Maintenance Documentation

Conient {an i-scream client}

Conient is an implementation of an i-scream client, it conforms to the i-scream Client-Server protocol v1.1 and allows data to be displayed in real-time in a Java Swing based graphical application. This documents aims to detail how the application works from a maintenance point of view.

Revision History

27/03/01	Initial creation				
		Committed by:	ajm4	Verified by:	tdb1
				Date:	28/03/01
28/03/01	Proof Read and Improved Readability				
		Committed by:	ab11	Verified by:	tdb1
				Date:	29/03/01
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	

Introduction

Conient is an i-scream real-time client. It is a Swing based Java application that connects to the i-scream server through the client interface. The client interface sits next to the root filter and receives a live stream of data as it passes through the system. The client interface then passes this stream to any clients that are connected.

Conient communicates to the client interface using the Client Control and Data protocols, as detailed in the protocol specification. Conient receives its data over the data link and displays the data in a graphical form. For examples of this display and information on how to use it, please refer to the Conient user documentation.

Conient is written in Java and is recommended to be only used with Java 1.3 and later. Although it does work with 1.2, Conient makes use of a large number of Swing components to display its data and there are a large number of Swing bugs that have been fixed and is not guaranteed to display its data correctly under a 1.2 JVM.

The application is self-contained, and can be run without installing anything other than Java. It can be on any platform that supports Java and uses no native libraries or system calls. However the easiest and recommended way of compiling Conient is through the use of the supplied Makefiles, and this really restricts compilation to any system that is able to use this system, typically only Unix style operating systems.

This document aims to discuss each section of the client in a manner that would allow other developers to maintain it. It should also be noted that the code itself contains extensive javadoc, which is also viewable on the i-scream website.

Overall structure

The diagram at the end of this document details how the various classes link together within Conient. The class Conient is the root of the system and contains the standard Java bootstrap method “public static void main(String[] args)”. This method initialises the singleton Configuration, the main Conient frame, the DataPanel and the ControlPanel. The ControlPanel then sets up the menu bar and tool bar, it then initialises the ConnectionHandler class passing it a reference to the Queue it will use to pass actions to the ConnectionHandler. The ConnectionHandler is also passed a reference to the DataPanel and from then on is effectively the centre of operations, allowing the various parts of the system to communicate and operate.

This document will detail the interactions between these classes and how they operate. It should be noted again that for detail descriptions of methods and algorithms in use, both javadoc and the code itself are heavily commented.

Conient

The static main method of this class is used to initialise Conient. It starts by initialising the Configuration class using the file specified, the default file is held in the attribute DEFAULT_CONFIG_FILE. This class is a singleton and is used by all areas of the system to obtain configuration properties from a local configuration file. This file can be specified on the command line, and thus the Conient class passes this option (either the default or the one from the command line) to the Configuration object. It should be noted that if the initial load of the default configuration fails, Conient will display an error and exit. For more information on the Configuration class please see later.

Once the Configuration system has been initialised, Conient then creates the DataPanel and ControlPanel and passes them as arguments to its own constructor. This constructor creates an instance of the Conient class. It adds the panels (which extend JPanel) in the appropriate places, and it obtains the menu bar from the ControlPanel and adds it to the frame. It then sets up the other components of the display and displays the all important splash screen. Once display of the splash screen has completed, it disposes of it and shows the main Conient class which extends JFrame. The default size of this frame is defined by the attributes DEFAULT_HEIGHT and DEFAULT_WIDTH.

Key Features

addMessage(String message)

This is a method most almost all classes in Conient contain. At the bottom of the main display is the “Messages” box. This is a JTextArea in a JScrollPane. This box is used to display what are in effect “logging” messages, and allow components to indicate to the user their status (including errors) in a non-intrusive manner. This method is a static method and can thus be called by all classes, the message they pass in is simply added to the first line of the text area, thus new messages appear at the top.

setDataStatus(String status) & setControlStatus(String status)

These methods are called by the ConnectionHandler class. The control and data status boxes are JLabel’s which are positioned just under the messages box. They are used to indicate to the user the status of the two channels that connect Conient to the server. These methods allow the ConnectionHandler to indicate to the user the current state of these links by calling them with the new status to be displayed. By default they are “Disconnected”.

setQueueStatus(int currentQueue, int numElements)

This method is called by the DataPanel class. The queue status box is a simply JLabel that sits below the control and data status boxes. This method is called when the DataPanel processes an item of data from its queue, it then sets the status of its queue, passing how many elements it still has in it and the number it has processed to date respectively. See the DataPanel section later for information about its data processing.

