Management

Table of Contents

In addition to managing the associations and servers, management also persists the state of each in XXX_sctp.xml file, where XXX is unique name given to management instance.

If there is system crash, management is responsible to bring the associations and servers back to same state it was before the crash. For example if client side association was connected to peer server before crash, management will try to connect back to peer server after restoration

API

The `Management.java` API looks like

```
/*
 * TeleStax, Open Source Cloud Communications
 * Copyright 2011-2014, Telestax Inc and individual contributors
 * by the @authors tag.
 *
 * This program is free software: you can redistribute it and/or modify
 * under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation; either version 3 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>
 *
 */

package org.mobicents.protocols.api;

import java.util.List;
import java.util.Map;

/**
 * <p>
 * {@link Management} class manages the underlying {@link Association} and
 * {@link Server}.
 * </p>
 * <p>
 * Management should persist the state of {@link Server} and {@link Association}
 * </p>
 * <p>
 * Management when {@link #start() started} looks for file <tt>XXX_sctp.xml</tt>
containing serialized state of underlying
```

```
 * {@link Association} and {@link Server}. Set the directory path by calling {@link
#setPersistDir(String)} to direct Management to look at specified
 * directory for underlying serialized file.
 * </p>
 * <p>
 * If directory path is not set, Management searches for system property
 * <tt>sctp.persist.dir</tt> to get the path for directory
 * </p>
 * <p>
 * Even if <tt>sctp.persist.dir</tt> system property is not set,
 * Management will look at System set property <tt>user.dir</tt>
 * </p>
 *
 * @author amit bhayani
 *
 */
public interface Management {

    /**
     * Returns the name of this {@link Management} instance
     *
     * @return
     */
    public String getName();

    /**
     * Get persist dir
     * @return
     */
    public String getPersistDir();

    /**
     * Directory where the XXX.xml will be searched
     * @param persistDir
     */
    public void setPersistDir(String persistDir);

    /**
    * The AssociationListener set for this Association
    *
    * @return
    */
    public ServerListener getServerListener();

    /**
     * The {@link AssociationListener} to be registered for this Association
     *
     * @param associationListener
     */
    public void setServerListener(ServerListener serverListener);
```

```
    /**
     * Adding ManagementEventListener.
     * This listener is notified when adding/removing servers and associations
     * @param listener
     */
    public void addManagementEventListener(ManagementEventListener listener);

    /**
     * Removing ManagementEventListener.
     * This listener is notified when adding/removing servers and associations
     * @param listener
     */
    public void removeManagementEventListener(ManagementEventListener listener);

    /**
     * Start the management. No management operation can be executed unless
     * {@link Management} is started. If {@link Server} and {@link Association}
     * were defined previously, Management should recreate those {@link Server}
     * and {@link Association}.
     *
     * @throws Exception
     */
    public void start() throws Exception;

    /**
     * Stop the management. It should persist the state of {@link Server} and
     * {@link Associtaion}.
     *
     * @throws Exception
     */
    public void stop() throws Exception;

    /**
     * returns true if Management is started
     * @return
     */
    public boolean isStarted();

    /**
     * This method stops and removes all registered servers and associations
     * Management should be started
     * Use this method only for test purposes or after total crashes
     *
     * @throws Exception
     */
    public void removeAllResourses() throws Exception;

    /**
     * Add new {@link Server}.
     *
     * @param serverName
```

```
     *            name of the Server. Should be unique name
     * @param hostAddress
     *            IP address that this server will bind to
     * @param port
     *            port that this server will bind to
     * @param ipChannelType
     *            IP channel type: SCTP or TCP
     * @param acceptAnonymousConnections
     *            true: this Server accepts Anonymous connections, false: no
     * @param maxConcurrentConnectionsCount
     *            A count of concurrent connections that can accept a Server. 0 means
an unlimited count.
     * @param extraHostAddresses
     *            When SCTP multi-homing configuration extra IP addresses can be put
here
     *            If multi-homing absence this parameter can be null
     * @return new Server instance
     * @throws Exception
     *             Exception if management not started or server name already
     *             taken or some other server already has same ip:port
     */
    public Server addServer(String serverName, String hostAddress, int port,
IpChannelType ipChannelType, boolean acceptAnonymousConnections,
            int maxConcurrentConnectionsCount, String[] extraHostAddresses) throws
Exception;

    /**
     * Add new {@link Server}. Server can not accept anonymous connections.
     *
     * @param serverName
     *            name of the Server. Should be unique name
     * @param hostAddress
     *            IP address that this server will bind to
