

---

# Slingshot Version 2.4.x Administration Guide

Version 2.4  
Document Version v1.10

Date 11.11.2008

---

**For more information contact:**

Swissrisk Financial Systems GmbH  
Technical Support  
Holzhausenstrasse 44  
D-60322 Frankfurt  
Germany

Phone +49 69 50952 111  
Fax: +49 69 50952 199  
Email [hotline@swissrisk.com](mailto:hotline@swissrisk.com)  
Website [www.swissrisk.com](http://www.swissrisk.com)

---

## Copyright Information

This document is protected by copyright law and may not be reproduced or distributed either in part or in total. The licensee is not allowed to pass on the software or the accompanying written materials to third parties or make them otherwise available without prior written agreement of the licensor. Information in this document that refers to possible product extensions or to available accessories is not legally binding, especially because the products are subject to continuous adaptation and because the information may also relate to future development. The contents of this document can change without prior notice and does not represent any legal obligation on the part of Swissrisk Financial Systems GmbH.

Swissrisk Financial Systems GmbH cannot be made liable for the correctness of information in this document nor for damages resulting from the use of this information or the impossibility of using this information. All other legal regulations for using the software and the corresponding documentation are set in the applicable license agreement.

**InVision** is a trademark of Swissrisk Financial Systems GmbH. All other product and company names mentioned in this manual are trademarks of their respective companies.

### **Published by:**

Copyright © Swissrisk Financial Systems GmbH |  
All rights reserved

Swissrisk Financial Systems GmbH  
Holzhausenstrasse 44  
D-60322 Frankfurt  
Germany

Phone: +49 69 50952-0  
Fax: +49 69 50952-333  
Website: [www.swissrisk.com](http://www.swissrisk.com)

## TABLE OF CONTENTS

<b>1 INTRODUCTION.....</b>	<b>2</b>
1.1 WDS Server.....	2
1.2 Client API Module.....	4
<b>2 WDS FUNCTIONALITY AND ADMINISTRATION.....</b>	<b>6</b>
2.1 InVision Requester API Module.....	6
2.2 Main Module.....	6
2.3 HTTP Server Module.....	6
2.4 License.....	7
2.5 Client API.....	7
2.6 Recovery.....	7
2.7 Record Merging and Output Bandwidth Restriction.....	8
2.8 HTTP Web Server.....	8
2.9 WDS Administration.....	8
2.10 WDS Services.....	14
<b>3 SLINGSHOT DATA PERMISSIONS.....</b>	<b>16</b>
3.1 Permissions Data Format.....	16
3.2 Sections.....	17
3.3 Permission Priority.....	19
3.4 Update Filtering.....	19
<b>4 WDS CONFIGURATION EXAMPLES.....</b>	<b>20</b>
4.1 Example 1 - BASIC Authorization with WDS making HTTP Request.....	20
4.2 Example 2 - AUTHORISATION_ID with WDS making a HTTP Request.....	21
4.3 Example 3 - AUTHORISATION_ID with WDS making LDAP Requests.....	22
4.4 WDS Configuration of Delayed Data for Services.....	23
<b>5 WDS CONFIGURATION – WDS.INI.....</b>	<b>25</b>
5.1 Logging configuration.....	25
5.2 Main WDS Logging.....	26
5.3 Commented wds.ini file.....	34

# 1 INTRODUCTION

The Slingshot Web Distribution System (WDS) is an HTTP server, which can deliver real time market data to the internet. It consists of two basic components, the *Server* and the *Java client API Module* or a *C++ client API Module*. Communication between the two is via HTTP tunnelling technique. This will allow it to operate in the internet environment through proxy servers and firewalls. The *client API* can not only request records (either by monitoring or by snap shots) but can also do inserts or contributions to fields of a record. The WDS can operate to two modes either using normal http requests or in Secured Socket Layer (SSL) mode. WDS uses OpenSSL for its Secured Socket Layer. For further information on OpenSSL refer to :- <http://www.openssl.org/>

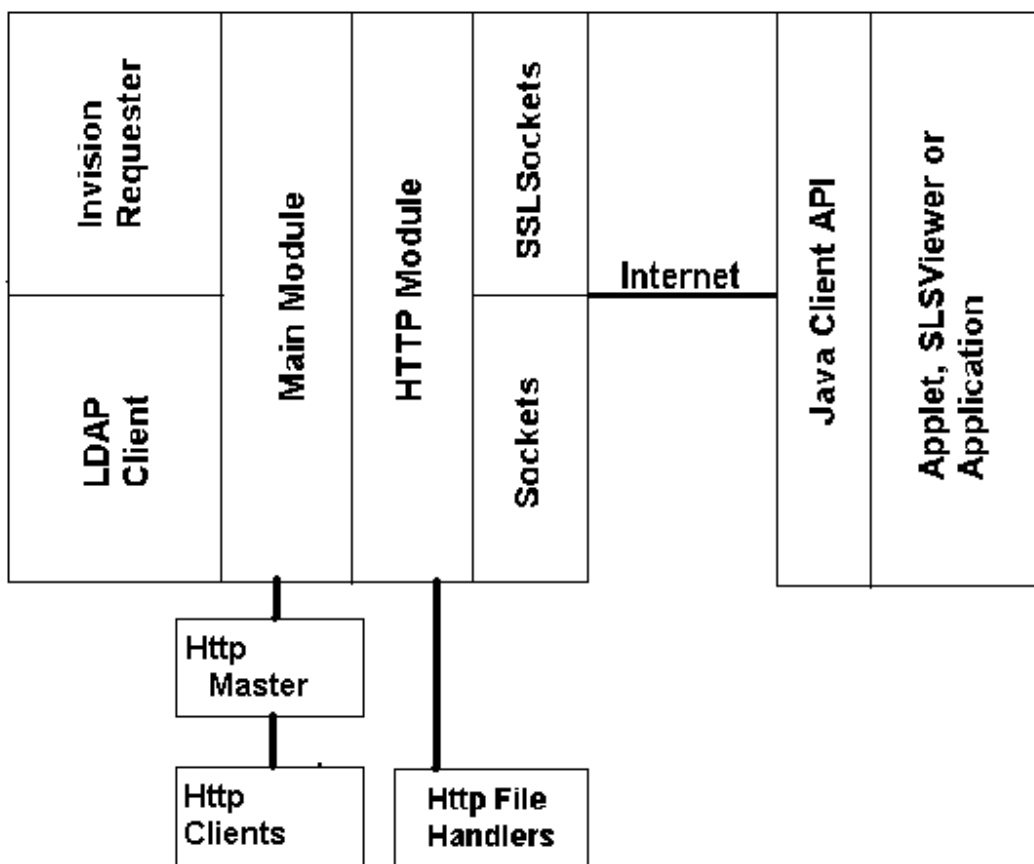


Figure 1. WDS and Java Client API

## 1.1 WDS Server

Currently the WDS provides the following facilities

- **Administration Port**

The WDS provides a port that a telnet session can connect to. It provides the following commands as subcommand of the WDS main command:

Stats – This will return stats about the various sections in the server.

Reset – This will reset all stats.

Get / Set – Get a list of variables that can be accessed by this session or change them.

User – Displays information about connected users.

Group – Displays information about groups

Perm – Displays information about permissions

Watched – Displays what records are watched by Connection or User

Mod – Displays Module information

Conn – Displays information about current connections.

Key – Displays information about watchers of records.

Lic – Displays WDS License information

Ver – Displays WDS Version

Shutdown – This will shutdown the WDS.

- **WDS Service**

This is a service in the WDS that contains all the statistics as records. Currently these stats consist of three records MAIN, HTTP and REQUESTER. These records represent the three sections in the WDS. These statistics can be treated like a normal service and record.

- **Secured Socket Layer (SSL)**

The WDS allows for SSL connections. It uses the OpenSSL implementation of SSL. refer to :- <http://www.openssl.org/>

- **HTML Echo Page**

The WDS will echo a GET request in a HTML page. This is accessed by requesting /WDS/echo to the server, e.g. by requesting <http://www.name.com:8000/WDS/echo> using a browser like Internet Explorer or Firefox – the port number in the request is the port number the WDS is attached to. This page is for testing purposes. It allows the client to know whether a connection to the WDS by an applet is possible.

There are two ways to setup WDS to access this functionality. WDS.ini can be configured with the parameter [WDS] ECHOPAGE=YES in the WDS section. The default for this parameter is NO. At run time one can enable or disable this functionality through the Administration interface with the command “WDS SET ECHOPAGE 1” or “WDS SET ECHOPAGE 0”. Some people may consider this functionality to be a potential security risk.

- **User Authorization**

User authorization can be done by a variety of methods. The WDS has two client interfaces (HTTP client and LDAP Client) that can be used to make external request to authenticate a User. Also the WDS can load static permission files which can contain username - password information.

- **User Request Permissions**

User access to records can be controlled by loading permission data for that user. The permission data can either be loaded from a file or by a request to an external server (HTTP or LDAP Server). The WDS can also load permission data in the form of lists. It can then get User permission information in relation to the loaded permission lists. See Slingshot2 Data Permissions

- **Response Message Priority**

The client API can set the priority of all response messages. The WDS has enabled this functionality before.

See WDS Parameter:- USE\_PRIORITY\_QUEUEING

- **Updates Paused**

The Client API can cause updates for a record to be paused for that connection. The default action is that the updates will be merged and queued at the connection output. The default behavior of the WDS can be changed to queuing updates. See WDS Parameter:- MERGE\_ON\_PAUSE. Care must be taken if turning merge on pause off. If a connections maximum queue size is reached, the connection will be disconnected.

- **Client API Server logging**

APIs can log messages to the server log file. This feature has to be enabled in the WDS.

- **Clients API Field List Request**

APIs can request a complete list of fields of a record.

## 1.2 Client API Module

The C++ Client API Module currently provides the following functionality in normal HTTP or SSL modes.

### Monitors

It allows clients to make monitor requests for particular fields of a record. The Client will then receive all updates for those fields. All responses to monitor requests can have a priority set.

**Snaps**

A client can make a snap request for a field of a record. The client will receive the current value of that field. All responses to snap requests can have a priority set.

**Inserts**

A client can make an insert request to a field of a record. This request will be passed to a RDS connected to the WDS.

**Name change**

Record name change has been implemented. This will allow sessions and symbol search requests to be made.

**Field List Request**

APIs can request A complete list of fields of a record. The response to this request is a string with each field id separated by a field separator. This field separator is set in the WDS.

## 2 WDS FUNCTIONALITY AND ADMINISTRATION

The WDS consists of three basic parts:

- the InVision Requester API module
- the Main module and
- the HTTP module.

It provides an interface between the InVision requester API and the Internet. It allows the fields of a record to be monitored or a snapshot of a field to be made. All requests are made for a field of a record, not on the entire record (field level based requests). The server will only deliver the fields that have been requested. All other fields will be cached in case of future requests. All data is considered as record based. For services that supply page data, the page is converted to a record format with rows represented by fields starting at field one.

The server has no limit on the number clients that can be connected, except for the limit within the purchased license. But it is limited by the number of file descriptors that are available on the machine. In cases where a limit is required, a parameter can be used in the configuration file to limit the number of clients.

### 2.1 InVision Requester API Module

This provides a simple interface to the InVision Requester API.

### 2.2 Main Module

This module is the interface between the HTTP Server module and the InVision Requester API. It consists of a cache and pending queues. The pending queue keeps track of all outstanding requests and the cache contains all monitored records. The cache keeps a watch list of the fields that are being watched by the clients. The main module is responsible for starting the other modules and for administration functions. All logging of events in the server is done via a separate thread (except the InVision Requester log messages, which are written directly by the InVision Requester).

### 2.3 HTTP Server Module

This module interfaces with the Internet. Because it is designed to connect only to the client API, it only utilizes a subset of the HTTP 1.0 standard. The client initiates all communication between the client and the WDS. The client will connect to the WDS using two connections. Data will be sent to the client when the client issues a GET HTTP request to the WDS. The WDS will respond to this GET request with a multipart message response. The multipart response allows the data to be streamed between the WDS and the client. Data will be sent to the WDS by the client issuing a POST HTTP request. The HTTP server module administers these connections, making sure that both the GET and the POST requests are associated with the same client.

