common for all port types, client adjuncts, etc. Each client type may override any or all of these properties.

ClientType=

Internal clients use this in lieu of the deprecated Class property (Class property is still supported).

Upstream=false

Indicates this port feeds an upstream server. This implies FullFeed=true as all packets except local-only and packets received via an upstream client will be sent upstream. Also implies packets received from the upstream server are LocalOnly=true

FullFeed=false

Indicates all packets except those received from this port should be sent to the client.

ReadOnly=false

Indicates that no packets will be accepted from the client.

LocalOnly=false

Packets received from this client will only be sent to clients that are not upstream or are connected via server-to-server connections.

MessageHoldTime=30

Time (minutes) to hold a message packet reference to send an associated posit to the client.

LastHeardTime=-1

Time (minutes) to hold station references to stations received from client (last heard). This facilitates support of IGate clients so messages can be sent “last heard” stations and packets from stations both connected directly to APRS-IS and RF will have the directly connected packets sent to the IGate (helps lower RF usage). This defaults to 60 for non-full-feed ports.

ServerAdjuncts=false

This enables server adjunct command(s) for the client (may be overridden by client).

ServerCommand=

This sets server adjunct command(s) for the client (may be overridden by client) (sets ServerAdjuncts=true if present)

FixedCommand=false

If true, the ServerCommand is forced. If false, enables LoginCommands.

LoginCommands=

This is the file name containing station-specific server commands. This file is a properties file with the callsign-SSID as the property name and the server adjunct command(s) as the property value.

AllowedList=

(L)If set, this is a list of the **only** callsign-SSIDs allowed to log into this port.

Internal Client Properties

These properties are global to all interactive (able to send to APRS-IS) internal clients (client adjuncts). Messaging capable clients support ?APRST, ?APRSP, ?APRSS, and other queries if respective packet type is defined.

(M)StationCall=

Callsign-SSID for this client. Each client callsign-SSID MUST be unique world-wide as APRS-IS dupe processing requires it.

StatusText=

If defined, a status packet with this text will be sent every 20 minutes. If client is messaging capable, this packet will be sent in response to a ?APRSS query.

TelemetrySource=

If defined, telemetry properties will be read and the message and telemetry packet tasks will be started. If this property starts with “file,”, the file name (can be full path) will be monitored for changes and the telemetry data will be updated upon change. Format of the telemetry.txt file (default name) is the exact same as the APRS telemetry data packet. The sequence number must be numeric. It will try to read telmessages.txt in the same folder as the telemetry file and will set the message properties if found and containing the various messages.

telBITSMsg=

This is the BITS telemetry message data sans “BITS.”.

telEQNSMsg=

This is the EQNS telemetry message data sans “EQNS.”.

telPARMMsg=

This is the PARM telemetry message data sans “PARM.”.

telUNITMsg=

This is the UNIT telemetry message data sans “UNIT.”.

Internal Client Posit Properties

If a valid posit is defined (either using GPS or manually), the posit will also be sent in response to a ?APRSP or ?PING? query.

Latitude=

This specifies the latitude of the station.
The latitude is specified in decimal degrees, south is negative.

Longitude=

This specifies the longitude of the station.
The longitude is specified in decimal degrees, west is negative.

Compressed=false

This specifies if posits are to use compressed format.

Ambiguity=-1

This specifies how many digits of are considered valid (-1 = all).

Altitude=

Altitude in feet MSL of station.

DataExtension=

This is the 7 byte extension for PHG, etc.

GPSIntfName=

If defined, this is the same name as the SerialInterface IntfName that is connected to a NMEA GPS. If defined, this overrides the above information with information from the GPS.

Symbol=

This specifies the APRS symbol to use in the posit.
I& is the preferred symbol for an IGate.

Comment=

This specifies text to follow the posit data.

PositInterval=20

This specifies the position beacon rate in minutes.

Delay=30

This specifies how long to initially wait before beaconing in seconds.

TCP (non-HTTP) Port Properties

These properties are included in the TCP port listener properties file. They include port listener properties and client properties. Port listener and client properties not listed are the same as the (common) settings shown above.

Class=

(Depricated) Set to TCPListenerPort

PortType=

Set to TCP

(M)Upstream=false

TCP listener ports cannot be upstream.