Inner class Splash

This is a very small inner class that simply displays a borderless window containing an image relating to the project. It displays for only a small amount of time (defined by DEFAULT_SPLASH_TIME_SECONDS). This window could be used to display information about the loading of Conient if it is ever expanded in the future.

ControlPanel

The ControlPanel is a small and relatively simple class that extends a JPanel, it is added to the North border of the main Conient frame. It is a central point for all Conient control interaction and defines both a JToolbar and a JMenuBar. On initialisation it constructs both these bars and adds the appropriate ActionListeners to them. Its main task then is to construct the ConnectionHandler class. Commands are passed to the ConnectionHandler through the use of an “ActionQueue”. This is so that the thread of control for the GUI is not lost while the ConnectionHandler carries out some potentially long operations. More detail on the use of the ActionQueue is provided later under the section that deals with the ConnectionHandler. The only physical display that this panel has is the JToolBar.

Key Features

getMenuBar()

This method is called by the Conient frame when it initialises, it simply return the menu bar so that Conient can add it to its frame.

The JToolBar

The JToolBar has four main buttons to control the two network links as detailed in the user documentation. All these four have action listeners attached to them which add appropriate commands to the ConnectionHandler’s action queue.

The JMenuBar

The JMenuBar has two main menus, “Conient” and “Connection”. The first has options to reconfigure Conient. The action listeners attached to these menu items make appropriate calls to the singleton Configuration object to carry out configuration tasks. The second menu is identical in operation to the tool bar and has items that add appropriate commands to the ConnectionHandlers action queue.

ConnectionHandler

This class provides the links to the i-scream server. One important point to note though at this point is the low coupling between the obtaining data, being handed instructions and passing them on to be displayed in an attempt to follow a Model, View , Controller paradigm. The whole connection handling mechanism makes full use of the uk.org.iscream.util.Queue that has been heavily used throughout the project. This allows data to be read from the data link and simply added to a queue, it has little or no knowledge of the display mechanism, and the knowledge it does have is simply to pass on references to the queue and inform the data panel that the data link has been stopped or restarted, both of which can easily be changed. These features mean that it could be possible for this class to be plugged in as the connection handling class for a variety of Java based clients, indeed an early proof of concept client was one that would work on a command line interface (CLI), and this class would have been easily placed into this application to provide full v1.1 protocol support.

During initialisation the ConnectionHandler sets up its action queue and informs the Configuration singleton that it is up by passing a reference to itself, this allows the Configuration object to obtain configuration through the control channel.

Once constructed the ControlPanel then starts the ConnnectionHandler running. Information about the two links this classes manages can be found in the protocol specification document.

Key Features

run()

This class extends the Thread class and as such contains a run method to carry its control. This method starts by checking if any default actions have been specified in the configuration (such as to start the control link on startup). It then enters its main loop. This loop reads from the action queue for new actions that have been sent to it. These actions are then used to carry out the requested command.

Actions

The action queue is used by the ControlPanel and the ConnectionHandler itself to tell its main thread to carry out commands. The ConnectionHandler defines a range of static integers that can be referenced to add the appropriate commands to the action queue. As the Queue class can only handle Objects, theses primitive integers must therefore be wrapped in the standard Integer wrapper class before being added to the action queue.

It should be noted that if any of the actions have any problems completing they should take steps to remove all other actions from the queue as appropriate, as some actions depend on others to complete.

The following actions are available, and can be referenced by ConnectionHandler.<action>.

DONOTHING

This is the default action and is simply there to account for a 0 value int. As is implied by its name, this action does nothing.

CONNECT

This instructs the ConnectionHandler to attempt to establish a control channel with the i-scream server. It first checks that the link is not already established. If it isn't it will then proceed to try and establish the link. It first obtains the appropriate configuration options from the Configuration object, if the firewall options are in use it will call the handleFirewall()

method to setup any tunnelling that is required and return the new server and port to connect to. It then attempts to open a socket to the i-scream server. Once the connection is established, it will go through any handshaking that is required by the protocol in use. The protocol that the ConnectionHandler is currently written to support is specified by the final value PROTOCOL_VERSION and is hard coded into this class. It should be noted that the protocol is backwards compatible, so if the server is using a newer protocol, Conient will warn the user but will not fail the connection. That is the end of this action and the control link will either be up and an appropriate error will have been displayed.