## 2.4 License

The WDS requires a licence string to function. This licence will be supplied on a installation basis (because licence information relates to the installation). The license will also determine how many concurrent users are allowed to connect.

## 2.5 Client API

The client API operates using a call back method. A request is made for a field of a record and a class is supplied to be called when an event occurs for that field in the request. A connection to a host consists of an instance of the `HttpConnection` class. Each connection can contain any number of services with any number of fields in records monitored. A `HttpConnection` consists of a cache, a pending queue and a insert queue. All monitored fields are cached and all requests for a field are queued. With only one request made to the server for multiple client requests. The `HttpConnection` class is responsible for establishing the connection and for maintaining it. It can handle redirections and after a redirection it will try to continue to communicate with this redirected server. If this fails it will revert back the original server in the request.

## 2.6 Recovery

### WDS

Once a client has connected and made requests, the server will consider it a valid connection until informed by the HTTP module that the client connection is no longer required. This will usually occur after a timeout period when the client fails to reconnect with a GET request or when the client disconnects. If a failure occurs between the server and ODS or linefeed, the server is responsible for recovery. All clients will remain valid and they will be notified of the failure. So each field is notified of a change of state.

### Client API

After a successful connection has been established and a successful response has being received for a monitor request, the client considers it a valid object until it is either dropped by the server or is unmonitored by the client application. If the WDS fails or a network failure occurs, the client will first try to reconnection to the old server. In this time period the application is not notified of any failure. If the reconnection fails the API will notify the application that all records are stale. The API will continue to try to reconnect to server. After all records are marked stale a successful reconnection to the server will result in all fields being requested. If a connection is made to a server that is not its original server (incorrect server signature or connection id), the client will consider this a new connection and request all fields again.

## 2.7 Record Merging and Output Bandwidth Restriction

The WDS has the ability to restrict the output bandwidth to a client. This bandwidth restriction will cause records that have frequent updates to be merged together at the output before sending. The WDS can be set to a default output bits per second bandwidth. The client also have the ability to turn merging on or off for particular client connections or for particular record requests.

The bandwidth usage of a connection can be calculated using the following formula. It should only be used as a base or minimum usage. This does not take into account the merging of updates.

n - Average number of records to receive a update per second  
 f - Average number of fields in the records to receive a update per second  
 s - Length of service name  
 k - Average length of a record names  
 d - Average length of data in fields

Bandwidth in bytes/sec =  $67 + n * (20 + s + k + f * (5 + d))$

Example: - If we have monitored 4 fields in 100 different records for service reuter. All the records have a name like DEM= or EUR=. Three fields in a average of 9 records are updated every second. All the fields being watched are numbers with an average data length of 5 bytes.

From the information above:-

n = 9  
 f = 3  
 s = 6  
 k = 4  
 d = 5

$67 + 9 * (20 + 6 + 4 + 3 * (3 + 5)) = 553$  bytes/sec.

## 2.8 HTTP Web Server

WDS can deliver HTMLI pages and files like a simple Http Web Server. There are two restrictions on the directory names that can be used: the directory names `SlingShot2000` and `WDS` are reserved for internal use. To activate this simple Http Web Server ability two parameters in the configuration file have to be set:

DOCUMENT\_ROOT = <document root directory>  
 HANDLER\_THREADS = <default number of threads for handling HTTP requests>

## 2.9 WDS Administration

The WDS provides an administration port (if enabled, by default it is disabled) where administration requests can be made. There are eight administration command groups:

STATS	Displays the WDS statistics.
RESET	Resets WDS Statistics
GET	Displays a list of variables that can be altered.
SET	Set a variable to a new value.
USER	Displays information about connected users.
WATCHED	Display the watchers of records.
CONN	Displays information about connections.
MOD	Displays Module information
PERM	Displays information about Permissions
GROUP	Displays information about User groups
LIC	Displays License Information
VER	Displays WDS Version
SHUTDOWN	Shutdown the WDS.

**All administration commands must be preceded by the word “WDS”, e.g. “WDS STATS”. The exception is the SHUTDOWN command which can be used as WDS SHUTDOWN or SHUTDOWN.**

The different admin commands are illustrated in details in the following sub-paragraphs.

#### **WDS Administration STATS**

The following is an example of the output of a statistics request to the Administration port. The first line states the process name version and date of build. Then there are three sections of statistics – HttpServer, Main and Requester.

```

HttpServer                               :
Current Outgoing Msg to Main (in queue)  : 0
Accumulated Number of Outgoing Msg to Main : 0
Maximum Outgoing Msg to Main             : 0
Actual Number of Current Clients          : 0
Maximum Number of Concurrent Clients      : 0
Accumulated Number of Clients Logins      : 0
Accumulated Number of Clients Timed out   : 0
Accumulated Number of Clients Sess. Stopped : 0
Actual Number of Current Connections      : 0
Maximum Number of Concurrent Connections  : 0
Accumulated Number of Connections made    : 0
Accumulated Number of Timed out Connections : 0
Number of Network Msgs Sent               : 0
Actual Network Msgs Sent/Sec              : 0
Maximum Network Msgs Sent/Sec            : 0
Number of Network Bytes Sent              : 0
Number of Network Bytes Recv              : 0
Output Messages Merged                    : 0

```

```

Output Messages Queued      : 0
Total of Publish Msgs      : 0
Publish Msgs Acutally Queued : 0
GET File Requests          : 0

ReqTask                    :
Number of Symbols          : 0
Msg recv                   : 0
Msg Sent                   : 2
Service Status Msgs       : 1
Object Status Msgs        : 0
Object Data Msgs          : 0
Misc. Msgs                 : 1
Session Requests          : 0

Main                        :
Service 01                 : ReutersSim
Service State              : ACTIVE
Current Number in Cache    : 0
Adds to Cache              : 0
Deletes from Cache         : 0
Number Pending             : 0
Number of Requests         : 0
Request Sent External      : 0
Number of Unmonitors       : 0
Number of Drops Received   : 0
Number of Updates          : 0
Number of Recaps           : 0
Number of Snaps            : 0
Number of Insert           : 0

```

<b>HTTPSERVER</b> :	HttpServer statistics section
Current Outgoing Msg to Main (in queue):	The current number of messages queued and waiting to be processed by main. If this is a large value, it indicates that main is not handling incoming messages from clients.
Accumulated Number of Outgoing Msg to Main:	The total number of messages sent to Main.
Maximum Outgoing Msg to Main :	The maximum number of messages in the queue to Main. This is a high water mark of message going to main. It can indicate if main is having problem processing messages from clients.
Actual Number of Current Clients:	The current number of active clients connected.
Maximum Number of Concurrent Clients:	The maximum number of connected active clients. This is a high water mark of connected clients.
Accumulated Number of Clients Logins:	The total number of client connections. This is incremented for every client connection and just gives a indication of the total number of client connections the WDS has handled.
Accumulated Number of Clients Timed out:	The total number of clients which have timed out.

Accumulated Number of Clients Sess. Stopped:	The total number of clients which have terminated their session.
Actual Number of Current Connections:	The current number of TCP connections. Usually indicates the number of active GET connections
Maximum Number of Concurrent Connections:	The maximum number of tcp connections at one time. This is the high water mark of active tcp connections.
Accumulated Number of Connections made:	The total number of connections. This is incremented for every tcp connection.
Accumulated Number of Timed out Connections	The total number of time-out connections
Number of Network Msgs Sent	The total number a messages sent to the network
Actual Network Msgs Sent/Sec	The current number of messages sent per second
Maximum Network Msgs Sent/Sec	The maximum number of messages sent per second
Number of Network Bytes Sent :	The total number of bytes sent over the network.
Number of Network Bytes Recv :	The total number of bytes received from the network.
Output Messages Merged	The total messages merged at the output.
Output Messages Queued	The total number of messages queued at the output.
Total of Publish Msgs	The total of data msgs currently in use.
Publish Msgs Actually Queued	The number of total data msgs queued waiting to be used.
GET File Requests	The number of HTTP file requests received.
ReqTask	Requester stats section
Number of Symbols	The number of record being watched.
Msg recv	The total number of messages received from Main.
Msg Sent	The total number of messages sent to Main.
Service Status Msgs	The total number of Service Status messages sent.
Object Status Msgs	The total number of object status messages sent.
Object Data Msgs	The total number of object data messages.
Misc. Msgs	The total number of misc. messages.
Session Requests	The number of session requests.
<b>Main</b>	Main stats section.
<b>SERVICE 01</b> <b>: REUTER</b>	The service name of the service.
Service State : ACTIVE	The state of the service.
Current Number in Cache :	The current number of records in the cache.

Adds to Cache	:	The total number of adds to the cache.
Deletes from Cache	:	The total number of deletes from the cache.
Number Pending 0	:	The current number of requests pending for this service.
Number of Requests	:	This is the total number of requests received
Request Sent External	:	Number of external request
Number of Unmonitors	:	The total number of unmonitors received
Number of Drops Received	:	The total number of drops sent.
Number of Updates	:	The total number of updates received.
Number of Recaps	:	The total number of recaps received.
Number of Snaps	:	The total number of snap requests received.
Number of Insert	:	The total number of insert requests received.

### WDS Administration GET and SET

Below is an example of the get administration request.

```

DEBUG                : MAIN HTTP
BIT_RATE             : 24000
MERGE                : 1
CLIENT_LOGGING      : 0
AUDIT                : NO
AUDIT_DEFINES        : Not Valid
MAX_BYTES            : 2000000
CONN_TIMEOUT         : 900
CLIENT_TIMEOUT       : 120
WAIT                 : 500
BLOCK                : 150
ECHOPAGE             : 1

```

### WDS Administration USER

The User command is used to obtain information about the user or delete a user entry. The command can be used the following ways.

### WDS USER or WDS USER ALL

This will display all the users currently connected, using the following format.

```

User Name | Connect Time | BandWidth | Queue | Merge | Group | Max:Current | Connect IDs
User1 | 09:57:50 19/02/2004 | 10 | 2000 | Yes | Default Test Group | 2:1 | 1

```

User Name - the user login name.  
Connect Time - the time the user established this connection.  
Bandwidth - the bandwidth of this users connections.  
Queue - the maximum size of users output queue

Merge	- is the output merging updates
Group	- users current group
Max:Current	- users maximum and current number of connections
Connect Id	- a list of connection Ids that this user is associated with.

## WDS USER \_NAME

This will return information about the user "NAME". The format of the response will be the same for WDS USER ALL.

## WDS USER DELETE USER\_NAME

This command will delete the user entry "USER\_NAME". This will not stop the user from reconnecting but will force the WDS to requests user information again.

## WDS Administration CONN

This command returns information about current connections. The command can be used in the following ways.

## WDS CONN or WDS CONN ALL

This will display all connections using the following format.

```
Id | Address | OutBytes | OutputRate | Merge | NumMerged | Buffered
1 | 192.168.152.15 | 31695 | 8 | Yes | 335 | No
```

Id	- the connection Id.
Address	- the ip address of the current physical connection.
OutBytes	- the number of bytes sent on this connection.
OutputRate	- the output bandwidth of this connection.
Merge	- is merging off or on.
NumMerged	- the current number of messages that have been merged.
Buffered	- is working through Buffered Proxy

## WDS CONN ID

This command will return information about the connection "ID". It will use the format stated above.

## WDS Administration WATCHED

This command returns information about the watchers of the records in the cache. It has three sub commands.

**WDS WATCHED CONN ID**

This command group will display watch information in relation to a connection.

**WDS WATCHED USER Name**

This command group will display watch information in relation to a user.

**WDS WATCHED KEY**

This command will display watch information in relation to a record.

**WDS WATCHED CONN or WDS WATCHED CONN ALL**

This will display the record name, the field id and the field value of all current connections.

**WDS WATCHED CONN ID**

This will display the record name, the field id and the field value of the connection "ID".

**WDS WATCHED USER or WDS WATCHED USER ALL**

This will display the record name, the field id and the field value of all connected users.

**WDS WATCHED USER NAME**

This will display the record name, the field id and the field value of the user "NAME".

**WDS WATCHED KEY SERVICE RECORD\_NAME**

This will display the user and connection ids of the watcher of the record identified by "SERVICE" and "RECORD\_NAME".