History=false

If this is set to true, ReadOnly is set to true and LocalOnly is set to false.

UDPMandatory=false

Client logging in must specify a UDP receive port when logging on if true. Sets offerUDP to true.

OfferUDP=false

Allow clients to specify a UDP port to receive packets on if true. The packets will be sent via the UDP port at the same port number defined in NICPort (see Port Listener Properties).

MaximumConnects=0

Maximum number of concurrent connections allowed on this port.
If MaximumConnects < 0, the port always allows a connection.
If MaximumConnects = 0, the port looks at the total remote connections compared to MaxTotalConnections.
If MaximumConnects > 0, the port will use MaximumConnects for its test using only connections to this port.

AlwaysAllowIP=

IP addresses which will always be allowed to connect regardless of number of connections. DNS names may be used but will be resolved only upon startup.

AlwaysBlockIP=

IP addresses which will always be blocked. DNS names may be used but will be resolved only upon startup. Because this is only read at startup and because it allows the connection before disconnecting the client, I recommend using your firewall settings to block IP addresses instead.

BufferSize=512

This is the maximum allowed packet size.

KeepAliveInterval=20

This is the quiet interval (seconds) on the TCP connection before a keep-alive string is sent to the client.

KeepAliveMessage=

The keep-alive message sent to the client consists of the comment line character '#' followed by "javAPRSSrvr 4.0.1b01" followed by a UTC timestamp followed by the server callsign-SSID followed by the TCP port number followed by this KeepAliveMessage text.

HTTP TCP Port Properties

These properties are included in the TCP port listener properties file. They include port listener properties and client properties. Port listener and client properties not listed are the same as the (common) settings shown above.

Class=

(Depricated) Set to TCPListenerPort

PortType=

Set to HTTP or HTTPS

HTTPClient=true

Defaults to true if using PortType HTTP or HTTPS. Set to true for ports that accept HTTP connections if using Class..

PKCS12Cert=

Set to the PKCS12 certificate/key pfx file for https/wss connections. If the certificate is a self-signed certificate, you must set -Djavax.net.ssl.trustStore=path/to/cacerts.jks to point to your CA keystore. This is also necessary if you are trying a secure connection upstream to a self-signed certificate.

certpwd=changeit(Base64 encoded)

If PKCS12Cert is set, you can set this to the password used by the pfx file by running the password through pwdenc.jar

(M)Upstream=false

TCP listener ports cannot be upstream.

(M)History=false

HTTP ports do not send packets to the client. If WebSocket client and history protocol, this is set to true.

(M)FullFeed=false

HTTP ports do not send packets to the client.

(M)ServerAdjuncts=false

HTTP ports do not send packets to the client. If WebSocket client, this is set to true.

(M)MessageHoldTime=-1

HTTP ports do not send packets to the client. If WebSocket client, this is set to default if not history protocol.

(M)LastHeardTime=-1

HTTP ports do not send packets to the client. If WebSocket client, this is set to default if not history protocol.

AlwaysBlockIP=

IP addresses which will always be blocked. DNS names may be used but will be resolved only upon startup. Because this is only read at startup and because it allows the connection before disconnecting the client, I recommend using your firewall settings to block IP addresses instead.

BufferSize=512

This is the maximum allowed packet size.

UDP Port Properties

This port ignores all other properties than those shown on this page.

Class=
(Depricated) Set class to UDPListenerPort

PortType=
Set to UDP

BufferSize=512
Maximum packet length in bytes.

AlwaysAllowIP=
IP addresses which will always be allowed to send regardless of valid login.

AlwaysBlockIP=
IP addresses which will always be blocked. DNS names may be used but will be resolved only upon startup.

Server to Server Port Properties

This “Port Listener” is used to create a hub matrix using UDP as the transport mechanism.

Class=

(Depricated) Set class to SrvrSrvrPort

PortType=

Set to Srvr-Srvr

(M)Upstream=true

These ports are considered “upstream” to the other servers.

(M)LocalOnly=true

Packets received via this port will not be sent out to other servers.

(M)History=false

No history packets.

(M)FullFeed=true

All non-upstream packets are passed to the other servers. Note this includes LocalOnly packets not from upstream ports.

