User Documentation

Server

The i-scream server is the core of the i-scream Distributed Central Monitoring System. This document provides a guide to using starting and configuring the server to suite your needs.

Revision History

28/03/01	Initial creation				
		Committed by:	ajm4	Verified by:	tdb1
				Date:	29/03/01
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	

Introduction

The i-scream server acts as the core of the i-scream system, it connects hosts that feed data to clients, which interpret, display and process data. It is made up of a range of components that carry out different tasks and provide a range of features to the rest of the system. This document will provide information on obtaining, configuring and the range of distribution options that are available.

Obtaining the i-scream Server

The latest build of the i-scream server may be downloaded from the *Builds* section of the i-scream project website: -

http://www.i-scream.org.uk/builds/

The website also contains other information that you may find useful in setting up an i-scream monitoring system.

An Overview of the i-scream Central Monitoring System

The i-scream system monitors status and performance information obtained from machines feeding data into it and then displays this information in a variety of ways.

This data is obtained through the running of small applications on the reporting machines. These applications are known as “Hosts”. The i-scream system provides a range of hosts which are designed to be small and lightweight in their configuration and operation. See the website and appropriate documentation to locate currently available Host applications. These hosts are simply told where to contact the server at which point they are totally autonomous. They are able to obtain configuration from the server, detect changes in their configuration, send data packets (via UDP) containing monitoring information, and send so called “Heartbeat” packets (via TCP) periodically to indicate to the server that they are still alive.

It is then fed into the i-scream server. The server then splits the data two ways. First it places the data in a database system, typically MySQL based, for later extraction and processing by the i-scream report generation tools. It then passes it onto to real-time “Clients” which handle the data as it enters the system. The system itself has an internal real-time client called the “Local Client” which has a series of Monitors running which can analyse the data. One of these Monitors also feeds the data off to a file repository, which is updated as new data comes in for each machine, this data is then read and displayed by the i-scream web services to provide a web interface to the data. The system also allows TCP connections by non-local clients (such as the i-scream supplied Conient), these applications provide a real-time view of the data as it flows through the system.

The final section of the system links the Local Client Monitors to an alerting system. These Monitors can be configured to detect changes in the data past threshold levels. When a threshold is breached an alert is raised. This alert is then escalated as the alert persists through four live levels, NOTICE, WARNING, CAUTION and CRITICAL. The alerting system keeps an eye on the level and when a certain level is reached, certain alerting mechanisms fire through whatever medium they are configured to send.

Key Features of The System

- A centrally stored, dynamically reloaded, system wide configuration system
- A totally extendable monitoring system, nothing except the Host (which generates the data) and the Clients (which view it) know any details about the data being sent, allowing data to be modified without changes to the server architecture.
- Central server and reporting tools all Java based for multi-platform portability
- Distribution of core server components over CORBA to allow appropriate components run independently and to allow new components to be written to conform with the CORBA interfaces.
- Use of CORBA to create a hierarchical set of data entry points to the system allowing the system to handle event storms and remote office locations.
- One location for all system messages, despite being distributed.
- XML data protocol used to make data processing and analysing easily extendable
- A stateless server which can be moved and restarted at will, while Hosts, Clients and reporting tools are unaffected and simply reconnect when the server is available again.
- Simple and open end protocols to allow easy extension and platform porting of Hosts and Clients.
- Self monitoring, as all data queues within the system can be monitored and raise alerts to warn of event storms and impending failures (should any occur).
- A variety of web based information displays based on Java/SQL reporting and PHP on-the-fly page generation to show the latest alerts and data
- Large overhead monitor Helpdesk style displays for latest Alerting information

Installation & Booting

Unpack Server

The first step is to download the server archive from the Builds section of the i-scream website. There are .zip and .tar.gz versions of the archive available. Next extract the archive to a suitable location, ensuring the directory structure is retained. The server does not mind where on the file system it is extracted, as long as the sub directories remain in the same relative location.

Configuration

The server configuration will need adjusting. Review the rest of this document for details on how to do this. One essential piece of configuration that needs creating is the mySQL settings, if you intend to use the database features. If you do not, skip this section.

There should already be a line in the system.conf file that specifies the “mySQL” configuration is in “mySQL.conf”. If not, review the relevant section later in this document for details on what to include. Assuming the configuration file is “mySQL.conf”, add the following to a file with that name in the “etc” directory under the main server directory.

```
mySQL.Host=<mysql_server>
mySQL.Database=<mysql_database>
mySQL.User=<mysql_user>
mySQL.Password=<mysql_password>
```

Replace the sections between < and > with the appropriate details. You will need to setup a table in the database give. Details about this are in the database documentation.

Component Manager

The Component Manager is responsible for starting up the server, based on the information given in the “default.properties” file. This file lists which components the Component Manager should attempt to startup. Details of which components are which can be found later, along with details of the contents of the “default.properties” file.

