

javAPRSSrvr User's Guide

3.15b01

javAPRSSrvr is Copyright © 2009 - Pete Loveall AE5PL pete@ae5pl.net

Use of the software is acceptance of the agreement to not hold the author or anyone associated with the software liable for any damages that might occur from its use. javAPRSSrvr and its adjuncts are offered free of charge only to amateur radio operators for amateur radio related use and may not be redistributed in any form without the express written prior approval of Peter Loveall AE5PL.

APRS is the registered trademark of Bob Bruninga WB4APR

Other trademarks included in the following text are recognized as belonging to the respective trademark holders.

Table of Contents

Section 1 - Introduction	1
Section 2 - Program Requirements and Description	2
Section 3 - Configuration Parameters	4
General Parameters	5
userCall=	5
AdminInfo=	5
password=	5
stderr=	5
logAppend=true	5
isBackground=false	5
utilityTimer=30	5
bufferSize=512	5
OSBufferSize=8192 or 16384	5
DisplayReadErrors=false	5
countInvalidAddr=true	5
skipYields=false	5
Duplicate Check Parameters	6
dupeTime=30	6
writeTO=dupeTime/2	6
Outbound Connection Parameters	7
inactivityTime=30	7
upstreamHubs=	7
upstreamHubsRO=	7
hubs=	7
upstreamUDPPort=	7
loginCommand;address;port=	7
Port Parameters	8
maxConnects=200	8
maxConnectsXXXXX=-1	8
minConnectsXXXXX=-1	8
connectBacklog=50 or maxConnects	8
Console	9
consolePorts=	9
consoleTO=5	9
Status Ports	10
statusPorts=	10
plainStatusPorts=	10
Log Ports	11
logList=	11
logCalls=	11
logPorts=	11
errorPorts=	11
dupePorts=	11
loopPorts=	11
callLogPorts=	11
connLogPorts=	11
TNLogPorts=	11
Server Ports	12
svrToSvrPorts=	12
Client Port Parameters	13
UDP Support	13
specialIPs=	13
excludeList=	13
excludeCalls=	13
keepAlive=20	13
keepAliveComment=	13

loginRequired=true	13
maxNoLogin=60	13
singleClientOnly=false	13
historyTime=0 or 30	13
msgHoldTime=30	13
useHistoryQueue=true	14
Client Ports	15
noEchoPorts=	15
echoPorts=	15
historyPorts=	15
Client Ports with Server Adjunct Interface	16
msgOnlyPorts=	16
clientOnlyPorts=	16
IGatePorts=	16
readOnlyPorts=	16
filteredHistoryPorts=	16
Special Client Ports	17
verifiedPorts=	17
verifiedList=	17
UDPPorts=	17
UDPAllowed=127.0.0.1	17
multicastPorts=	17
multicastTTL=0	17
localOnlyPorts=	17
globalUDPPorts=	17
localOnlyUDPPorts=	17
globalHTTPPorts=	17
localOnlyHTTPPorts=	18
Data Stream Filter Parameters	19
filterCalls=false	19
filterNOGATE=true	19
filter3rdParty=false	19
invalid3rdParty=TCPIP;TCPXX	19
filterMsgList=	19
passDXSpots=false	19
filterAPRS=false	19
passOldObjects=false	19
disallowUnverified=false	19
Q Construct Parameters	20
qProtocol=A	20
qAllowed=A	20
traceEnable=false	20
traceCalls=	20
qMultiConnects=	20
HTML Status Page Parameters	21
bkgdColor=606060	21
tableBkgColor=d0d0d0	21
titleBkgColor=ffd700	21
headerBkgColor=909090	21
HTMLTop=	21
HTMLBottom=	21
borderWidth=2	21
HTMLRefresh=0	21
statusLinkUserDefined=	21
statusLinkPostfix=	21
statusLinkHost=jfindu	21
portTable=	22
portTableFile=	22
Server Adjunct Parameters	23
ServerAdjunct=	23

adjunctCommands;IPAddress;port=	23
adjunctCommands;port=	23
IGate Adjunct Parameters	24
IGateAdjunct=	24
IGateCall=	24
Section 4 - Recommended Configurations	25
Section 5 - Installation Instructions	26
Sun and other non-Microsoft JVM's (Combined JAR):	26
Microsoft .NET:	26
Section 6 - Console Commands	27
General Commands	27
H – Display All Commands	27
? – Display All Commands	27
B – Exit Console	27
Q – Quit Program	27
Port Commands	27
F ipaddr ipport command parameters – Filter Command for an Inbound Connection	27
K ipaddr ipport – Kill Inbound Connection	27
G login message – Send General Announcement Message	27
U – Disconnect and go to next upstream connection	27
Configuration Commands	28
R – Reload Configuration File	28
W – Write Configuration File	28
L – List Set Parameters	28
P param – Print Parameter Value	28
S param value – Set Parameter Value	28
Display Commands	29
A – Display Attached Port Counts	29
C – Display Connection List	29
D – Display Statistics	29
M – Display Memory Usage	29
T – Threads	29
V – View System Properties	29
Section 7 – Status Page	30
Section 8 - Server Adjuncts	32
Section 9 - IGate Adjuncts	33

Section 1 - Introduction

javAPRSSrvr was written to provide the amateur radio APRS community a simple, but effective core server for the Internet. It provides a multi-platform application tuned to present cleanly formatted packets to the APRS-IS feed. It does this with effective duplicate checking to reduce bandwidth requirements.