(M)ReadOnly=false

Port is bidirectional.

(M)ServerAdjuncts=false

FullFeed=true so no server adjunct is used.

(M)MessageHoldTime=-1

FullFeed=true.

(M)LastHeardTime=-1

FullFeed=true.

ServerList=

(L)This is a list of remote servers. The format of each entry is address,port,callsign-SSID. “address” is either an IP address or DNS name which is resolved at startup, “port” is the remote UDP port that packets will be sent to and received from, and “callsign-SSID” is the remote server’s callsign-SSID. This allows for better loop detection. The list may contain this server’s information which will be ignored based on callsign-SSID. This is done so core sysops can use a single list file (core.lst, for instance) amongst all core servers.

BufferSize=512

This is the maximum allowed packet size.

Status Port Properties

The status port listener provides an XML status page to the HTTP requestor. The XML content is described in further detail later in this document. There are two types of status ports: general and detail. All status ports provide the base (general) information. If DetailXSLName exists in the properties file, detail information will be included in the returned page as well.

There is a favicon.ico included in the APRSSrvr.jar file. If you wish to use your own, place your favicon.ico file in the same directory you started javAPRSSrvr in (user.dir).

Class=

(Depricated) Set to StatusListenerPort

PortType=

Set to Status

XSLPath={Java system property user.dir}

This is the path to the XSL format files, if they exist. XSL files can be used to transform the XML content to HTML type presentation format.

XSLName=

This is the XSL file to be used for formatting the general XML page. Note that this is just the file name so it can be served up properly in the XML page and by javAPRSSrvr.
Use **internal** if you wish to use the default general.xsl file contained in aprssrvr.jar.

DetailXSLName=

This is the XSL file to be used for formatting the detail XML page. Note that this is just the file name so it can be served up properly in the XML page and by javAPRSSrvr.
If this property exists (does not have to have a value) in the properties file, the port will provide a detail XML file if requested by the remote client. The detailed page is requested by requesting **/detail** from javAPRSSrvr. To request the detailed raw XML page, use **/detail.xml**
Use **internal** if you wish to use the default detail.xsl file contained in aprssrvr.jar.

PageCache=

Set this to a path to a directory (relative or absolute) that contains only files that will be available to status page HTML such as images. Because every file in the folder will be cached in memory, the fewer, the better. The operating system will be used to identify the MIME type of the file and all identified files will be cached and noted in the log. This folder will be monitored for changes.

HTTPClient=false

When set to true, this status port can also support HTTP send-only and WebSocket connections. Add a Description (see HTTP TCP Port Properties) to advertise availability.

Multicast Port Properties (Internal Client)

All **client** port properties apply.

Class=
(Depricated) Set to net.ae5pl.aprssrvr.MulticastPort

ClientType=
Set to Multicast

(M)Upstream=false
Multicast ports cannot be upstream.

(M)LocalOnly=false
Multicast ports do not accept packets.

(M)ReadOnly=true
Multicast ports do not accept packets.

NICAddress=
IP address of local NIC if server is multihomed.

NICPort=-1
Port to use to send from and to.

MulticastAddress=
Multicast address to send to.

TTL=0
Time to live – number of hops packet will “live”.

Send-only Upstream Properties

This client type is intended to be used in low bandwidth environments such as SatGates which are receive-only from RF. This class makes use of either the HTTP or UDP packet submission port available on later versions of javAPRSSrvr with a validated login.

Class=

Set to UniUpstreamDialer (prior full package format accepted)

(M)Upstream=true

These ports are considered “upstream” to the other servers.

(M)History=false

No history packets.

(M)FullFeed=true

All non-upstream packets are passed to the other servers.

(M)ReadOnly=true

Port is bidirectional.

(M)ServerAdjuncts=false

FullFeed=true so no server adjunct is used.

(M)MessageHoldTime=-1

FullFeed=true.

(M)LastHeardTime=-1

FullFeed=true.

NICAddress=

IP address of local NIC if server is multihomed.

NICPort=-1

Port to use to send from.

Protocol=UDP

UDP or HTTP are the options.

RemoteHost=

IP or DNS address of remote server. Note this is resolved at startup.

RemotePort=8080

Port to send to on the remote server (8080 is de facto standard).Defaults to 8443 if Secure=true.