Running the Component Manager is simple. On some systems you can run the jar file directly, on others you can run the included run script. If both of these fail, you can start it directly from the command line.

```
java -jar iscream-server.jar -f <filter_name>
```

This starts the server, giving the Filter (if one is running) the name given. The server will then start up, and the following output will be given if you are using a FileLogger;

```
----------------------------------------
--- i-scream Server Component Manager ---
---   (c) 2001 The i-scream Project   ---
---    (http://www.i-scream.org.uk)   ---
----------------------------------------
---          Starting System          ---
ComponentManager{static(v1.37)}: initialising - using ./etc/default.properties
ComponentManager{static(v1.37)}: coming up
ComponentManager{static(v1.37)}: using component start timeout of 5 seconds
ComponentManager{static(v1.37)}: dependency checking component - {dbinterface.DBInterface(v1.13)}
ComponentManager{static(v1.37)}: starting component - {dbinterface.DBInterface(v1.13)}
ComponentManager{static(v1.37)}: dependency checking component -
{clientinterface.ClientInterfaceMain(v1.24)}
ComponentManager{static(v1.37)}: starting component - {clientinterface.ClientInterfaceMain(v1.24)}
ComponentManager{static(v1.37)}: dependency checking component - {rootfilter.RootFilter(v1.37)}
ComponentManager{static(v1.37)}: starting component - {rootfilter.RootFilter(v1.37)}
ComponentManager{static(v1.37)}: dependency checking component -
computingFilter{filter.FilterMain(v1.30)}
ComponentManager{static(v1.37)}: starting component - computingFilter{filter.FilterMain(v1.30)}
ComponentManager{static(v1.37)}: dependency checking component -
FilterManager{filtermanager.FilterManager(v1.22)}
```

```
ComponentManager{static(v1.37)}: starting component -
FilterManager{filtermanager.FilterManager(v1.22)}
ComponentManager{static(v1.37)}: running
```

When the Component Manager reports “running” it has successfully started all it’s components. No further output will be given, unless something goes wrong.

Component description

Core

The core provides the central services to the i-scream server components. It contains the central Logger CORBA interface which allows all the components to send logging messages to the single logging destination. It also contains the ConfigurationManager that is the centre of the dynamic configuration system. Components can use the core to obtain any configuration they require, and can also ask if their configuration has changed since.

FilterManager

The FilterManager is the Hosts first contact with the i-scream server. A hosts only local configuration is the details of where to find its local FilterManager. Once contacted, it can then obtain its required configuration from the central store by using the FilterManager as a proxy for the requests. The FilterManager will also determine which Filter the connecting Host should send its data to and determine if that Filter is available to receive data.

Filter

The Filter is the entry point to the i-scream server for all data flowing from a Host. A Host sends two main types of data. The first is a UDP packet, which contains system information from the machine on which the sending host is running. This is normally sent at regular interval (often about every ten seconds). The second is a TCP communication (often called a heatbeat), during which a Host can check if its configuration has changes (and re-initialise if needed) and also indicate to the server that it is still alive. Once a host has heartbeated, the Filter then runs a series of Service Checks against the calling hosts essential network services to determine if they are running or not. This data is then sent on through the system.

RootFilter

The RootFilter is where all the data from Filters eventually makes its way back to. This data is then split by the RootFilter to the active interfaces, the DBInterface, the Client Interface or both.

ClientInterface

This interface manages distribution of data to connected clients. These are either locally connected CORBA clients (such as the Local Client), or remote TCP clients (such as Conient).

DBInterface

This interface takes only pure data packets (i.e. no heartbeat or any other type of packet) and places the data in an SQL database. This data can later be queried to generate historical reports and post analysis of the data the Hosts are sending into the system.

Local Client

This is effectively the main “monitoring” aspect of the whole system. It takes inbound packets and distributes them to a number of Monitors. The Monitors then examine the data and ensure that it is all at appropriate levels. It also maintains alert levels if data is deemed to be above valid thresholds. These alerts are then escalated over time if they persist. This component also has a section to dump data out to a web feed, which can then be read by dynamic PHP webpages in the i-scream web services to generate webpages of the latest

data. When alerts are raised it passes the alerts onto a set of Alerters. The alerters are configured to send alerts according to Host and alert level. They then send on the alert information through a variety of channels. The final stage is to dump the alerting information to a web feed, which can also be read by dynamic PHP webpages in the i-scream web services to generate webpages of the latest alert information.

Recommended system distribution options

The following details a selection of possible components that could be distributed and why. For more information about distributing components, see the installation and booting notes earlier and the default.properties configuration options.

Core

This could easily be run separately, as it has no dependencies with any other component in the system, but yet almost all components are dependant on the Core for the essential services. Thus this could be run while the rest of the system can be modified, shutdown, restarted or any other change in system distribution.

DBInterface

This is a primary candidate for distribution, indeed it is recommended to distribute this component to the same machine that the mySQL server is running on. This should prove to have better performance when it comes to interacting with the database system.