The current APRS-IS network consists of 3 core servers and multiple second tier and lower tier servers which may also act as IGates. javAPRSSrvr is designed to act as an APRS-IS server and also has the capability to act as an IGate. javAPRSSrvr may also be extended by other 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 recent Java Virtual Machine. It has been successfully tested with the Microsoft JVM for Windows (JVM 1.1.4), with the Sun 1.1.8 JVM and higher, and with many 3rd party JVMs. It has also been tested with Microsoft .NET 2.0 (and higher) with J# 2.0 SE installed.

javAPRSSrvr is comprised of a number of classes which Java looks at as objects. The main class is javAPRSSrvr. This class is called at startup, loads the parameter file, and begins execution of the different socket 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 supports two types of connections: upstream connections (to other servers) and client connections (from other servers or clients). We will look at client connections first.

There are multiple types of client ports. 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.

The first type of client port is the “full feed” port. There are multiple versions of this port: noEchoPorts and echoPorts are equivalent and provide a full feed (all non-duplicate packets seen by the server) to connected clients. echoPorts is supported only as a configuration file entry for prior compatibility and presents a feed where client-initiated packets are NOT echoed back to the client (same as noEchoPorts). This redesign of echoPorts prevents loop conditions (by definition, an echo is a loop). historyPorts provides an initial dump of historyTime minutes of posits, objects, and items. This port is output-only (does not accept data from the client) to prevent historical data from being re-propagated throughout APRS-IS.

The second type of client port is the “adjunct-enabled” port. These ports provide a default limited feed of packets seen by the server. This limited feed can therefore be expanded by a server adjunct to include more packets than the base feed. This is very useful for adjuncts such as javAPRSFilter. The base types of adjunct-enabled client ports are messageOnlyPorts, IGatePorts, and clientOnlyPorts. clientOnlyPorts does not keep a history of stations reported by the attached client and will not work for IGates. Use clientOnlyPorts for very specific purposes such as SatGate connections, etc. There are two output-only adjunct-enabled ports called readOnlyPorts and filteredHistoryPorts. The output-only ports are targeted for display-only clients such as javAPRS (not related to javAPRSSrvr).

The third type of “client” port is the log port. There are multiple types of log ports feeding different aspects of the server data. These ports may only be connected to by clients at IP addresses specified in logList.

The fourth type of “client” port is the status port. There are two types of status ports: statusPorts and plainStatusPorts. statusPorts sends an HTML page to an HTTP browser. plainStatusPorts sends a text status page. The text status page can be retrieved from statusPorts by appending /status.txt to the URL. There are other specialty ports described in section 3.

There are two outbound connection types. The first is the hub connection. upstreamHubs provides a bidirectional connection to a **single** server at a time. If upstreamHubsRO or hubs is defined, upstreamHubs will be ignored. These are receive-only connections and are designed to provide unidirectional communication. Multiple concurrent bidirectional connections are not allowed due to the looping and bandwidth issues that such antiquated configurations impose.

The second outbound connection type is the svrToSvrPorts “connection”. This is actually a UDP predefined feed between two or more servers. This is primarily for use between core servers to ensure that all core servers see all packets.

All ports and connections can be set to any port number and can be attached to any IP address if the server supports multiple IP's. All client ports can support UDP output (see client ports description and upstreamUDPPort).

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 characters and no less than 3 ASCII 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.

There are multiple types of filtering defined under Section 3 – Data Stream Filter Parameters. Please see that section for more information.

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 3.10b10
```

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

```
# javAPRSSrvr 3.10b10 Port full.
```

```
# javAPRSSrvr 3.10b10 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 password is valid, 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 parameters reside in a configuration file which, by default, is called javaprssvr.cfg. You can use any text file if you pass the name into javAPRSSvr as a command line parameter.

The parameters are CASE SENSITIVE. Defaults are shown below.

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

List parameters may be defined on the line or may be defined in a text file. 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. Do not put blank lines in the file. The file must have the extension .lst For instance, this would be the definition for hubs where you want to connect to first.aprs.net and second.aprs.net port 1313:

```
hubs=first.aprs.net:1313;second.aprs.net:1313
```

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

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

You would then put the following line in your configuration file:

```
hubs=hubs.lst
```

(R) at the beginning of the parameter description means that the parameter can be changed on-the-fly from the console with either the S or R commands.

General Parameters

userCall=

This is the name used for logging into other servers (max 9 alphanumeric characters).

AdminInfo=

(R)This is a general use string that is displayed next to the title "Sys Op" on the status page.

password=

(R)This is the encrypted password for console access. DO NOT SET MANUALLY. ONLY SET VIA THE SET COMMAND AND THE WRITE COMMAND. CAUTION: This password crosses the Internet in plain text so do not use a password you might use elsewhere.

stderr=

If set, this is the path of the file where all error messages are sent.

logAppend=true

If true, the error log file will be appended to instead of being replaced.

isBackground=false

This tells javAPRSSrvr whether to monitor stdin and write to stdout. Error messages are still written to stderr. Set this to true if you are running javAPRSSrvr as a background daemon or Windows service.

utilityTimer=30

(R)This sets when memory cleanup should occur (in seconds). It defaults to 30 seconds as this seems to be a good number to prevent the garbage collector from taking too many CPU cycles at one time.

bufferSize=512

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

OSBufferSize=8192 or 16384

(R)This tells javAPRSSrvr how deep the operating system receive buffer is. javAPRSSrvr uses this number to prevent "hanging reads". The default is 8192 for all Windows-based servers with a JVM less than 1.4.0. All others default to 16384. This number should not be changed unless "hanging" connections occur.