Base64Encode=false

If true, login line is sent using Base64 encoding (for javAPRSSrvr upstream servers). For HTTP, the login line is sent in the HTTP header “Authorization”, APRS-IS type, instead of in the content.

HTTP-specific Properties

Compress=false

gzip encode data if true.

URL=

(new in 4.3.3b31) If present and protocol is not UDP, will be used to connect to remote host. This allows connecting to a web server that requires a path other than / to post a packet to. If set, RemoteHost, RemotePort, and Secure are ignored.

Secure=false

Use HTTPS if set to true.

AuthType=

Sets authorization type. Do not set if using a login line in the body text (non-javAPRSSrvr updstream servers). For javAPRSSrvr upstream servers, APRS-IS and Basic are valid types.

Serial Interface Properties (common)

The serial interface properties are part of the base APRS server architecture to allow sharing of serial interfaces between local clients. For instance, NSRDigi and APRSIGate can share the same TNC without knowledge of each other or any direct interaction.

InterfaceType=

Set to individual interface type (replaces deprecated Class property). Case-insensitive.

IntfName=

This is the name used by the various clients to access this serial interface. This can be any alphanumeric string.

InitFile=

The contents of this file are sent to the serial interface as-is at start-up.

TXDELAY=

Format is port,ms;port,ms. The port is the TNC port number, starting at 0. Ms is the delay in 10 ms increments. This generates a KISS command that is sent after the contents of InitFile to the TNC.

P=

Format is port,p;port,p. The port is the TNC port number, starting at 0. P is a persistence number between 0 and 256. This generates a KISS command that is sent after the contents of InitFile to the TNC.

SLOTTIME=

Format is port,ms;port,ms. The port is the TNC port number, starting at 0. Ms is the delay in 10 ms increments. This generates a KISS command that is sent after the contents of InitFile to the TNC.

TXTAIL=

Format is port,ms;port,ms. The port is the TNC port number, starting at 0. Ms is the delay in 10 ms increments. This generates a KISS command that is sent after the contents of InitFile to the TNC.

FULLDUPLEX=

Format is port,fd;port,fd. The port is the TNC port number, starting at 0. If fd = 1, full duplex; fd = 0, half duplex. This generates a KISS command that is sent after the contents of InitFile to the TNC.

RFSpeed=1200

This is used to pace output to the serial interface.

SharedTransmit=true

Sets whether clients will see each other's packets.

KISSMode=false

If true, a number of optimizations are implemented at the serial interface to enhance KISS TNC support.

TCPPorts=

(L)Either IP:port or port list. Each defined port opens a listener which provides bidirectional access to the serial interface.

File Interface Properties

The File interface provides a generic Random Access File interface to a serial port. Because this interface is dependent on the OS providing a generic file interface for a serial port, its performance will vary.

Class=

(Deprecated) Set to net.ae5pl.serintf.FileIntf

InterfaceType=

Set to File

ConfigFile=

Batch/shell/executable file that configures the serial port. This file will be run before javAPRSSrvr connects to the port.

SerialFileName=

OS format file name for serial port.

TCP Interface Properties

The TCP interface provides a generic “serial” access to a TCP port.

Class=

(Deprecated) Set to net.ae5pl.serintf.TCPIntf

InterfaceType=

Set to TCP

NICAddress=

IP address of NIC to use if multihomed.

SerialIPAddress=

IP or DNS address of remote TCP port. This is resolved at start-up.

SerialIPPort=-1

Remote TCP port number.

Section 4 – Configuration Recommendations/Requirements

Use default settings for as many properties as possible by leaving the properties out of the properties file. Do not adjust timing parameters as they may affect APRS-IS performance.

Core servers **cannot** have any upstream connections defined. Each core server must have all of the other core servers defined in Server to Server definitions (note it is now possible to have multiple server to server port definitions, even reusing the same UDP port).

Non-core servers **only** use upstream connections APRS-IS. Non-core servers should **never** use server to server ports.

ServerCall must be a unique identifier in APRS. Using a callsign with an SSID provides clarity and may be required.