Filter (filter hierarchies)

A Filter has support to read data sent from a Filter and then send it onto another. This allows Filters to be chained together. A Filter can also receive data from multiple Filters thus enabling a tree hierarchy to build up. This hierarchy can be designed around the network architecture to ensure that data is fed in most efficiently. It also allows Hosts to send UDP data packets into the system at remote locations without the increased risk of packet loss by internet travel. A final point is that it could be used to allow Hosts to report from machines beyond a firewall, a filter could be configured to use an “Appligator” (for more information please refer to the JacORB documentation) to establish a connection from outside over CORBA. The Hosts could then connect to the outside Filter to feed data in.

FilterManager

Of course, if the last scenario was used for Filters, then a FilterManager would need to be placed outside the firewall (or even useful at remote office) to also allow hosts to be configured properly. As FilterManagers have no state and interact little with other components (aside from obtaining configuration and check a Filter is alive), multiple FilterManagers can be run in remote locations to provide services to remote Hosts.

LocalClient

This component is the most flexible of all the components in the i-scream system. No components have a dependency on it. And the Client Interface supports it deregistering from the system. Thus this component is recommended to be almost always run in a separate JVM so that it can be stopped and restarted at will should the need arise, however if it is not envisaged that this will need happen, it would probably best reside in the same JVM as the Client Interface that supplies its data.

RootFilter & ClientInterface

These two components should remain close to each other due to their heavy exchange of data. Often these would remain inside the Core JVM or possibly with the DBInterface (or, of course, with both). Though because they are distinct components, they can be run independently and anywhere.

Plugin Descriptions

The following is a list of the various plugins available throughout the system. For information configuring these, please refer to the Configuration section.

Filter Plugins

Filter plugins are run against any new data entering the system from an external source. They can be identified by the "__Plugin.class" extension.

EnforceEssentialData

Ensures that the packet data defined as essential in the XML data specifications for the system are in the packet. If this check fails, the packet is dropped.

TypeChecker

Ensures that the packet type is one of the allowed packet types as specified in the XML data specifications.

Service Checks (plugins)

Service checks are performed when a Host performs a heartbeat with the system to indicate it is still alive, the system then performs a series of Service Checks against the host to ensure essential network services are active. The available Service Checks are as follows, and can be identified by the "__ServiceCheck.class" extension.

FTP

Simply connects and ensures a header is received from port 21.

HTTP

Simply connects on port 80 and performs an HTTP 1.1 HEAD request to ensure that the server is still responding.

SSH

Simply connects and ensures a header is received from port 22.

POP3

Simply connects and ensures a header is received from port 110.

IMAP

Simply connects and ensures a header is received from port 143.

Telnet

Simply connects and ensures a header is received from port 23.

SMTP

Simply connects and ensures a header is received from port 25.

Monitor Plugins

Monitor plugins are used to check specific data contained within data that is sent through the system. They can be identified by the "__Monitor.class" extension.

CPU

Can be configured to monitor breaches of thresholds for CPU in use percentages. It also has sub-monitor support to break it into User CPU, Kernel CPU, I/O Wait CPU and Swap CPU for those hosts that send that data.

Memory

Can be configured to monitor breaches of thresholds in memory usage. It can be configured for either monitoring percentage in use, or the amount in use in kilobytes.

Disk

Can be configured to monitor breaches of thresholds in memory usage. It can be configured for either monitoring percentage in use, or the amount in use in kilobytes.

Queue

Can be configured to monitor the levels of the internal queue system that passes data between server components. This allows the system to effectively monitor itself for even storms and possible data flow problems. This too can be either specified as a percentage or an absolute value as a threshold level.

Heartbeat

Can be configured to check periodically to see if a Host has ceased sending heartbeat packets, which is thus an indication that it has died.

Load

Can be configured to monitor load data as sent from Unix flavour machines.

Process

Can be configured to monitor load data as sent from Unix flavour machines. It also has sub-monitor support to break it into total number of processes, CPU (or running) processes, sleeping processes, stopped processes and zombie processes for those hosts that send that data.

Services

Can be configured to simply monitor if the service information being sent reports a failure.

Swap

Can be configured to monitor the swap space usage on a Host.

UserCount

Can be configured to monitor the number of users logged onto a Host

WebFeeder

Can be configured to run to provide a feed to the system WebFeeder. This feeder supplies the i-scream web services with all information flowing into the Local Client.

Alerter Plugins

Alerter plugins are written to provide a method of delivery for alerts that have been raised by Monitors. There are currently only a couple of simple alerters available, but for regular use they cover most eventualities.

Email

As the name indicates, this Alerter can be configured to send email alerting information. Email destinations can of course be anywhere, an individual, a mailing list, a page or even a mobile phone.

IRC

Allows alerts to be sent via an IRCbot on an IRC channel. The bot can be interacted with in a variety of ways to ask it information about alerts, as well as it broadcasting alerts to the channel. Simply ask the IRCbot for “help” once it has been started in the channel, for a list of commands.

WebFeeder

Can be configured to run to provide a feed to the system WebFeeder. This feeder supplies the i-scream web services with all alerts that are being raised by the system.