     * @param port
     *            port that this server will bind to
     * @param ipChannelType
     *            IP channel type: SCTP or TCP
     * @param extraHostAddresses
     *            When SCTP multi-homing configuration extra IP addresses can be put
here
     *            If multi-homing absence this parameter can be null
     * @return new Server instance
     * @throws Exception
     *             Exception if management not started or server name already
     *             taken or some other server already has same ip:port
     */
    public Server addServer(String serverName, String hostAddress, int port,
IpChannelType ipChannelType, String[] extraHostAddresses) throws Exception;

    /**
     * Add new {@link Server}. IP channel type is SCTP. Server can not accept
```

```
anonymous connections.
     *
     * @param serverName
     *            name of the Server. Should be unique name
     * @param hostAddress
     *            IP address that this server will bind to
     * @param port
     *            port that this server will bind to
     * @return new Server instance
     * @throws Exception
     *             Exception if management not started or server name already
     *             taken or some other server already has same ip:port
     */
    public Server addServer(String serverName, String hostAddress, int port) throws
Exception;

    /**
     * Remove existing {@link Server}
     *
     * @param serverName
     * @throws Exception
     *             Exception if no Server with the passed name exist or Server
     *             is started. Before removing server, it should be stopped
     */
    public void removeServer(String serverName) throws Exception;

    /**
     * Start the existing Server
     *
     * @param serverName
     *            name of the Server to be started
     * @throws Exception
     *             Exception if no Server found for given name or Server already
     *             started
     */
    public void startServer(String serverName) throws Exception;

    /**
     * Stop the Server.
     *
     * @param serverName
     *            name of the Server to be stopped
     * @throws Exception
     *             Exception if no Server found for given name or any of the
     *             {@link Association} within Server still started. All the
     *             Association's must be stopped before stopping Server
     */
    public void stopServer(String serverName) throws Exception;

    /**
     * Get the list of Servers configured
```

```
     *
     * @return
     */
    public List<Server> getServers();

    /**
     * Add server Association.
     *
     * @param peerAddress
     *            the peer IP address that this association will accept
     *            connection from
     * @param peerPort
     *            the peer port that this association will accept connection
     *            from
     * @param serverName
     *            the Server that this association belongs to
     * @param assocName
     *            unique name of Association
     * @return
     * @throws Exception
     */
    public Association addServerAssociation(String peerAddress, int peerPort, String
serverName, String assocName) throws Exception;

    /**
     * Add server Association. IP channel type is SCTP.
     *
     * @param peerAddress
     *            the peer IP address that this association will accept
     *            connection from
     * @param peerPort
     *            the peer port that this association will accept connection
     *            from
     * @param serverName
     *            the Server that this association belongs to
     * @param assocName
     *            unique name of Association
     * @param ipChannelType
     *            IP channel type: SCTP or TCP
     * @return
     * @throws Exception
     */
    public Association addServerAssociation(String peerAddress, int peerPort, String
serverName, String assocName, IpChannelType ipChannelType)
            throws Exception;

    /**
     * Add Association. IP channel type is SCTP.
     *
     * @param hostAddress
     * @param hostPort
```

```
     *      If hostPort==0 this mean the local port will be any vacant port
     * @param peerAddress
     * @param peerPort
     * @param assocName
     * @return
     * @throws Exception
     */
    public Association addAssociation(String hostAddress, int hostPort, String
peerAddress, int peerPort, String assocName)
            throws Exception;

    /**
     * Add Association
     *
     * @param hostAddress
     * @param hostPort
     *      If hostPort==0 this mean the local port will be any vacant port
     * @param peerAddress
     * @param peerPort
     * @param assocName
     * @param ipChannelType
     *            IP channel type: SCTP or TCP
     * @param extraHostAddresses
     *            When SCTP multi-homing configuration extra IP addresses can be put
here
     *            If multi-homing absence this parameter can be null
     * @return
     * @throws Exception
     */
    public Association addAssociation(String hostAddress, int hostPort, String
peerAddress, int peerPort, String assocName, IpChannelType ipChannelType,
            String[] extraHostAddresses) throws Exception;

    /**
     * Remove existing Association. Association should be stopped before
     * removing
     *
     * @param assocName
     * @throws Exception
     */
    public void removeAssociation(String assocName) throws Exception;

    /**
     * Get existing Association for passed name
     *