STARTDATA

This action instructs the ConnectionHandler to establish the data channel with the server. This action will first decide if the control channel is open, if it is not, it queues the CONNECT action to start the channel as it is required to negotiate the data channel, it also adds STARTDATA action to the queue so that this action is immediately returned to when the control link has been established. This mechanism happens in two stages. It first needs to negotiate with the server a port to connect the data channel to. This is done as specified in the protocol, the server will then return the port that it is listening on for the new data channel. This action will then ask the handleFirewall method to modify these details as needed if firewall tunnelling is in use and then it will establish a socket connection with the i-scream server. The next task is to start the DataReader class reading the data as it flows in from the socket, this DataReader is passed a reference to a Queue. This queue is constructed with a limit defined from the configuration, this prevents it from filling up and crashing the system. The default algorithm for the Queue (see the Queue javadoc for information on the algorithms available) is FIRST and is defined by the final attribute QUEUE_ALGORITHM, the default queue size is defined by DEFAULT_QUEUE_LIMIT and is 500 by default, though it uses the Configuration value by preference. This queue is then also passed to the DataPanel class which is responsible for reading the data out of the queue and displaying it appropriately. The DataPanel is then instructed to clear up its display and both the DataPanel and the DataReader classes have their threads started. Data then commences to flow into the system.

STOPDATA

This action instructs the ConnectionHandler to stop the data flow into the system. After making checks that it is appropriate to stop the link, it then starts to close it. It first shuts down the DataReader, by calling the shutdown method. This is only actioned by the DataReader once per data read and so to prevent this holding up the shutdown of the link, the action waits for DATAREADER_SHUTDOWN_TIMEOUT time in seconds for the reader to shutdown, after that it checks if the thread is still running and warns appropriately before terminating its I/O at which point the DataReader will throw an exception and die. The action then closes the data socket and then instructs the firewall handling methods to close the firewall tunnelling process. The DataPanel is not informed that the link has gone, as it will simply stop updating once its queue is empty.

DISCONNECT

This action instructs the ConnectionHandler to disconnect the control link. If the data link is detected to be open, it will queue up the STOPDATA action and re-queue itself for once that action has finished. This action then closes the socket and instructs the firewall handling methods to close the firewall tunnelling process.

getConfigFromServer(String configName, String propertyName)

This method is called by the Configuration object in order to obtain configuration options from the servers configuration. This carries out the command on the control channel as specified

in the protocol specification. This method will return an empty String if the configuration is not found in the server.

shutdownDataLink()

This method is called by the DataReader if it detects a fault on its data stream. This can occur with any interruption to the data link with the server. This method then enqueues the STOPDATA action to shutdown the link.

setHostList(String hostList)

This performs the SETHOSTLIST command as specified in the protocol specification. This is called by the STARTDATA to set up the data link. One of the options available to Conient is to be able to receive only data that it requires. This allows the user to enter a list of hosts that they want to see data for and then this list is set on start of the link. This can greatly reduce traffic to the Client as only a subset of the data is sent. For more information about the options available, please see the user guide.

handleFirewall(String server, int port, Process firewallProcess)

One of the nicer extra features of Conient, is its ability to use a third party tool to establish a tunnel (often through a firewall) to gain access to the i-scream client interface. This is not a fully supported feature, but one which has proved to be very useful. This method takes a server name, a port and a reference to a process to hold the firewall process in. It then consults the documentation to obtain options relating to the firewall tools. If firewall tunnelling has been requested, it then executes the “firewall command” as specified in the Configuration. This system then waits a set amount of time before trying to use the ports that this command should have opened. It then returns the server name that the port is open on to the calling method. It is assumed to be localhost, but it maybe that the command executed on a remote machine to set the tunnel up. For more information on the configuration options available for this, please refer to the user guide. This method is called by the CONNECT and STARTDATA actions.

closeFirewall(Process firewallProcess)

This method calls the destroy method on the given firewall process the process to null, thus closing the tunnelling process. This method is called by the DISCONNECT and STOPDATA actions.

DataPanel

The DataPanel is the core of the data displaying mechanism. It is added to the “Center” panel of Conient’s main display. It extends JSplitPane which means it is split into two distinct halves down the vertical axis. This split is adjustable and makes use of the usual features supplied by the JSplitPane.