**WDS Administration PERM**

This command returns stored permission information Users. The command has two options "ALL" or User Name. The "ALL" command option will return all users in the permission data. The User name command option will return the permission data for that User.

## 2.10 WDS Services

This is a service in the WDS that contains all the statistics as records. Currently these stats consist of three records MAIN, HTTP and REQUESTER. These records represent the three sections in the WDS. Listed below there are the field ids for each record.

**Record HTTP**

<b>Field</b>	<b>Description</b>
1 :	Current Outgoing Msg to Main
2 :	Num. of Outgoing Msg to Main
3 :	Maximum Outgoing Msg to Main
4 :	Number of Current Clients
5 :	Number of Max. Clients
6 :	Number of Clients
7 :	Number of Time out Clients
8 :	Number of Current Connections
9 :	Number of Max. Connections
10 :	Number of Connections
11 :	Number of Time out Connections
12 :	Number of Network Bytes Sent
13 :	Number of Network Bytes Recv
14 :	Number of messages Merged
15 :	Number of messages Queued
16 :	Total of Publish Msgs
17 :	Publish Msgs Queued
18 :	GET File Requests
19 :	Number of Network Msgs Sent
20 :	Maximum Network Msgs Sent/Sec

Record REQUESTER

<b>Field</b>	<b>Description</b>
1 :	Number of Symbols
2 :	Msg recv
3 :	Msg Sent
4 :	Service Status Msgs
5 :	Object Status Msgs
6 :	Object Data Msgs
7 :	Misc. Msgs
8 :	Session Requests

**Record MAIN**

All service fields are service number x 100 + field Id.

<b>Field MAIN 0x</b>	<b>Description</b>
1 :	Service State
2 :	Current Number in Cache
3 :	Adds to Cache
4 :	Deletes from Cache
5 :	Number Pending

<b>Field MAIN 00</b>	<b>Description</b>
1 :	Number of Requests
2 :	Request Sent External
3 :	Number of Unmonitors
4 :	Number of Drops Received
5 :	Number of Updates
6 :	Number of Recaps
7 :	Number of Snaps
8 :	Number of Insert

## 3 SLINGSHOT DATA PERMISSIONS

This permission scheme provides:

- Read Permissions
- Write Permissions
- Group Permission
- Delete Permissions ( not used )
- Update Filtering

### 3.1 Permissions Data Format

The following is a list of keywords used in the permission data.

#### **Keywords**

PE\_READ\_BAR – Read bars a list of PE Codes

PE\_READ\_PERM – Read permissions a list of PE Codes

PE\_WRITE\_BAR – Write bars a list of PE Codes

PE\_WRITE\_PERM – Write permissions a list of PE Codes

PE\_DELETE\_BAR – Delete bars a list of PE Codes

PE\_DELETE\_PERM – Delete permissions a list of PE Codes

OBJ\_READ\_BAR – Read bars a list of object names

OBJ\_READ\_PERM – Read permissions a list of object names

OBJ\_WRITE\_BAR – Write bars a list of object names

OBJ\_WRITE\_PERM – Write permissions a list of object names

OBJ\_DELETE\_BAR – Delete bars a list of object names

OBJ\_DELETE\_PERM – Delete permissions a list of object names

REG\_READ\_BAR – Read bars a list of regular expressions

REG\_READ\_PERM – Read permissions a list of regular expressions

REG\_WRITE\_BAR – Write bars a list of regular expressions

REG\_WRITE\_PERM – Write permissions a list of regular expressions

REG\_DELETE\_BAR – Delete bars a list of regular expressions

REG\_DELETE\_PERM – Delete permissions a list of regular expressions

INFO – contains user specific information

## 3.2 Sections

Sections are used to divide the permission data into specific categories.

### Service

(Service:Reuters)

This specifies that this section refers to Service Reuters

### User

{USER:Admin}

This specifies that this section is for User Admin.

### Group

This is used to set group permission information.

Example:

{GROUP:User Group No 1}

[INFO]

.0,0,0,0,1000,1,5000

[PE\_READ\_PERM]

0-551

559-10000

11000-11700

### Lists

There are two possibilities for specifying a User's permissions. The first is to specify directly, each User's permissions in his or her own section. The second method, which is more efficient if Users have common permissions, is to specify a list at the start of the file and then specify that a dealer has his permissions set according to this list. Both methods allow multiple component lists to be defined.

Example:

(Service:Reuters)

{List:Normal1}

[PE\_READ\_PERM]

0-10000, 12000-32000

[OBJ\_READ\_BAR]

DEM=, GBP=

[REG\_READ\_BAR]

0#\*, DEM???

{List:Write1}

```
[PE_WRITE_BAR]
200-300
500-750

{List:Old1}
[PE_BAR]
200-300
500-750
[OBJ_PERM]
DEM=,FRF=,ESP=,IEP=
```

**Normal1:** This allows a User with this list read access to everything with a PE Code between 0 and 10,000 and between 12,000 and 32,000 except for the objects DEM= and GBP=. All Objects beginning with 0# are barred and all 6 character objects beginning with DEM are also barred.

**Write1:** This allows a User with this list write access to everything except items with PE codes between 200 and 300 and between 500 and 750. A read permission list with corresponding read permissions should be provided. More read objects can of course be specified.

**Old1:** This demonstrates that permissions using the old keywords refer to read access. The newer keywords should be used when generating new permissions files.

### Keyword - INFO

The section INFO contains information relevant to the user. It consists of a coma separated list of parameters. These are described below:-

1. password - contains the users password
2. Maximum Objects Watched - not in use as yet
3. Maximum Pages Watched - not in use as yet
4. Maximum Connections Allowed - 0 for any number
5. SuperUser Variable - not in use as yet
6. User Bandwidth - Bandwidth allowed to the user( bits/sec.)
7. Output Merge - 0 - no merge, 1 – merge
8. Maximum Output Queue Size – measured in messages
9. Users Group

### Example:

```
(Service:Reuters)
{USER:Admin}
[INFO]
admin,0,0,0,0,10000,1,2000,UserGroup_1
[PE_READ_BAR]
0

{USER:JimmyN}
[INFO]
password,0,0,0,0,5000
```

[LIST]

Write1, Normal1

In the example above the User 'Admin' is allowed read and write access to everything except those objects with PE\_CODES of 0. This is the easiest way to permission everything. If an object has a PE Code of zero then it is ignored by the system and it is barred. All other objects will be permissioned and available for use by the User.

The User JimmyN has password "password" and has a bandwidth of 5000 bits/sec. His permissions correspond to those mentioned above for the lists Write1 and Normal1.

### 3.3 Permission Priority

In all cases if an item is barred in any of a User's lists, it will be barred. Even if the item is permissioned in another list the barring takes priority.

This is implemented as follows. The method is the same for both object names and PE codes and for read and write permissions.

- 1 All of the User's bar lists are searched and if the item is in any of those lists it is barred.
- 2 All of the User's perm lists are then searched and if the item is not in any of them then it is barred.
- 3 If the User has no perm lists and the item is not in a bar list then the item is permissioned.

### 3.4 Update Filtering

Update filtering is the process of allowing or barring particular updates for particular users. This can be used, as in the Reuters headline broadcast record (N2\_UBMS), to bar users from seeing particular updates (news stories in this case). The process of permissioning users for updates is as follows.

Search for the "CSKPermissions" Field in all updates of an object. This Field ID can be changed with parameter [APIs] PermissionFieldID= in the WDS configuration file.

For each user associated with an object which has been marked as having the CSKPermissions FID all updates are checked for their permissions as follows:

- If the CSKPermissions FID is empty send the data to the user.
- Otherwise only send the update to the user if the user's permissions allow at least one of the PE codes in the CSKPermissions FID.

A user is permitted for an update according to the normal rules as described in the section "Priority" above.

## 4 WDS CONFIGURATION EXAMPLES

In this section we will consider a few configuration scenarios. The WDS allows user connections to the WDS to be authenticated. To do this the client API supplies information about the user to the WDS. The information is in two forms either using normal Http basic authentication( User name and password) or a user Id that is supplied to the API. User authorization can also be done using a user certificate. But in this case authorization of the user is done by the SSL layer.

To enable authorization the parameter AUTH= is set to the method that the WDS will use, either BASIC, AUTHORISATION\_ID or CERTIFICATE.

Authorization of a user is done either by the WDS making an external request to either a HTTP server or a LDAP Server to obtain user information. The WDS can also use password information loaded from permission files.

Example 1 will examine a User using BASIC authorization and the WDS making HTTP request to get authorization for the User.

Example 2 will examine a User using AUTHORISATION\_ID authorization and the WDS making a HTTP request to get authorization for the User.

Example 3 will examine a User using AUTHORISATION\_ID authorization and the WDS making a HTTP LDAP request to get password information.

The WDS can also control user access to data using permission information loaded from a file or supplied from a external request to a Http Server or LDAP Server.

### 4.1 Example 1 - BASIC Authorization with WDS making HTTP Request

In this example the user will use the normal Http Basic authorisation method. The WDS will use the supplied User Name and password to make a Http request to a Http server to gain authorisation for this user. The WDS will store the user name and password for future connection.

[AUTHORISATION]

# The authorisation identification method.

AUTH=BASIC

# This specifies the external authorisation method

METHOD=HTTP

[HTTPPERMCONFIG]

# Basic and Authorisation Id: Http Request options %USER %PASSWD

# More than one http server can be specified using URL=, URL1=, URL2=, etc

# A poll address can also be specified so the WDS can determine if the host

# is available, using POLL=, POLL\_1=, POLL\_2=, etc. The POLL and URL number

# must be the same host. A maximum of 30 hosts can be set.

```
URL_1=http://192.168.152.11/%USER.html
POLL_1=http://192.168.152.11/index.html

# Set some format information in HTTP Authorisation Server Response
# Password is present in http client response.
# IN_DATA - if password is in .db style permission data
PASSWORD=IN_DATA

# Default group for permissioning
DEFAULT_PERM_GROUP='Default Test Group'
```

## 4.2 Example 2 - AUTHORISATION\_ID with WDS making a HTTP Request

In this example a Authorisation Id will be supplied by the Client Api. The WDS parameter AUTH will be set to AUTHORISATION\_ID. The WDS will expect all connection requests to contain this authorisation id. If it is absent the connection request will be rejected. The WDS treats the Id as a user name. A request to the Http Server can then be made using the authorisation id. If the response is successful, the connection is allowed and the Id is stored for future connections.

### [AUTHORISATION]

```
# The authorisation identification method.
AUTH=AUTHORISATION_ID

# This specifies the external authorisation method
METHOD=HTTP
```

### [HTTPPERMCONFIG]

```
# This will set the proxy if one is being used.
#PROXY= proxy:8000

# Basic and Authorisation Id: Http Request options %USER %PASSWD
# More than one http server can be specified using URL=, URL1=, URL2=, etc
# A poll address can also be specified so the WDS can determine if the host
# is available, using POLL=, POLL_1=, POLL_2=, etc. The POLL and URL number
# must be the same host. A maximum of 30 hosts can be set.
URL_1=http://192.168.152.11/%USER.html
POLL_1=http://192.168.152.11/index.html
```

```

# Does the HTTP Authorisation Server Response contain permission data
#
# RESPONSE
# Default: NO
# RESPONSE = PERMISSION_DATA

# Set some format information in HTTP Authorisation Server Response
# Password is present in http client response.
# IN_DATA - if password is in .db style permission data
PASSWORD=IN_DATA

# Default group for permissioning
DEFAULT_PERM_GROUP='Default Test Group'

```

### 4.3 Example 3 - AUTHORISATION\_ID with WDS making LDAP Requests

In this example a Authorisation Id will be supplied by the Client Api. The WDS parameter AUTH will be set to AUTHORISATION\_ID. The WDS will expect all connection requests to contain this authorisation id. If it is absent the connection request will be rejected. The WDS treats the Id as a user name. A request will be made to the Ldap Server to authorise the user and get user permission and Group data. If the Ldap request is successful the connection is allowed.