     * @param assocName
     * @return
     * @throws Exception
     */
    public Association getAssociation(String assocName) throws Exception;
```

```
    /**
     * Get configured Association map with name as key and Association instance
     * as value
     *
     * @return
     */
    public Map<String, Association> getAssociations();

    /**
     * Start the existing Association
     *
     * @param assocName
     * @throws Exception
     */
    public void startAssociation(String assocName) throws Exception;

    /**
     * Stop the existing Association
     *
     * @param assocName
     * @throws Exception
     */
    public void stopAssociation(String assocName) throws Exception;

    /**
     * Get connection delay. If the client side {@link Association} dies due to
     * network failure or any other reason, it should attempt to reconnect after
     * connectDelay interval
     *
     * @return
     */
    public int getConnectDelay();

    /**
     * Set the connection delay for client side {@link Association}
     *
     * @param connectDelay
     */
    public void setConnectDelay(int connectDelay) throws Exception;

    /**
     * This method is not used more.
     *
     * @return
     */
    public int getWorkerThreads();

    /**
     * This method is not used more.
     *
     * @param workerThreads
```

```
     */
    public void setWorkerThreads(int workerThreads) throws Exception;

    /**
     * This method is not used more.
     *
     * @return
     */
    public boolean isSingleThread();

    /**
     * This method is not used more.
     *
     * @param singleThread
     */
    public void setSingleThread(boolean singleThread) throws Exception;

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3.
     * Threshold 1.
     *
     * @return
     */
    public double getCongControl_DelayThreshold_1();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3.
     * Threshold 2.
     *
     * @return
     */
    public double getCongControl_DelayThreshold_2();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3.
     * Threshold 3.
```

```
     *
     * @return
     */
    public double getCongControl_DelayThreshold_3();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3. Array must have 3 values.
     * Threshold 1.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_DelayThreshold_1(double val) throws Exception;

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3. Array must have 3 values.
     * Threshold 2.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_DelayThreshold_2(double val) throws Exception;

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
more then value 0, 1 or 2 of the array
     * CongControl_DelayThreshold, the Association's congestion level becomes to 1, 2
or 3. Array must have 3 values.
     * Threshold 3.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_DelayThreshold_3(double val) throws Exception;

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
```

```
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2.
     * Threshold 1.
     *
     * @return
     */
    public double getCongControl_BackToNormalDelayThreshold_1();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2.
     * Threshold 2.
     *
     * @return
     */
    public double getCongControl_BackToNormalDelayThreshold_2();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2.
     * Threshold 3.
     *
     * @return
     */
    public double getCongControl_BackToNormalDelayThreshold_3();

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2. Array must have 3
     * values.
     * Threshold 1.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_BackToNormalDelayThreshold_1(double val) throws
Exception;
```

```
    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2. Array must have 3
     * values.
     * Threshold 2.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_BackToNormalDelayThreshold_2(double val) throws
Exception;

    /**
     * For outgoing messages congestion control we need to have 3 thresholds - delays
of outgoing messages before it will be
     * sent to IP channel (3 levels - 1, 2, 3). If a delay time in seconds becomes
less then value 0, 1 or 2 of the array
     * CongControl_BackToNormalDelayThreshold, the Association's congestion level
reduces to 0, 1 or 2. Array must have 3
     * values.
     * Threshold 3.
     *
     * @param val
     * @throws Exception
     */
    public void setCongControl_BackToNormalDelayThreshold_3(double val) throws
Exception;

    /**
     * SCTP option: Enables or disables message fragmentation.
     * If enabled no SCTP message fragmentation will be performed.
     * Instead if a message being sent exceeds the current PMTU size,
     * the message will NOT be sent and an error will be indicated to the user.
     *
     * @return
     */
    public Boolean getOptionSctpDisableFragments();

    /**
     * SCTP option: Enables or disables message fragmentation.
     * If enabled no SCTP message fragmentation will be performed.
     * Instead if a message being sent exceeds the current PMTU size,
     * the message will NOT be sent and an error will be indicated to the user.
     *
     * @param optionSctpDisableFragments
     */
    public void setOptionSctpDisableFragments(Boolean optionSctpDisableFragments);
```

```
    /**
     * SCTP option: Fragmented interleave controls how the presentation of messages
occur for the message receiver.
     * There are three levels of fragment interleave defined
     * level 0 - Prevents the interleaving of any messages