Configuration Options

default.properties

As previously mentioned this file contains options that are either local to the JVM that is running the components, or options that must be readable before the Core component comes online. Typically it is only the Core that will have to read its options from the file, although some system properties are set e.g. the CORBA system in use and the SQL driver details.

All options in this file are REQUIRED options and all should be looked at and checked before the system has started. The format for this file is specified in the Java JDK API documentation as the system makes use of the java.util.Properties object to read configuration objects. Documentation for this can be found here:

http://java.sun.com/j2se/1.3/docs/api/java/util/Properties.html#load(java.io.InputStream)

However a brief outline of the formatting options follows.

Each line should be terminated by a line terminator (\n or \r or \r\n). A line that contains only whitespace or whose first non-whitespace character is an ASCII # or ! is ignored (thus, # or ! indicate comment lines). Every line other than a blank line or a comment line describes one property to be added to the table (except that if a line ends with \, then the following line, if it exists, is treated as a continuation line, as described below). The key consists of all the characters in the line starting with the first non-whitespace character and up to, but not including, the first ASCII =, :, or whitespace character. All of the key termination characters may be included in the key by preceding them with a \. Any whitespace after the key is skipped; if the first non-whitespace character after the key is = or :, then it is ignored and any whitespace characters after it are also skipped. All remaining characters on the line become part of the associated element string. Within the element string, the ASCII escape sequences \t, \n, \r,\\, \", \', \ (a backslash and a space), and \uxxxx are recognised and converted to single characters. Moreover, if the last character on the line is \, then the next line is treated as a continuation of the current line; the \ and line terminator are simply discarded, and any leading whitespace characters on the continuation line are also discarded and are not part of the element string.

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

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

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

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

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

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

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

cheeses

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

System Drivers

```
org.omg.CORBA.ORBClass= jacorb.orb.ORB
org.omg.CORBA.ORBSingletonClass= jacorb.orb.ORBSingleton
jdbc.drivers= org.gjt.mm.mysql.Driver
```

These specify the CORBA orb in use, they are specific to the JacORB ORB and would possibly change should you decide to use a different ORB. If however you are unfamiliar with CORBA concepts it is recommended that you make use of the supplied CORBA Services utilities which will establish a CORBA environment for you. For more information about CORBA services, please refer to the CORBA services user guide.

Logging System

```
uk.org.iscream.Verbosity= 5
```

This specifies the logging verbosity to be used by the system, i.e., how much information will be printed to the log file, where the number is one of :

FATAL	- (0) Fatal or Critical Errors
ERROR	- (1) All Errors
WARNING	- (2) Warnings
SYSMSG	- (3) System Component messages
SYSINIT	- (4) System Component initialisation
DEBUG	- (5) All debugging messages

It defaults to level 3, which provides a reasonable amount of output to ensure the system is working correctly.

```
uk.org.iscream.LoggerPackage=uk.org.iscream.core.loggers
uk.org.iscream.LoggerClass=FileLogger
```

The above allow you to specify the package that will be used to provide an implementation of the Logger (see maintenance documentation for further details). The default package as shown above is where the bundled loggers are located. There are a range available :

ScreenLogger	- Logs everything to the system console [Params = none]
FileLogger	- Logs everything to a file [FileLogger.filename = logfile name]
MultiLogger	- Logs to both the screen and a file [uses parameters of the loggers it uses]
SimpleGUILogger	- Logs to a nice GUI ! [SimpleGUILogger.maxMessages = maximum number of lines to hold in the window]
SimpleSwingLogger	- Logs to a nicer GUI! [SimpleSwingLogger.maxMessages = maximum number of lines to hold in the window]

The default logger that will be used is the ScreenLogger, however it is recommended that you use the FileLogger for more persistent logging. This will also allow you to make use of the log.cgi web script which can format and process the system log files. It also has “grep” facilities to allow you to search for specific options.

The “Param” options detailed above should be entered for the logger you have chosen. For example, for the FileLogger class the following line should be present to indicate where the log should be stored.

```
uk.org.iscream.LoggerClass.FileLogger.filename=/home/iscream/main.log
```

Configuration Locations

```
uk.org.iscream.ConfigurationLocation=./etc
uk.org.iscream.SystemConfigurationFile=system.conf
```

The configuration locations define where the central configuration system in the Core component should look for its configuration files and which file it should use as its root for configuration options. It defaults to the current working directory, but it is recommended that you use the above settings to keep it all in the same place. However this option does allow you to switch to say a test configuration if needed.

