NAVSYSTEMS (IOM) LIMITED

USER MANUAL

NAVSYSTEMS (IOM) LTD
Commerce Chambers
Bowring Road
Ramsey
Isle of Man
IM8 2LQ

27/05/2013

Email: dh@navsystems-uk.com

Blue Spider Logger User Manual

Ver 1.23

1. Introduction

This software has been designed to fulfill a requirement to obtain information from a fleet of ships regarding fuel consumption, weather, route travelled, vessel attitude, and many other statistics to be made accessible to office personnel. On each vessel a server application collects and log the required information. The data is recorded as a set of simple CSV files the format of each is configurable. Log files are automatically named with the start date and are periodically copied to a network folder. The files are replicated to shore using 3rd party software.

The software runs as a windows service and can be configured to collect data from com ports and from OPC servers. The software can be configured to automatically check for software updates and download and install these automatically.

This user manual is intended as a guide to use and configuration of the software.

1.1. Summary of key features

- Record up to 32 log files each with different layouts.
- Capture data from OPC, COM ports and TCP/IP streams.
- Write to log files at regular rates or triggered by arrival of incoming data
- Unattended fully automated software updates
- Remote configuration
- Upload of completed log files
- Output of custom messages
- AIS filtering
- Javascript engine can be used to process incoming data (for instance calculation of vessel roll period)
- Logging of system alarms and events
- Low CPU and memory usage.

1.2. Functional overview

Serial data is collected by the BSPNet service and this can be run on more than one computer to make ports on remote machines available.

The BSPNet service acts as an agent for BSPLogger.

Serial ports on several remote machines may be used as long as a network connection exists to these machines.

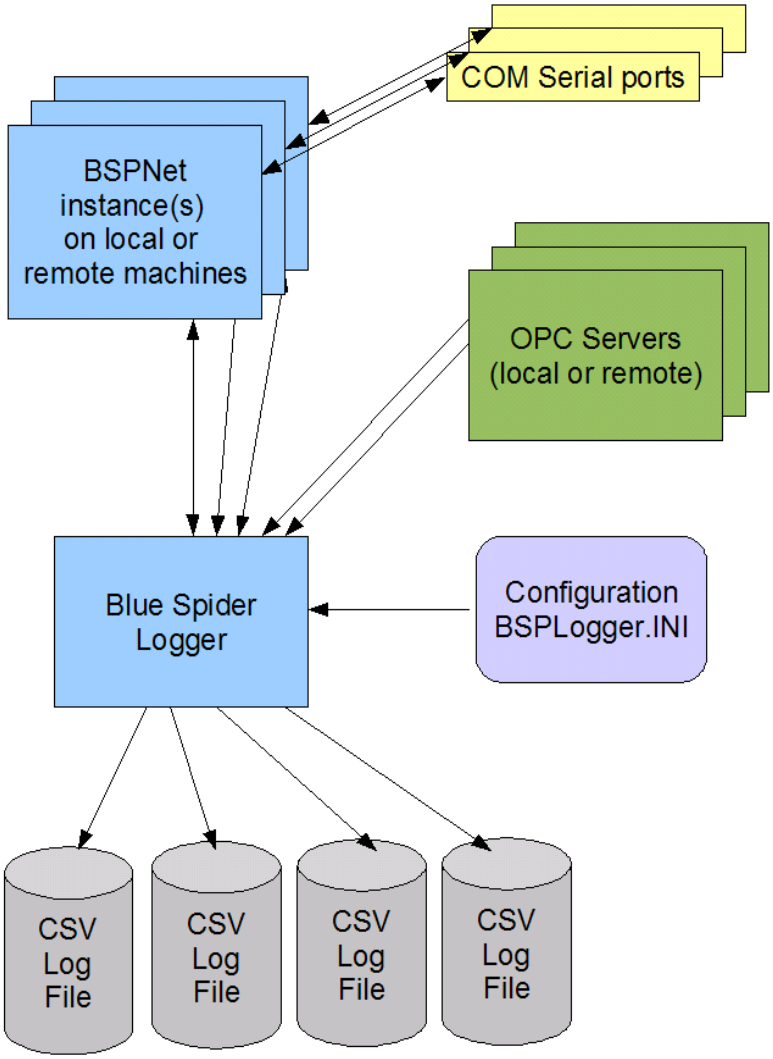

Figure 1 – Functional Overview

1.3. Installation of the software

The software is packaged as a windows installer executable. Running the initial installation offers options for installing the Logger and the BSPNet service. The Logger is typically installed on just one machine. The BSPNet service can be installed on additional machines and acts as an agent to make serial comms data available to the Logger.

Once the software has been installed automatic updating must be configured before the automatic update mechanism will work.

You may run the install again manually to upgrade but see sections 1.5. and 3.2.3.4. for implications relating to the service account name.

1.4. Deployed files

Name	Purpose
BSPLogger.exe	The logger service
BSPNet.exe	The network communications service
BSPUpdater.exe	The updater service (Checks for software updates)
BSPNetLink.dll	Internal DLL (used for comms interfacing)
QtCore4.dll	3rd party DLL (QT core runtime)
QtScript4.dll	3rd party DLL (QT script runtime)
restart_services.bat	A batch file used to restart services after automatic updates complete
WTclient.dll	3rd party DLL (Connectivity to OPC servers)
wyUpdate.exe	3rd party EXE (Installs updates)
Client.wyc	Used for automatic updating
_versions.txt	List of installed file versions

The Client.wyc is used to determine the installed versions of the files and is used by the updater programs.

restart_services.bat is also used by the updater to restart the isntalled services following an upgrade.

_versions.txt records the installed versions and this should match the installed file versions.

The Qt files are part of the Qt runtime used by all the BSP applications.

1.5. Windows services

There are 3 windows services installed:

- BSPLogger
- BSPNet
- BSPUpdater

(Note1: Depending on install options you may have only BSPNet on some machines)

(Note2: Due to the way the automatic updates currently work, regardless of installation choice, all files are deployed).

The windows services are automatically installed depending on installation choices. These can be manually started from the service control manager or from an administrator command prompt e.g. `net start bsplogger`.

The services are configured to start automatically when the machine is booted.

The services run on the following accounts (by default)

Service	Account
BSPLogger	Network Service
BSPNet	Network Service
BSPUpdater	System

You can change the account that BSPLogger runs on if necessary. BSPUpdater must always run on the System account and requires high privilege in order to be able to update the software.

It is recommended however that the default account settings are used if possible. The only possible reason for changing the BSPLogger service account would be for remote folder share permissions or issues with OPC server connectivity.

If you do not start (or disable) the BSPUpdater service then automatic updates will not work.

Note that if you do change the account for BSPLogger that automatic updates will preserve such settings. A re-run of the install program will however set it back to running on the Network Service account. Using the shortcut to manually check for updates is preferred. See section 2. Configuring the update location

1.6. Install location

On a 32 bit machine the installation location will typically be:

```
C:\Program Files\NavSystems\Blue Spider Logger
```

On a 64 bit machine the location will typically be

```
C:\Program Files (x86)\NavSystems\Blue Spider Logger
```

The only additional file that needs to be placed in this location is the **updateloc.cfg** file which is needed for configuring the software updates location. (See 2. Configuring the update location)

1.7. Configuration data

Configuration data is not supplied by the install program and must first be created before the software can perform useful functions. However the location of the configuration data needs to be known.

1.7.1. BSPLogger configuration location

On a XP or Server2003 machine this will typically be:

```
C:\Documents and Settings\NetworkService\Application
Data\NavSystems\Blue Spider Logger\System Config
```

On a Windows 7 or Server 2008 machine:

```
C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Nav
Systems\Blue Spider Logger\System Config
```

The System Config folder is where the initial **BSPLogger.ini** file must be placed.

Without this configuration file the service will start but will not perform any useful functions.

WARNING The configuration location will be different if the BSPLogger service is configured to run on a named account.

1.7.2. System Alert Logs location

The Blue Spider logger always records significant configuration events warnings and errors to a set of alert log files. The last few alert logs are always retained and older logs are automatically deleted. These built in logs require no further configuration. You can also set up your own alert log files for replication to the remote logging folder.

The alert logs provide a means of checking that the system is working correctly and make it possible to see if there are any configuration errors.

The latest alert logs will be found in...

On a XP or Server2003 machine this will typically be:

```
C:\Documents and Settings\NetworkService\Application
Data\NavSystems\Blue Spider Logger\System Logs\Alerts
```

On a Windows 7 or Server 2008 machine:

```
C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Nav
Systems\Blue Spider Logger\ System Logs\Alerts
```

WARNING The built in alert log location will be different if the BSPLogger service is configured to run on a named account.

1.8. Variables

The BSPLogger program works by assigning incoming data to variables and by logging these variables. These variables can be created in the INI file and can also be the result of arithmetic or other operations or in fact any valid Javascript expression. There are a number of built in variables are variables that are created automatically but you can also create your own variables.

Variable names use a dotted notation.

e.g.

```
Ship.WindSpeed
```

Variables can be used in expressions in the INI file such as

```
Ship.WindDirection = deg_offset(Ship.Heading,
      InputChannel5.Message1.Field1)
```

deg_offset() is a built in function for adding angles (in degrees) together (modulo 360). In this example InputChannel5.Message1.Field1 contains the wind direction from the anemometer but this is relative to the vessel heading so to get the north relative wind direction we need to add two angles.

You can also add variables together with + or perform any other arithmetic.

Variable can be numbers or strings so you can also do things like concatenate two string variables. Variables can also be arrays or composite values.

If you need to perform a more complex transformation of an input variable you can write your own Javascript code and use it

For more information see the list of available built in variables and functions.

2. Configuring the update location

In order for the automatic software updates feature to work a file called **updateloc.cfg** must be created in the program folder (see 1.6. Install location)

The **updateloc.cfg** file is a simple text file and contains just the location of the software updates.

You can set the location to a local drive e.g. c:\bspupdates. Or a network share e.g. \\server1\share\bspupdates. Or to a ftp or http address.

Note that using mapped drive letters may not work as the BSPUpdater service runs on the system account and may not be able to access mapped drives by drive letter.