DisplayReadErrors=false

(R)This turns on displaying of read exceptions which cause disconnects. You can use this to determine if OSBufferSize may be set too low.

countInvalidAddrs=true

(R)This turns on displaying of invalid connection exceptions caused by a 0.0.0.0 address for the local address in Java. These messages can indicate a denial of service attack although no firm confirmation of this has been done. These occurrences showed up in Java 1.4.2.

skipYields=false

(R)True causes manual thread sequencing to be skipped. This will cause CPU usage to increase under load, but can improve packet throughput. Caution: Setting this to true can cause 100% CPU usage on a heavily loaded system.

Duplicate Check Parameters

Duplicate checking allows for only the first packet to pass through the server. Duplicate checking is based on the from call, unproto call, length of the data part, and a CRC of the data part of the packet.

Duplicate checking also stops duplicate packets that have been mangled by certain software and hardware configurations. These mangled packets include packets with trailing white space removed, packets with 8 bit characters either removed or replaced with spaces, Mic-E packets with non-printable characters either removed or replaced with spaces, and Mic-E packet translations when the Mic-E packet has already been passed.

dupeTime=30

(R)This is the "hold" time in seconds for duplicate packet comparisons. This time divided by 2 is the depth of the individual write queues.

writeTO=dupeTime/2

(R)This is the allowed depth of individual output queues in seconds before disconnecting the remote connection.

Outbound Connection Parameters

Format for the hub definitions are as follows (note: IP address or DNS name can be used):

hubIP:hubPort or

hubIP:hubPort,localIP:localPort

Multiple ports are separated by a semicolon. For example:

hub1IP:hub1Port;hub2IP:hub2Port

inactivityTime=30

(R)This is how long a lapse in data receipt will be considered a loss of the remote server for all outbound connections.

upstreamHubs=

(R)These are servers which you want to both send and receive data from through this connection. Only one server will be connected at a time. Append a "u" (no quotation marks) to each server definition if you want to use UDP. Round-Robin DNS names are supported.

upstreamHubsRO=

(R)These are servers which you want to only receive data from through this connection. Only one server will be connected at a time. Append a "u" (no quotation marks) to each server definition if you want to use UDP. This parameter overrides upstreamHubs (mutually-exclusive). Round-Robin DNS names are supported.

hubs=

These are servers which you want to only receive data from through these connections. Append a "u" (no quotation marks) to each server definition if you want to use UDP. This parameter overrides upstreamHubs (mutually-exclusive). Round-Robin DNS names are supported.

upstreamUDPPort=

(R)This defines a UDP listen port to use in conjunction with upstreamHubs.

Define a port number (can be NIC specific by prepending the IP address of the NIC followed by the port number; IP:port) that upstreamHubs connections can use to get packets from upstream servers. Be sure that you are on a full duplex connection and any firewalls have been opened to receive UDP packets to this port from the designated upstreamHubs server ports.

loginCommand;address;port=

This allows you to send a command as part of the login string to an upstreamHubs or upstreamHubsRO. For instance, if you want to use javAPRSSrvr to support some local clients and only want local data, you can do something similar to the following:

```
upstreamHubs=aprswest.net:14580
```

```
loginCommand;aprswest.net;14580=filter r/33/-97/200
```

This will cause javAPRSSrvr to only receive packets near Dallas, TX.

;address must match the upstreamHubs or hubs definition. If an IPv6 address is used, replace the colons with semicolons (loginCommand;1080;0;0;0;8;800;200C;417A;14580=)

Port Parameters

Format for the port definitions are as follows:

Port or

IP:Port

Multiple ports are separated by a semicolon. For example:

Port1;Port2

maxConnects=200

(R)This defines the maximum total number of connections to all non-log server ports.

maxConnectsXXXXX=-1

(R)Replace the XXXXX with the port number which you want to specifically further restrict connections to. A negative number means that no port specific test will be done.

minConnectsXXXXX=-1

(R)Replace the XXXXX with the port number which you want to specifically authorize potentially more connections to. This number overrides maxConnects. For instance, set maxConnects=50 and minConnects14580=25. If 40 connections occur to port 10152 and 10 to port 14580, when the next connect attempt goes to port 10152, it will be rejected. When the next connect attempt to 14580 occurs, it will succeed because the total connects to port 14580 is below 25. A negative number means that no port specific test will be done.

connectBacklog=50 or maxConnects

This defines the backlog parameter for the Java ServerSocket. This parameter tells Java how many unprocessed connect requests will be queued before the OS will dump any further requests.

Console

consolePorts=

The recommended port is 14500. This port is password protected. Local echo is recommended by the telnet client as echo is not provided by this port. Only one console session is allowed at a time.

consoleTO=5

(R)This is the timeout before disconnecting remote consoles in minutes.