The right hand pane consists of a JPanel which is configured to use the CardLayout layout manager. This area is used to hold multiple HostDisplayPanel objects, each of which displays data about a specific host. The panels are layered using the CardLayout so that only one is visible at a time. These panels are contained in a JScrollPane to allow their data to move outside the visible area, but still allow the user to scroll to view the data.

The left hand pane consists of a JList which holds an alphabetical sorted list of hostnames. This list has an ActionListener attached to it which detects a change in selection on the list. When it sees the user has made a selection change it looks up the name of the host selected and then asks the CardLayout to bring the card holding the appropriate JPanel to the front.

Key Features

run()

The DataPanel class implements the Runnable interface and as such has to implement a run method. This method is used for the thread of control for the DataPanel and is started by the ConnectionHandler.

The first thing this method does is setup the host list and panels. If the user has requested a fixed host list to be used, the DataPanel can setup the list and add appropriate panels before the data is read in. See the refreshHostList() method. Next it displays the QueueFrame if requested, see later for the use of this. It then enters its main loop.

The main loop starts by reading an object out of the data queue. These objects will have been placed in by the DataReader. It then casts this object into a String that will contain XML text which needs to be converted into an XMLPacket by the XMLPacketMaker, for information on these classes please refer to the server maintenance documentation.

At this point if the packet dump option is selected, it will dump the toString() of the XMLPacket to the system console. This is typically used for debugging purposes only.

It then asks the XMLPacket for the type of packet it contains, if it is heartbeat or data packet the method then decides if it should already know about the host or not. It looks the hostname up in a hashmap and obtains a reference to the HostDisplayPanel that is looking after data for that host, it then calls the updateHost(XMLPacket) method on the HostDisplayPanel. Note that if a fixed host list is not being used it will have to create a new HostDisplayPanel if the host it encountered is not present in the display. If it is configured to “discover” new hosts, it will also add the name to a list of known hosts, see the user guide for information about this feature.

If the packet is a queueStat packet and the queue monitoring is in use, then it passes this packet to the update(XMLPacket) of the QueueFrame class.

Note that if any uncaught exceptions are thrown by the HostDisplayPanel, this method will print the appropriate message to the Messages window and cease operation until the data link is started again.

refreshHostList()

Constructs the JList on initialisation of the run() method. This reads in the host list from the configuration and adds the hosts and their HostDisplayPanel's to the display. Note that if the list is empty, it assumes that all data has been requested (as indeed it will according to the protocol spec) and it will then set the run method to add new hosts as it finds them using the _usingConfiguredList attribute.

addHostPanel(String host)

When an unknown host is detected and the user is not using a fixed host list, this method is called with the hostname of the new host. This method will then create a new HostDisplayPanel for the host and then added to the CardLayout panel on screen, associating it with the given hostname. It will then add the name to the JList and update the JLists display. This is done by taking the keySet from the HashMap that holds the HostDisplayPanels and performing and passing it to a TreeSet which will then sort the items alphabetically. This resulting TreeSet is then set to be the data source of the JList, thus resulting in a sorted host list.

cleanUpTabs()

This method is called by the ConnectionHandler when a new data link is established. This resets the JList, the main JPanel and the CardLayout, effectively clearing the display ready for new data that is about to come in.

setQueue(Queue queue)

This method is called by the ConnectionHandler when a new data link is established, it assigns the DataPanel the new Queue that the current DataReader will be using to pass the data through to the DataPanel.

HostDisplayPanel & DataComponent architecture

A HostDisplayPanel displays and maintains the display of all the data entering Conient for a specific host. It uses as its base a JPanel which it extends. To this it adds two PacketTimer objects, which are used to give an indication of when the next data and heartbeat packets will arrive, and one JButton to launch a platform information window and a large number of DataComponent objects.

The PacketTimer objects are simply threads which update a JprogressBar. The HostDisplayPanel asks the Configuration class to lookup on the i-scream server the time intervals between data packets and heartbeat packets arriving for the host this HostDisplayPanel is responsible for, it then passes these values to the PacketTimers to give them a scale.

The platform information JButton is assigned an action listener which spawns a new JFrame containing simple StringDataComponents that display non critical system specific information such as operating system version and uptime. This is done to keep largely unused data available but hidden.