Until the update location has been configured the automatic updates feature will not work.

Note that as this file has to be placed in the program folder on Windows 7 and server 2008 you will have to first create the file elsewhere and then copy it to this folder. Windows 7 does not permit editing of files directly in program folders.

NOTE
The updater (BSPUpdater service) checks for updates every day at precisely 1 minute past midnight (UTC time) .

If updates are available and need to be applied to multiple machines they will all be applied at the same time.

This is useful to minimize downtime if BSPUpdater and BSPNet (on remote machines) all need to be updated.

After the BSPUpdater service is started it may be up to 1 day before it first checks for updates.

You can also check for updates manually by using the shortcut provided in the start menu for BSPLogger.

3. BSPLogger.INI

The BSPLogger.INI file contains all the configuration required by the Blue Spider Logger. This is where you configure the inputs/outputs, OPC server addresses, variables, log files etc. The INI file is placed in the (1.7.1. BSPLogger configuration location directory). Once you have created this file you can also configure a remote location where this INI file will be replicated to/from.

3.1. INI File Sections

The INI file can contain some or all of the following sections:

Section name	Purpose
[System]	Can specify the remote configuration location and a few other options.
[ComPorts]	Specifies COM port settings for each required input.
[OpcServer1]…[OpcServer32]	Specifies OPC server names and locations.
[OpcGroup1]…[OpcGroup64]	Specifies OPC group rates and connects OPC variables to remote OPC tags.
[CustomInputFormat1]…[CustomInputFormat64]	Specify decoders for custom input messages
[CustomInputChannel1]…[CustomInputChannel32]	Assigns message(s) (input formats) to individual comms channels.
[CustomOutputFormat1]…[CustomOutputFormat64]	Defines custom output formats and assigns these to given output channels (or even to log files).
[Nav1][Nav2][Nav3]	(optional) can specify decoders for built in GPS decoder
[Gyro1][Gyro2]	(optional) can specify decoders for built in Gyro decoder
[Depth1][Depth2]	(optional) can specify decoders for built in Echo sounder decoder
[Variables]	Defines additional variables to be available for logging.
[VarHistory]	Allows history to be recorded for specified variables so that min, max, average and other statistics can be logged.
[Logging]	Defines the location where log files will be recorded and can specify a remote location where completed log files will be copied.
[LogFile1]…[LogFile32]	Defines individual log files, their types, layout, names, and recording strategy.

3.2. INI File Reference

3.2.1. [System]

The [System] section can specify a remote configuration location

Example:

```
[System]
RemoteConfig = "\\ls-chlbea\share\bspini\bsplogger.ini"
```

You should use a UNC for remote folders.

3.2.1.1. RemoteConfig=

The RemoteConfig key specifies the location of the remote configuration and must reference an accessible folder. It the folder does not initially exists it is created. If errors occur then warning will be logged to the alert log.

If changes are made to BSPLogger.INI either in the remote location or in the local configuration folder then the INI file will be automatically copied to the other location.

Note whenever the INI file is reloaded or copied between these locations an event is recorded in the alert log.

3.2.1.2. DefaultInputTimeout=

The default input timeout specifies the amount of time for all custom inputs after which if no new message is received (for a given input message) the input variables will be set to blank.

You can override the input timeout for individual message formats associated with input channels.

If no input timeout is specified here or for an individual input then the input data variables will remain valid indefinitely.

It is often worthwhile specifying a default timeout as most input data has a limited useful lifetime. If no data is received on a particular input for a period of time it is usually best to log it as blank (e.g unknown)

3.2.1.3. OPCTraceEnabled=

If this keys value is present and set to 1 then additional OPC trace messages will be written to the alert log. This will result in a lot more output but it is useful for debugging.

3.2.2. [ComPorts]

This section maps port channels to physical com ports on the same or on remote machines and specifies the settings for each.

Example:

```
[ComPorts]
GPS1  = COM1 virtual
GYRO1 = COM2 virtual
IP_01 = COM5 9600N81
IP_02 = COM6 on WKSTA1 9600N81
IP_02 = COM2 on WKSTA1 9600N81
```

The port names on the left are communications device names see the section regarding port names.

NOTE
The settings for each port are remembered by the system and by remote BSPNet machines. When adding a new port to the INI file for the first time you can expect an attempt to be made to open the port before it has actually been configured by BSPNet. This will result in initial the port open attempt to fail and an alert will be logged. The program will however re-attempt to open the port and should succeed on a subsequent attempt.

If you no longer wish to use a particular COM port then removing the setting from the INI file should be sufficient providing the BSPLogger service is running at the time. If you ever have problems with BSPNet continuing to hold ports (that are no longer used) open then temporarily add a line like:

```
IP_08 = COM2 DISABLED
```

or

```
IP_08 = COM2 on WKSTA1 DISABLED
```

3.2.2.1. Keys

The keys in this section must match communication device names.

The values can take several different formats depending on how the port should be configured.

The allowed syntax is as follows:

DeviceName = ComPortName on MachineName DCB

or

DeviceName = ComPortName DCB

or

DeviceName = ComPortName virtual

where

DeviceName is a valid communications device name
ComPortName is COM1 to COM32
DCB is a string of the form

{baudrate}PDS

where P is parity – one of:

- N - none
- O - odd
- E - even
- M - mark
- S - space

and D is number of data bits one of:

- 7
- 8

and S is number of stop bits one of:

- 1
- 1.5
- 2

3.2.3. [OpcServer1]…

The OpcServer sections configure remote OPC servers that will be used to gather data.

Example:

```
[OpcServer1]
Server = WtSvrTst2

[OpcServer2]
Server = DLLTestSvr
Machine = WKSTA1
```

You should configure a section for each OPC server you wish to connect to. The sections must be numbered sequentially from 1.

The machine name is needed only if the OPC server is running on a different machine to BSPLogger.

3.2.3.1. Server=

This key specifies the OPC server name. The value can be placed in quotes if needed.

3.2.3.2. Machine=

This key should be used if the OPC server is on another machine. For a local OPC server this key should be omitted.
The value should be the name of a machine on the local network or an IP address.

3.2.3.3. Using Matrikon OPC Tunneller

Specifying a machine name for a remote machine will work providing the machines are in the same domain or trust. To connect to OPC servers in different domains the best option is to use tunnelling software such as the tunneller product offered by Matrikon. This makes remote OPC servers look like local servers and actually also improves the performance of remote connections.

When using the Matrikon tunneller the server name for a server called “OPCServer1” on a remote machine becomes:

Tunneller::MachineName::OPCServer1

where machine name is the remote machine name.
Simply enter this after the Server= and do not specify a remote machine name.

3.2.3.4. DCOM Configuration

The BSPLogger is designed to run on the Network Service account. See 1.5. Windows services. Some OPC servers by default will not give sufficient permission to allow access from this account. If the OPC server is located on the same machine or within the same domain and you are not using tunnelling software then it is possible to change the permissions using **dcomcnfg**.

However you need to work out the name of the OPC server (as listed in **dcomcnfg**) in order to change the correct settings.

For example the Iconics simulator has the name “Simulator OPC-DA Server and Simulator OPC-AE Server”

Example:

1. Start run **dcomcnfg** OR press windows key (between CTRL and ALT) and the letter R at the same time
2. Goto Console root->Component services->My computer->DCOM config
3. Select details view on the toolbar
4. Scroll down to find "Simulator OPC-DA Server and Simulator OPC-AE Server"
5. Click right on it and select properties
6. Select the security tab
7. In launch and activation permissions select customize and press edit
8. Under groups or user names select add.
9. Type in Network Service and press OK.
10. Check all access permissions for network service and press OK.
11. repeat for the other two boxes "Access permissions" and "Configuration Permissions"
12. Press OK.

This will enable BSPLogger to access the ICONICS simulator.

Similar changes can be made for other servers.

The other option you have is to change the account that BSPLogger runs on but see section 1.5.

3.2.4. [OpcGroup1]…

The OpcGroup sections specify sets of variables to be retrieved from OPC servers

Example:

```
[OpcGroup1]
OpcGroup.Name=MyGroup1
OpcGroup.Rate=1.0
OpcGroup.ServerSection=OpcServer1
; variables from OPC tags
Train1.Node1.Tag1.PV=UNIT1.TRAIN1.NODE1.TAG1.PV
Train1.Node1.Tag1.ID=UNIT1.TRAIN1.NODE1.TAG1.ID

[OpcGroup2]
OpcGroup.Name=MyGroup2
OpcGroup.Rate=2.0
OpcGroup.ServerSection=OpcServer2
; variables from OPC tags
Thing.Fred=fred
```

3.2.4.1. OpcGroup.Name=

This specifies a unique name for the OPC group. Each group must have a unique name.

3.2.4.2. OpcGroup.Rate=

This specifies the update rate in seconds

3.2.4.3. OpcGroup.ServerSection=

This specifies the OPC server to be used for this group

3.2.4.4. All other keys are variables

Any other key in this section is a variable declaration. Variable declarations here map OPC tags to local variable names.

3.2.5. [CustomInputFormat1]…

Custom input formats specify how to decode an ASCII input message and break it up into fields

Example:

```
;  Anemometer
;
[CustomInputFormat5]
MsgName          = $IIMWV
MsgType          = 1,0,6
Field1           = 2,0,0 ;Direction
Field2           = 3,0,0 ;
Field3           = 4,0,0 ;Speed M/S
Field4           = 5,0,0 ;
```

Custom input formats typically identify a message by name and by using more than one format for a given input channel it is possible to decode inputs where more than one message is being received.

If a message does not have a name it cannot be distinguished from other messages so in this particular case only a single format should be used. Most messages can be identified by a unique name so this restriction is not normally a problem.

By default a message is assumed to have its fields delimited by a comma character. The message is also assumed to be terminated with a carriage return line feed pair “\r\n”

3.2.5.1. MsgName=

This key specifies the name at the beginning of the message. The value may be optionally enclosed in double quotes. If a double quote character is part of the message name it should be escaped by using \”

3.2.5.2. MsgType=

This optional key specifies the length and position of the message name.