#### [AUTHORISATION]

```

# The authorisation identification method.
AUTH=AUTHORISATION_ID

# This specifies the external authorisation method
METHOD=LDAP

```

#### [LDAPPERMCONFIG]

```

# Ldap server host name
SERVER=bilby

# User authorization dn. Use %USER in dn to represent user name.
# For more than one dn use:- USRAUTH_DN1=, USRAUTH_DN2=, etc.
# A maximum of thirty can be specified.
USRAUTH_DN_1=cn=%USER,cn=admin,dc=accounts,dc=local
USRAUTH_DN_2=cn=%USER,cn=front_office,dc= accounts,dc=local
USRAUTH_DN_3=cn=%USER,cn=cleaners,dc= accounts,dc=local

# These parameters are used to request user permission data.
USRPERM_SEARCHFILTER_3=, etc.

```

```

USRPERM_SEARCHBASE_1=cn=admin,dc=accounts,dc=local
USRPERM_SEARCHFILTER_1=cn=%USER
USRPERM_SEARCHBASE_2=cn= front_office,dc=accounts,dc=local
USRPERM_SEARCHFILTER_2=cn=%USER
USRPERM_SEARCHBASE_3=cn= cleaners,dc=accounts,dc=local
USRPERM_SEARCHFILTER_3=cn=%USER

# This is attribute of the permission data for requests using USRPERM_SEARCHBASE;
USRPERM_ATTR=streetAddress

# These parameters are used to request group information.
GRPNAME_SEARCHBASE_1='OU=WDS Test Group,DC=accounts,DC=local'
GRPNAME_SEARCHFILTER_1=(&(objectclass=group)(member=cn=%USER,cn=
admin,dc=accounts,dc=local))
GRPNAME_SEARCHBASE_2='OU=WDS Test Group,DC=pottery-row,DC=local'
GRPNAME_SEARCHFILTER_2=(&(objectclass=group)(member=cn=%USER,cn=
front_office,dc=accounts,dc=local))
GRPNAME_SEARCHBASE_3='OU=WDS Test Group,DC=pottery-row,DC=local'
GRPNAME_SEARCHFILTER_3=(&(objectclass=group)(member=cn=%USER,cn=
cleaners,dc=accounts,dc=local))

# This is the attribute for the group information;
GRPNAME_ATTR=cn

# The next group of parameters is used to load group permission data.
# GRPPERM_DN= - user authorization DN.
# GRPPERM_PWD= - user password in encrypted format
# GRPPERM_SEARCHBASE= - base search DN
# GRPPERM_SEARCHFILTER= - search filter
# GRPPERM_ATTR= - attribute for permission data
GRPPERM_DN=cn=user1,cn=Users,dc=accounts,dc=local
GRPPERM_PWD=PPJGRTLULFLRJKTJ
GRPPERM_SEARCHBASE=cn=Users,dc=pottery-row,dc=local
GRPPERM_SEARCHFILTER=cn=user1
GRPPERM_ATTR=streetAddress

# Default group for permissioning
DEFAULT_PERM_GROUP='Default Test Group'

```

#### 4.4 WDS Configuration of Delayed Data for Services

The services of the WDS can be configured to request delayed data when the request for a record is not permissioned. This is done by inserting the following parameter into the service section of the wds.ini.

DELAYEDDATA = delayed\_data\_string

**Example.**

```
[ReutersSim]
MACHINE_NAME = ReutersSim
DELAYEDDATA = /
```

If the user is not permissioned for DEM= in the above example. The WDS will request the record /DEM= automatically for the user. A flag bit is set in the call back class to signify that this is delayed data.

## 5 WDS CONFIGURATION – WDS.INI

The wds.ini file has the following sections.

[LICENSE]	- contain WDS and Module licenses
[WDS]	- general WDS parameters
[AUTHORISATION]	- authorization parameters
[HTTPPERMCONFIG]	- Http Permission Server Requester parameters
[LDAPPERMCONFIG]	- LDAP Permission Server Requester parameters
[SSL]	- SSL parameters
[LogMgr]	- logging configuration, together with [<Logger name>]

The sections below are for InVision Requester configuration. Refer to InVision Requester for setup information.

[APIs]  
[MONITOR]  
[REQUESTER]  
[InVision]  
[<Service Name>]  
[<Server Name>]

### 5.1 Logging configuration

The WDS has three completely different logging mechanisms:

- Main WDS logging using the LogManager (LogMgr)
- InVision Logging (InVision Requester messages only)
- AdminDebug logging (only special Admin debug messages, and only if switched on)

InVision logging and its configuration is explained in the InVision manual and will not be covered here.

AdminDebug logging is controlled by the parameters

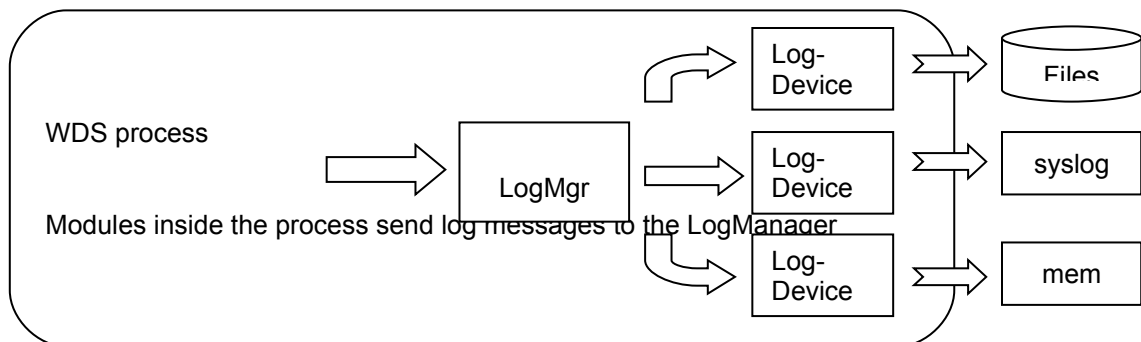
```
[WDS]
ENABLE_ADM_DEBUG
ADM_TIM_CHK_DIFF
ADM_TIM_CHK_HR
ADMIN_DEBUG_FILE
SWAP_FILES
FILE_LENGTH
```

See there for a detailed explanation.

## 5.2 Main WDS Logging

Starting with Slingshot WDS 2.4.0, the configuration of logging has been unified. The Slingshot WDS All I-Trader processes now use the same library for logging and its configuration, so this section applies to all I-Trader processes.

Within the WDS, the different modules use standardized function call for logging. All log messages (except to ones generated by the InVision Requester, and the AdminDebug messages, see previous section) are written in a memory queue. A separate thread reads them from there and puts them to the log manager (LogMgr), which is responsible for synchronizing the messages, filtering and forwarding them to the log devices (LogDev). A log device can also filter messages, and it processes the message, e.g. write it into a file, this depends on the type of the log device (a standard log device is the FileLogDevice). Any number of log devices can be configured:



Actually implemented log devices are:

- FileLogDevice: writes into log files, renames files if maximum size of logfile is reached
- Buffered FileLogDevice: like FileLogDevice, but writing messages is done in a separate thread
- ConsoleLogDevice: write to stdout or stderr
- SysLogDevice: write to syslog (Unix only)
- RemoteSysLog: send syslog messages to a remote syslog host (all platforms)
- SharedMemoryLogDevice: write (round robin) into a shared memory buffer, which can be read with other programs

### Filtering

Log messages can be filtered based on Priority and based on their message ID (message IDs are present, but not used in WDS, most messages just have ID 0). The LogMgr filters on message ID (e.g. all on except 5, 283 and 340-390), and the single log devices filter on message priority (e.g. log only fatal and error messages). The LogMgr knows the filter setting of the log devices and only messages which will be logged by at least one message logger will be constructed at all.

### Configuration

#### 5.2.1.1 List of log devices

The set of log devices is configured in section [LogMgr] of the configuration file of the process. Parameters:

`MaxLogDevices` Maximum number of log devices checked, default 20, range 0 – 100. No need to change this if you don't need more than 20 log devices.

`LogDevType_x` Type of log device, supported types are

`file` FileLogDevice, writing to a log file on disk

`bufferedfile` BufferedFileLogDevice, writing to a log file on disk, but in a separate thread (buffered in memory queue)

`console` ConsoleLogDevice, write to stdout or stderr

`memory` SharedMemoryLogDevice, write into a shared memory buffer

`syslog` SysLogDevice, write to syslog facility on Unix

`remotesyslog` SysLogDevice, send syslog message to a remote host (via UDP)

New log device types may be added. Unknown log file types are silently ignored.

`LogDevName_x` Name of section for configuring this log device. There is no restriction on this name, except that it must not contain white spaces and it must appear later on in this config file.

Replace the `x` by the number of the log device, e.g. `LogDevType_2 = console` to configure the second log device to be console logger.

Default configuration (if nothing is configured) is

```
LogDevType_1 = file
```

```
LogDevName_1 = FileLogger
```

that means that file one file log device is present, which is configured in section `[FileLogger]`.

### 5.2.1.2 LogMgr

The log manager itself is configured in section `[LogMgr]`. Parameters:

`MaxMsgId` range 1000 – 500000, default depends on process (usually about 50000). Maximum message ID which will be used for filtering. Specify only if (by accident) the default does not cover all message IDs in the process.

`InactMsgId` Message IDs which should be ignored. Default: empty (nothing is ignored) Specify a list of numerical ranges, comma separated. Example:

```
InactMsgId=206-210,1000,1250-1270
```

### 5.2.1.3 Log devices (generic configuration)

All log devices have some type-specific and some general parameters. In this section we cover the general parameters each log device understands. These parameters are set in the configuration file section of the log device which is specified with `LogDevName_x` in section `[LogMgr]`.

### 5.2.1.3.1.1 *Enabling/Disabling the logger*

`Enable` Boolean parameter, default `true`.

Is this log device enabled at all (if not, it will log no message at all)

### 5.2.1.3.1.2 *Filtering on message severities*

Each message severity class can be enabled or disabled separately.

Boolean parameters (true or false), default: all enabled (true)

<code>Debug</code>	Debug messages
<code>Operation</code>	Operation messages (currently not used)
<code>Info</code>	Information messages
<code>Warning</code>	Warning messages
<code>Error</code>	Error messages
<code>Fatal</code>	Fatal messages (currently not used)
<code>Plaintext</code>	Plain text messages (currently not used)
<code>Dump</code>	Dump messages (currently not used)
<code>Hexdump</code>	Hexdump messages (currently not used)
<code>Audit</code>	Audit messages
<code>Event</code>	Event messages (currently not used)
<code>Trace</code>	Trace messages (currently not used)

### 5.2.1.3.1.3 *Header fields to log*

Each log message has header fields (e.g. source file name), and each header field may be logged or not.

Boolean parameters (true or false), default: log all header fields (true) except ID, Class and Method.

<code>Header_Timestamp</code>	Timestamp, e.g. 12:58:03
<code>Header_ClockTick</code>	1/1000 seconds of actual time
<code>Header_ID</code>	numerical message ID, e.g. 25332 (currently not used)
<code>Header_Type</code>	severity, e.g. ERROR
<code>Header_ClassName</code>	C++ class which logs the message, e.g. CcompleteLogging. (currently not used)
<code>Header_MethodName</code>	C++ method with logs the message, e.g. writeLogConfiguration. (currently not used)
<code>Header_FileName</code>	C++ source file name, e.g. CCompleteLogging.cpp
<code>Header_Line</code>	line number in source file, e.g. 253

#### 5.2.1.3.1.4 Flush

Should the log device flush after each log message? This may not supported by every log device.

After each Error or Fatal log message flush is called without regarding this parameter.

`Flush` Boolean parameter, default false

#### 5.2.1.4 Buffered log device

All buffered log device understand the parameters in this section. Buffered log devices put the incoming log messages into a memory queue and let a separate thread do the real logging. This avoids blocking the logging thread if the real logging blocks (e.g. when writing to syslog or a remote file).

Actually, the `SysLogDevice`, the `RemoteSysLogDevice` and the `BufferedFileLogDevice` are buffered log devices.

`MaxMsgQueueLength` default 100000, range 0 – 2000000000. 0 = unlimited.

Maximum length of log message queue in memory. If the maximum queue length is reached, further log messages are discarded.