Status Ports

The status ports are HTTP ports supporting HTML and text formats. If a connection occurs to a status port and nothing is received from the remote client after 10 seconds, a straight text status page will be sent without HTTP headers. Both port types support the following URLs:

/ (default)

/status.htm & /status.html (HTML)

/status.txt (text)

/favicon.ico (favorite icon)

If any other URL or bad command is received, a 404 error is returned. 404 will be returned for /favicon.ico if there is not a file named favicon.ico in the javAPRSSrvr directory.

statusPorts=

This defaults to the HTML status page.

Recommended port is 14501.

plainStatusPorts=

This defaults to a plain text status page for rapid reporting to browsers.

Recommended port is 14511.

Log Ports

logList=

(R)This is a list of semicolon separated DNS names or IP addresses which are allowed to connect to the logPorts and errorPorts.

logCalls=

(R)This is a semicolon delimited list of logins to send packets to the callLogPorts.

logPorts=

This defines the "non-dupe valid packet" dump port.
Recommended port is 14502.

errorPorts=

This defines the "rejected packet" dump port. These are packets that have failed to meet the filters requirements.
Recommended port is 14503.

dupePorts=

This defines the "duplicate packet" dump port.
Recommended port is 14504.

loopPorts=

This port displays any packets that are determined to have arrived via a loop according to the q construct.
Recommended port is 14505.

callLogPorts=

This port displays any packets that enter the server from the logCalls login(s).
Recommended port is 14506.

connLogPorts=

This port displays an entry at connection time, at login, and at disconnection.
Recommended port is 14507.

TNCLogPorts=

This defines the port where all RF received, RF sent, and Internet sent packets are put.
The format is:

[<R:timestamp]packet for packets received on RF
[>R:timestamp]packet for packets sent on RF
[>I:timestamp]packet for packets sent to javAPRSSrvr (Internet)
Recommended port is 14508.

Server Ports

svrToSvrPorts=

(List parameter) Bidirectional UDP port

The format is localIP:localport;remoteIP1:remoteport1;remoteIP2:remoteport2 You can use DNS names for the remote IP addresses but they will only be reconciled at startup. Be sure to open your firewall for the UDP port.

Client Port Parameters

UDP Support

You can specify any or all client ports to support UDP feeds to the clients. Append a "u" (no quotation marks) to the port definitions you want to have support UDP. Those ports will operate as normal TCP ports unless the client sends a UDP xyz login command. If a UDP login command is received, the TCP connection will still operate as normal with keep-alive comments being sent at regular intervals; however, all packets (non-comments) will be sent via UDP to the remote UDP port specified in the UDP login command.

A lower-case "u" means that UDP is optional. An upper-case "U" means that UDP is mandatory. If UDP is mandatory on a port, the client cannot get logged-in without a valid UDP request. This is best used with loginRequired=true (default).

specialIPs=

(R)This is a list of semicolon separated DNS names or IP addresses which can connect to the server even after there are maxConnects connections.

excludeList=

(R)This is a list of semicolon separated DNS names or IP addresses which are excluded from connecting to the noEchoPorts and the echoPorts.

excludeCalls=

(R)This is a list of semicolon separated callsign-SSID's which are excluded from logging into the server and are excluded from passing packets through the server.

USERLIST;javaMSG;javaTITLE;javaTITL2;KIPSS;SERVER;userCall are automatically excluded.

keepAlive=20

(R)This specifies how long in seconds can elapse before a keep-alive comment line is sent to the client. This is to keep a connection active. Zero means no keep-alives are sent.

keepAliveComment=

(R)This specifies the text to attach to the keepAlive comment lines.

loginRequired=true

(R)This specifies whether a login is required before the server will pass packets to the client. The login can be verified or unverified. This helps reduce bandwidth loss due to port sniffers.

maxNoLogin=60

(R)This sets the maximum time in seconds that a client may remain connected without logging in.

This is only checked when loginRequired=true (default). Be careful to not make this too short as some clients are have a slow initialization after startup.

singleClientOnly=false

Setting this to true will restrict connections from any single IP or login to a clientOnlyPorts type port to one connection at a time.

This defaults to false if keepAlive is set to a value greater than 0 and true if keepAlive equals 0.

historyTime=0 or 30

(R)This is the depth, in minutes, of the history queue. This should never be more than 35 minutes. If historyPorts, msgOnlyPorts, hubAndMsgPorts, clientOnlyPorts, or filteredHistoryPorts are defined, historyTime defaults to 30 minutes.

msgHoldTime=30

(R)This is the maximum time in minutes that a message packet will be held for a corresponding position report.

useHistoryQueue=true

(R) This determines whether the history queue is used to find posits for packets.

Client Ports

noEchoPorts=

Bidirectional

This defines ports which do standard authentication and can receive data from verified clients. Data from the connected client is not echoed.

Recommended port is 10152 (port 23 on core servers).

echoPorts=

(deprecated) Bidirectional

This defines ports which do standard authentication and can receive data from verified clients. Non-duplicate data from the connected client is echoed back to the client.

Use not recommended. **These ports now act exactly like noEchoPorts (no echo data).**

historyPorts=

Output only.