If a message is terminated by an NMEA style checksum then add NMEA at the end of the MsgType value. This indicate that the last field should terminate before the *XX checksum.

See the section on field specifiers.

3.2.5.3. Field1=

For each field of a message a FieldN key should specify its position. See the section on field specifiers.

3.2.5.4. Delimiter=

The default delimiter is the comma character.
You can specify a different delimiter

e.g. If a message has fields delimited by a colon character then use:

```
Delimiter = ":"
```

Alternatively if the message has different delimiters you can specify an ordered set of delimiters using the syntax

For example a message has 4 fields delimited by a colon and semicolon

```
Delimiter = [":",";",";"]
```

3.2.5.5. Terminator=

The default terminator is “\r\n”. You can override this by specifying a different terminator.

3.2.5.6. Field specifiers

The FieldN and MsgType keys take a value of the form of 3 numbers
These 3 numbers (a,b,c) specify

a) The field index 1 based
b) The actual character position in the message. By default this should be 0 in order to use the position determined by the delimiters
c) The actual length of the field. Again if this is left at 0 the entire field is taken up to the next delimiter.

If the delimiter is set to nothing then absolute positions must be used. This is the way to decode messages where all fields are fixed size and no delimiters are used.

3.2.6. [CustomInputChannel1]…

Custom input channel sections assign messages to a given custom input channel. Custom input channels are ports with the device names IP_01 to IP_32.

Example:

```
[CustomInputChannel5]
Message1        = CustomInputFormat5
```

You can assign more than one message to a channel. Just add a Message2= and so on.

For each field in each message assigned to an input channel a variable with the name

InputChannelN.MessageN.FieldN

Will be created. These variables hold the raw input values of each field.
You may log them directly or assign to other variables first.

In addition a variable

InputChannelN.MessageN

is created for each message to hold the full decoded message

Another variable

InputChannelN.MessageN.Timestamp

is created to hold the last time the message was decoded.

3.2.6.1. Message1=

Keys Message1 to Message16 can be used to specify the message formats for each message to be received on the channel.

3.2.7. [CustomOutputFormat1]…

Custom output formats specify how output messages should be built and which channels they should be output on the output channels are the ports with the device names OP_01 to OP_16.

Example:

```
[CustomDataOutputFormat3]
CustomOutputChannel= 3
MsgName                                 = $TESTOUTPUT
Field1                                  = Train1.Node1.Tag1.PV
Field2                                  = Train1.Node1.Tag1.ID
NMEA                                    = checksum
```

3.2.7.1. CustomOutputChannel=

This key specifies the output channels this message should be sent to.

You can if desired send a message to more than one output channel.

Example:

```
CustomOutputChannel= 3, 5
```

3.2.7.2. LogToFile=

You can log messages to log files that have the log file type set to the Output type. This key specifies a log file by number (or more than one log file). Log files specified here must be of the correct type.

3.2.7.3. MsgName=

This key specifies the name of the message. You can place the value in double quotes if necessary. If no message name is required then you can omit this key

3.2.7.4. Field1=

This key specifies the variable (or expression) to be logged for each field.

3.2.7.5. Delimiter=

The default delimiter is the comma character.
You can specify a different delimiter

e.g. If a message has fields delimited by a colon character then use:

```
Delimiter = ":"
```

3.2.7.6. Terminator=

The default terminator is “\r\n”. You can override this by specifying a different terminator. The terminator key is optional but if not present the default \r\n will be used. If you want to omit the terminator entirely then specify

```
Terminator = ""
```

3.2.7.7. NMEA=checksum

If this key and value are present an NMEA style checksum is added to the end of the message

3.2.7.8. WhenTimeout=

WhenTimeout can be used to specify an interval between successive outputs of the message.

Example:

```
WhenTimeout = interval(5.0)
```

The value must be enclosed in interval()

The value specifies a period in seconds between each output of the message.

If neither WhenTimeout or Trigger are specified the message will be output at the default rate of once per second.

3.2.7.9. Trigger=

As an alternative to regular output intervals it is possible to trigger an output to occur on arrival of a specific input message or OPC group update.

You can actually specify more than one trigger condition

Valid trigger names are of the format

- InputChannel1.Message1
  An input channel message

- PortInput.DeviceName
  A raw comms input channel where device name is a valid device name

- OpcGroup.Name
  Arrival of data in an OPC group subscription where Name is the name assigned to an OPC group.

- Any valid variable name
  Any valid variable name can be used. When the variable is updated then this will trigger output.

You can have multiple trigger conditions for the output of a message.

Example:

```
Trigger=InputChannel2.Message3, PortInput.GYRO1
```

When using port device variable names such as PortInput.GYRO1 in field specifiers the Trigger should also specify the same variable name.

If neither WhenTimeout or Trigger are specified the message will be output at the default rate of once per second.

3.2.8. [Nav1]...

Configures the built in GPS decoder

Example:

```
[Nav1]
MsgName                = $GPGGA
MsgType                =  1,0,6
Time                   =  2,0,0   ; ->GPS1.Time (if no GGA)
Latitude               =  3,0,0   ; ->GPS1.Pos.Lat
LatitudeChar           =  4,0,0
Longitude              =  5,0,0   ; ->GPS1.Pos.Lat
LongitudeChar          =  6,0,0
GpsQuality             =  7,0,0   ; ->GPS1.Quality
NumSatellites          =  8,0,0   ; ->GPS1.NumSatellites
HorizontalDilution     =  9,0,0   ; ->GPS1.HDOP
Altitude               = 10,0,0   ; ->GPS1.Pos.Alt (see note)
GeoidalSeparation      = 12,0,0   ; ->GPS1.GeoidalSeparation
DGPSAge                = 14,0,0

[Nav2]
MsgName                = $GPVTG
MsgType                = 1,0,6
Heading                = 2,0,0    ; course made good from gps
SpeedKmh               = 8,0,0    ; speed calculated by gps

[Nav3]
MsgName                = $GPZDA
MsgType                = 1,0,0
Time                   = 2,0,0    ; ->GPS1.Time
Day                    = 3,0,0
Month                  = 4,0,0
Year                   = 5,0,0
```

These sections define the GPS messages that can be decoded. The values are decoded to built in variables and are then accessible for logging. You can have up to 3 GPS receivers but they must all use the same formats. Note that the number of sections in the INI file here refers to individual GPS messages and not the number of receivers.

Data to be decoded by the built in GPS decoder must arrive on ports GPS1 to GPS3 and will be decoded to the built in GPS variables.

The GPS1 port is considered as the primary receiver and in BSPLogger this cannot be overridden.

3.2.9. [Gyro1]...

Configures the built in Gyro decoder

Example:

```
[Gyro2]
MsgName       = $HEHDT
MsgType       = 1,0,6
Heading       = 2,0,0
```

Data to be decoded by the built in Gyro decoder must arrive on ports GYRO1 to GYRO3 and will be decoded to the built in GYRO variables.

The GYRO1 port is considered as the primary receiver and in BSPLogger this cannot be overridden.

3.2.10. [ScriptIncludes]

Script includes allow you to create your own functions in Javascript. By creating a .js file and placing it in the System Config\Scripts folder and adding it as an include in this ini file section any functions you define will be available to the system.

You will need to initially create the scripts folder.

Example:

```
[ScriptIncludes]
Include1 = myfunctions.js
```

Currently scripts placed in the scripts folder are not replicated to (or from) the remote repository location.

Script functions can access any variable defined in the INI file or built in variables. Variables can be passed as parameters to script functions defined in an external .js file.

You need a basic understanding of Javascript in order to write your own script functions. It is possible to write script functions that will have a detrimental affect on the performance of the data logger application or worse still cause it to crash or lock up. Care must therefore be taken when adding and using your own script functions.

3.2.11. [Variables]

The variables section allows you to create variables from other variables or extract internal values from variables.

You create variables by specifying a name for the variable as the key and any valid Javascript expression for the value.

Example:

```
Ship.WindDirection = deg_offset(Ship.Heading,
InputChannel5.Message1.Field1) {format="%.2f" heading = "Wind Direction"}
Ship.WindSpeed     = InputChannel5.Message1.Field3  {heading = "Wind
Speed"}
Fred.Quality = opcQualityOf(Train1.Node1.Tag1.PV)
Ship.AvgGPS.Pos = latlon(historyOf(GPS1.Pos.Lat).avg,
historyOf(GPS1.Pos.Lon).avg)
```

NOTE
Any variable you create in this section must be defined on a single line in the INI file. In the example above some of the lines have been wrapped.

There are a number of built in variables and these are documented in section 5. Built in Variables

You can assign to limited number of certain built in variables here.

You cannot define the same variable more than once.

Variables are accessible from Javascript code as values but they also have other properties and these can be accessed using the special functions described in section 7.2. Special functions

Variables have att that can also be defined in the INI file such as the heading (caption for log file columns), format specifiers and a number of other attributes. Attributes are defined in curly brackets after the definition.

User defined variables are recalculated when any of the variables on the RHS of the expression change. You can override this default behavior using the attributes.

For details on configuring variable attributes see section 6. Variable attributes.

3.2.12. [VarHistory]

Example

```
[VarHistory]
GPS1.Pos = {seconds=60 rate_hz=1}
Ship.Heading = {seconds=120 rate_hz=1}
```

The VarHistory section makes it possible to record the historical values of variables to an internal buffer. This in turn makes the historical data available to scripting and makes it possible to calculate statistics such as the average value, minimum and maximum etc.

In the above example GPS1.Pos actually refers to all variables starting with GPS1.Pos. GPS1.Pos will normally at least have GPS1.Pos.Lat and GPS1,Pos.Lon members.

When adding a variable like GPS1.Pos (which is a composite variable) it is effectively the same as saying

```
GPS1.Pos.Lat = {seconds=60 rate_hz=1}
GPS1.Pos.Lon = {seconds=60 rate_hz=1}
```

Unless a variable is explicitly added to the VarHistory section the history will not be available. The amount of data recorded to a variables history is defined by the parameters specified for the variable. You should not attempt to add history that covers an excessive number of samples or over a very long period of time as this can affect the performance of the application.