`WriteAllMessagesAtShutdown` Boolean, default false

On shutdown, wait for messages in memory queue to be written (true), or terminate immediately (false, default).

`LogDiscardedMessages` Boolean, default true

If messages are discarded, log this fact (with number of discarded messages).

#### 5.2.1.5 File log device

The file log device writes log messages to a log file. If the log file reaches a specific size (`LogSize`), this file is closed, renamed and a new file is started. The old file is renamed to *<old log file name>.00001*. If there is already a file with this name, this file is renamed to *<old log file name>.00002*, and so on.

`LogFileName` Default: Process name + “.log”, e.g. “DataServer.log”

Name of log file (also Parameters `LogFile` or `FileName` can be used for backward compatibility).

`LogCount` Default 3, range 1 – 500 (alternative parameter name `NumberOfSwapFiles` for backward comp.)

Number of old log files to keep. All older log files are deleted.

`LogSize` Default 500000, range 100, 2000000000 (alternative parameter name `SizeMax`)

Maximum size of a single log file. If this size is reached, a new log file is started, and the old one is renamed to *<old log file name>.00001*.

`NewFileOnStartup` Boolean, default true.

Start a new file on startup of the process (instead appending to an existing file), even if the existing log file has not reached the `LogSize` yet.

### 5.2.1.6 Buffered file log device

Simply a combination of the `FileLogDevice` and the buffered log device – no further parameters added.

### 5.2.1.7 Console log device

The console log device may log to `stderr` (default) or to `stdout`. Parameter:

`UseStdErr` Use `stderr` (instead of `stdout`) for logging. Boolean, default true.

### 5.2.1.8 Shared memory log device

The shared memory log device logs into a shared memory buffer (round robin), overwriting the oldest messages if the memory buffer becomes full. These memory buffer can be read at runtime (to have a kind of buffering between the logging process and the reading/file-writing one), or it can be used after shutdown/crash of the logging process (for a post-mortem analysis). See section below for the `logread` utility which is used to read the information from the shared memory. This log device could be very useful if a process sometimes crashes – with shared memory logging, all (or nearly all) log message can be logged with little overhead (much less than writing into a file), so in case of a crash, the last log messages (e.g. 10 MB) are still accessible (with `logread`) without having the overhead of writing them into a file the whole time.

Each shared memory block is associated to a file in the file system (necessary on MS Windows to keep the shared memory block after the process terminated, and for common behavior also used on Solaris/Linux platforms). These files will always have a name in the format

```
Swissrisk_shm_log_<process name>.<pid>.<count>,
```

where `Swissrisk_shm_log_` is a fixed prefix, the process name is configurable (see below), `<pid>` is the process ID in the system, and `<count>` is a number (starting with 01). The directory for these files is configurable (see below). Example:

```
Swissrisk_shm_log_TestMainProcess.1152.01
```

Parameters of the shared memory log device:

`SharedMemName` Default: Process name (e.g. `DataServer`)

Name of shared memory block to use (the `<process name>` part of it).

`ShmPath` Default: `“./”` (the actual directory)

Where the shared memory file are stored (the ones starting with `Swissrisk_shm_log_`).

`FilePath` Default: `“./”` (the actual directory)

If (by accident) a shared memory block should be created which already exists (e.g. because the process gets the same process ID as some time before), the shared memory block content is dumped to a file before it is overwritten. These files start with `shm_log_` followed by `<process name>.<pid>` and (if this file already exists) an optional `<counter>`. Example: `shm_log_TestMainProcess.1152.00001`

The parameter `FilePath` specify where these file are stored.

`NumberOfSwapShm` Default 2, Range 1 – 20

It will be possible to switch to another shared memory block at run-time using the LOG admin command. In the name of the new shared memory block, the `<count>` parameter will increase (it starts with 01). This parameter controls how many different files / memory blocks may be created before starting again with 01.

`MemSize` Default 10000000 (10 MB), Range 20000 – (system dependend, <= 10000000)

Size of shared memory region used for logging.

### 5.2.1.9 Syslog device

The syslog device uses the local syslog facility available on Unix platforms (e.g. Solaris, Linux) for logging. On Windows platforms, this device is available, but does no logging at all. The syslog facility uses different message priorities than I-Trader, so a configurable mapping is provided. Syslog messages may not exceed 1024 characters, so longer messages are truncated by the system (it is possible to split longer messages into multiple syslog messages, see `MaxMsgLen` below).

Parameters:

`MaxMsgLen` Default 950, range 0 – 60000

Maximum length of a single message sent to syslog – longer messages will be split into multiple messages. If set to 0, messages are not split (but perhaps truncated by the system).

`Ident` Default: process name

A string that is prepended to every message.

`LogPID` Boolean, default false

Log the process ID with each message

`LogCons` Boolean, default false

Write messages to the system console if they cannot be sent to syslogd.

`LogNoWait` Boolean, default false

Do not wait for child processes that have been forked to log messages onto the console.

#### 5.2.1.9.1.1 Log message priority mapping (for syslog)

Each message priority in I-Trader is assigned to a syslog message priority. This can be configured using the `MAP_XXX` parameters, possible values are:

`EMERG` A panic condition. This is normally broadcast to all users.

ALERT	A condition that should be corrected immediately, such as a corrupted system database.
CRIT	Critical conditions, such as hard device errors.
ERROR	Errors
WARNING	Warning messages
NOTICE	Conditions that are not error conditions, but that may require special handling.
INFO	Informational messages
DEBUG	Messages that contain information normally of use only when debugging a program.

#### Default configuration is

MAP_BASE	=	DEBUG
MAP_PLAIN	=	DEBUG
MAP_DUMP	=	DEBUG
MAP_OPER	=	DEBUG
MAP_DEBUG	=	DEBUG
MAP_WARN	=	WARNING
MAP_ERR	=	ERROR
MAP_FATAL	=	ALERT
MAP_HEX	=	DEBUG
MAP_AUDIT	=	INFO
MAP_INFO	=	INFO
MAP_MESSAGE	=	DEBUG
MAP_EVENT	=	INFO
MAP_TRACE	=	DEBUG

### 5.2.1.10 Remote syslog device

The remote syslog device sends its messages to a remote host via UDP as specified in RFC 3164. Note that sending messages via UDP is unreliable – messages may be lost without notice. To avoid losing messages, sending messages may be delayed (`SleepAfterSend` parameter).

Messages with multiple lines in it (containing the newline character) may be handled different on different receiving syslog daemons: On Solaris 8 or 10, the syslog daemon will escape the newline characters (so the output fits into a single line, but all newline characters are transferred to `\n` in this line. Solaris 2.6 will simply truncate the message at the first newline character.

The remote syslog device uses the same mapping facility as the “normal” syslog log device, so it reads in all the `MAP_xxx` parameters described there (with the same defaults).

Parameters used from normal syslog device (see there):

`MaxMsgLen` Default 950, range 0 – 60000

`Ident` Default: process name

LogPID Boolean, default false

Additional parameters of remote syslog device:

LogHost Default: localhost

Host where to send the syslog messages to. May be a host name or an IP address.

LogPort Default: 514 Range: 0 - 65535

Port where to send UDP syslog packages to. Default is the standard syslog port 514.

SleepAfterSend Default: 0 Range: 0 - 2000

Time to sleep after sending a message (in milliseconds)

### 5.2.1.11 Sample configuration

A typical configuration is a file logger writing everything (except debug messages) to disk and a console logger writing only fatal and error messages to stdout. The log file is named "log/MyLogFile.log" and will grow up to 500,000,000 bytes. 5 instances of old log files will be kept (older files will be deleted). On startup, a new log file is created (no matter how big the existing file is). On the Console, only fatal, error and warning messages are logged.

```
[LogMgr]
LogDevType_1 = file
LogDevName_1 = FileLogger
LogDevType_2 = console
LogDevName_2 = ConsoleLogger

[FileLogger]
LogFileName = log/MyLogFile.log
LogCount    = 5
LogSize     = 500000000
NewFileOnStartup = true

[ConsoleLogger]
Debug      = false
Operation  = false
Info       = false
Warning    = true
Error      = true
Fatal      = true
Plaintext  = false
Dump       = false
Hexdump    = false
Audit      = false
Event      = false
Trace      = false
```

### 5.2.1.12 logread – shared memory reading

The shared memory logger writes log messages into shared memory blocks. To get the log messages out again, use the utility `logread`. If called with `-h`, it will list all available options:

logread Version: 3.4.3

Utility to print shared memory log information.

Syntax:

```
logread [-c config filename] [-l] [-p output path] [-s shm path]
        [-t shm mem] [-d] [-r] [-m] [-o filename]
```

Options:

```
-c <file>  configuration file for option, default 'logread.ini'
           (see sample ini file for details)
-l         list current shared memory logs
-p <path>  set output Path, will overwrite ini file option
-s <path>  path where shared memory files are located
-t <name>  tail shared memory name
-d         dump shared memory to log file but do not remove shared
           memory and exit, using default file naming
-r         same as -d but remove shared memory and exit
-m         run continuously dump shm to logs and remove shared memory,
           using default file naming
-o <file>  set output file name from default
           use 'stdout', 'stderr' or a normal filename
```

The logread utility can run in three different modes:

1. list shared memory blocks (-l)
 

This lists the shared memory blocks and their status (attached to a logging process / unattached)
2. 'tail' messages actually written to the shared memory block (-t)
 

If the attached logging process terminates, the logread utility will also terminate.
3. dump a complete shared memory block (-d or -r)
 

This is useful only after the logging process has terminated.

### 5.3 Commented wds.ini file

```
#####
#
# Configuration file for SlingShot 2.4 WDS
#
# Please refer to the Installation and Administration Guide.
#
# Copyright (c) 2002-2008 Swissrisk Financial Systems
#
#####

#####
#
# Licences
#
```

Licences

```

#
#####

[LICENSE]
# License string supply by Swisrisk Financial Systems
# License String must be present.
LICENSE= Licence String

# Lists licensed modules
#
# Example: MODULES= NEWS_MOD DAVS_WEATHER NEWS_TEST
#         NEWS_MOD="Module Licence"
#         DAVS_WEATHER="Module Licence"
#         NEWS_TEST="Module Licence"
# Default: Nil
# MODULES=

#####
#
# WDS WDS
#
#####

[WDS]

# Base document directory that the WDS uses for serving file requests.
#
# Example: ROOT=. \public_html
# Default: (no default)
# ROOT=

# Domain for the JavaScript push page.
#
# Example: JS_DOMAIN=swisrisk.com
# default: (no default)
# JS_DOMAIN=

# A string that is used to separate field in admin requests.
#
# Example: ADMIN_SEP='# #'
# Default: ' | '
# ADMIN_SEP=

# Name of the Application passed on to InVision Requester.
# (no need to change this)
#
# Example: REQUESTER=ReqTask
# Default: ReqTask
# REQUESTER=

# User name that is passed to the InVision Requester.
#
# Example: LOGIN=GUEST
# Default: GUEST
# LOGIN=

# Password of the user passed to the InVision Requester.
#
# Example: PASSWORD=PASSWORD
# Default: PASSWORD
# PASSWORD=

# Disable inserts (to be forwarded to an InVision server).
#
# Example: DISABLE_INSERTS=YES
# Default: NO
# DISABLE_INSERTS=

# Debug modes - This enables debug messages on in the specified sections.
# HTTP - http thread
# REQUESTER - InVision requester
# MAIN - Main thread
# ALL - all debug
# or combinations like HTTP,MAIN
#