     * level 1 - Allows interleaving of messages that are from different associations
     * level 2 - Allows complete interleaving of messages.
     *
     * @return
     */
    public Integer getOptionSctpFragmentInterleave();

    /**
     * SCTP option: Fragmented interleave controls how the presentation of messages
occur for the message receiver.
     * There are three levels of fragment interleave defined
     * level 0 - Prevents the interleaving of any messages
     * level 1 - Allows interleaving of messages that are from different associations
     * level 2 - Allows complete interleaving of messages.
     *
     * @param optionSctpFragmentInterleave
     */
    public void setOptionSctpFragmentInterleave(Integer optionSctpFragmentInterleave);

    /**
     * SCTP option: The maximum number of streams requested by the local endpoint
during association initialization
     * For an SctpServerChannel this option determines the maximum number of outbound
streams
     * accepted sockets will negotiate with their connecting peer.
     *
     * @return
     */
    public Integer getOptionSctpInitMaxstreams_MaxOutStreams();

    /**
     * SCTP option: The maximum number of streams requested by the local endpoint
during association initialization
     * For an SctpServerChannel this option determines the maximum number of inbound
streams
     * accepted sockets will negotiate with their connecting peer.
     *
     * @return
     */
    public Integer getOptionSctpInitMaxstreams_MaxInStreams();

    /**
     * SCTP option: The maximum number of streams requested by the local endpoint
during association initialization
     * For an SctpServerChannel this option determines the maximum number of outbound
```

```
streams
     * accepted sockets will negotiate with their connecting peer.
     */
    public void setOptionSctpInitMaxstreams_MaxOutStreams(Integer maxOutStreams);

    /**
     * SCTP option: The maximum number of streams requested by the local endpoint
during association initialization
     * For an SctpServerChannel this option determines the maximum number of inbound
streams
     * accepted sockets will negotiate with their connecting peer.
     */
    public void setOptionSctpInitMaxstreams_MaxInStreams(Integer maxInStreams);

    /**
     * SCTP option: Enables or disables a Nagle-like algorithm.
     * The value of this socket option is a Boolean that represents whether the option
is enabled or disabled.
     * SCTP uses an algorithm like The Nagle Algorithm to coalesce short segments and
improve network efficiency.
     *
     * @return
     */
    public Boolean getOptionSctpNodelay();

    /**
     * SCTP option: Enables or disables a Nagle-like algorithm.
     * The value of this socket option is a Boolean that represents whether the option
is enabled or disabled.
     * SCTP uses an algorithm like The Nagle Algorithm to coalesce short segments and
improve network efficiency.
     *
     * @param optionSctpNodelay
     */
    public void setOptionSctpNodelay(Boolean optionSctpNodelay);

    /**
     * SCTP option: The size of the socket send buffer.
     *
     * @return
     */
    public Integer getOptionSoSndbuf();

    /**
     * SCTP option: The size of the socket send buffer.
     *
     * @param optionSoSndbuf
     */
    public void setOptionSoSndbuf(Integer optionSoSndbuf);

    /**
```

```
     * SCTP option: The size of the socket receive buffer.
     *
     * @return
     */
    public Integer getOptionSoRcvbuf();

    /**
     * SCTP option: The size of the socket receive buffer.
     *
     * @param optionSoRcvbuf
     */
    public void setOptionSoRcvbuf(Integer optionSoRcvbuf);

    /**
     * SCTP option: Linger on close if data is present.
     * The value of this socket option is an Integer that controls the action taken
when unsent data is queued on the socket
     * and a method to close the socket is invoked.
     * If the value of the socket option is zero or greater, then it represents a
timeout value, in seconds, known as the linger interval.
     * The linger interval is the timeout for the close method to block while the
operating system attempts to transmit the unsent data
     * or it decides that it is unable to transmit the data.
     * If the value of the socket option is less than zero then the option is
disabled.
     * In that case the close method does not wait until unsent data is transmitted;
     * if possible the operating system will transmit any unsent data before the
connection is closed.
     *
     * @return
     */
    public Integer getOptionSoLinger();

    /**
     * SCTP option: Linger on close if data is present.
     * The value of this socket option is an Integer that controls the action taken
when unsent data is queued on the socket
     * and a method to close the socket is invoked.
     * If the value of the socket option is zero or greater, then it represents a
timeout value, in seconds, known as the linger interval.
     * The linger interval is the timeout for the close method to block while the
operating system attempts to transmit the unsent data
     * or it decides that it is unable to transmit the data.
     * If the value of the socket option is less than zero then the option is
disabled.
     * In that case the close method does not wait until unsent data is transmitted;
     * if possible the operating system will transmit any unsent data before the
connection is closed.
     *
     * @param optionSoLinger
     */
```

```
    public void setOptionSoLinger(Integer optionSoLinger);

}
```