DataComponents

The DataComponent interface is used throughout this section of Conient. DataComponents are responsible for looking after a particular data item from a packet. DataComponents all implement the DataComponent interface and are added to the _components HashMap with a key linking it to the packet attribute it is responsible for. When a new packet comes in to the HostDisplayPanel, it simply iterates over the packet attributes and then passes it to the responsible DataComponent and therefore does not have to concern itself with how to handle each data item during the update of the display, it simply delegates that to the relevant DataComponent.

The next key thing to note about the DataComponent architecture is the distinction between visible and non-visible components. A visible data component should extend the VisibleDataComponent class to ensure that it extends JPanel as its base container. Whereas a non-visible DataComponent only needs to hold the data or perhaps process it. However, it turns out that all defined DataComponents are classed as visible, and the StringDataComponent is used to hold data for non-visible components but it is simply not added to the display.

Thus the HostDisplayPanel has two methods to add components to its component list, one to add the component to the display and the hash, and one to only add it to the hash.

It should be noted that by default all DataComponent should setVisible(false) when they initialise, then setVisible(true) when they process their first item of data. In this way components will only be visible if a host actually sends the data.

Display Construction

The display is constructed using the GridBagLayout to allow for specific positioning and sizing of components according to the constraints. The display is split into a number of JPanels which hold related data components, these are: “cpu”, “loadData”, “processesData”, “memory”, “users”, “_disks”, “_services “ and “_extra”. The “_extra” panel is used when the option to display extra packet data is selected. When the host updates its data components, it will also detect any attributes in the packet that it doesn’t know about and create a new data component in the “_extra” panel if this option is selected.
Disks and service checks are added to the display dynamically as it is impossible to easily determine in advance how many and the details about each item under these panels. The HostDisplayPanel has methods to handle this. When a packet attribute is detected that does not have a component available to handle it, it checks to see if it is either a disk or a service check before classing it as “_extra” data. It then calls the appropriate method to add the new disk or service check to the display.

Key Features

UpdateHost(XMLPacket packet)

This method is called by the DataPanel when it receives new data for this HostDisplayPanel, it passes in the XMLPacket that contains the data. This method then determines the packet type. If it's a heartbeat packet, it resets the heartbeat PacketTimer then calls the processPacket method, if it's a data packet, it resets the data PacketTimer and then calls the processPacket method.

processPacket(XMLPacket packet)

This method iterates over all of the attributes of the given packet, it looks in the _components HashMap to see if a DataComponent is present that can deal with the found attribute as described in the DataComponent architecture above.

addDataComponent(DataComponent dataComponent)

Simply adds a non-visible DataComponent to the _components HashMap, in effect registering the attribute that the DataComponent looks after.

addVisibleDataComponent(Container holder, VisibleDataComponent dataComponent)

Adds the given VisibleDataComponent to the given holder, then casts it to a DataComponent and passes it to addDataComponent.

createDiskPanel(String attribute)

Creates a new JPanel to hold disk information, this is called by the processPacket method when a new disk is detected. This method constructs appropriate DataComponents to handle all the needed disks attributes, including attributes that are sent but not visible, it then adds them appropriately.

createServicesPanel(String attribute)

This is almost identical to the above message, but processes service check attributes instead of disks.

The DataComponents package

This package contains all the available DataComponents in use in Conient. New DataComponents should be added to this package and implement and extend appropriately according to the DataComponent architecture described above.

DataComponent

This is the interface that all DataComponents should implement. It has two methods, one to set the value of the DataComponent is currently holding. The second to query what the main packet attribute the DataComponent is looking responsible for. This last method is called when adding the component to the _components HashMap.

It should be noted that the setValue method throws the DataFormatException. This should be thrown by the DataComponent when it detects data that it cannot process either due to corruption or incorrect values. The HostDisplayPanel will then display an error in the Messages box detailing the toString of the component that failed and what data it failed on and from which host.

VisibleDataComponent

An abstract class that extends JPanel and implements the above interface. This should be extended by all DataComponents that intend to display their data on the display. As you see from the system diagram, this is the root of all current DataComponents.

Another point to note here is that VisibleDataComponent also implements the Runnable interface. Please refer to the section titled “Swing Safety Utilities”.

StringDataComponent

This is the most basic of DataComponents. It simply displays a disabled JTextField with a JLabel side by side that will contain the data and name of the attribute respectively.

DateDataComponent

This extends StringDataComponent to convert the data value to a date format.

UptimeDataComponent

This extends StringDataComponent to convert the data value into an uptime format, this uses a static method in the uk.org.iscream.util.DateUtils class to achieve this.