Component Startup

```
uk.org.iscream.ComponentTimeout=5
uk.org.iscream.ComponentList=Core;DBInterface;ClientInterface;RootFil
ter;Filter;FilterManager;Client
```

The above settings provide details to the ComponentManager for the current JVM. It tells it which components to start and how long to wait (in seconds) between re-attempting components that failed due to a missing dependency. The component list is key to distributing the i-scream server across multiple JVM’s. To start a component on another machine you could simply copy a standard zip of the i-scream server to the other machine configure its local default.properties file to start a specific component and it will then start that component. Because the ComponentManager handles all dependencies between components, there is no fixed start up order for any components as they will just be restarted if a dependency for them failed. However the above list is a “recommended” start order as it does reduce dependency failures and thus the system will boot quicker.

system.conf – or the main configuration file

This file is loaded by the ConfigurationManager which resides in the Core component. This is the root of ALL configurations for the system (excluding local JVM options) and defines how the i-scream server operates. In some respects this is the user interface to the server. The format of this and all other configuration files that can be “included” are identical to the “default.properties”.

Sub-Configurations & Configuration Grouping

One of the key features of the configuration system is that it allows you to build a hierarchy of options for different hosts (and also filters, but this is used to a lesser extent) which always end in the system.conf file. The root of this hierarchy is defined in the system.conf. It starts by using the “config” property. This defines the location that the configuration for the requested component resides in, if an entry for a component does not exists it simply gets returned the system.conf as its configuration file. Thus the system.conf is ALWAYS the root of all the configurations. An example entry is:

```
config.Host.raptor.ukc.ac.uk=raptor.conf
```

This defines that the configuration for the component “Host.raptor.ukc.ac.uk” is in the file “raptor.conf”, thus they hierarchy would be “system.conf <- raptor.conf <-“, any properties in raptor.conf would override those in system.conf. The component names available are:

These define configurations for a specific host, note that as well as Host.<option> this also specifies the configuration for all Monitor.<option> and Alerter.<option>. This is because the individual monitors are linked closely to an individual hosts configuration.

Host.<hostname>	e.g. Host.raptor.ukc.ac.uk
Host.<host IP address>	e.g. Host.129.12.4.232

This allows you to define Filter configuration to be located in a separate file, for instance to configure a filter hierarchy.

Filter.<filtername>

All other configuration is not independently looked up and is expected to remain in the system.conf for centrality.

The second key feature of this part of the system is that of grouping by name. This allows you to use a wildcard anywhere in the configuration name to group them. This group can then be assigned a specific configuration file.

```
group.computing=Host.129.12.4.*;

group.rocks=Host.agate.ukc.ac.uk;Host.arkose.ukc.ac.uk;Host.bas
alt.ukc.ac.uk;Host.chalk.ukc.ac.uk;Host.chert.ukc.ac.uk;Host.gr
anite.ukc.ac.uk;Host.jade.ukc.ac.uk;Host.jasper.ukc.ac.uk;Host.
magnetite.ukc.ac.uk;Host.obsidian.ukc.ac.uk;Host.pumice.ukc.ac.
uk;Host.pyrite.ukc.ac.uk;Host.slate.ukc.ac.uk;Host.topaz.ukc.ac
.uk;

group.compsoc=Host.compsoc1.ukc.ac.uk

group.students=Host.stue*.ukc.ac.uk
```

In the above examples, the “computing” will match any machine with an IP address that starts 129.12.4. The “rocks” group matches a list of machines. The “compsoc” group matches a single machine. And finally the “students” group matches any machine that starts “stue” and ends “ukc.ac.uk”. Of course multiple wildcards can be used in a single group identifier, as well as multiple group identifiers defining different host matches within a grouping.

Thus configuration would be assigned like this:

```
config.computing=computing.conf
config.rocks=rocks.conf
config.compsoc1=rocks.conf
config.students=students.conf
```

Group searching and allocation ONLY occurs in the system.conf, the method of sharing configuration in sub-configuration files is achieved through use of the “include” option. This include option allows you to include further files (in a semi-colon separated format) which may have options you want to include, but override those in the system.conf. It is important to note that the options in the current configuration file override anything that may be found in included configurations. Thus the head of a configuration hierarchy starts in the file specified in “config” option, it then goes to any files in the first file of the include line (and all its includes) then the seconds and so on, finally the system.conf is checked for configuration. For example:

system.conf contains –
config.Host.raptor.ukc.ac.uk=computing.conf

computing.conf contains –
include=general.conf;server.conf

general.conf contains –
include=web.conf

server.conf contains only configuration options. Thus the configuration heirarchy returned would be :

order of checking →
computing.conf ; general.conf ; web.conf ; server.conf ; system.conf

Note carefully the ordering that takes place here.

A final point to note about this system is that a host can be specified as being a member of multiple groups, as well as being specified as having an individual configuration. Thus grouping can be used to specify a hierarchy also. The ordering is as follows :

order of checking→
Host.<hostname> ; Host.<ip> ; Host.<hostname wild card match> ; Host.<ip match>

The wildcard matches for wildcarded hostnames are returned in alphabetical order.

The above grouping systems are difficult to grasp at first, but offer a powerful and flexible configuration system. Indeed it is entirely possible that you could allow groups of users to configure monitoring options for their own hosts while securing the main configuration file to system administrators only.

Database Connection Configuration

```
config.mySQL=mySQL.conf
```

THIS LINE IS IMPORTANT, if you intend to use the database system for generating the historical reports. It must specify the name of the file which contains the configuration options for the database connection. This is described in the installation section above.