Management API is divided into two sections 1) managing the resources and 2) configuring management

API's to manage resources

`public void addManagementEventListener(ManagementEventListener listener)`

Adding a listener for management events (adding/removing servers and associations).

`public void removeManagementEventListener(ManagementEventListener listener)`

Removing a listener for management events (adding/removing servers and associations).

`public Association addAssociation(String hostAddress, int hostPort, String peerAddress, int peerPort, String assocName, IpChannelType ipChannelType, String[] extraHostAddresses)`

Add's a new client side association to the management. The underlying protocol (SCTP or TCP) depends on IpChannelType passed. Association when started will create underlying SCTP/TCP socket that will bind to hostAddress:hostPort and tries to connect to peerAddress:peerPort. Each association is identified by unique name. The connection attempt be will made after every `connectDelay` milliseconds till the connection is successfully created. If SCTP socket is being created, extraHostAddresses can be passed for multi-home machines. SCTP Socket will bind to "hostAddress" as primary address and use "extraHostAddresses" as fall-back in case if primary network goes down. Appropriate Exception's are thrown if other association with same name already exist or if other association is already bound to same hostAddress:hostPort or other association is already configured to connect to same peerAddress:peerPort.

`public Association addServerAssociation(String peerAddress, int peerPort, String serverName, String assocName, IpChannelType ipChannelType)`

Add's a new server side association to the management. A server by name `serverName` should already have been added to the management before adding server side association. Only Association from peerAddress:peerPort will be accepted by underlying server socket. If connection request is coming from any other ip:port combination it's gracefully closed and error message is logged. If connect request comes for configured peerAddress:peerPort, but underlying association is not started, it's gracefully closed and error message is logged. The IpChannelType should match with that configured for server. Appropriate Exception's are thrown if other association with same name already exist or if other association is already configured to receive connection request from same peerAddress:peerPort.

`public Server addServer(String serverName, String hostAddress, int port, IpChannelType ipChannelType, String[] extraHostAddresses)`

Add's a new server to the management. Server will be bound to hostAddress:port when started. Type of underlying protocol (SCTP/TCP) depends on IpChannelType passed If SCTP server socket is being created, extraHostAddresses can be passed for multi-home machines. SCTP Socket will bind to "hostAddress" as primary address and use "extraHostAddresses" as fall-back in case if primary network goes down. Each server is identified by unique name. Appropriate Exception's are thrown if other server with same name already exist or if other server is already configured to bind to same hostAddress:port

`public void startAssociation(String assocName)`

Start's the association with name `assocName`. `AssociationListener` should be set before starting this association Appropriate Exception's are thrown if there is no association with given name or if association with given name is found but is already started.