CPUDataComponent

This contains a label and a JProgressBar to visually display the value (which should be passed as a %) of CPU data.

UsersDataComponent

This again contains a label for the component, but this time has a JComboBox to display a drop down list of users logged onto a system.

ServiceDataComponent

This component has a label which is set to the name of the service check that it is responsible for. It then prints the status and the result of the service check in a JTextField. This component effectively looks after two packet attributes, both of which are passed in on construction. Which means this component must be added twice to the HostDisplayPanel, once for each attribute, but only the main attribute should be added as a VisibleDataComponent to ensure that it only appears once on the display.

ProcessesDataComponent

This DataComponent is also used to display load data. It simply returns two JTextFields stacked upon each other, the top showing the label and the bottom the data value. When several of these are combined it thus displays in a tabular format.

StorageDataComponent

The storage DataComponent is used to display a storage resource such as memory. Visually it is similar in appearance to the CPU component, but also has support for displaying the size as well as the percentage in use, and as such it has a constructor which can take unit information (e.g., Mb) and a divider which takes a value to divide the raw packet data by to obtain the display size. It takes two attributes, one for the value and one for the max value, and as such it should register itself for both, with only the main attribute being registered as a VisibleDataComponent

DiskDataComponent

This component extends the StorageDataComponent and simply adds more information to the label, such as disk mount point and the disks device name. It takes four attributes, one for the value, one for the max value (which are both passed to the StoreageDataComponent constructor), one for the mount point of the disk and one for the device the disk is on. Thus it should register itself for all of these attributes, with only the main attribute being registered as a VisibleDataComponent.

Configuration

The Configuration object is a singleton and is used by almost every component in the system to obtain configuration details. It provides this configuration mainly through a Java Properties object, which has support for reading and writing a properties table in a file. It also allows components to obtain configuration options from the server, this is done once the ConnectionHandler has been brought up, the ConnectionHandler then forwards configuration request on behalf of the Configuration object.

Key Features

initialise(String configFile)

This constructs the Configuration object and should be called before any other method is called on the singleton. It will throw a RuntimeException if this is not obeyed. This reads in the default configuration file which and will error and perform a System.exit(1) if the initial configuration file cannot be loaded. This file is defined in the Conient class which can also obtain the name off the command line on startup.

getServerProperty(String configName, String propertyName)

This takes the name of a server side configuration (e.g. Host.raptor.ukc.ac.uk) and a property name (e.g. TCPUpdatetime) and asks the ConnectionHandler to obtain the configuration from the server. It will return null if the ConnectionHandler is not running (which should never occur according to bootstrap order in Conient) or an empty String if the server either does not have the property or has trouble obtaining it.

GUIReconfiguration()

This is called by the “Modify Configuration” option on the JMenuBar. It displays a ConfigurationDialog.

saveNewConfiguration()

This displays a JFileChooser to allow the user to type in the name of a new file to store the configuration in. It then sets the file in use to be this new filename and calls the saveConfiguration method.

saveConfiguration()

This method calls the saveFileConfiguration with the currently loaded filename.

loadConfiguration()

This prompts the user with a JFileChooser to select the configuration file they would like to load. This method then calls the readFileConfiguration to load in the selected file. It will show appropriate errors to the user should a problem occur.

readFileConfiguration(File inputFile) throws FileNotFoundException, IOException

This method is called to load a specified configuration into the Properties file. It uses the Properties classes own methods for parsing the file. It throw any errors out to the calling method should any exceptions be thrown.

saveFileConfiguration(File outputFile)

This saves the current Properties object to the specified output file using the Properties store method. It will display any errors to the user should any occur. Note it makes use of the CONFIG_HEADER attribute to write an appropriate header into the file.

getProperty(String key)

This is used by any part of the system that requires configuration properties. It behaves exactly as the java.util.Properties object does.

setProperty(String key, String value)

This allows any part of the system to modify a configuration entry. This should only really be called by the ConfigurationDialog class, although in future it maybe useful for other areas of the class to be able to modify their configuration.

setConnectionHandler(ConnectionHandler connectionHandler)

This is used by the ConnectionHandler class to tell the Configuration singleton that the ConfigurationHandler is now online. It also supplies a reference to the handler so that the Configuration object can use the handler to obtain requested configuration properties from the server.