This defines the history dump ports.

This port is output-only to reduce people attempting to connect IGates to it which cause potential loops on APRS-IS. Only posits and objects are archived.

Recommended port is 10151 if required.

Client Ports with Server Adjunct Interface

msgOnlyPorts=

Bidirectional

This defines ports which only output : data type packets (messages) and position type packets (including Mic-E) if there is an associated message that has been sent.

Recommended port is 1314.

clientOnlyPorts=

Bidirectional

This defines ports which only output : data type packets (messages) destined for the logged in callsign-SSID and position type packets (including Mic-E) if there is an associated message that has been sent.

IGatePorts=

Bidirectional

This defines ports which only output : data type packets (messages) destined for the logged in callsign-SSID OR destined for stations heard from the client within the last historyTime minutes and position type packets (including Mic-E) if there is an associated message that has been sent.

Recommended port is 14580.

readOnlyPorts=

Output only

This port is used to provide a filtered feed to clients with no input capability. It does support Server Adjunct definitions in the login (see below).

filteredHistoryPorts=

Output only

This port is used to provide a filtered feed with filtered historical posits and objects to clients with no input capability. It does support Server Adjunct definitions in the login (see below).

Special Client Ports

verifiedPorts=

Bidirectional

This defines ports which do not do authentication and consider all connects as verified clients. This port is restricted to access from hosts listed in verifiedList.

verifiedList=

(R)This is a list of semicolon separated DNS names or IP addresses which are allowed to connect to the verifiedPorts.

UDPPorts=

Input only

This defines ports where UDP packets can be sent. The packets must meet the APRS standard of having a proper header, a data type of either a posit or weather packet, and terminated by a null (0), a cr and/or a lf, or no termination character.

Recommended port is 1315.

UDPAllowed=127.0.0.1

(R)This is a list of semicolon separated DNS names or IP addresses which are allowed to send packets to the UDPPorts.

multicastPorts=

Output only

This defines output only multicast ports. The format is MulticastIP,Port,TTL,HostIP where ,HostIP is optional. Multiple ports may be defined by separating the entries with semicolons.

multicastTTL=0

(R)Time-to-live for multicast packets.

If multicastTTL is greater than 0, it overrides the TTL parameter in the multicastPorts definition.

localOnlyPorts=

Bidirectional

This port type is an IGate port which packets from the client only appear on local ports and are not placed on upstreamHubs connections.

globalUDPPorts=

Input Only

This defines ports where UDP packets can be sent from any client. Each UDP packet must contain a standard login line (user xxx pass nnn) terminated with a cr/lf sequence. The remainder of each packet must meet the APRS standard of having a proper header, a data type of either a posit or weather packet, and terminated by a null (0), a cr and/or a lf, or no termination character and the APRS "packet" must be from the logged-in station.

localOnlyUDPPorts=

Input Only

This defines ports where UDP packets can be sent from any client. Each UDP packet must contain a standard login line (user xxx pass nnn) terminated with a cr/lf sequence. The remainder of each packet must meet the APRS standard of having a proper header, a data type of either a posit or weather packet, and terminated by a null (0), a cr and/or a lf, or no termination character and the APRS "packet" must be from the logged-in station. APRS packets are not placed on upstreamHubs connections.

globalHTTPPorts=

Input Only

This defines ports where a client can connect using the HTTP POST. Each POST data (application/octet-stream) must contain a standard login line (user xxx pass nnn) terminated with a cr/lf sequence. The remainder of the data must meet the APRS standard of having a proper header, a data type of either a posit or weather packet, and terminated by a null (0), a cr and/or a lf, or no termination character and the APRS "packet" must be from the logged-in station.

localOnlyHTTPPorts=

Input Only

This defines ports where a client can connect using the HTTP POST. Each POST data (application/octet-stream) must contain a standard login line (user xxx pass nnn) terminated with a cr/lf sequence. The remainder of the data must meet the APRS standard of having a proper header, a data type of either a posit or weather packet, and terminated by a null (0), a cr and/or a lf, or no termination character and the APRS "packet" must be from the logged-in station. APRS packets are not placed on upstreamHubs connections.

Data Stream Filter Parameters

filterCalls=false

(R)This enables filtering out from calls that must be ASCII alphanumeric or _ A single hyphen may also exist (for a SSID).

filterNOGATE=true

(R)This drops packets with NOGATE or RFONLY in the path.

filter3rdParty=false

(R)This drops all packets with a data type of } or packets where the header begins with a right brace “}”. This parameter has been superseded by invalid3rdParty which is more selective.

invalid3rdParty=TCPIP;TCPXX

(R)If filter3rdParty=false, third-party packets will be dropped if an invalid3rdParty entry is found in the third-party header. The first setting indicates a verified login; the second indicates an unverified login.

filterMsgList=

(R)This drops message packets addressed to any of the calls in the list. The default list includes messages automatically generated by aprsD and APRS+SA which were never meant to be propagated throughout APRS-IS. **USERLIST;javaMSG;javaTITLE;javaTITL2;KIPSS;SERVER;userCall** are automatically filtered. APRServe packets which begin with USERLIST, javaMSG, javaTITLE, or javaTITL2 are also filtered.

passDXSpots=false

(R)This enables payloads that start with "DX de " to be passed.

filterAPRS=false

(R)This enables rudimentary packet payload interpretation.

passOldObjects=false

(R)This enables payloads with alphanumeric or space data types to be passed. Only checked when filterAPRS=true.

disallowUnverified=false

Do not pass any packets from non-verified sources.