```

```
# Example: DEBUG=REQUESTER,MAIN
# Default: NO
# DEBUG=

# Echo log messages to stdout.
#
# Example: LOG_ECHO=TRUE
# Default: FALSE
# LOG_ECHO=

# Cache hash table size.
#
# Example: HASH_SIZE=1024
# Default: 1024
# HASH_SIZE=

# The WDS contains a default service where stats and parameters can be
# monitored. This sets the name of that service.
#
# Example: STATUS_SRV=WDS
# Default: WDS
# STATUS_SRV=

# HTTP module loop sleep time in milliseconds.
#
# Example: SLEEP_TIME=2
# Default: 2
# SLEEP_TIME=

# Interval to wait for another update.
# Range: 60 - 500
#
# Example: BLOCK=150
# Default: 150
# BLOCK=

# Maximum wait time for updates.
# Restriction: Wait time can not be set less than block time.
# Range: 300 - 2000
#
# Example: WAIT=500
# Default: 500
# WAIT=

# Requester retry time if record not available and retry is not set in
# InVision.
#
# Example: WDS_RETRY=15
# Default: 0
# WDS_RETRY=

# Time that a client connect will remain active in the server with
# no physical connection to that client.
#
# Example: CLIENT_TIMEOUT=120
# Default: 120
# CLIENT_TIMEOUT=

# Time a socket connection will stay open. Used only in persistent
# connections mode. ( This is disabled at the moment because of problem with MSIE)
# default 40
# CONN_TIMEOUT=40

# Maximum number of bytes that can be sent in a GET connection
# before the line is terminated. This will force the client to reconnect.
#
# Example: MAX_BYTES=2000000
# Default: 2000000
# MAX_BYTES=

# Maximum time in seconds that each GET connection can stay alive.
# This will force the client to reconnect.
#
# Example: CONNECT_TIME=900
# Default: 900
# CONNECT_TIME=
```

```
# Output bit rate of the network connection.
#
# Example: BIT_RATE=24000
# Default: 24000 bits per second rate
# BIT_RATE=

# The maximum size of the output queue before the client is disconnected.
#
# Example: MAX_QUEUE=2000
# Default: 2000
# MAX_QUEUE=

# This sets the number of initial startup messages. This is used to evaluate
# the connection between the client and the host. This should not be set
# higher than the default value.
# default 8
# INIT_HELLO_CNT=8

# Only applicable for InternetExplorer/Flash(ActionScript 3) clients.
# If these clients are seeing delays during initial connection setup, this
# parameter could be set to YES. This turns on a workaround for
# these clients in WDS which will, if possible, speed up the connection
# setup and also avoid spurious connection data buffering.
# default NO
# SND_BIG_HELLO=YES

# Admin Channel:
# TCP port number of the administration port (0 = no admin channel)
#
# Example: ADMIN_PORT=4740
# Default: 0 (no admin channel)
# ADMIN_PORT=

# Sets the maximum number of admin clients that can connect at one time.
# Absolute maximum is 20.
#
# Example: ADMIN_MAX_CLIENTS=10
# Default: 10
# ADMIN_MAX_CLIENTS=

# Set admin client connection password.
#
# Example: ADMIN_PASSWD=PASSWORD
# Default: Nil
# ADMIN_PASSWD=

# Maximum number of clients that can connect.
# 0 - no limit
# This value is determined by the Licence and cannot be changed
# Example: MAX_CLIENTS=0

# Timeout value the WDS waits before deleting surplus publish
# messages.
#
# Example: DELETE_PUBLISH=3600
# Default: 3600
# DELETE_PUBLISH=

# Request time out value in seconds.
#
# Example: REQUEST_TIMEOUT=150
# Default: 150
# REQUEST_TIMEOUT=

# If you wish WDS to run as a different user or group on a non-Windows platform,
# you must start it as root initially (otherwise it will not be able to
# open a TCP port below 1024) and it will switch after start.
# The use of uid and group ids is allowed eg. #-1
#
# Example: USER=#-1
#          GROUP=#-1
# Defaults: empty = no change of user or group
# USER=
# GROUP=

# Default output action to merge records.
```

```

#
# Example: MERGE=YES
# Default: YES
# MERGE=

# The number of threads for handling file requests.
#
# Example: HANDLER_THREADS=5
# Default: 5
# HANDLER_THREADS=

# The maximum number of threads for handling file requests.
#
# Example: MAX_THREADS=5
# Default: 5
# MAX_THREADS=

# Priority queuing, on or off.
#
# Example: PRIORITY_QUEUE=NO
# Default: NO
# PRIORITY_QUEUE=

# Record merging when it is paused.
#
# Example: MERGE_ON_PAUSE=YES
# Default: YES
# MERGE_ON_PAUSE=

# Allows clients to log messages to WDS.
#
# Example: CLIENT_LOGGING=NO
# Default: NO
# CLIENT_LOGGING=

# Field separator character(s) for field list responses.
#
# Example: FIELD_LIST_SEP=;
# Default: ;
# FIELD_LIST_SEP=

# Log file names:
# See new section [LogMgr] and the sections refered in that section.
# The old parameters LOG_FILE, DEBUG_FILE, AUDIT_FILE, SWAP_FILES, FILE_LENGTH are gone.

# Field separator character in Audit log file.
#
# Example: AUDIT_FIELD_SEP=|
# Default: |
# AUDIT_FIELD_SEP=

# Audit logging defines.
# MonitorSuccess - log all successful monitors
# MonitorFail - log all monitor failures
# SnapSuccess - log all successful snaps
# SnapFail - log all snap failures
# InsertSuccess - log all successful inserts
# InsertFail - log all insert failures
# PermissionFail - log all permission failures
#
# Example: AUDIT_DEFINES=MonitorSuccess MonitorFail
# Default: Nil
# AUDIT_DEFINES=

# Set the wds to attach user name in defined fid for inserts.
#
# Example: INSERT_USER_FID=11104
# Default: (empty = do not attach a user name in inserts)
# INSERT_USER_FID=

# Maximum number of current connections a client can have with
# this WDS. Only valid when user authorisation is used. A value of zero means
# any number of connections.
#
# Example: MAX_CONNECTIONS=0
# Default: 0

```

```
# MAX_CONNECTIONS=

# Force the WDS to use old admin channel formats.
#
# Example: USE_OLD_ADMIN=NO
# Default: NO
# USE_OLD_ADMIN=

# This is the IP address that the WDS will bind with.
# eg. wdshost or 192.168.1.12
# This is overwritten by the Licence and cannot be changed.
# Example: IP_ADDRESS=INADDR_ANY

# The following is a set of Field ID (FID) parameters.
# It is used to set InVision record data to fids.

# Fid Priority.
#
# Default: Nil
# PRIORITY_FID=

# Mask fid.
#
# Default: Nil
# MASK_FID=

# Object Type fid.
#
# Default: Nil
# OBJTYPE_FID=

# Page Type fid.
#
# Default: Nil
# PAGETYPE_FID=

# Foreground colour fid.
#
# Default: Nil
# FORE_COL_FID=

# Background colour fid.
#
# Default: Nil
# BACK_COL_FID=

# Columns fid.
#
# Default: Nil
# COL_FID=

# Row fid.
#
# Default: Nil
# ROW_FID=

# Record PE fid.
#
# Default: Nil
# PE_FID=

# Set a record fieldlist fid.
#
# Default: -32768
# FIELDLIST_FID=

# Specify this to explicitly set the socket option SO_SNDBUF
#
# default 0, socket option not explicitly set
# SOC_SND_BUF_SZ=32768

# Specify this to explicitly set the socket option SO_RCVBUF
#
# default 0, socket option not explicitly set
# SOC_RCV_BUF_SZ=32768
```

```

# Backlog parameter on socket listen call
# (number of connections which are "pre-accepted" by the operating system
# before rejecting additional connections).
# default 50
# CONN_LISTEN_BACKLOG = 100

# Send the HTML echo page or not (WDS/Echo)
#
# default: NO
# ECHOPAGE=YES

# Send the load response or not
#
# default: NO
# ECHOLOAD=YES

#####
#Description of an elementary load response scheme in WDS
#####
#
#A load balancer can make a request GET /WDS/load. This request is
#put in the request queue and processed after all the prior requests
#have been processed. Also, the response to this query could be delayed
#(mS) based on some parameters which are described below.
#
#Specify these to delay (mS) response to the query "GET /WDS/load"
#
#
#COMPONENT 1 (default, no delay due to this component)
#
#
#Delay (mS) for this component is calculated as
#( (number of connections/CONN_LOAD_BLK_SZ) * CONN_LOAD_FACTOR)
#
#CONN_LOAD_BLK_SZ=
#default 1
#
#Delay (mS) per block of connections
#CONN_LOAD_FACTOR=
#default 0 (mS), no delay due to this component
#
#
#COMPONENT 2 (experimental, as of yet not so reliable,
#             default, no delay due to this component)
#
#
#Here an attempt is made to delay (mS) the response based on a sampling
#of network bytes sent over a period of time. One could say a sort of
#historical part.
#
#LOAD_FACTOR=
#default 0 (mS), no delay due to this component
#
#LOAD_SCALE_FACTOR=
#default HTTP_OUTPUT_BIT_RATE/8000 (bytes/mS), maybe too low if many
#clients connected
#
#LOAD_SAMP_TIME=
#default 60000 (mS)
#
#Delay (mS) for this component is calculated as
#( (Sample network bytes sent / LOAD_SAMP_TIME) / LOAD_SCALE_FACTOR) * LOAD_FACTOR
#
#Total delay (mS) is the sum of delays due to components 1 and 2.
#####

#####
# Small enhancement to debug logging. An administrator has the option to set an
# hour in the day when information regarding connections and records
# being watched by each connection can be written to a debug file
# default (WDS_ADMIN_DEBUG.log). Ini file entries are:
#
# ADMIN_DEBUG_FILE =
#
# Enable the functionality, default NO (disabled).
#
# ENABLE_ADM_DEBUG = YES

```

```

#
# The hour during which to write the output. Valid values 0-23. Default 0 (midnight).
#
# ADM_TIM_CHK_HR = 15
#
# How often (in seconds) to check if the specified hour has been reached, default 60
seconds.
# Minimum value that can be set is 60 seconds. Recommended 1800 seconds.
#
# ADM_TIM_CHK_DIFF = 900
#
# The maximum size and number of swap file settings are the same as for other debug
files.
#
#////////////////////////////////////
//
#       Examples:
#
#       ADM_TIM_CHK_DIFF = 900
#       ADM_TIM_CHK_HR = 14
#
#       Let's say the WDS is started at 13:55. Then for the day in question the
#       first log will happen around 14:10, the next at 14:25, again at 14:40
#       and finally a log at 14:55.
#
#       If ADM_TIM_CHK_DIFF = 1800
#
#       Let's say the WDS is started at 13:55. Then for the day in question the
#       first log will happen around 14:25 and finally a log at 14:55.
#////////////////////////////////////
//

#####
# Option to use a new cache removal scheme. In this scheme a separate list of potential
# records for removal from a service cache is maintained. This list is periodically
# checked to see
# if records need to be deleted. In the old scheme the whole service cache in batches of
10 is
# periodically checked for records that can be deleted. Ini file options are:
#
# Enable new cache removal scheme, default 0 (disabled)
#
# NEW_CACHE_REM_SCHEME = 1
#
# Maximum number of records to delete in one attempt, default 10
#
# CACHE_DEL_BLK_SZ = 20
#
# When this parameter is used with the old cache removal scheme, this would be the
number
# of records in a service cache that would be checked for deletion in one attempt.

# Maximum number of application outgoing messages (to InVision) that can be handled in
one go
# on the InVision thread.
# default 10, minimum 3
# REQUEST_OUTQUEUE_HANDLED=10

# Maximum number of requester messages that can be handled in one go on the main thread
# default 10, minimum 3
# MAX_REQUESTER_MSGS=20

# Maximum number of incoming InVision messages that can be handled in one go on the
InVision thread
# default 10, minimum 3
# MAX_INVISION_MSGS=20

# Maximum number of client connections to check for over which data can be sent in one
go
# default 30, minimum 30
# MAX_CONN_DATA_SEND = 30

#####
#####
##### DO NOT CHANGE ANY PARAMETER IN FOLLOWING AREA #####
#####
#####