Miscellaneous system-wide options

```
ConfigurationProxy.updateTime=60
```

The entire system uses a component called the ConfigurationProxy to obtain configuration options from the central manager. This proxy maintains cached information about the configuration but also is configured to regularly update its cache. This property allows you to specify how regularly (in seconds) the proxy checks the configuration system for configuration changes. Put simply this is the time it will take for the entire configuration to propagate to system components once the configuration has been changed.

```
Queue.MonitorInterval=15
```

This is the time in seconds that the internal queue monitoring system sends data to allow you to monitor the internal queue states. Queue monitoring is turned off if you comment this line.

```
Queue.SizeLimit=1000
Queue.RemoveAlgorithm=FIRST
```

Queues can and should be limited in the maximum size that they can reach. Once this limit is reached, it uses specified algorithm to remove old data items from the queue, thus always ensuring that the latest data can be added. The algorithms available are :

FIRST	- removes the oldest item of data at the head of the queue
LAST	- removes the newest item from the tail of the queue
DROP	- leaves the queue in its present (full) state and drops the new item
RANDOM	- removes a random entry from the queue

FilterManager Options

```
FilterManager.listenPort=4567
```

This sets the port that the FilterManager listens on for connections by Hosts wanting to be configured. The above value is recommended not to be changed unless needed (as this is the standard named port).

Filter Options

```
Filter.UDPListenPort=4589
Filter.TCPListenPort=4589
```

These are the ports that the Filter should listen on for data from Hosts. These values are sent to the Hosts by the FilterManager, however the above value is recommended not to be changed unless needed (as these are the standard named ports).

```
Filter.PluginsPackage=uk.org.iscream.filter.plugins
Filter.Plugins=TypeChecker;EnforceEssentialData
```

These detail the package that should be searched for FilterPlugins and the plugins to load. The package above contains the default plugins bundled with they system, these plugins were detailed earlier. However, these two plugins should ALWAYS be used. They are crucial to the operation of the system as they ensure that only correct packets enter the system, the Filter is the ONLY place where this occurs.

```
Filter.parentFilter=root
```

The is the name of the Filter that this Filter should pass its data onto. It is this mechanism that allows filter hierarchies to be built up. Combine this with the line “config.Filter.<filtername>=afilter.conf” to achieve this. IMPORTANT NOTE: At least ONE or more filters at the base of the Filter hierarchies MUST point to the RootFilter in order that data reaches the main system. ANOTHER IMPORTANT NOTE: You should take care to ensure that you do not create a cyclic filter hierarchy, this would not be very sensible.

RootFilter Options

```
RootFilter.name=root
```

This is the name given to the root filter, the above example is sensible. This should appear as the Filter.parentFilter for all filters at the lowest level of the hierarchy that are feeding the RootFilter.

```
RootFilter.realtimeInterfaceName=realtimeclients
RootFilter.dbInterfaceName=database
```

These options specify the names of the two types of client interface that are available, the dbInterface (for storing persistent Host data) and the realtimeInterface for clients that deal with the i-scream data in real time. The names assigned here should be the ones assigned to the respective interface as this is the name they use to register themselves over CORBA.

ClientInterface Options

```
ClientInterface.listenPort=4510
ClientInterface.name=realtimeclients
```

These specify the default port that TCP based clients should connect to, the example above is the standard port and should remain unchanged unless needed. The name is used to identify the interface to the RootFilter.

DBInterface Options

```
DBInterface.name=database
```

The database connectivity configuration should be kept in a separate file (as detailed earlier), however this line specifies the name of the interface to identify it over CORBA and thus to the RootFilter.

Host Options

These options specify the options a Host should use when sending data. The Host protocol is configured to check for changes in the configuration on every TCP heartbeat, thus changes that affect a Hosts configuration may only be reflected after this period has expired. The UDP time indicates how often UDP data packets should be sent to the system, in effect this could be seen as being a sampling time.

```
Host.UDPUpdateTime=10
Host.TCPUpdateTime=60

Host.filter=computingFilter;defaultFilter
```

The above attribute is a semi-colon separated list of names of Filters that this host can try and connect to. When a host connects to the FilterManager, it will check to see if the configured Filters for a host are available according to the order of this list. It will return to the host the first Filter that it finds, if no configured filters are found it returns an error to the calling Host. It is then up to the Host to decided what to do next, but typically they will wait for a timeout period then try again.

```
Host.serviceChecksPackage=uk.org.iscream.filter.plugins
Host.serviceChecks=FTP;SSH;SMTP;Telnet;
```

These detail the package the service checks reside in, and the service checks that should be run against the host being configured. Typically the system configuration will not contain any service checks by default so that by default a machine is not checked for any services, then in sub-configuration you would specify specific services for a Host or group of Hosts.