The included example properties files provide a foundation for configuring your server. Remember, this software is designed to be started and left alone. **Modifications to configuration, program files, etc. will require a restart of the software for the changes to take effect.**

Java Properties Format Description

This information is copied from the Oracle Javadoc description for Properties.load():

> Reads a property list (key and element pairs) from the input stream. The stream is assumed to be using the ISO 8859-1 character encoding; that is each byte is one Latin1 character. Characters not in Latin1, and certain special characters, can be represented in keys and elements using escape sequences similar to those used for character and string literals (see §3.3 and §3.10.6 of the *Java Language Specification*). The differences from the character escape sequences used for characters and strings are:
>
> - Octal escapes are not recognized.
> - The character sequence `\b` does *not* represent a backspace character.
> - The method does not treat a backslash character, `\`, before a non-valid escape character as an error; the backslash is silently dropped. For example, in a Java string the sequence `"\z"` would cause a compile time error. In contrast, this method silently drops the backslash. Therefore, this method treats the two character sequence `"\b"` as equivalent to the single character `'b'`.
> - Escapes are not necessary for single and double quotes; however, by the rule above, single and double quote characters preceded by a backslash still yield single and double quote characters, respectively.
>
> An `IllegalArgumentException` is thrown if a malformed Unicode escape appears in the input.
>
> This method processes input in terms of lines. A natural line of input is terminated either by a set of line terminator characters (`\n` or `\r` or `\r\n`) or by the end of the file. A natural line may be either a blank line, a comment line, or hold some part of a key-element pair. The logical line holding all the data for a key-element pair may be spread out across several adjacent natural lines by escaping the line terminator sequence with a backslash character, `\`. Note that a comment line cannot be extended in this manner; every natural line that is a comment must have its own comment indicator, as described below. If a logical line is continued over several natural lines, the continuation lines receive further processing, also described below. Lines are read from the input stream until end of file is reached.
>
> A natural line that contains only white space characters is considered blank and is ignored. A comment line has an ASCII `'#'` or `'!'` as its first non-white space character; comment lines are also ignored and do not encode key-element information. In addition to line terminators, this method considers the characters space (`' '`, `'\u0020'`), tab (`'\t'`, `'\u0009'`), and form feed (`'\f'`, `'\u000C'`) to be white space.

If a logical line is spread across several natural lines, the backslash escaping the line terminator sequence, the line terminator sequence, and any white space at the start the following line have no affect on the key or element values. The remainder of the discussion of key and element parsing will assume all the characters constituting the key and element appear on a single natural line after line continuation characters have been removed. Note that it is *not* sufficient to only examine the character preceding a line terminator sequence to see if the line terminator is escaped; there must be an odd number of contiguous backslashes for the line terminator to be escaped. Since the input is processed from left to right, a non-zero even number of 2*n* contiguous backslashes before a line terminator (or elsewhere) encodes *n* backslashes after escape processing.

The key contains all of the characters in the line starting with the first non-white space character and up to, but not including, the first unescaped `'='`, `':'`, or white space character other than a line terminator. All of these key termination characters may be included in the key by escaping them with a preceding backslash character; for example,

```
\:\=
```

would be the two-character key `":="`. Line terminator characters can be included using `\r` and `\n` escape sequences. Any white space after the key is skipped; if the first non-white space character after the key is `'='` or `':'`, then it is ignored and any white space characters after it are also skipped. All remaining characters on the line become part of the associated element string; if there are no remaining characters, the element is the empty string `""`. Once the raw character sequences constituting the key and element are identified, escape processing is performed as described above.

As an example, each of the following three lines specifies the key `"Truth"` and the associated element value `"Beauty"`:

```
 Truth = Beauty
        Truth:Beauty
 Truth                  :Beauty
```

As another example, the following three lines specify a single property:

```
 fruits                           apple, banana, pear, \
                                  cantaloupe, watermelon, \
                                  kiwi, mango
```

The key is `"fruits"` and the associated element is:

```
"apple, banana, pear, cantaloupe, watermelon, kiwi, mango"
```

Note that a space appears before each `\` so that a space will appear after each comma in the final result; the `\`, line terminator, and leading white space on the continuation line are merely discarded and are *not* replaced by one or more other characters.

As a third example, the line:

```
cheeses
```

specifies that the key is `"cheeses"` and the associated element is the empty string `""`.