If the rate_hz is omitted then the history will be added to on each update of the given variable.

For additional information see section 8. History Objects

3.2.13. [Logging]

The Logging section defines a local destination directory for log files and an optional remote destination for completed log files.

Example:

```
[Logging]
Folder = c:\bsplogging
FinalDestination = \\ls-chlbea\share\logcopy
```

3.2.13.1. Folder=

This key specifies the folder where log files should be recorded directly. It should be on a drive on the local machine.

The file name can be enclosed in double quotes.

3.2.13.2. FinalDestination=

This is the location where completed logs will be copied. It can be a folder on the local machine or a UNC path to a remote drive.

Using a UNC path is recommended for accessing shares on another machine. Mapped drive letters cannot be used by programs running on the “NetworkService” account.

If completed logs cannot be copied an alert will be raised.

To copy log files to remote shared folder you may need to change the permissions on the share to allow full access to “NT AUTHORITY\NetworkService”

The alternative is to run BSPLogger on a named account but if choosing this option note that it affects the location of the configuration files.

See section 1.5. Windows services for more information.

3.2.14. [LogFile1]…

The LogFileN sections set up individual log files

Example:

```
[LogFile1]
Title                                = GPS
Type                                 = Standard
BaseFileName                         = GpsTest_
DurationInHours                      = 0.2
RateInSeconds                        = 60(60)
MaxFileSizeInBytes = 1600000
Field1 = System.Date          {heading = "System Date"}
Field2 = System.Time          {heading = "System Time"}
Field3 = GPS1.Pos.Lat         {heading = "Lat" }
Field4 = GPS1.Pos.Lon         {heading = "Lon" }
```

Another example:

```
[LogFile3]
Title                                = GyroData
Type                                 = Message
BaseFileName                         = GyroData_
DurationInHours                      = 0.2
MaxFileSizeInBytes                   = 1200000
Trigger                         = GYRO1
Field1 = System.Date                 {heading = "System Date"}
Field2 = System.Time                 {heading = "System Time"}
Field3 = bin2hex(PortInput.GYRO1)    {heading = "Gyro Data"}
```

And another:

```
[LogFile4]
Title                    = OPC1Data
Type                     = Message
BaseFileName             = Opc1Data_
DurationInHours          = 0.2
MaxFileSizeInBytes       = 1200000
Trigger           = OpcGroup.MyGroup1
Field1 = System.Date                 {heading = "System Date"}
Field2 = System.Time                 {heading = "System Time"}
Field3 = Train1.Node1.Tag1.PV        {heading = "PV" }
```

3.2.14.1. Title=

This key allows the log file to be given a meaningful name. It is a required field but it has limited uses and is not used for naming the files

3.2.14.2. Type=

This key identifies the type of log file must be set to a one of the following values

- Standard
  Standard logs are written at a periodic interval such as once a second. You define the fields of a standard log in the {LogFileN] section.

- Message
  Message logs are typically written when triggered to write by the arrival of an incoming message. The fields are defined in the same way as for standard logs.

- Output
  Output logs are like custom outputs and can therefore also be written at a regular rate or triggered by incoming data. However the data written to an output log has no column headings and does not have to be in CSV format. An output log can also have multiple messages just like a custom output. For an output log the fields making up the data are defined in custom output format(s) and not in the log file definition itself. Logging rates and triggers are ignored and the ones defined for the custom output are used instead.

- Alert
  Alert logs are exactly the same as the built in alert logs except they are recorded to your chosen location and subject to your own size and time limits.

3.2.14.3. BaseFileName=

The BaseFileName key specifies the prefix on the name of the log file.

Log files are named by combining the prefix here with the date and time when the log file is started.

3.2.14.4. Extension=

The extension key specifies a file name extension. The default is CSV. If you are logging raw binary data using a Type=Output log file then you may wish to specify a different file extension.

3.2.14.5. DurationInHours=

This keys value sets the maximum duration of the log file in hours. When this time period expires a new log file will be started and the previous one will be copied to the remote destination.

3.2.14.6. RateInSeconds=

This keys value sets the recording rate for the log file. Fractions of a second are allowed. A rate of zero is not allowed. Setting the rate at close to zero will mean that logging will be performed as fast as possible but if set to fast then the required rate might not be achievable and warnings may occur.

The RateInSeconds does not need to be specified as logging can be triggered to occur on arrival of incoming messages using . However you must specify either a rate in seconds or a trigger using Trigger.

You can specify both a rate in seconds and a trigger condition but this will mean that logging will occur at the specified rate AND when the trigger condition is true.

For Type=Output log files the trigger and rates are ignored. The log file is written to as a result of the output being triggered.

By default the rate in seconds specifies that logging will occur at a particular rate but it does not specify the exact times at which logging will occur. If you need to ensure logging occurs at exact time boundaries then you need to include a second value in brackets after the rate value.

For instance if you wish to log data at 5 minute intervals on exact 5 minute boundaries then specify the RateInSeconds as follows:

```
RateInSeconds=300(300)
```

This specifies that logging should occur every 300 seconds and that the interval starts when the current time is exactly divisible by 300 seconds. In other words logging will occur every 5 minutes and on a 5 minute boundary.

Without the value in brackets logging will still occur every 5 minutes but its unlikely to start on a boundary exactly divisible by 5 minutes as the first interval will start when the logger program first begins.

3.2.14.7. MaxFileSizeInBytes=

This keys value places a constraint on the file size. When the file is about to exceed the specified size a new log file will be started and the previous one copied to the remote destination.

If DurationInHours (or MaxLines) are also specified then it is the first of these conditions that will cause a new log file to be created.

3.2.14.8. MaxLines=

To limit the maximum number of lines in a file you can specify this limit here.

3.2.14.9. Trigger=

This keys value sets a number of possible trigger conditions that will cause a write to the log file to occur. You can use any variable name or a list of variable names as the value of this key.

If you specify more than one variable name then logging will occur as each of the variables change. In most cases a single variable should be specified.

For Type=Output log files the trigger and rates are ignored. The log file is written to as a result of the output being triggered.

4. Communications Device Names

Device Name	*Purpose*
GPS1	Primary GPS
GPS2	Additional GPS
GPS3	Additional GPS
GYRO1	Primary Gyro
GYRO2	Additional Gyro
GYRO3	Additional Gyro
ECHO1	Primary echo sounder
ECHO2	Additional echo sounder
ECHO3	Additional echo sounder
RP01	Primary motion sensor
RP02	Additional motion sensor
RP03	Additional motion sensor
OP_01	General purpose outputs
...	...

Device Name	Purpose
OP_16	...
IP_01	General purpose inputs
...	...
IP_32	...
RTT_01	Special positioning inputs
...	...
RTT_08	...
AIS_01	AIS input

For each communications device name there is a variable called PortInput.DeviceName where DeviceName is replaced with the name from the above table. These variables hold the last received data on the corresponding port. If you wish to log raw data directly from these variables or turnaround an incoming stream on an input port you should trigger this on update of the PortInput.DeviceName variable.

For PortInput.AIS_01 there is a related variable called PortInput.AIS_01.Filtered which contains the resulting filtered AIS data if filtering has been enabled. If filtering has not been configured then the value of this variable is the unfiltered AIS data. For more information see 11. AIS Filtering

5. Built in Variables

This section describes the built in variables that can be used for logging or in Javascript expressions to create additional variables.

Note all variables with a number suffix in the name can also be accessed as arrays in Javascript. Arrays use a 0 based index but the number suffixes are 1 based.

GPS1.Pos.Lat

is the same variable as

GPS[0].Pos.Lat

The array notation is more useful when you want to use another variable as the array index.

Array aliases are also created for your own user defined variables if you apply a number suffix when defining the variable.

Do not use a number suffix in variable names if this is not what was intended.

5.1. Variable Names

Quick reference summary of built in variables

Variable Name	Description
Alert.Description	Last recorded system alert string.
GPS1.Altitude	Altitude reported by the GPS receiver #1.
GPS1.AltitudeWGS84	Altitude reported by the GPS receiver #1. BSPLogger expects all GPS receivers to output position in WGS84 and this means the altitude is also with respect to the WGS84 geoid.The altitude here is the altitude of the antenna not the CRP.
GPS1.Date	Time (date) of GPS1 (DD/MM/YYYY) as received from GPS receiver #1.
GPS1.GeoidalSeparation	Geoidal separation reported by the GPS receiver #1.
GPS1.HDOP	HDOP (horizontal dilution of precision) as received from the GPS receiver #1.
GPS1.PDOP	PDOP (dilution of precision) as received from the GPS receiver #1.
GPS1.Pos.Lat	Geodetic position reported by GPS1 (latitude).
GPS1.Pos.Lon	Geodetic position reported by GPS1 (longitude).
GPS1.Quality	Quality indicator reported by GPS receiver #1.
GPS1.Sats	Number of satellites/ground stations in view to GPS receiver #1
GPS1.Time	Time (time) of GPS1 (HH:MM:SS.SS) as received from GPS receiver #1.
GPS1.VDOP	VDOP (vertical dilution of precision) as received from the GPS receiver #1.
Gyro1.Heading	Adjusted heading in degrees reported by GYRO #1
Gyro1.Raw.Heading	Raw heading in degrees reported by GYRO #1
Logging.Config1.LogType	Configured type of the primary log file #1
Logging.Config1.Name	Caption name of log file #1
Logging.Primary1.AnticipatedSize	Anticipated size of the primary log file #1 in bytes
Logging.Primary1.FileSize	Size of the primary log file #1 in bytes
Logging.Primary1.Unc	UNC filename of the primary log file #1
MRU1.Heave	The heave reported by motion sensor #1 device (in metres).
MRU1.Pitch	Pitch reported by motion sensor #1 device (in degrees).
MRU1.Roll	Roll reported by motion sensor #1 device (in

Variable Name	Description
	degrees).
RTT_01.Altitude	Altitude reported by the RTT_01 input.
RTT_01.Heading	Raw heading in degrees reported by RTT #1
RTT_01.Pos.Lat	Geodetic position of RTT_01 (latitude).
RTT_01.Pos.Lon	Geodetic position of RTT_01 (longitude).
RTT_01.WaterDepth	Water depth reported by the RTT_01 input.
Ship.EchoSounderDepth1	Raw water depth reading obtained from echo sounder #1.
Ship.Heading	Vessel heading from the primary gyro (possibly adjusted with a fixed calibration offset).
Ship.RawEchoSounderDepth	Raw water depth reading obtained from the primary echo sounder.
System.CommsScannerState	State of the communications accesiblilty scanner thread in BSPLogger.
System.Date	System date in DD/MM/YYYY format.
System.DBR.VarsRevision	System variable table revision.
System.Diag.PSC.L1CacheHits	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L1CacheMisses	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L1CacheWrites	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L2CacheCollisions	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L2CacheHits	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L2CacheMisses	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L2CacheOvers	PSC Cache diagnotics for internal use and debugging
System.Diag.PSC.L2CacheWrites	PSC Cache diagnotics for internal use and debugging
System.Time	System time in HH:MM:SS.SS format.
System.Timestamp	System date and time.
System.VMUsage	Number of bytes of virtual memory currently used by BSPLogger.