ConfigurationDialog

The ConfigurationDialog is the main method for the user to interact with the configuration settings. It is recommended that the user is NOT asked to modify the raw configuration details and that any additional changes to required configuration options be made in this class.

This class displays a modal dialog box with four tabbed pane options for “client”, “server”, “firewall” and “data”. For details on the options available currently, it is recommended that you consult the Conient user guide.

Key Features

getCurrentSettings()

This method is called on construction and contains reads all the configurable options from the Configuration object. It then updates all the displaying JComponents with their current configured values.

prepareLists()

On initialisation this class reads two variables from the configuration that hold a set of known hosts and a set of hosts to monitor. The hosts to monitor is always a subset of the known hosts, but the lists are split for displaying purposes so that only the known hosts that are not being monitored are displayed in the known hosts list. This method performs this sorting when the lists are first loaded.

setListDisplay()

The host lists are maintained using an ArrayList as the data model, however the JList component that is used to display this data does not yet have full support for using the Collection classes as its underlying model. Thus whenever a change to the list occurs this method should be called to both sort the new list into alphabetical order and update the display.

setNewSettingsAndClose()

Once the user has finished their modifications to the configuration they can then click “OK” to exit. This buttons’ ActionListener is attached to this method. This method then writes back all the values held by the display components into the configuration. It also combines the monitored hosts back with the list of known hosts and writes them back also.

Miscellaneous Utilities

QueueFrame

The queue frame class is a very simple class that will not be discussed in detail. Essentially it operates in a similar fashion to the HostDisplayPanel, however it is simply for displaying QueueStat packets from the server. If the user selects the option to display this window in the configuration, this is displayed on the start of a data channel. It then displays the source and current queue information from all areas of the server and allows the user to keep an eye on server data statistics and queue build ups. This data is supplied by the QueueMonitor class within the server, for more information please refer to the server maintenance documentation.

Swing Safety Considerations & Utilities

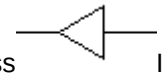
One of the important things to note when using Swing classes is that they are not thread safe. A system wide work thread called the Swing Event Dispatcher runs to continually update the display and make changes as the windowing environment around the application changes. This then means that if two parts of the system try and update the display, it could result in a corruption of the visual layout. To resolve this problem it is necessary to add all tasks to the event dispatcher using the SwingUtilities static methods invokeLater and invokeAndWait. These methods take a class that implements Runnable and executes their commands in the order that they are received, thus no simultaneous updates occur to the display and thus prevent corruption.

Within Conient this problem is dealt with in two clear ways. The first is that the VisibleDataComponent abstract implements the Runnable interface, thus all extending classes must implement this. All VisibleDataComponents then place all their updating of the display in their run() method for the event dispatcher to execute as it sees fit. The second is the use of two utility classes to add objects to JPanels. Often a JPanel will be visible when a new component is added, often when a DataComponent is added to a display dynamically. To solve Swing issues with this problem a small utility classes exists to perform the component addition in a Swing safe way, called SwingSafeAdd. It simply takes a container and a component and adds the component to the container. Another utility is the SwingSafeAddCard class which is used by the DataPanel to dynamically add new JPanels to its CardLayout'd JPanel.