Section 5 - Installation Instructions

Java 8 through 10 JVM's

1. Install the latest JRE or JDK your OS. Some JREs are already optimized for server operation. Server optimization is not required for IGates and other client-type installations although I still recommend it if available.

2. Place javAPRSSrvr.jar and the Properties files into a unique directory. Create a sub directory called lib and place all other JAR files you are going to use in that directory.

3. Modify the Properties files to meet your unique requirements. Do not use the ClassPath property for internal clients or serial interfaces.

4. Run javAPRSSrvr using your Java VM. This is done with the following command line in the directory you created above (Windows format; 'ix uses colons ':' in place of semicolons ';').

   JDKPath\bin\java -server -cp lib/* javaprssrvr.SrvrMain

   The –server switch should not be used if you do not have a server JVM installed. Other switches are available for various versions of Java and for various manufacturers. Please refer to the Java manufacturer's web site for more information on the various command line switches that can be used to fine tune your Java Virtual Machine (JVM).

Java 11 (and later) JVM's

1. Install the latest JRE or JDK your OS. Some JREs are already optimized for server operation. Server optimization is not required for IGates and other client-type installations although I still recommend it if available.

2. Place javAPRSSrvr.jar and the Properties files into a unique directory. Create a sub directory called lib and place all other JAR files you are going to use in that directory.

3. Modify the Properties files to meet your unique requirements. Set GCTimer=0 as the G1 garbage collector is the default. Do not use the ClassPath property for internal clients or serial interfaces.

4. Run javAPRSSrvr using your Java VM. This is done with the following command line in the directory you created above (Windows format; 'ix uses colons ':' in place of semicolons ';')..

   JDKPath\bin\java -server -XX:-UsePerfData --add-modules ALL-MODULE-PATH -p lib -m javaprssrvr.main

   The –server switch should not be used if you do not have a server JVM installed. Other switches are available for various versions of Java and for various manufacturers. Please refer to the Java manufacturer's web site for more information on the various command line switches that can be used to fine tune your Java Virtual Machine (JVM).

   If you are using EmailGate or WhoIsGate, you will need to add jakarta.activation and jakarta.xml.bind modules to your lib directory (see specific user guides) and use the 4.4.1 versions to work with Jakarta EE 9 and later.

Java 19 (and later) JVM's (4.3.3b15 and later)

Java 19 introduced virtual threads. Because javAPRSSrvr uses individual threads for receive and transmit on TCP connections as well as threads to monitor sockets, virtual threads can significantly reduce loading on the operating system which was supplying threads at a 1 to 1 ratio. On some operating systems, this could lead to resource exhaustion. javAPRSSrvr 4.3.3 build 15 and later automatically uses virtual threads for all application generated threads using the Executors.newVirtualThreadPerTaskExecutor() method instead of Executors.newCachedThreadPool().

The default parallelism is the number of CPUs. It may need to be increased using the system property jdk.virtualThreadScheduler.parallelism. For instance, I use -Djdk.virtualThreadScheduler.parallelism=32.

Section 6 – XML Status Page Information

The following XML elements are included in the general and, if (D) is specified, detail XML pages served by the Status Listener Port. If DetailXSLName is defined (may be the empty string) in the status port properties file, the status port will support requesting detail.xml. Otherwise, a request for detail.xml will receive a “404 File not found.” status response.

The following is from my IGate. The first is the general XML and the second is the detail.xml page. I have highlighted common (all clients, for instance) settings. Non-highlighted lines are class-specific.

If a style sheet is named in the properties file for this port, the following line will appear so the client can know what style sheet to request and apply:
<?xml-stylesheet href='stylesheet.xsl' type='text/xsl'?>
“stylesheet.xsl” is replaced by the name you have defined in this port’s properties page.

General XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<javaprssrvr name="javAPRSSrvr" revision="b01" version="4.0.0">
<software version="4.0.0">
javAPRSSrvr
</software>
<java>
<jre version="23.1-b03">
Oracle Corporation
</jre>
<time>
<current utc="1339878952884"/>
</time>
<os>
Windows Server 2008 R2
</os>
</java>
<dupeprocessor>
<servercall>
AE5PL-JF
</servercall>
<duplicatepackets packets="85"/>
<goodpackets packets="129787"/>
<blockedpackets packets="0"/>
<loopedpackets packets="0"/>
<dupequeuedepth ms="0" packets="0"/>
<dupechecklist msholdtime="30000" size="1176"/>
<historylist msholdtime="1800000" size="54545"/>
<micepackets msholdtime="30000" size="171"/>
</dupeprocessor>
<listenerports total="6">
<portlistener>
<rcvd bytes="1696" lines="35" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections total="0"/>
</portlistener>
<portlistener>
<rcvd bytes="15971" lines="254" packets="253">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="11440" lines="171" packets="0">
<udp bytes="0" lines="0" packets="0"/>
```

```
</xmtd>
<connections current="1" total="1"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections current="0" total="0"/>
</portlistener>
<portlistener>
<rcvd bytes="72781" lines="2230" packets="159">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="19373" lines="354" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections current="0" total="177"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections current="0" total="0"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">
<udp bytes="21889" lines="242" packets="91"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections total="121"/>
</portlistener>
</listenerports>
<serialinterfaces/>
<clients current="9" total="9">
<rcvdtotals bytes="101762" lines="2666" packets="419">
<udp bytes="11386773" lines="129881" packets="129461"/>
</rcvdtotals>
<xmtdtotals bytes="12539566" lines="138839" packets="138667">
<udp bytes="0" lines="0" packets="0"/>
</xmtdtotals>
<clientrcv>
<time>
<connect utc="1339875538140"/>
<lastlinein utc="1339878898246"/>
</time>
<upstream>
false
</upstream>
<readonly>
false
</readonly>
<login>
<callssid verified="true">
```

AE5PL-WX
</callssid>
<software version="1.4">
APRSWxSrvr
</software>
</login>
<rcvdfrom bytes="15971" lines="254" packets="253">
<udp bytes="0" lines="0" packets="0"/>
</rcvdfrom>
<clientxmt>
<upstream>
false
</upstream>
<sentto bytes="11440" lines="171" packets="0">
<lastlinems>
1339878939362
</lastlinems>
<udp bytes="0" lines="0" packets="0"/>
</sentto>
<xmtqueue depth="0" depthms="0"/>
<adjuncts/>
</clientxmt>
</clientrcv>
<clientrcv>
<time>
<connect utc="1339875476915"/>
<lastlinein utc="1339878952875"/>
</time>
<upstream>
true
</upstream>
<readonly>
false
</readonly>
<login>
<callssid verified="true">
FIFTH
</callssid>
<software version="3.15b08">
javAPRSSrvr
</software>
</login>
<rcvdfrom bytes="12518" lines="175" packets="0">
<udp bytes="11364884" lines="129639" packets="129370"/>
</rcvdfrom>
<clientxmt>
<upstream>
true
</upstream>
<sentto bytes="43959" lines="509" packets="508">
<lastlinems>
1339878947403
</lastlinems>
<udp bytes="0" lines="0" packets="0"/>
</sentto>
<xmtqueue depth="0" depthms="0"/>
<adjuncts/>
</clientxmt>
<remoteserver port="23000">
rotate.aprs.net
</remoteserver>

</clientrcv>
</clients>
</javaprssrvr>

Detail XML


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<javaprssrvr name="javAPRSSrvr" revision="b01" version="4.0.0">
<software version="4.0.0">
javAPRSSrvr
</software>
<java>
<jre specversion="1.7" version="23.1-b03">
Oracle Corporation
</jre>
<time>
<current utc="1339879765087"/>
<start utc="1339875475204"/>
<up millis="4289883"/>
</time>
<os architecture="amd64" version="6.1">
Windows Server 2008 R2
</os>
<processors>
4
</processors>
<memory>
<heap committed="192937984" used="109951232"/>
<nonheap committed="27983872" used="17979448"/>
</memory>
<threads current="32" groups="9" peak="33" total="44"/>
</java>
<dupeprocessor>
<class name="DupeProcessing">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<servercall>
AE5PL-JF
</servercall>
<duplicatepackets packets="105"/>
<goodpackets packets="159321"/>
<blockedpackets packets="0"/>
<loopedpackets packets="0"/>
<dupequeuedepth ms="0" packets="0"/>
<dupechecklist msholdtime="30000" size="999"/>
<historylist msholdtime="1800000" size="54785"/>
<micepackets msholdtime="30000" size="159"/>
</dupeprocessor>
<listenerports total="6">
<portlistener>
<rcvd bytes="2795" lines="56" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="174" lines="2" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections total="0"/>
<class name="StatusListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="14501">
0.0.0.0
</networkinterface>
</portlistener>
<portlistener>
```