5.1.1. Alert.Description

[String]

Default caption in log files: "Alert Description"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_NOLOG

The last recorded system alert string.

5.1.2. GPS1.Altitude

[Real Number]

Default caption in log files: "GPS1 Altitude"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

The altitude reported by the GPS receiver #1. BSPLogger expects all GPS receivers to output position in WGS84 and this means the altitude is also with respect to the WGS84 geoid. The altitude here is the altitude of the antenna not the CRP.

5.1.3. GPS1.AltitudeWGS84

[Real Number]

Default caption in log files: "GPS1 Altitude (WGS84)"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

The altitude reported by the GPS receiver #1. BSPLogger expects all GPS receivers to output position in WGS84 and this means the altitude is also with respect to the WGS84 geoid.The altitude here is the altitude of the antenna not the CRP.

5.1.4. GPS1.Date

[String]

Default caption in log files: "GPS1 Date"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VTYPE2_DATE | SVY_VASSOC_NEXT | SVY_VSCAN_NOLOG

The time (date) of GPS1 (DD/MM/YYYY) as received from GPS receiver #1.

5.1.5. GPS1.GeoidalSeparation

[Real Number]

Default caption in log files: "GPS1 GeoidalSeparation"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

The geoidal separation reported by the GPS receiver #1.

5.1.6. GPS1.HDOP

[Real Number]

Default caption in log files: "GPS1 HDOP"
Default format specifier: "%.1f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VLOG_VALUE

The HDOP (horizontal dilution of precision) as received from the GPS receiver #1.

5.1.7. GPS1.PDOP

[Real Number]

Default caption in log files: "GPS1 PDOP"
Default format specifier: "%.1f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VLOG_VALUE

The PDOP (dilution of precision) as received from the GPS receiver #1.

5.1.8. GPS1.Pos.Lat

[Real Number]

Default caption in log files: "GPS1 Latitude"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VTYPE2_LAT | SVY_VUNIT_ANGLE | SVY_VLOG_VALUE

The geodetic position reported by GPS1 (latitude). The position given here is the raw position in WGS84.

5.1.9. GPS1.Pos.Lon

[Real Number]

Default caption in log files: "GPS1 Longitude"

Default format specifier: NULL
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VTYPE2_LON | SVY_VUNIT_ANGLE | SVY_VLOG_VALUE

The geodetic position reported by GPS1 (longitude). The position given here is the raw position in WGS84.

5.1.10. GPS1.Quality

[String]

Default caption in log files: "GPS1 Quality"
Default format specifier: "%.1f"
Attribute flags: SVY_VTYPE_STRING

The quality indicator reported by GPS receiver #1.

5.1.11. GPS1.Sats

[Integer]

Default caption in log files: "GPS1 Sats"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE

The number of satellites/ground stations in view to GPS receiver #1

5.1.12. GPS1.Time

[String]

Default caption in log files: "GPS1 Time"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VTYPE2_TIME | SVY_VASSOC_PREV

The time (time) of GPS1 (HH:MM:SS.SS) as received from GPS receiver #1.

5.1.13. GPS1.VDOP

[Real Number]

Default caption in log files: "GPS1 VDOP"
Default format specifier: "%.1f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VLOG_VALUE

The VDOP (vertical dilution of precision) as received from the GPS receiver #1.

Three GPS inputs are available. All of the previous variables are repeated for each.

5.1.14. Gyro1.Heading

[Real Number]

Default caption in log files: "GYRO1 Heading"
Default format specifier: "%.2f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

The adjusted heading in degrees reported by GYRO #1

5.1.15. Gyro1.Raw.Heading

[Real Number]

Default caption in log files: "Raw GYRO1 Heading"
Default format specifier: "%.2f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

The raw heading in degrees reported by GYRO #1

Up to 3 gyro inputs may be used and the previous two variables are repeated for each.

5.1.16. Logging.Config1.LogType

[String]

Default caption in log files: "Log File #1 Type"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_SLOWLOG

Configured type of the primary log file #1

5.1.17. Logging.Config1.Name

[String]

Default caption in log files: "Log File #1 Name"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_NOLOG

Caption name of log file #1

There are up to 32 log files available and the previous two variables are repeated for each.

5.1.18. Logging.Primary1.AnticipatedSize

[String]

Default caption in log files: "Primary Log #1 Anticipated File Size"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_NOLOG

Anticipated size of the primary log file #1 in bytes

5.1.19. Logging.Primary1.FileSize

[String]

Default caption in log files: "Primary Log #1 Filesize"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_NOLOG

Size of the primary log file #1 in bytes

5.1.20. Logging.Primary1.Unc

[String]

Default caption in log files: "Primary Log #1 UNC"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_SLOWLOG

UNC filename of the primary log file #1

There are up to 32 log files available and the previous three variables are repeated for each.

5.1.21. MRU1.Heave

[Real Number]

Default caption in log files: "MRU1 Heave"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

The heave reported by motion sensor #1 device (in metres).

5.1.22. MRU1.Pitch

[Real Number]

Default caption in log files: "MRU1 Pitch"
Default format specifier: "%.2f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

The pitch reported by motion sensor #1 device (in degrees).

5.1.23. MRU1.Roll

[Real Number]

Default caption in log files: "MRU1 Roll"
Default format specifier: "%.2f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

The roll reported by motion sensor #1 device (in degrees).

There are 3 motion sensor inputs available and variables for MRU2 and MRU3 are also available.

5.1.24. RTT_01.Altitude

[Real Number]

Default caption in log files: "RTT_01 Altitude"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VSCAN_SLOWLOG | SVY_VLOG_VALUE

The altitude reported by the RTT_01 input.

Note there are 8 RTT inputs available RTT_01.Altitude though RTT_08.Altitude and all the other RTT variables are available for the other RTT inputs

5.1.25. RTT_01.Heading

[Real Number]

Default caption in log files: "RTT 01 Heading"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

The raw heading in degrees reported by RTT #1

5.1.26. RTT_01.Pos.Lat

[Real Number]

Default caption in log files: "RTT_01 Latitude"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VTYPE2_LAT | SVY_VUNIT_ANGLE | SVY_VSCAN_SLOWLOG | SVY_VLOG_VALUE

The geodetic position of RTT_01 (latitude). The position given here is in your selected working datum which is not necessarily WGS84.

5.1.27. RTT_01.Pos.Lon

[Real Number]

Default caption in log files: "RTT_01 Longitude"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VTYPE2_LON | SVY_VUNIT_ANGLE | SVY_VSCAN_SLOWLOG | SVY_VLOG_VALUE

The geodetic position of RTT_01 (longitude). The position given here is in your selected working datum which is not necessarily WGS84.

5.1.28. RTT_01.WaterDepth

[Real Number]

Default caption in log files: "RTT_01 Water Depth"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VSCAN_SLOWLOG | SVY_VLOG_VALUE

The water depth reported by the RTT_01 input.

There are 8 RTT inputs available see 5.1.24. RTT_01.Altitude

5.1.29. Ship.EchoSounderDepth1

[Real Number]

Default caption in log files: "Ship Echo Sounder Depth 1"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

Raw water depth reading obtained from echo sounder #1.

There are 3 echo sounder inputs and variable Ship.EchoSounderDepth1, Ship.EchoSounderDepth2, Ship.EchoSounderDepth3

Alternatively accessible as Ship.EchoSounderDepth[0], Ship.EchoSounderDepth[1], Ship.EchoSounderDepth[2]

5.1.30. Ship.Heading

[Real Number]

Default caption in log files: "Ship Heading"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_ANGLE | SVY_VUNIT2_DEGREES | SVY_VLOG_VALUE

Vessel heading from the primary gyro (possibly adjusted with a fixed calibration offset).

5.1.31. Ship.RawEchoSounderDepth

[Real Number]

Default caption in log files: "Ship Echo Sounder Depth"
Default format specifier: "%.3f"
Attribute flags: SVY_VTYPE_DOUBLE | SVY_VUNIT_DISTANCE | SVY_VLOG_VALUE

Raw water depth reading obtained from the primary echo sounder.

5.1.32. System.CommsScannerState

[Integer]

Default caption in log files: ""SERVICE_NAME" comms scanner status"
Default format specifier: "%u"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE

The state of the communications accesiblilty scanner thread in BSPLogger. This variable is provided purely for debugging purposes to allow for diagnosis of issues in the field it is of no use for any other purpose.

5.1.33. System.Date

[String]

Default caption in log files: "Date"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VTYPE2_DATE | SVY_VASSOC_NEXT | SVY_VSCAN_NOLOG

The system date in DD/MM/YYYY format. See also System.Time

5.1.34. System.DBR.VarsRevision

[String]

Default caption in log files: "Variable Table Revision"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VSCAN_SLOWLOG

The system variable table revision. This is a GUID value that changes whenever the variable list is updated due to a reload of the INI file.