Monitor Options

```
Monitor.PluginsPackage=uk.org.iscream.client.monitors
Monitor.Plugins=CPU;Load;Process;Disk;Memory;Swap;Services;\
Heartbeat;Queue;UserCount;WebFeeder;
```

These options specify which package to look in for Monitors and which to load. All Monitor configuration is looked up on a per Host basis and can/should be specifically overridden to configure specific monitoring properties for specific hosts.

```
Monitor.alertTimeout.NOTICE=60
Monitor.alertTimeout.WARNING=900
Monitor.alertTimeout.CAUTION=1800
Monitor.alertTimeout.CRITICAL=3600
```

Or

```
Monitor.CPU.alertTimeout.NOTICE=60
Monitor.CPU.alertTimeout.WARNING=900
Monitor.CPU.alertTimeout.CAUTION=1800
Monitor.CPU.alertTimeout.CRITICAL=3600
```

The above options configure the timeout before raising an alert to the next alert level. These values are in seconds. Note that it is possible to set a default alertTimeout value for ALL monitors, as well as setting individual alertTimout values for specific monitors, should the need arise.

```
Monitor.CPU.threshold.LOWER=90
Monitor.CPU.threshold.UPPER=99
```

Threshold values are used to indicate that a problem has occurred. When a monitor detects its value (e.g.,. CPU) lifts above the LOWER level, it raises an alert at the NOTICE level, this alert then escalates according to the alertTimeout value. However if the level raises above the UPPER threshold level, alertTimout values are halved and the alert will be escalated twice as quick. It should be noted that the Services Monitor should have NO thresholds configured, as the system deals with this internally.

```
Monitor.Disk.thresholdMeasure=PERCENTAGE
```

Or

```
Monitor.Disk.thresholdMeasure=VALUE
```

The above option is for Disk, Memory and Queue monitors ONLY. It allows you to indicate that the threshold values are quoted in “kb” for disk or number of items for Queues, rather than the standard, which is as a percentage.

```
Monitor.Heartbeat.reachFINALcount=5
```

This above option allows you to specify how many CRITICAL alerts are raised before it moves to a FINAL alert. Typically this would only be useful for machines that you know are likely to disappear, and it ensures that the system does not keep complaining about them disappearing. It might be interesting to note that you can use this option on any Monitor, but it is only really useful for heartbeats, as that indicates a machine has gone.

```
Monitor.Heartbeat.checkPeriod=50
```

The above option is for Heartbeat monitors only, determines how often (in seconds) the Heartbeat monitor should check when a host last sent a heartbeat. This is special because it is the only monitor that is not stimulated by the arrival of data.

A final point to note about monitors is that the CPU and Process monitors have a subset of monitors within it them can independently be set a Monitor configuration. These are “user”, “kernel”, “iowait” and “swap” for CPU levels and “total”, “cpu”, “sleeping”, “stopped” and “zombie”. These can be specified as in the following example:

```
Monitor.CPU.alertTimeout.NOTICE=60
Monitor.CPU.idle.alertTimeout.WARNING=900
Monitor.CPU.idle.alertTimeout.CAUTION=1800
Monitor.CPU.idle.alertTimeout.CRITICAL=3600

Monitor.CPU.idle.threshold.LOWER=90
Monitor.CPU.idle.threshold.UPPER=99
```

Note that the WebFeeder monitor has no specific configuration of its own, it simply connects to the central WebFeeder

Alerter Options

The first thing to note is that the messages that are sent from the alerting mechanisms, all use a string replacement before they get sent. The following variables are available to be used in the Alerter messages:

```
%level%            - the alert level (eg, WARNING)
%threshold%        - the threshold broken (eg, LOWER)
%source%           - the source of the alert (eg, raptor.ukc.ac.uk)
%value%            - the value reached (eg, 95)
%thresholdValue%   - the value of the threshold broken (eg, 90)
%attributeName%    - the attribute that has caused the alert (eg, CPU User)
%timeTillNextAlert%      - the time the next alert will be sent out
%timeSinceFirstAlert%    - the time elapsed since the first alert for this problem
%timeOfFirstAlert%       - the time the first alert was sent
```

It should be noted that the Alerter configurations are also loaded (like the Monitors) on a per Host basis, allowing configuration of the alerting mechanisms to be different for different hosts, or groups of hosts.

The three available alerting mechanisms can be started as follows.

```
Alerter.PluginsPackage=uk.org.iscream.client.alerters
Alerter.Plugins=EMail;IRC;WebFeeder;
```

As with the other plugin systems in the server, you first specify the package to load the plugins from, then the semi-colon separated list of plugins to load. Note that the WebFeeder alerter has no specific configuration of its own, it simply connects to the central WebFeeder

The Email alerter options will be dealt with first.

```
Alerter.EMail.level = WARNING
```

A standard configuration option, this tells the alerter what the minimum level of alerts it should send is. It should be noted that as soon as this level of alert is broken, if an OK level alert is sent, the alert will also send that to indicate the broken alert level is now OK.

```
Alerter.EMail.destList = dev@i-scream.org.uk
```

This is the email address to send the alert messages to.

```
Alerter.EMail.sender = dev@i-scream.org.uk
```