No packets are ever passed from non-logged in stations. When false, only posit and weather packets from invalid logins (passcode is invalid) are allowed to pass. When true, no packets from invalid logins are passed. Invalid logins (read-only clients) are still able to connect and receive data, just not send it when this switch is true. Setting this to true will also exclude any TCPXX or qAX packets from passing through the server.

Q Construct Parameters

Further definition can be found at <http://www.aprs-is.net/q.htm>

qProtocol=A

This sets the second letter of the q construct for packets where this server adds the q construct.

qAllowed=A

This defaults to the value of qProtocol. This specifies which protocols/networks this server should pass.

traceEnable=false

(R)This enables full trace information in the header for all packets.

traceCalls=

(R)This enables tracing of packets from specific calls.

qMultiConnects=

(R)Clients allowed to use verified logins on multiple servers, **including** this one.

HTML Status Page Parameters

bkgdColor=606060

tableBkgColor=d0d0d0

titleBkgColor=ffd700

headerBkgColor=909090

(R)These set the background colors for the page, tables, titles, and column headers respectively. These are hex RGB as specified in the HTML spec.

Note: You can put more in this text than just the background color. For instance, setting

`tableBkgColor=606060 cellspacing=1`

will reduce the space between cells from the default of 2 pixels to 1 pixel. Be careful not to "break" the page.

HTMLTop=

(R)This sets the text that follows the <BODY> tag.

HTMLBottom=

(R)This sets the text that precedes the </BODY> tag.

borderWidth=2

(R)This sets the width of the border (in pixels) around each table.

HTMLRefresh=0

(R)This tells the browser to reload the HTML status page every X seconds using the HTTP Refresh setting in the HTTP (not HTML) header. Zero causes the Refresh setting to be omitted.

statusLinkUserDefined=

(R)This defines the URL prefix to be used for the callsign links on the HTML status page.

For example, `statusLinkUserDefined=http://www.qrz.com/callsign/`

statusLinkPostfix=

(R)This defines the URL suffix to be used for the callsign links on the HTML status page.

For example, `statusLinkPostfix=%25` This value is only used if `statusLinkUserDefined` has a value other than an empty string.

statusLinkHost=jfindu

(R)This causes a predefined `statusLinkUserDefined` and `statusLinkPostfix` to be created.

Valid entries are `jfindu`, `findu`, `aprsworld`, and `qrz`. This value is ignored if `statusLinkUserDefined` has a value other than an empty string.

portTable=

(R) This is a semicolon delimited line allowing you to display 2 columns of information on the HTML status page. The first column is titled "Port Number" and the second column is "Description". Any text can go in either column. For instance: `portTable=10152;No Echo Port;10153;Read Only Port;14501,14511;Status Ports` will display:

Available Ports	
Port Number	Description
10152	No Echo Port
10153	Read Only Port
14501,14511	Status Ports

If a column is left blank, it is merged with the other column. If both columns are left blank, the column will appear as a separator row.

portTableFile=

This parameter points to a file containing sysop-defined HTML which will be inserted in place of the portTable on the HTML status page.

Server Adjunct Parameters

ServerAdjunct=

This is the name of the adjunct class file. For instance ServerAdjunct is the name for ServerAdjunct.class in javAPRSFilter. This is case sensitive. See below for more information about server adjunct programs.

adjunctCommands;IPAddress;port=

adjunctCommands;port=

(R)This setting allows the definition of files per defined filtered port for predefined filter definitions per individual logins. For instance, adjunctCommands;14580=predef.lst tells javAPRSSrvr to load the setting in predef.lst for port 14580. The file format is similar to the javaprssrvr.cfg format. For instance, predef.lst might contain the line AE5PL-TS=filter r/33/-96/100

;IPAddress may be either missing to apply to a non-NIC specific port, an IPv4 address, or an IPv6 address with colons replaced by semicolons (adjunctCommands;1080;0;0;0;8;800;200C;417A;14580=)

IGate Adjunct Parameters

IGateAdjunct=

This is the name of the adjunct class file.

For instance IGateAdjunct is the name for IGateAdjunct.class in javAPRSIGate. This is case sensitive. See below for more information about IGate adjunct programs.

IGateCall=

This is the callsign-SSID for the IGate.

It must conform to AX.25 standards, must be unique throughout APRS, and it must be different from userCall (the server's callsign-SSID).

Section 4 - Recommended Configurations

Use default settings for most parameters. Do not adjust timing parameters as they may affect APRS-IS performance. Default parameters should be left out of javaprssvr.cfg for ease of reading.

Core servers cannot have any upstreamHubs defined. Each core server must have the other core servers defined in svrToSvrPorts.

Non-core servers must only use upstreamHubs or, if not supporting bidirectional clients, upstreamHubsRO to connect to APRS-IS. Non-core servers should **never** use svrToSvrPorts.

userCall must be a unique identifier in APRS. Using a callsign provides clarity to assist sysops of the servers that you connect to.