5.1.35. System.Diag.PSC.L1CacheHits

[Integer]

Default caption in log files: "SystemDiagPSC1"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.36. System.Diag.PSC.L1CacheMisses

[Integer]

Default caption in log files: "SystemDiagPSC2"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.37. System.Diag.PSC.L1CacheWrites

[Integer]

Default caption in log files: "SystemDiagPSC3"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.38. System.Diag.PSC.L2CacheCollisions

[Integer]

Default caption in log files: "SystemDiagPSC7"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.39. System.Diag.PSC.L2CacheHits

[Integer]

Default caption in log files: "SystemDiagPSC4"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.40. System.Diag.PSC.L2CacheMisses

[Integer]

Default caption in log files: "SystemDiagPSC5"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.41. System.Diag.PSC.L2CacheOvers

[Integer]

Default caption in log files: "SystemDiagPSC8"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.42. System.Diag.PSC.L2CacheWrites

[Integer]

Default caption in log files: "SystemDiagPSC6"
Default format specifier: "%d"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE | SVY_VSCAN_NOLOG

PSC Cache diagnotics for internal use and debugging

5.1.43. System.Time

[String]

Default caption in log files: "Time"
Default format specifier: NULL
Attribute flags: SVY_VTYPE_STRING | SVY_VTYPE2_TIME | SVY_VASSOC_PREV

The system time in HH:MM:SS.SS format. See also System.Date

5.1.44. System.Timestamp

[String]

Default caption in log files: "System Timestamp"
Default format specifier: "*dd/mm/yyyy hh:nn:ss.ss"
Attribute flags: SVY_VTYPE_STRING | SVY_VTYPE2_DATETIME | SVY_VSCAN_NOLOG

System date and time.

5.1.45. System.VMUsage

[Integer]

Default caption in log files: “BSPLogger virtual memory usage"
Default format specifier: "%u"
Attribute flags: SVY_VTYPE_LONG | SVY_VLOG_VALUE

The number of bytes of virtual memory currently used by BSPLogger. This variable is provided for debugging of BSPLogger especially with respect to potential memory leaks.

6. Variable attributes

Variable attributes can be applied to variables when they are defined in the INI file. The most commonly used attribute for a variable is the ability to give a variable a heading (e.g. caption) for its column in a log file. Variables in BSPLogger are actually complex Javascript objects and have more properties than just the value of the variable.

Variable attributes are defined in the INI file in curly braces after the variable definition

```
MyVariable.Value = SomeOther.VariableValue + 1   { attributes go here }
```

To give the variable named MyVariable.Value a heading for use in log files we declare it as follows:

```
MyVariable.Value = SomeOther.VariableValue + 1   { heading = “My Value” }
```

You can also define other special properties of variables in a similar way. This includes the ability to give a variables a format specifier to indicate how the value should be formatted when outputting.

All attribute definitions are placed in curly brackets and most are in the format:

{ key1=value key2=value}

For example:

```
{ heading=”My Heading” format=”%.3lf” }
```

Gives a variable a heading and a numeric format specifier indicating it should be printed to 3 decimal places.

Of course variables can also be formatted as strings using Javascript functions

6.1. heading

The heading specifier gives the variable a caption. The text for the heading should be placed in double quotes.

This attribute can be applied to variables defined in the [Variables] section, and for fields defined for a given [LogFile1], [LogFile2] etc.

Example:

```
heading=”a heading”
```

6.2. format

The format specifier makes it possible to alter the formatting of a variable when it is output. It is typically used to specify the number of decimal places to be printed.

Example:

format=”%.4lf”

6.2.1. Numeric formats

Numeric formats are similar to the C (and PHP) language printf format specifiers. As long as a variable has a value that is a valid number then a numeric format can be used.

Some examples:

- %.3lf *prints the value to 3 decimal places*
- %.2lf *prints the value to 2 decimal places*
- %lg *prints the value in exponential format (default decimal places)*
- %lf *prints the value in default format (default decimal places)*
- %.4lg *prints the value in exponential format (4 decimal places)*
- %d *prints the value as an integer*
- %03d *prints as an integer 3 digits padded with leading zeroes.*
- %x *prints the value as a hexadecimal number (lower case)*
- %08x *prints as an 8 digit hex number with leading zeroes.*
- 0x%08x *same as above but prefixed with 0x*

For more information look up printf on google.

6.2.2. Special formats

For certain types of variables latitude, longitude, time and date it is possible to use alternative format specifiers.

6.2.2.1. Date and time formats

Date and time variables can be formatted using a format specifier such as:

yyyy-mm-dd

or

hh::nn:ss.sss

6.2.2.2. Latitude and longitude formats

Latitude or longitude variables can be formatted using format specifiers such as

DDD mm.mmm H

DDD.ddddddd

HDDD mm ss.ssss

6.3. Variable calculation dependencies

By default variables defined in the INI file are automatically recalculated whenever any of the variables used in the expression are changed. In most cases this is an acceptable default behaviour. For expressions involving many variables or conditionally using different input variables this default behaviour may result in an excessive number of recalculations of the output variable. This also means that if the output variable has an associated history object that the number of entries added to the history is increased and this can results calculated statistics being artificially skewed.

Consider a variable that depends on ship position and heading. Ship position and heading typically arrive in different messages at different times and possibly different rates. You may only wish to calculate such a variable when the GPS position is updated. To prevent the gyro heading being used as well you need to override this default behaviour.

To do this add an additional attribute

#calc_after[var1,var2,var3...]

where var1, var2, var3 etc. are replaced with the names of variables you want to trigger the recalculation. You can have one or more variables listed here. The variables used in the expression itself are no longer considered when determining when the output variable should be recalculated and instead your supplied list effectively says calculate the output variable whenever any of the listed variables changes.

7. Built in functions

This sectin describes the built in functions that are available. You can define additional functions by writing your own scripts.

7.1. Standard functions

The following functions may be used in expressions e.g. when defining custom variables in the INI file:

7.1.1. abs(x)

Compute absolute value.
Returns the absolute value of x.

Parameters
x
Floating point value.

Return Value
The absolute value of x.

7.1.2. acos(x)

Compute arc cosine.
Returns the arccosine of x.

Parameters
x
Floating point value.

Return Value
The arccosine value of x.

7.1.3. acosh(x)

Compute hyperbolic arc cosine.
Returns the hyperbolic arccosine of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic arccosine value of x.

7.1.4. asin(x)

Compute arc sine.
Returns the arcsine of x.

Parameters
x
Floating point value.

Return Value
The arcsine value of x.

7.1.5. asinh(x)

Compute hyperbolic arc sine.
Returns the hyperbolic arcsine of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic arcsine value of x.

7.1.6. atan(x)

Compute arc tangent
Returns the arctan of x.

Parameters
x
Floating point value.

Return Value
The arctan value of x.

7.1.7. atan2(y,x)

Compute arc tangent with two parameters.
Returns the principal value of the arc tangent of y/x, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant.

Parameters
y
Floating point value representing an y-coordinate.

x
Floating point value representing an x-coordinate.

If both arguments passed are zero, an error occurs and the result will be infinity.

Return Value
Principal arc tangent of y/x, in the interval [-pi,+pi] radians.

7.1.8. atanh(x)

Compute hyperbolic arc tangent
Returns the hyperbolic arctan of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic arctan value of x.

7.1.9. bin2hex(s)

Convert (binary) string to hexadecimal string
Returns the hexadecimal encoded string.

Parameters
s
String value.

Return Value
String containing hexadecimal encoded digits.

7.1.10. ceil(x)

Round up value
Returns the smallest integral value that is not less than x.

Parameters
x
Floating point value.

Return Value The smallest integral value not less than x.

7.1.11. chr(n)

Convert numeric value to ASCII character
Returns converted value as string containing a single character.

Parameters
n
Numeric value (in the range 0 to 255)

Return Value String with one character.

7.1.12. cos(x)

Compute cosine
Returns the cosine of x.

Parameters
x
Floating point value.

Return Value
The cosine value of x.

7.1.13. cosh(x)

Compute hyperbolic cosine
Returns the hyperbolic cosine of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic cosine value of x.

7.1.14. deg_offset(a, b)

Performs modulo 360 addition to add angle values.

Parameters
a
Numeric angle value in the range 0 <= a < 360.

b
Value to add (can be negative)

Return Value
The resulting angle value.

7.1.15. exp(x)

Compute exponential function
Returns the base-e exponential function of x, which is the e number raised to the power x.

Parameters
x
Floating point value.

Return Value
Exponential value of x.

7.1.16. floor(x)

Round down value
Returns the largest integral value that is not greater than x.

Parameters
x
Floating point value.

Return Value
The largest integral value not greater than x.

7.1.17. hexdec(s)

Convert hexadecimal string to integer.

Parameters
s
String containing hexadecimal number.

Return Value
Integer value

7.1.18. iif(b,v1,v2)

Conditional evaluation.

Parameters
b
Boolean 0 or 1 (non zero is considered true) this parameter is usually the result of a comparison.

v1
Value to be returned if b is true.

v2
Value to be returned if b is false.

Return Value
Either v1 or v2 depending on the value of b.

7.1.19. hex2bin(s)

Convert hexadecimal string to binary/ASCII
Returns the resulting binary string. Note that this can potentially contain non-printable characters.

Parameters
s
String value (binary data).

Return Value
String containing decoded characters.

7.1.20. ln(x)

Compute natural logarithm
Returns the natural logarithm of x. The natural logarithm is the base-e logarithm, the inverse of the natural exponential function (exp). For base-10 logarithms, a specific function log exists.

Parameters
x
Floating point value. If the argument is negative, a domain error occurs, and the result will be NAN If it is zero, the function returns -infinity.

Return Value Natural logarithm of x.

7.1.21. log(x)

Compute base 10 logarithm
Returns the base 10 logarithm of x.

Parameters
x
Floating point value. If the argument is negative, a domain error occurs, and the result will be NAN If it is zero, the function returns -infinity.

Return Value Base 10 logarithm of x.

7.1.22. ord(c)

Get ASCII code of first character in string.
Returns ascii code of a character value.

Parameters
c
String value (if the string is longer than once character the remaining characters are ignored).

Return Value
An integer containing the ASCII code value. Note if the string is empty the returned value is -1.