This is the email address that the alert messages will appear from

```
Alerter.EMail.smtpServer = mercury.ukc.ac.uk
```

This is the server that the alerter will use for SMTP in order to send the mail.

```
Alerter.EMail.subject = i-scream alert: %level% alert on
%source% for %attributeName%
```

The subject of the email to be sent. Note that the string replacement variables can be used here also.

```
Alerter.EMail.message = The i-scream distributed central
monitoring system has\nraised a %level% alert for the host
%source%.\n\nThe value for %attributeName% of %value% has
exceeded the\n%threshold% threshold value of
%thresholdValue%.\n\nThis alert was originally raised at
%timeOfFirstAlert%,\nwhich was %timeSinceFirstAlert%
ago.\n\nThe next alert (should one occur) will be sent in
%timeTillNextAlert%.
```

This is the message to send, this is totally configurable, but is advised to make as much use of the replacement variables as possible in order that the problem is conveyed. Note also the use of “\n” standard escape code for formatting.

The second main type of Alerter is the IRC alerter.

```
Alerter.IRC.level = OK
```

It too has a minimum level of alert to send. Note here that setting this to OK means it will send ALL alert levels sent, including simple NOTICE’s which are raised immediately a threshold is broken.

```
Alerter.IRC.IRCServer = compsoc1.ukc.ac.uk
```

This is the name of the IRC server the bot should connect to.

```
Alerter.IRC.IRCPort = 6667
```

This is the IRC port that the IRC server is running on

```
Alerter.IRC.nickList = iscreamBot;_iscreamBot;i-screamBot
```

This is a semi-comma separated list of optional nicknames that the bot can use, should one be taken.

```
Alerter.IRC.user = i-scream
```

This is the user-id to report to the IRC server.

```
Alerter.IRC.comment = i-scream alerting bot
```

This is the comment to attach to this bot for IRC.

```
Alerter.IRC.channel = #i-scream
```

This is the IRC channel that the bot should join.

```
Alerter.IRC.message = %level%: %attributeName% on %source% has
passed %threshold%(%thresholdValue%) threshold with %value% -
time till next alert (should one occur), %timeTillNextAlert%
```

This is the message that should be sent to the channel. Note that this message should not be too long, but enough to convey the problem.

```
Alerter.IRC.reconnectDelay = 30
```

This is the time in seconds that the IRC bot should wait if it had trouble connecting before trying again.

```
Alerter.IRC.startupNotice = i-scream alerting bot activated
```

This is the notice message that should be broadcast to the channel on activation of the bot.

```
Alerter.IRC.stopCommand = stop alerts
Alerter.IRC.startCommand = start alerts
Alerter.IRC.lastAlertCommand = last alert
Alerter.IRC.joinCommand = join
Alerter.IRC.nickChangeCommand = nick
Alerter.IRC.statCommand = statistics
Alerter.IRC.uptimeCommand = uptime
Alerter.IRC.timeSinceLastAlertCommand = time since last alert
Alerter.IRC.versionCommand = version
Alerter.IRC.helpCommand = help
```

All the above are the messages people can type to ask the bot to execute various (self explanatory) commands.

```
Alerter.IRC.rejectMessage = sorry, I don't understand your
request
```

This is the message a user will receive if they try and ask something of the bot that it doesn't understand.

WebFeeder Options

These are the various configuration options for the WebFeeder. The WebFeeder is receives information from the WebFeeder Monitor (which contains the latest data flowing through the system) and the WebFeeder Alerter (which contains the latest alerting information).

```
WebFeeder.latestActive = true
WebFeeder.alertActive = true
```

The above options turn on the two sections of the WebFeeder. To disable either the latest data or the alerting webpage updates, simply comment the above lines.

```
WebFeeder.alertLevel = OK
```

Like other alerters, this configures what alert levels the WebFeeder cares about.

```
WebFeeder.checkPeriod = 120
```

Periodically the WebFeeder will check its directory structure and remove old and stale alerts from the system. This value is the time in seconds between carrying out this task. These are alerts that have either gone "FINAL", have had a their host go "FINAL" on a heartbeat and thus non of the alerts will apply and finally OK alerts, as they are only displayed for a short period to indicate that all is well.

```
WebFeeder.alertDeleteOlderThan = 300
```

When the checkPeriod run occurs, this value is checked against the time stamp. It is measured in seconds and if the stale alert is older than this timeout, it is removed.

```
WebFeeder.rootPath = /usr/local/proj/co600_10/webroot
```

This is the full local system path to the root folder that the alert data and latest data will placed, appended by their specific paths below.

```
WebFeeder.latestSubDir = latest
WebFeeder.latestFileName = latest_data
```

These configure the names of the directory structures and files to use for the latest data.

```
WebFeeder.alertSubDir = alert
WebFeeder.alertFileName = alert_data
```

These configure the names of the directory structures and files to use for the alerting data.

For more information about the i-scream web services and how the web feeder supplies them with information, please refer to their documentation.