<rcvd bytes="18191" lines="292" packets="291">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="14187" lines="212" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<class name="TCPListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="14583">
0.0.0.0
</networkinterface>
<connections current="1" total="1"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<class name="TCPListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="10155">
0.0.0.0
</networkinterface>
<connections current="0" total="0"/>
</portlistener>
<portlistener>
<rcvd bytes="92737" lines="2845" packets="196">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="24660" lines="454" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<class name="TCPListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="8080">
0.0.0.0
</networkinterface>
<connections current="0" total="227"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<class name="TCPListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="10153">
0.0.0.0
</networkinterface>
<connections current="0" total="0"/>
</portlistener>
<portlistener>
<rcvd bytes="0" lines="0" packets="0">

<udp bytes="27676" lines="308" packets="112"/>
</rcvd>
<xmtd bytes="0" lines="0" packets="0">
<udp bytes="0" lines="0" packets="0"/>
</xmtd>
<connections total="154"/>
<class name="UDPListenerPort">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<networkinterface port="8080">
0.0.0.0
</networkinterface>
</portlistener>
</listenerports>
<serialinterfaces/>
<clients current="9" total="9">
<rcvdtotals bytes="126890" lines="3360" packets="494">
<udp bytes="13969857" lines="159455" packets="158942"/>
</rcvdtotals>
<xmtdtotals bytes="15387879" lines="170468" packets="170255">
<udp bytes="0" lines="0" packets="0"/>
</xmtdtotals>
<clientrcv>
<time>
<connect utc="1339875538140"/>
<lastlinein utc="1339879738510"/>
</time>
<class name="TCPClientRcv">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<messages callssids="0" unacked="0"/>
<upstream>
false
</upstream>
<readonly>
false
</readonly>
<login>
<callssid verified="true">
AE5PL-WX
</callssid>
<software version="1.4">
APRSWxSrvr
</software>
</login>
<rcvdfrom bytes="18191" lines="292" packets="291">
<udp bytes="0" lines="0" packets="0"/>
</rcvdfrom>
<clientxmt>
<class name="TCPClientXmt">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<upstream>
false
</upstream>
<sentto bytes="14187" lines="212" packets="0">
<lastlinems>
1339879759672
</lastlinems>
<udp bytes="0" lines="0" packets="0"/>
</sentto>

<xmtqueue depth="0" depthms="0"/>
<adjuncts/>
</clientxmt>
<client port="63317">
127.0.0.1
</client>
<networkinterface port="14583">
0.0.0.0
</networkinterface>
</clientrcv>
<clientrcv>
<time>
<connect utc="1339875476915"/>
<lastlinein utc="1339879765067"/>
</time>
<class name="UpstreamClientRcv">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<messages callssids="0" unacked="0"/>
<upstream>
true
</upstream>
<readonly>
false
</readonly>
<login>
<callssid verified="true">
FIFTH
</callssid>
<software version="3.15b08">
javAPRSSrvr
</software>
</login>
<rcvdfrom bytes="15470" lines="216" packets="0">
<udp bytes="13942181" lines="159147" packets="158830"/>
</rcvdfrom>
<clientxmt>
<class name="UpstreamClientXmt">
<package name="net.ae5pl.aprssrvr" revision="b01" title="javAPRSSrvr" version="4.0.0"/>
</class>
<upstream>
true
</upstream>
<sentto bytes="51862" lines="604" packets="603">
<lastlinems>
1339879743934
</lastlinems>
<udp bytes="0" lines="0" packets="0"/>
</sentto>
<xmtqueue depth="0" depthms="0"/>
<adjuncts/>
</clientxmt>
<remoteserver port="23000">
rotate.aprs.net
</remoteserver>
<networkinterface port="0">
0.0.0.0
</networkinterface>
</clientrcv>
</clients>
</javaprssrvr>