7.1.23. sgn(x)

Obtains the sign of a numeric value

Parameters
x
Numeric value.

Return Value
The value -1, 0 or 1 depending on the whether x is negative, zero, or positive respectively.

7.1.24. sin(x)

Compute sine
Returns the sine of x.

Parameters
x
Floating point value.

Return Value
The sine value of x.

7.1.25. sinh(x)

Compute hyperbolic sine
Returns the hyperbolic sine of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic sine value of x.

7.1.26. sqrt(x)

Compute square root
Returns the square root of x.

Parameters
x
Floating point value.

If the argument is negative, a domain error occurs, a NAN will be returned.

Return Value
Square root of x.

7.1.27. strcat(s1,s2)

Concatenate two strings.

Parameters
s1
First string.

s2
Second string.

Return Value
String, s1 concatenated with s2

7.1.28. strcmp(s1,s2)

Compare two strings.
This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until the end of one of the the strings is reached.

Parameters
s1
First string.

s2
Second string.

Return Value
Returns an integral value indicating the relationship between the strings: A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

7.1.29. stricmp(s1,s2)

Compare two strings (case insentitive).
This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until the end of one of the the strings is reached.

Parameters
s1
First string.

s2
Second string.

Return Value
Returns an integral value indicating the relationship between the strings: A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

7.1.30. stripos(s1, s2, [index])

Find position of substring (case insenstive version)
Searches s1 for the first occurance of s2.

Parameters
s1
String to be seached

s2
String to search for

index [optional]
position to start seaching (0 is the first character position)

Return Value
Integer, -1 if not found, otherwise the 0 based index of the begining of the string s2 in s1.

7.1.31. strlen(s)

Get string length
Returns the length of s.

Parameters
s
String

Return Value
Integer, length of the string.

7.1.32. strpos(s1, s2, [index])

Find position of substring
Searches s1 for the first occurance of s2.

Parameters
s1
String to be seached

s2
String to search for

index [optional]
position to start seaching (0 is the first character position)

Return Value
Integer, -1 if not found, otherwise the 0 based index of the begining of the string s2 in s1.

7.1.33. strtolower(s)

Convert string to lowercase
Returns a string containing all upper case letters replaced with lower case equivalents.

Parameters
s
String

Return Value
String

7.1.34. strtoupper(s)

Convert string to uppercase
Returns a string containing all lower case letters replaced with upper case equivalents.

Parameters
s
String

Return Value
String

7.1.35. substr(s, start, [len])

Extracts a part of a string
Returns the portion of string specified by the start and length parameters.

Parameters
start
Zero based index of the start position.

len [optional]
Number of characters to extract. If this parameter is omitted then the remainder of s will be returned.

Return Value
Returns the extracted part of string.

7.1.36. tan(x)

Compute tangent
Returns the tan of x.

Parameters
x
Floating point value.

Return Value
The tan value of x.

7.1.37. tanh(x)

Compute hyperbolic tangent
Returns the hyperbolic tan of x.

Parameters
x
Floating point value.

Return Value
The hyperbolic tan value of x.

7.1.38. value(x)

Converts a string to a number
A string containing spaces between digits or sign or containing comma's can be converted to a well formed number using this function.

Parameters
x
String.

Return Value
Returns the number or blank if not a valid number

7.2. Special functions

Special functions can typically only be to variable objects defined in the INI file

7.2.1. timestampOf(var)

Obtains the timestamp for a given variable

Parameters
var
Variable.

Return Value
Timestamp (real number)

(Windows FILETIME value converted to seconds)

7.2.2. sourceTimestampOf(var)

Obtains the source timestamp for a given variable. This only works with OPC variables. Note that this is the timestamp provided by the remote OPC server

Parameters
var
Variable.

Return Value
Timestamp (real number)

(Windows FILETIME value converted to seconds)

7.2.3. historyOf(var)

Obtains the history object for a given variable

Parameters
var
Variable.

Return Value
History Object

See Variable History Objects

7.2.4. flagsOf(var)

Obtains the type code flags for a given variable

Parameters
var
Variable.

Return Value
Long

7.2.5. opcQualityOf(var)

Obtains the OPC quality code for a given variable. The variable should be defined in an OPC group

Parameters
var
Variable.

Return Value
Long

7.2.6. variableUpdated(var)

Determines if the specified variable has been updated during the current execution loop of BSPLogger. This function is intended to be used in scripts that conditionally update variables dependant on the value of at least one other variable.

Parameters
var
Variable.

Return Value
Boolean

True if the variable in question has been updated.

7.2.7. freq(hist)

Frequency counter.
freq(var_history [, min_period_hz [, min_magnitude [, crossing]]]);

Example:

```
Ship.RollFrequency = freq(historyOf(Ship.Roll), 5, 0.1);
```

Returns:
frequency in Hz. Zero if data is rejected.

var_history:
History object for a variable. This can be obtained for a variable by using the historyOf function. The variable must be present in the VarHistory section of the INI file.

min_period_hz (optional):
The minimum period at which average point crossings are rejected.

crossing (optional):
crossing value at which edges are counted. If omitted or 'null', then the average value of the signal is used.

min_magnitude (optional):
Rejects data that does not exceed this magnitude. If data is rejected then the return value will be zero. Use to avoid small vibrations resuling in a high frequency (e.g. if calculating roll then this could occur due to engine vibration when the vessel is on glass flat water).

8. History Objects

History objects are created by defining them in the [VarHistory] section of the INI file. See 3.2.12. [VarHistory]

You can create one history object per variable. If you ever need more than one history object for a given variable just create another variable assigned from the original.

History objects record history for a given variable and allow certain statistics to be calculated. You can also access the raw recorded data. History objects store the data in memory so there is a limit to the amount of data that can be stored. Storing too much data in a history object can seriously impact performance so should be avoided.

The properties of history objects are detailed in the following sections.

Timestamp values are in seconds and are effectively windows FILETIME values converted to seconds.

You can write your own Javascript functions to make use of history objects for more sophisticated analysis if needed.

To obtain the history object for a given variable use the expression:

```
historyOf(var)
```

Where var is the name of the variable with the history you want to access.

8.1. Properties

History objects provide the following (read-only) properties:

8.1.1. length

This, like a Javascript array, returns the number of elements

8.1.2. timeRange

This returns the time range of the recorded data in seconds

8.1.3. rateHz

This is the sampling interval or 0 if only updated as triggered by updates to the specified variable.

8.1.4. secondsPerSample
Analogous to rate but the interval instead of the frequency. e.g. (1 / rateHz)

8.1.5. min
The arithmetic minimum value.

8.1.6. max
The arithmetic maximum value.

8.1.7. avg
The arithmetic average value

8.1.8. avgmod2pi
The average angle value for angles in radians 0 to 2pi. This special average is computed by summing the unit vectors for each angle and then computing the resultant.

8.1.9. avgmod360
The average angle value for angles in degrees 0 to 360. This special average is computed by summing the unit vectors for each angle and then computing the resultant.

8.2. Accessing elements
Elements can be accessed using Javascript array syntax. The values are returned.

e.g.
historyOf(var)[0] returns the value of the first element.

In addition the following two methods can be used:

- **valueAt**(index)
  Returns the value at the given index

- **timestampAt**(index)
  Returns the timestamp at the given index

8.3. Additional functions

The history object also supports a method:

- **lowerBound**(ts)
  This method returns the index prior to or at the given timestamp value.

9. Using scripts

Scripts can be defined for a variety of purposes

e.g.

- Defining your own constants.
- Creating your own functions to use in expressions in the INI file
- Updating variables conditionally

Scripts are defined by adding a [ScriptInclude] section to the INI file. See section 3.2.10. [ScriptIncludes]

In addition to the functions defined in section 7. Built in functions you can also use standard Javascript functions.

9.1. Defining your own constants

Self explanatory really but by defining your own constants in a .js file you will actually get slightly faster script execution than by defining them in the INI file. All variable objects defined in the INI file are actually somewhat more heavyweight than regular Javascript variables.

9.2. Defining your own functions

You can define your own functions (taking arguments) as regular Javascript functions and call them from expressions in the INI file. All expressions in the INI file have to fit on one line but functions in a .js file are not subject to this restriction.

9.3. Updating variables conditionally

In the INI file you can define one variable as being the result of operations on another but its unconditional. Suppose you are receiving a message which contains a value and an item identifier. For example the item identifier is a tank number and the value is the fluid level in the tank. You have 4 tanks and you want to record the level in each. You cannot do this directly in the INI file.

Lets say that we decode the incoming message (it doesn't matter what this mystery message looks like) and we have the required data in InputChannel1.Message1.Field1 and Field2. Field1 is the level value, Field2 is the tank number.

To split this data out into 4 tank level variables we first declare the tank level variables in the INI file

Example:

```
[Variables]
Tank1.Level = _INPUT
Tank2.Level = _INPUT
Tank3.Level = _INPUT
Tank4.Level = _INPUT
```

Here instead of specifying an expression for each variable we use the placeholder **_INPUT** this means that we will be able to assign to these variables from a script.

Next we must create some Javascript code.

<<tank_levels.js>>

```
Server.calcUserVarsStage1.connect(updateTankVars);

function updateTankVars() {
      if (variableUpdated(InputChannel1.Message1.Field1) &&
          variableUpdated(InputChannel1.Message1.Field2)
      ) {
            var t = Number(InputChannel1.Message1.Field2);
            if (t > 0 && t <= 4) {
                  --t;
                  Tank[t].Level = InputChannel1.Message1.Field1;
            }
      }
}
```

The first line in the js code is executed once for initialization. It connects an event called calcUserVarsStage1 to our function. It means that BSPLogger will call our function on every cycle of its main processing loop.

In our function we must check to see if the input variables were updated. This will only be true if the mystery message has been received and spilt into the input fields. It is very important to do this check or we will end up using a lot of processing power and achieve nothing.

Next we extract the tank number. Notice that the input field variables are strings so we must coerce the value to a number. We then check that the tank number is valid and in the range 1 to 4.