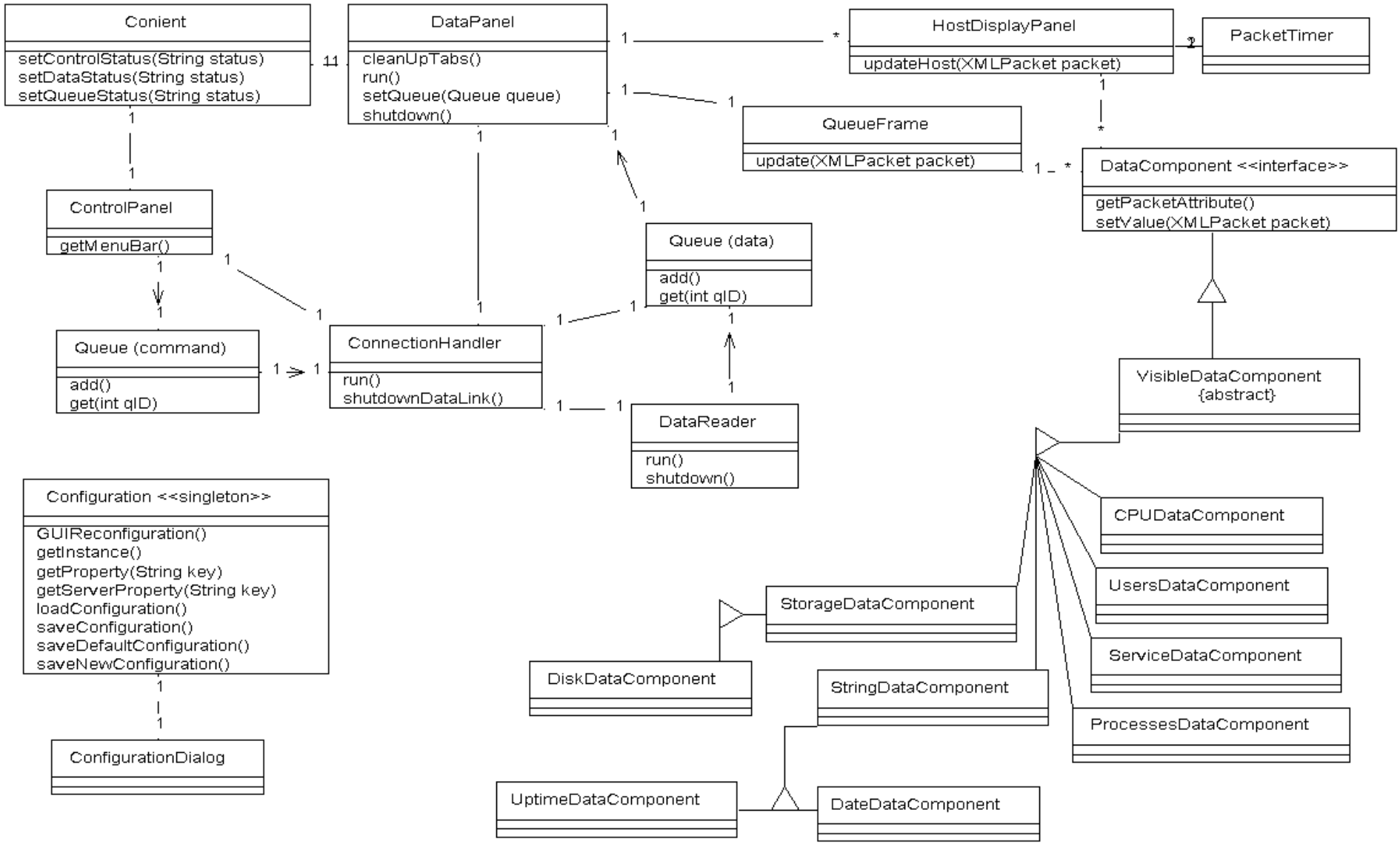
It should be noted that while these utilities may not always be needed, the Java documentation is very vague about which components are Swing thread safe and which are not. So these small changes were added to ensure that thread safety is maintained throughout.

Conient System Diagram

This diagram is intended to show the connections between the classes and the main methods available within the Conient system. Although it is based on a UML class diagram it has been modified to act as a simple visual aid to describe how the various parts of the system relate to each other. Some standard indicators do remain, such as the links between classes, e.g. 1-*. There are also some added non-standard features. The arrow heads on the links between some classes (such as the Queue class), are intended to show the direction of data flow and the following symbol is used as an indicator of “Extends” and “Implements” relationships between classes.



Interface/Super Class Implementing/Extending class


Conient
setControlStatus(String status)
setDataStatus(String status)
setQueueStatus(String status)
DataPanel
cleanUpTabs()
run()
setQueue(Queue queue)
shutdown()
HostDisplayPanel
updateHost(XMLPacket packet)
PacketTimer
QueueFrame
update(XMLPacket packet)
DataComponent <<interface>>
getPacketAttribute()
setValue(XMLPacket packet)
ControlPanel
getMenuBar()
Queue (data)
add()
get(int qID)
Queue (command)
add()
get(int qID)
ConnectionHandler
run()
shutdownDataLink()
DataReader
run()
shutdown()
VisibleDataComponent
{abstract}
Configuration <<singleton>>
GUIReconfiguration()
getInstance()
getProperty(String key)
getServerProperty(String key)
loadConfiguration()
saveConfiguration()
saveDefaultConfiguration()
saveNewConfiguration()
ConfigurationDialog
CPUDataComponent
UsersDataComponent
ServiceDataComponent
ProcessesDataComponent
StorageDataComponent
DiskDataComponent
StringDataComponent
UptimeDataComponent
DateDataComponent