`public void startServer(String serverName)`

Start's the server with name `serverName`. Appropriate Exception is thrown if there is no server with given name or if server with given name is found but is already started.

`public void stopAssociation(String assocName)`

stop's the association with name `assocName`. The underlying socket is closed. Appropriate Exception is thrown if there is no association with given name.

`public void stopServer(String serverName)`

stop's the server with name `serverName`. Appropriate Exception is thrown if there is no server with given name. Throws exception if the server is found for given name but there are association's for this server which are still in "started" state.

`public void removeAssociation(String assocName)`

Removes the association with name `assocName`. Appropriate Exception is thrown if there is no association with given name. Throws exception if association is found with given name but is started.

`public void removeServer(String serverName)`

Removes the server with name `serverName`. Appropriate Exception is thrown if there is no server with given name. Throws exception if server is found with given name but is started.

`public Association getAssociation(String assocName)`

Returns the association with name `assocName`. Appropriate Exception is thrown if there is no Association with given name.

`public Map<String, Association> getAssociations()`

Returns the unmodifiable Map of association. Key is association name and value is association instance

`public List<Server> getServers()`

Returns the unmodifiable list of servers.

`public void removeAllResourses()`

This method stops and removes all registered servers and associations. Management should be started before this operation can be called. Use this method only for test purposes or after total crashes.

Configuration

`setPersistDir`

Management when started looks for file XXX_sctp.xml containing serialized state of underlying association and server. Set the directory path to direct Management to look at specified directory for underlying serialized file. If directory path is not set, Management searches for system property sctp.persist.dir to get the path for directory. Even if `sctp.persist.dir` system property is

not set, Management will look at System set property `user.dir`

`setConnectDelay`
: Time in milli seconds that underlying SCTP socket will wait before attempting to connect to peer. This is only applivable for clien side sockets. This parameter can be updated only at the SCTP stack running time, including GUI.

`congControl_DelayThreshold_1`, `congControl_DelayThreshold_2`, `congControl_DelayThreshold_3`
: Delay time in seconds between a time when an outgoing message has been submitted for sending to a IP peer and time when the message has been sent to IP network. The more this time the more panding messages are in an outgoing buffer and the more is IP network congestion. These parameters are thresholds that trigger Association congestion level to the next level. This parameter can be updated only at the SCTP stack running time, including GUI.

`congControl_BackToNormalDelayThreshold_1`, `congControl_BackToNormalDelayThreshold_2`, `congControl_BackToNormalDelayThreshold_3`
: These parameters are thresholds that trigger Association congestion level back the previous level. This parameter can be updated only at the SCTP stack running time, including GUI.

`optionSctpDisableFragments` SCTP stack level option: Enables or disables message fragmentation.

`optionSctpFragmentInterleave` SCTP stack level option: Fragmented interleave controls how the presentation of messages occur for the message receiver.

There are three levels of fragment interleave defined:

- level 0 - Prevents the interleaving of any messages
- level 1 - Allows interleaving of messages that are from different associations
- level 2 - Allows complete interleaving of messages.

`optionSctpInitMaxstreams_MaxOutStreams` SCTP stack level option: The maximum number of outbound streams requested by the local endpoint during association initialization

`optionSctpInitMaxstreams_MaxInStreams` SCTP stack level option: The maximum number of inbound streams requested by the local endpoint during association initialization

`optionSctpNodelay` SCTP stack level option: Enables or disables a Nagle-like algorithm (true means disabling).

`optionSoSndbuf` SCTP stack level option: The size of the socket send buffer.

`optionSoRcvbuf` SCTP stack level option: The size of the socket receive buffer.

`optionSoLinger` SCTP stack level option: Linger on close if data is present.