```

```
#####
#####

# This is the time interval in seconds between ping messages to clients.
#
# Example: PING_TIME=40
# Default: 40
# PING_TIME=

# This sets the interval for the hello message. A hello message is a message
# set to the client that the client will not respond to. This differs from a
# ping message which the client is forced to respond to. A hello message is
# set to the client when there is no activity to the client in the specified
# time. This basically lets the client know that the server is still there.
#
# Example: NOACTIVITY_HELLO=15
# Default: 15
# NOACTIVITY_HELLO=

# Requester idle time in loop.
#
# Example: POLL_TIME=10
# Default: 10
# POLL_TIME=

#####
#
# Authorisation Authorisation
#
#####

[AUTHORISATION]

# The authorisation identification method.
# Can be either BASIC, AUTHORISATION_ID or empty (for no authorisation)
#
# Example: AUTH=BASIC
# Default: NO
# AUTH=

# Is the password in base64 encoding?
#
# Example: PASSWD_BASE64=YES
# Default: NO
# PASSWD_BASE64=

# Time the wds wait before deleting user information when
# the user has no connections.
#
# Example: TIMEOUT=3000
# Default: 3000
# TIMEOUT=

# External authorisation method (specify to turn on).
# Can be either HTTP or LDAP.
#
# Example: METHOD=LDAP
# Default: (empty = no external authorisation)
# METHOD=

# This specifies how permission files are to be loaded.
# Load permissions options
# - FILE          this will load the permissions from file specified in option
#   PERMISSION_FILE in each service configuration section or in a file
#   ServiceName.db in the default directory
# - FileName     this will load permissions from a single file called FileName.
#
# Example: LOAD=FILE or LOAD=.\WDS.db
# Default: Nil
# LOAD=

# Delete user permission data when the user times out?
#
# Example: CLEAR_CACHE=YES
# Default: NO
# CLEAR_CACHE=
```

```

# WDSs Authorization realm (turn on Authorisation).
# default WDS
# REALM=WDS

#####
#
# Http Permission Configuration Http Permission Configuration
#
#####

[HTTPPERMCONFIG]

# Proxy if one is being used.
#
# Example: PROXY=proxyhost:3127
# Default: Nil (no proxy)
# PROXY=

# Maximum number HTTP client handlers to be used.
#
# Example: HANDLERS=5
# Default: 5
# HANDLERS=

# Format of the HTTP client request
# Basic and AuthorisationId:
# Http Request options %USER %PASSWD
# More than one http server can be specified using URL_1=, URL_2=, etc. A poll
# address can also be specified so the WDS can determine if the host is
# available, using POLL_1=, POLL_2=, etc. The POLL and URL number must be the
# same host. A maximum of 30 hosts can be set.
#
# Example:
# URL_1=http://www.dav.com/myfile.html#user=%USER,passwd=%PASSWD
# POLL_1=http://www.dav.com/index.html
# URL_2=http://www.dav.com/myfile.html#user=%USER,passwd=%PASSWD
# POLL_3=http://www.dav.com/index.html
# Default: Nil
# URL_1=
# POLL_2=

# Select Permission Server randomly.
#
# Example: RANDOM=YES
# Default: NO
# RANDOM=

# Does the HTTP Authorisation Server Response contain permission data
#
# RESPONSE
# Default: NO
# RESPONSE = PERMISSION_DATA

# Set some format information in HTTP Authorisation Server Response
# Password is present in http client response.
# YES if password is in response, use PREFIX & POSTFIX
# highlight password if necessary
# IN_DATA - if password is in .db style permission data
#
# Example= PASSWORD=IN_DATA
# Default: NO
# PASSWORD=
# PREFIX=
# POSTFIX=

# Default group for permissioning.
#
# Example: DEFAULT_PERM_GROUP='The Default Group'
# Default: Nil
# DEFAULT_PERM_GROUP=

# When the AUTHORISATION is set to BASIC or AUTHORISATION_ID
# and AUTHORISATION_METHOD is HTTP, AUTH_REQUERY_INT
# parameter can be set so that a user is reauthenticated at the
# specified second intervals. If the reauthorisation of a user
# fails, all connections referring to this user are sent an

```

```

# unauthorised message and closed. The user is removed from
# the cache. If user had some permissioning data, it is
# removed as well. Please note that WDS only
# checks for the availability of the requested resource
# (for example, a HTML page) and does not try to update
# the permissioning data. In other words, no dynamic permissioning
# is done. If the original resource that validated the user
# exists, the user is considered valid.
# default value is 0 (seconds), do not reauthenticate users.
#
# AUTH_REQUERY_INT = 900

#####
#
# Ldap Permission Configuration                Ldap Permission Configuration
#
#####

[LDAPPERMCONFIG]

# Ldap Server host name.
#
# Example: SERVER=ldapsrv
# Default: Nil
# SERVER=

# Ldap Server port number.
#
# Example: PORT=389
# Default: 389
# PORT=

# Ldap handler threads.
#
# Example: HANDLERS=5
# Default: 5
# HANDLERS=

# Permission data line entry separator.
#
# Example: LDAPPERMDATA_LINESEP=#
# Default: #
# LDAPPERMDATA_LINESEP=

# Ldap search scope.
#
# Example: SEARCHSCOPE=sub
# Default: sub
# SEARCHSCOPE=

# This will always force authorisation of client
# Default: NO
# Example:
# FORCE_AUTH=YES

# Default ldap user and password. This is used in ldap requests if
# USRAUTH_DN is not specified. The password has to be in a encrypted format.
#
# Example: USR_DN=davo
#          USR_PWD=SDEIGKDSYYHSNEXCYUUSJSNTWG
# Default: Nil
# USR_DN=
# USR_PWD=

# User authorization dn. Use %USER in dn to represent user name. For more than
# one dn use:- USRAUTH_DN_1=, USRAUTH_DN_2=, etc. A maximum of thirty can be
# specified.
#
# Example: USRAUTH_DN_1=cn=%USER,ou=trading,o=csksoftware,c=de
#          USRAUTH_DN_2=cn=%USER,ou=accounts,o=csksoftware,c=de
# Default: Nil
# USRAUTH_DN_1=

# Parameters to request user permission data. For multiple dns
# use USRPERM_SEARCHBASE_1=, USRPERM_SEARCHBASE_2=, etc and
# USRPERM_SEARCHFILTER_1=, USRPERM_SEARCHFILTER_2=, etc. A maximum of thirty

```

```

# dns can be specified.
#
# Example: USRPERM_SEARCHBASE_1=ou=trading,o=csksoftware,c=de
#         USRPERM_SEARCHFILTER_1=cn=%USER
#         USRPERM_SEARCHBASE_2=ou=accounts,o=csksoftware,c=de
#         USRPERM_SEARCHFILTER_2=cn=%USER
# Default: Nil
# USRPERM_SEARCHBASE_1=
# USRPERM_SEARCHFILTER_1=

# Attribute of the permission data for requests using USRPERM_SEARCHBASE.
#
# Example: USRPERM_ATTR=description
# Default: Nil
# USRPERM_ATTR=

# Parameters to request group information. For multiple DNs use
# GRPNAME_SEARCHBASE_1=, GRPNAME_SEARCHBASE_2=, etc and
# GRPNAME_SEARCHFILTER_1=, GRPNAME_SEARCHFILTER_2=, etc. A maximum of thirty
# DNs can be specified.
#
# Example: GRPNAME_SEARCHBASE_1=ou=groups,ou=trading,o=csksoftware,c=de
#         GRPNAME_SEARCHFILTER_1=(&(objectclass=group)(member=cn=
%USER,ou=trading,o=csksoftware,c=de))
#         GRPNAME_SEARCHBASE_2=ou=groups,ou=accounts,o=csksoftware,c=de
#         GRPNAME_SEARCHFILTER_2=(&(objectclass=group)(member=cn=
%USER,ou=accounts,o=csksoftware,c=de))
# Default: Nil
# GRPNAME_SEARCHBASE_1=
# GRPNAME_SEARCHFILTER_1=

# Attribute for the group information.
#
# Example: GRPNAME_ATTR=uid
# Default: cn
# GRPNAME_ATTR=

# Group of parameters to load group permission data.
# GRPPERM_DN= - user authorization DN
# GRPPERM_PWD= - user password in encrypted format
# GRPPERM_SEARCHBASE= - base search DN
# GRPPERM_SEARCHFILTER= - search filter
# GRPPERM_ATTR= - attribute for permission data
#
# Example: GRPPERM_DN='cn=David A User,ou=admins,o=csksoftware,c=de'
#         GRPPERM_PWD=UENHSFGFKEKTFMXKSKKERMFDKS
#         GRPPERM_SEARCHBASE=ou=admins,o=csksoftware,c=de
#         GRPPERM_SEARCHFILTER='cn=David A User'
#         GRPPERM_ATTR=uid
# Default: Nil
# GRPPERM_DN=
# GRPPERM_PWD=
# GRPPERM_SEARCHBASE=
# GRPPERM_SEARCHFILTER=
# GRPPERM_ATTR=

# Default group for permissioning.
#
# Example: DEFAULT_PERM_GROUP='The Default Group'
# Default: Nil
# DEFAULT_PERM_GROUP=

#####
#
# SSL Configuration SSL Configuration
#
#####

# The following section is for SSL use only. This section is used only when the
# SERVER_TYPE is set to SSL.
[SSL]

# Set the WDS to run in SSL mode.
#
# Example: ACTIVE=YES
# Default: NO

```

```

# ACTIVE=NO

# This directive can be used to control the SSL protocol flavors WDS should use
# when establishing its server environment. Clients then can only connect with
# one of the provided protocols.
#
# The available (case-insensitive) protocols are:
#
# SSLv2
#
# This is the Secure Sockets Layer (SSL) protocol, version 2.0. It is the
# original SSL protocol as designed by Netscape Corporation.
#
# SSLv3
#
# This is the Secure Sockets Layer (SSL) protocol, version 3.0. It is the
# successor to SSLv2 and the currently (as of February 1999) de-facto
# standardized SSL protocol from Netscape Corporation. It's supported by
# almost all popular browsers.
#
# TLSv1
#
# This is the Transport Layer Security (TLS) protocol, version 1.0. It is the
# successor to SSLv3 and currently (as of February 1999) still under
# construction by the Internet Engineering Task Force (IETF). It's still not
# supported by any popular browsers.
#
# all
#
# This is a shortcut for ``+SSLv2 +SSLv3 +TLSv1'' and a convenient way for
# enabling all protocols except one when used in combination with the minus
# sign on a protocol as the example above shows.
#
# Example:
#
# enable SSLv3 and TLSv1, but not SSLv2
# PROTOCOL all -SSLv2
#
# Default: all
# PROTOCOL= all

# This complex directive uses a colon-separated cipher-spec string consisting
# of OpenSSL cipher specifications to configure the Cipher Suite the client is
# permitted to negotiate in the SSL handshake phase. Notice that this directive
# can be used both in per-server and per-directory context. In per-server
# context it applies to the standard SSL handshake when a connection is
# established. In per-directory context it forces a SSL renegotiation with the
# reconfigured Cipher Suite after the HTTP request was read but before the HTTP
# response is sent.
#
# An SSL cipher specification in cipher-spec is composed of 4 major attributes
# plus a few extra minor ones:
#
# Key Exchange Algorithm:
# RSA or Diffie-Hellman variants.
#
# Authentication Algorithm:
# RSA, Diffie-Hellman, DSS or none.
#
# Cipher/Encryption Algorithm:
# DES, Triple-DES, RC4, RC2, IDEA or none.
#
# MAC Digest Algorithm:
# MD5, SHA or SHA1.
#
# For more details refer to mod_ssl documentation.
#
# Example: RC4+RSA:+HIGH:
# Default: ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
# CIPHER_SUITE=

# This directive points to the PEM-encoded Certificate file for the server and
# optionally also to the corresponding RSA or DSA Private Key file for it
# (contained in the same file). If the contained Private Key is encrypted the
# Pass Phrase dialog is forced at startup time. This directive can contain