Section 5 - Installation Instructions

Sun and other non-Microsoft JVM's (Combined JAR):

Install the latest JDK from Sun for your OS.

Place javAPRSSrvr.jar (from the combined jar .zip file) and javaprssrvr.cfg into a unique directory.

Modify javaprssrvr.cfg to meet your requirements.

Run javAPRSSrvr.jar using your Java VM. This is done with the following command line.

```
JDKPath\bin\java -server -cp javAPRSSrvr.jar javAPRSSrvr
```

Microsoft .NET:

(No support for javAPRSSDB or non-.NET interfaces)

Unzip the .msi file and run it. If you need any Microsoft components, the installation program will walk you through installing them.

Modify javaprssrvr.cfg to meet your requirements.

Enter "net start javAPRSSvc" on a command line to start the service the first time.

You may make copies of javAPRSSvc under different names and manually install them to run as services. This allows you to run multiple instances of javAPRSSrvr as .NET Windows services. To manually install additional instances, copy javAPRSSvc.exe to a new directory and rename javAPRSSvc.exe to the name of the service you want to install. Create a new javaprssrvr.cfg for your new service. Be sure to **not** duplicate port numbers. Copy in any javAPRSSrvr .dlls that you need. Do **not** rename the .dlls. Run the following line to install your new service:

```
%windir%\Microsoft.NET\Framework\v2.0.50727\installutil.exe -i yournewsrvc.exe
```

You will need to run "net start yournewsrvc" to have your new service start the first time.

Note: javAPRSSvc requires .NET version 2.0 which will be asked for during full installation if not already installed. .NET 3.0 (Vista) is .NET 2.0 with more features so it will already be installed. If you have previously installed javAPRSSvc under .NET 1.1, run:

```
"%windir%\Microsoft.NET\Framework\v1.1.4322\installutil.exe" -u javAPRSSvc.exe
```

at the command prompt in the directory that has the previous javAPRSSvc. This must be done before installing the new javAPRSSvc. You must also stop javAPRSSvc before executing the above uninstall command.

Section 6 - Console Commands

The default console password (must be entered before any console commands can be entered) is CR or LF (Enter Key). If you have enabled a console port, change this using the S command followed by the W command.

These commands are case insensitive and require a CR to take effect.

General Commands

H – Display All Commands

? – Display All Commands

B – Exit Console

Q – Quit Program

Port Commands

F ipaddr ipport command parameters – Filter Command for an Inbound Connection

Changes the server adjunct command for an inbound connection

K ipaddr ipport – Kill Inbound Connection

Closes the inbound connection from ipaddr:ipport

If ipport is omitted or set to zero, all connections from ipaddr will be disconnected.

G login message – Send General Announcement Message

Sets/Clears a message for all verified clients at login or sends an immediate message to the specified client.

G by itself clears the general announcement message, thereby disabling it.

A login of SET will set the general announcement message which will be sent to each verified client at login. A login of ALL will send the general announcement message immediately to all verified logins.

Using a login of a current connection will cause the message to be sent immediately to the specified client.

The client must be verified, not an outbound connection, and not a server to receive a message.

U – Disconnect and go to next upstream connection

Disconnects current upstream connection and goes to next in list.

Configuration Commands

R – Reload Configuration File

This will re-read the configuration file and update the parameters indicated with a (R) mark.

W – Write Configuration File

This will write current parameters to config file

L – List Set Parameters

This lists all set parameters, their default values, and their current values.

P param – Print Parameter Value

Displays the value of param

S param value – Set Parameter Value

Sets the value of param

Display Commands

A – Display Attached Port Counts

This displays current connection counts by port.

C – Display Connection List

This displays the active inbound and outbound connections.

D – Display Statistics

This is the same page that you get when you connect a browser to the plainStatusPort. The statistics are connection related.

M – Display Memory Usage

This displays the memory usage as seen by the application. The total memory used will always be lower than what the operating system shows javAPRSSrvr is using. This is due to the JVM overhead.

T – Threads

This displays a list of all of the threads running in javAPRSSrvr. The number of threads will be lower than what the operating system reports due to JVM threads that javAPRSSrvr does not know about. Below are sample lines and what they mean:

main

This is the main javAPRSSrvr thread.

Outbound Connection to third.aprs.net:10152

This is the thread that controls the "dialing" of the outbound connection to third.aprs.net port 10152.

Listening on 127.0.0.1:1313

This is a thread that is listening for connections on port 1313 on address 127.0.0.1

Listening on 14504

This is a thread that is listening for connections on port 14504 on all server addresses.

Utility Thread

This is the memory cleaning thread.

Input Queue

This thread processes each line of received data.

ame-main.ametx.com/208.199.181.250:56240->third.aprs.net/198.88.21.2:10152

This is a thread that is sending data to third.aprs.net port 10152

ame-main.ametx.com/208.199.181.250:56240<-third.aprs.net/198.88.21.2:10152

This is a thread that is receiving data from third.aprs.net port 10152

V – View System Properties

This displays all entries in System.getProperties(). This information can be useful in determining working directories, etc.