After this check we could have used a switch statement but all numbered variables declared in the INI file e.g. Tank1, Tank2, Tank3, Tank4 are also accessible as arrays. We must however use a zero based index. So Tank1.Level can also be accessed as Tank[0].Level. We take advantage of this in our script.

10. Reserved Words

The following words cannot be used in variable names either because they are reserved in the Javascript language or because they would conflict with existing objects.

10.1. Reserved by Javascript language

(Primary language keywords)

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- debugger
- default
- delete
- do
- double
- else
- enum
- export
- extends
- false
- final
- finally
- float
- for
- function
- goto
- if
- implements
- import
- in
- instanceof
- int
- interface
- let
- long
- native
- new
- null
- package
- private
- protected
- prototype
- public
- return
- short
- static
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- typeof
- var
- void
- volatile
- while
- with
- yield

(Classes that should be reserved)

- Array
- Date
- JavaArray
- JavaClass
- JavaObject
- JavaPackage
- Math
- NaN
- Number
- Object
- String

(Special Functions / properties)

- toString
- valueOf
- length
- assign
- eval
- isFinite
- isNaN
- isPrototypeOf
- parseFloat
- parseInt
- getClass
- hasOwnProperty
- constructor
- Infinity
- parent
- undefined
- propertyIsEnum
- self

10.2. Reserved by BSPLogger

- Vars
- Server

10.2.1. Vars

This object actually holds all other variables although they are all available at global scope as well. Vars.GPS1.Pos.Lat is the same as GPS1.Pos.Lat is the same as GPS[0].Pos.Lat.

10.2.2. Server

This is an object that can be used in scripts. See 9.3. Updating variables conditionally.

11. AIS Filtering

AIS filtering is an option that can be used to significantly reduce the amount of AIS data needing to be logged. For the raw device input variable PortInput.AIS_01 there is another corresponding variable called PortInput.AIS_01.Filtered. This variable exists regardless of whether filtering has been configured. If filtering has not been configured then this variable just holds the unfiltered raw input data. See section 4. Communications Device Names

11.1. Configuring AIS filtering

To configure AIS filtering you set up a new section in the INI file as follows:

```
[AIS_01]
FilterRadius = 50000
FilterUnknown = true
FilterDeferIdents = true;
FilterInclude = "$AIALR"
```

There are a number of options for the AIS filter:

11.1.1. FilterRadius=

This sets up a distance filter with the specified distance in metres. This is the primary means of reducing the amount of data that needs to be logged.

11.1.2. FilterUnknown=

If this key is present (and set to true) then any message that is not a known AIS message will be automatically excluded from the output. Most AIS systems will output a number of additional messages that BSPLogger does not recognize or decode. There is often very little need to log these messages. If a particular message needs to be logged it can be added to the FilterInclude list. See below.

11.1.3. FilterDeferIdents=

Raw AIS messages come in a variety of types but broadly speaking there are two categories: A) messages that give details about the vessel associated with a particular MMSI. B) messages that tell us about the position of an object with a given MMSI number. The category A messages do not tell us the position of an object but only its name and other information. If FilterDeferIdents is set to true then category A messages will not be passed through until a category B message for the same MMSI number arrives and the position is within the filter radius. This way we avoid logging data (the category A messages) for objects that are outside the radius. If FilterDeferIdents is missing or not set to true then the position messages

(category B) will be filtered but the information messages (category A) will only be filtered once a category B message has arrived.

11.1.4. FilterInclude=

This key allows any message with a given name to be passed through the filter. For instance if you want to log AIS system alarm messages then include $AIALR as in the above example. Normally you will not have this key present. If you are only interested in positions of nearby vessels then you do not need to pass through any additional messages.

12. Some Worked Examples
12.1. Logging of vessel track and roll period

Relevant parts of INI file

```
[Nav1]
MsgName             = $GPGGA
MsgType             =  1,0,6
Time                =  2,0,0
Latitude            =  3,0,0
LatitudeChar        =  4,0,0
Longitude           =  5,0,0
LongitudeChar       =  6,0,0
GpsQuality          =  7,0,0
NumSatellites       =  8,0,0
HorizontalDilution  =  9,0,0
Altitude            = 10,0,0
GeoidalSeparation   = 12,0,0
DGPSAge             = 14,0,0

[Nav2]
MsgName             = $GPVTG
MsgType             = 1,0,6
Heading             = 2,0,0
SpeedKmh            = 8,0,0

[Nav3]
MsgName             = $GPZDA
MsgType             = 1,0,0
Time                = 2,0,0
Day                 = 3,0,0
Month               = 4,0,0
Year                = 5,0,0

[Gyro1]
MsgName              = $HEHDT
MsgType              = 1,0,6
Heading              = 2,0,0

[ComPorts]
GPS1  = COM1 9600N81
Gyro1 = COM5 9600N81
RP01 = COM6 19200N81

[VarHistory]
Ship.Motion.Roll = { seconds=120 }

[Variables]
Ship.Motion.Roll = MRU1.Roll
Ship.Motion.RollPeriod = 1.0 / freq(historyOf(Ship.Motion.Roll))

[LogFile1]
Title                                = GPS
Type                                 = Standard
BaseFileName                         = VesselTrack_
DurationInHours                      = 2
RateInSeconds                        = 4
Field1 = System.Date  {heading = "Date"}
Field2 = System.Time  {heading = "Time"}
Field3 = GPS1.Pos.Lat  {heading = "Latitude" }
Field4 = GPS1.Pos.Lon  {heading = "Longitude" }
Field5 = Gyro1.Heading  {heading = "Ship Heading" format="%.3f"}
Field6 = Ship.Motion.Roll {heading = "Ship Roll" format="%.3f"}
Field7 = Ship.Motion.RollPeriod {heading = "Roll Period" format="%.3f"}
```

Sample output

```
Date,Time,Latitude,Longitude,Ship Heading,Ship Roll,Roll Period
12/10/2012,10:24:10.89,24 15.236628 S,148 46.451858 E,193.300,0.488,10.005
12/10/2012,10:24:14.95,24 15.310595 S,148 46.354297 E,197.100,0.088,10.005
12/10/2012,10:24:18.99,24 15.394518 S,148 46.247395 E,196.500,-0.635,9.994
12/10/2012,10:24:23.04,24 15.459977 S,148 46.160460 E,197.100,0.965,10.001
12/10/2012,10:24:27.09,24 15.535678 S,148 46.064371 E,194.600,-0.953,9.992
12/10/2012,10:24:31.13,24 15.610784 S,148 45.966254 E,196.800,0.617,10.004
12/10/2012,10:24:35.19,24 15.684840 S,148 45.869845 E,194.600,-0.066,9.994
12/10/2012,10:24:39.24,24 15.760531 S,148 45.771607 E,187.500,-0.525,9.994
12/10/2012,10:24:43.29,24 15.835372 S,148 45.674750 E,192.600,0.901,10.005
12/10/2012,10:24:47.33,24 15.909588 S,148 45.575479 E,187.500,-0.989,10.000
12/10/2012,10:24:51.38,24 15.985196 S,148 45.479281 E,179.800,0.711,10.005
```

[Nav1] to [Nav3] sections configure the built in GPS decoder

[Gyro1] section configures the built in gyro decoder.

The [ComPorts] section configures GPS, Gyro and motion sensor (RP01) device inputs.

The [VarHistory] section sets up history recording for the Ship.Motion.Roll variable

The [Variables] section defines this variable and the RollPeriod variable is defined using the freq() function which performs a frequency counter operation on the history of the Ship.Motion.Roll

[LogFile1] section defines the layout and headings for the log file, sets up the logging rate and duration of the log file.

12.2. Logging of raw or filtered AIS data

Relevant parts of INI file:

```
[ComPorts]
.
AIS_01 = COM7 on WKSSTA3 9600N81
.
.
.
[CustomDataOutputFormat1]
Delimiter = ""
Terminator = ""
Field1 = PortInput.AIS_01.Filtered
Trigger = PortInput.AIS_01.Filtered
LogToFile = 2
.
.
.
[LogFile2]
Title           = AIS
Type            = Output
BaseFileName    = FilteredAIS_
Extension       = txt
MaxLines        = 65536
```

Sample output:

```
!AIVDM,1,1,,A,35D7EH5000O`msFLdjD<Lp@J0000,0*20
!AIVDM,2,1,5,B,53P7oa02=;KQI51SN21LPU@<P4m0Ttr22222221@000005KeI<1PC3Cm,0*4D
!AIVDM,2,1,6,B,55D7EH2W3Oo0Pw?33:0t<D4r04hE9B222222221S2Pk836?os=QPC3Cm,0*35
!AIVDM,2,2,6,B,E28888888888880,2*56
!AIVDM,1,1,,A,35D7EH5000O`msFLdjD<Lp@J0000,0*20
!AIVDM,1,1,,A,333d341000O`sd`Lc7AM=2nP0000,0*0A
!AIVDM,1,1,,B,14V?fN0000O`pSVLdgWS>F:l20SW,0*58
!AIVDM,1,1,,B,14V?fN0000O`pSVLdgWS>F:l20SW,0*58
!AIVDM,1,1,,B,14V?fN0000O`pSVLdgWS>F:l20SW,0*58
!AIVDM,1,1,,A,33udhF5P00O`oU8Ldg0h0?w00000,0*0E
!AIVDM,1,1,,A,33udhF5P00O`oU8Ldg0h0?w00000,0*0E
!AIVDM,1,1,,B,13P9cvPOh0O`l6`Ldd37rUdv00S;,0*1C
```

To log filtered or raw AIS data you need to configure a port as the AIS input by assigning an port to the AIS_01 device name.

To configure logging of raw input data to be logged as received and without CSV formatting it is easiest to configure a custom output format and specify that this be logged to a particular output file (LogToFile=2)

[LogFile2] is configured as a Type=Output log and the file extension is set to .TXT

Finally we place a limit on the number of lines.

To log the raw AIS input data or any other raw input data in a similar way simply replace PortInput.AIS_01.Filtered with PortInput.AIS_01 or a different device name.

12.3. Logging of OPC data

TBD

12.4. Logging of system alerts

TBD

12.4.1. Decoding and logging more complex messages