```

```
# two files (referencing different filenames) when both a RSA and a DSA
# based server certificate is used in parallel.
#
# Example: CERT_FILE=file1.pem,file2.pem
# Default: Nil
# CERT_FILE=..\certs\server.pem

# This directive points to the PEM-encoded Private Key file for the server. If
# the Private Key is not combined with the Certificate in the
# SSLCertificateFile, use this additional directive to point to the file with
# the stand-alone Private Key. When CERT_FILE is used and the file contains
# both the Certificate and the Private Key this directive need not be used. But
# we strongly discourage this practice. Instead we recommend you to separate
# the Certificate and the Private Key. If the contained Private Key is
# encrypted, the Pass Phrase dialog is forced at startup time. This directive
# can contain two files (referencing different filenames) when both a RSA and a
# DSA based private key is used in parallel.
# Example: PRIVATE_KEY=file1.pem,file2.pem
#
# Example: file1.pem
# Default: Nil
# PRIVATE_KEY=

# This directive sets the all-in-one file where you can assemble the
# Certificates of Certification Authorities (CA) whose clients you deal with.
# These are used for Client Authentication. Such a file is simply the
# concatenation of the various PEM-encoded Certificate files, in order of
# preference. This can be used alternatively and/or additionally to CA_DIR.
#
# Example: CA_FILE=..\certs\CA\allcacerts.crt
# Default: Nil
# CA_FILE=

# Used for Http Client Only
# Example: CA_CLIENT_FILE=..\certs\CA\allcacerts.crt
# Default: Nil
# CA_CLIENT_FILE=

# This directive sets the directory where you keep the Certificates of
# Certification Authorities (CAs) whose clients you deal with. These are used
# to verify the client certificate on Client Authentication.
#
# The files in this directory have to be PEM-encoded and are accessed through
# hash filenames. So usually you can't just place the Certificate files there:
# you also have to create symbolic links named hash-value.N. And you should
# always make sure this directory contains the appropriate symbolic links. Use
# the Makefile which comes with mod_ssl to accomplish this task.
#
# Example: CA_DIR=..\certs\CADir
# Default: Nil
# CA_DIR=

# Used for Http Client Only
# Example: CA_CLIENT_DIR=..\certs\CADir
# Default: Nil
# CA_CLIENT_DIR=

# This directive sets the all-in-one file where you can assemble the
# Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose
# clients you deal with. These are used for Client Authentication. Such a file
# is simply the concatenation of the various PEM-encoded CRL files, in order of
# preference. This can be used alternatively and/or additionally to CRL_DIR.
#
# Example: CRL_FILE=..\certs\CA\crl.pem
# Default: Nil
# CRL_FILE=

# This directive sets the directory where you keep the Certificate Revocation
# Lists (CRL) of Certification Authorities (CAs) whose clients you deal with.
# These are used to revoke the client certificate on Client Authentication.
#
# The files in this directory have to be PEM-encoded and are accessed through
# hash filenames. So usually you have not only to place the CRL files there.
```

```

# Additionally you have to create symbolic links named hash-value.rN. And you
# should always make sure this directory contains the appropriate symbolic
# links. Use the Makefile which comes with mod_ssl to accomplish this task.
#
# Example: CRL_DIR=..\certs\CA\CRL
# Default: Nil
# CRL_DIR=

# This directive sets the Certificate verification level for the Client
# Authentication. It applies to the client authentication process used in the
# standard SSL handshake when a connection is established.
#
# The following levels are available for level:
#
# none: no client Certificate is required at all
# optional: the client may present a valid Certificate
# require: the client has to present a valid Certificate
#
# In practice only levels none and require are really interesting, because
# level optional doesn't work with all browsers.
#
# Example: VERIFY=none
# Default: Nil
# VERIFY=

# Maximum number of threads that will be used in the SSL accept processing.
#
# Example: MAX_THREADS=10
# Default: 10
# MAX_THREADS=

# Optional encrypted pass phrase that can be used instead of the Pass
# Dialog at startup.
#
# Example: PASS_PHRASE=SDEIGKDSYYHSNEXCYUUSJSNTWG
# Default: Nil
# PASS_PHRASE=

#####
#
# LogMgr LogMgr
#
#####
#
#####
[LogMgr]
# main configuration section, default name is LogMgr,
# but the default name can overwritten in constructor of CCompleteLogging
#
# Maximum number of log devices (default: 10)
# up to this number, the LogDevName_x and LogDevType_x are tried.
# Example:
# MaxLogDevices = 20
#
# List of log devices
# Each log device has a type (file, memory, console, ...)
# and a name. The type determines the code used for this log device,
# the name specifies the name of the configuration section of
# this log device. This way, it is possible to have multiple
# log devices of the same type.
# Default configuration:
# LogDevType_1 = file
# LogDevName_1 = FileLogger

#####
# Sections of different log Devices:
# The section names are specified above with LogDevName_x,
# so here we use sample names.
#

```

```

# Common parameters in all sections described below
# (read in, but not necessarily used by all loggers)
#
# Enable/Disable complete Logger
# Default: true
# Example:
# Enable = false
#
# Message severities to be logged
# Optional parameters
# Default: true for all
# Values allowed: true or false
# Examples:
# Debug = false
# Operation = true (not used at WDS)
# Info = true
# Warning = true
# Error = true
# Fatal = true (not used at WDS)
#
# Plaintext = true
# Dump = true (not used at WDS)
# Hexdump = true (not used at WDS)
# Audit = true
# Event = true (not used at WDS)
# Trace = true (not used at WDS)
#
# Header fields to be loggen
# Optional parameters
# Default: all enabled
# Values allowed: true or false
# Examples:
# Header_Stamp = true
# Header_ClockTick = true
# Header_Type = true
# Header_FileName = true
# Header_Line = true
#
# Flush
# Optional parameters
# Default: false
# Values allowed: true or false
# Examples:
# Flush = true

# [FileLogger]
# Log to (a set of) log files. Once the file reaches a configurable
# size, it will be closed, renamed, and a new file is started.
#
# LogFileName = <process name>.log
#
# [Console]
# Output to stdout (console)
#
# Use stderr (or stdout)?
# Default: true
# UseStdErr = true

# Backward compatibility:
# To get the previous behaviour of the WDS with 3 different log files,
# use the following configuration:
#
# [LogMgr]
# LogDevType_1 = file
# LogDevName_1 = ErrorFileLogger
# LogDevType_2 = file
# LogDevName_2 = DebugFileLogger
# LogDevType_3 = file
# LogDevName_3 = AuditFileLogger
#
# [ErrorFileLogger]
# FileName = WDS_Error.log
# NumberOfSwapFiles = 10
# SizeMax = 500000

```

```

# Debug      = false
# Info       = false
# Warning    = true
# Error      = true
# Audit      = true
#
# [DebugFileLogger]
# FileName = WDS_Debug.log
# NumberOfSwapFiles = 10
# SizeMax = 500000
# Debug     = true
# Info      = true
# Warning   = false
# Error     = false
# Audit     = false
#
# [AuditFileLogger]
# FileName = WDS_Audit.log
# NumberOfSwapFiles = 10
# SizeMax = 500000
# Debug    = false
# Info     = false
# Warning  = false
# Error    = false
# Audit    = true
#
#####
#
# InVision InVision
#
#####
# For further information please refer to the InVision documentation.

####
# The [APIs] section defines generic stuff of the InVision API,
# such as permission field and admin port
[APIs]
# Admin Port (prompt, password)
# default: no admin port, no password, simple prompt
# enables admin access (via telnet) to the InVision Requester
# Admin_Port      = 6513
# Admin_Prompt    = WDS >
# Admin_Password  = WDSAdmin
#
# Permissioning field ID
# If set, all updates are searched for this field ID.
# If present, this field contains PE codes which have to be checked
# against the user permissioning before forwarding this update.
# Example: PermissionFieldID=CSKPermissions
# Default: CSKPermissions

####
# The [MONITOR] section defines the amount and type of InVision logs
# which are recorded.
# Note that the InVision Requester writes its own set of logfiles,
# and this is completely independent from the logfiles written
# by the rest of the WDS, which are defined in section [LogMgr].

[MONITOR]
# The directory where the log files, if applicable, are written.
#
# Example: LOGPATH=Logs
# Default: . or environment variable LOGNAME
# LOGPATH=
LOGPATH=.

# Name prefix of the log files.
# "xxx.log" will always be added, xxx = number of logfile, e.g. "002.log"
#
# Example: LOGNAME=WDS
# Default: Nil
# LOGNAME=
LOGNAME=WDS

```

```
# The different InVision log sections. Loags are written according to whether
# or not there defined level exceeds the provided level:
# 9 means errors
# 7 means warnings & errors & important information & statistics.
# 5 means all informations
# 0 means all logs
# Everything below 5 just adds more debugging information, useful
# only for Swissrisk service personal.
# For production systems, keep between 7 and 9.
API      = 8
API,INI  = 5
API,STS  = 9
REQ      = 8
REQ,INI  = 5
REQ,STS  = 9
INV      = 8
INV,INI  = 5
INV,STS  = 9
CP       = 8
CP,INI   = 5
CP,STS   = 9

# Whether or not to write the logs to disk.
# Options = "ON" or "OFF"
#
# Example: DISK=On
# Default: Off
DISK=ON

# Whether or not to force the log to disk after every write. Recommended when
# you want to ensure you have the latest logs. Not recommended in high load
# situations or in production.
# Options = "TRUE" or "FALSE"
#
# Example: FLUSH=TRUE
# Default: False
# FLUSH=

# Whether or not to show the logs on screen.
# Options = "TRUE" or "FALSE"
#
# Example: ECHO=True
# Default: False
# ECHO=

# Maximum number of log files to write. The oldest files will be deleted
# when the logs exceed this number.
#
# Example: MAX_FILE_NO=50
# Default: 10
# MAX_FILE_NO=

# Maximum number of lines in a log file before a new one is opened.
#
# Example: MAX_FILE_LENGTH=10000
# Default: 1000
# MAX_FILE_LENGTH=
MAX_FILE_LENGTH=50000

#####
# The [REQUESTER] section defines the services which are available to the WDS
# [REQUESTER]
# A space delimited list of service names. Each of this services require a
# configuration section of this name below.
#
# Example: SERVICES=REUTERS VWD
# Default: Nil
# SERVICES=
SERVICES=REUTERS

# Name should be unique within the InVision system,
# used to identify the WDS in the logs/admin channel of the
# InVision servers where it connects to.
#
# Example: REQUESTER_NAME=WDSserver
```

```

# Default: hostname of the machine
# REQUESTER_NAME=

# Example: MAX_TERMINALS=10000
# Default: 0 (virtually unlimited)
# MAX_TERMINALS=

# Check for errors in the data in transport. Should be left at FALSE
# in production as it reduces performance
#
# Example: VALIDATE_FIDSTREAMS=True
# Default: False
# VALIDATE_FIDSTREAMS=

####
# The [InVision] section defines the servers where the SERVICES can be found.
[InVision]
# Servers list, either server names, IP addresses or references to the LHOST
# sections below.
#
# Example: Servers=LHOST1
# Default: Nil
# Servers = LHOST1
Servers = LHOST1

##          Server configuration section providing the services below

# The LHOST1 Server is the Reuters simulator running on the local machine.
[LHOST1]
# T:          = TCPIP
# 127.0.0.1 = The IP Address of the Server
# 9000       = The port number of the Server
#
# Example: MachineName=T:rtrsimhost:9000 or MachineName=T:127.0.0.1:9000
# Default: Nil
# MachineName = T:rtrsimhost:9000
MachineName = T:rtrsimhost:9000

# The heartbeat time (heartbeat messages are sent both directions if no other traffic)
#
# Example: HB_UPSTREAM=5
# Default: 10
# HB_UPSTREAM=
#HB_UPSTREAM=5

##          Services configuration section

# The name of the Service.
[ReutersSim]
# Name of the InVision service available on on of the servers configured above.
#
# Example: MACHINE_NAME=ReutersSim
# Default: Nil
# MACHINE_NAME=ReutersSim
MACHINE_NAME=[INV]ReutersSim

# File containing access permissions, referenced only if the parameter LOAD in
# the authorization section above is set to FILE and the authorization is
# activated there as well.
#
# Example: PERMISSION_FILE=.\ReutersSim.db
# Default: file name consisting of the current service name with the extension
#          '.db'
# PERMISSION_FILE=ReutersSim.db

# Prefix for requesting delayed data from the current service.
#
# Example: DELAYEDDATA=/
# Default: Nil
# DELAYEDDATA=

[SheetServer]
MACHINE_NAME = [INV]SheetServer
PERMISSION_FILE = SheetServer.db

```

```
CODEPAGECHARMAPPING = FALSE
```

```
[REUTERS]
```

```
MACHINE_NAME = [INV]REUTERS
```

```
DELAYEDDATA = /
```

```
CODEPAGECHARMAPPING = FALSE
```