Section 7 – Status Page

Server ID	AE5PL-JS	Server login/callsign
Sys Op	Pete Loveall pete@ae5pl.net	Sysop information
OS	Windows 2003 5.2	Operating System
JVM	Sun Microsystems Inc. 1.5.0-beta2 class.zip: 49.0	Java Virtual Machine information
Total Up Time	29.749s	Total time since last restart

Total Bytes In	42,225	Bytes received by server
Total Bytes Out	32,737	Bytes sent by server
Average bps In	11,648	Average bits per second received (does not include TCP/IP overhead)
Average bps Out	9,030	Average bits per second sent (does not include TCP/IP overhead)

Total Valid Packets	450	Total packets received which are available to be passed to clients
Total Duplicate Packets	4	Total duplicate packets (includes mangled packets)
Total Rejected Packets	0	Total packets which are not valid packets (header is mangled, no information field, etc.)
Total Looped Packets	0	Packets received which are considered loop packets
Total Packets Processed	454	Total packets processed from the received line

Total 3rd Party Packets	0	Total 3 rd party packets (data type = ;)
Total NOGATE Packets	0	Total packets with NOGATE or RFONLY in the header
Total Server Message Packets	0	Total packets generated by aprsd, APRServe, or AHub
Generic Queries Blocked	0	Total generic queries from RF (data type = ?)
Generic Queries Passed	0	Total generic queries generated on APRS-IS
Total OpenTrac Packets	0	Total packets with unproto = OPNTRK

Total Inbound Connects	7	Cumulative count of connections to server
Total Inbound Disconnects	0	Cumulative count of disconnections from server
Peak Inbound Connections	7	Peak number of connections to server
Maximum Inbound Connections	100	maxConnects setting
Current Inbound Connections	7	Current count of connections to server
Current Outbound Connections	1	Current count of outbound connections from server

Receive Queue Depth (ms)	0	Depth of receive line queue in milliseconds
--------------------------	---	---

Outbound Connections											
Server	q Addr	Verified	Software	Connected	Bytes Rcvd	Bytes Sent	Rcv bps	Send bps	Last Packet In	Looped Packets	Queue Depth(ms)
Remote server IP address and port	Login name used by q algorithm	Is bi-directional?	Remote server software name and version	Total connection time	Total bytes received	Total bytes sent	Average receive bits per second	Average send bits per second	Time since last packet received	Packets received that fail q loop test	Transmit queue depth

Inbound Connections													
Port	Client	Callsign	Verified	Software	SA Setting	Connected	Bytes Rcvd	Bytes Sent	Rcv bps	Send bps	Last Packet In	Looped Packets	Queue Depth(ms)
Server port	Client IP address and port	Client login	Is bi-directional?	Remote server software name and version	Server Adjunct setting	Total connection time	Total bytes received	Total bytes sent	Average receive bits per second	Average send bits per second	Time since last packet received	Packets received that fail q loop test	Transmit queue depth

Section 8 - Server Adjuncts

With the release of 2.0, javAPRSSrvr now supports externally developed software to provide filtering and possibly other future features. This is done via the ServerAdjunctInterface interface. This interface is only used with msgOnlyPorts, IGatePorts, clientOnlyPorts, readOnlyPorts, and filteredHistoryPorts. The interface basically works as follows:

For any packets which would normally be dropped going to the client on a particular port type, a method is now called which will allow the adjunct software to say "send it to the client anyway". This is known as additive filtering. For instance, if the adjunct software is configured to filter packets for a particular region for msgOnlyPorts 14579, the client will see all messages and associated posits worldwide PLUS all packets originated within the specified region.

The filtering is done on a connection basis. The sysop can define default filters for ports via parameters in the config file (see the server adjunct software manual for details). The client can specify dynamic filters as follows:

Via the login. For instance:

```
user mycall pass -1 vers testsoft 1.1.1 filter r/36/-75/2400
```

Via a comment line. For instance:

```
# filter r/36/-75/2400
```

Via a sequenced APRS message (message requiring an ack):

```
MyClient>APRS::AE5PL-JT :filter r/36/-75/2400
```

For the message method to work with most clients, the server's call must be all upper case or the client must use SERVER as the message To field. javAPRSSrvr will ack the client's message and send a message to the client with the result of the setFilter method.

Only the login and comment line methods are available for readOnlyPorts and filteredHistoryPorts.

Filtering is only one application that this could be used for. This basically opens up the server to other processing on packets.

Section 9 - IGate Adjuncts

With the release of 3.5, javAPRSSrvr now supports externally developed software to function as an IGate using the IGateInterface interface. Unlike the Server Adjunct, the IGate Adjunct interface is bidirectional.

The IGate Adjunct interface offers the ability for the adjunct application to use the parsing capabilities of javAPRSSrvr so minimal parsing need be done within the adjunct application.

The basic flow on receive is for javAPRSSrvr to wait for a line of data; the adjunct supplies a line; javAPRSSrvr parses the line; and javAPRSSrvr returns the parsed information to the adjunct for processing.

The basic flow for sending is javAPRSSrvr puts all parsed packets that would go to a noEchoPort client on the adjunct's queue; the adjunct determines if it is to gate the packet to RF; and the adjunct transmits to RF if the packet passes the IGate's tests.

An IGate adjunct may do other bidirectional functions instead of IGate functions. For instance, javAPRSQRZ and javAPRSEmail are IGate adjuncts that do lookups and email, respectively.