DMIS DISTRIBUTED SERVICE ARCHITECTURE

DMIS Core Framework v3.0.1

DMIS
Application Server Reference

July 2005

Revision History

Date	Version	Author	Remarks
Dec 2004	1.0.1	J. Kessler	Initial draft
Jan 2005	1.0.2	J. Kessler	Initial version
Feb 2005	1.0.3	J. Kessler	Corrections and additions to original draft
April 2005	1.0.4	J. Kessler	Adjusted layout of document set
July 2005	1.1.1	J. Kessler	Updated, added UML diagrams as Appendices

Table Of Contents

1. Introduction

1.1. Purpose

This document is an overview of the DMIS Application Server as it relates to the Distributed Services architecture.

1.2. Scope

Areas covered include the functionality and configuration of the DMIS Application Server.

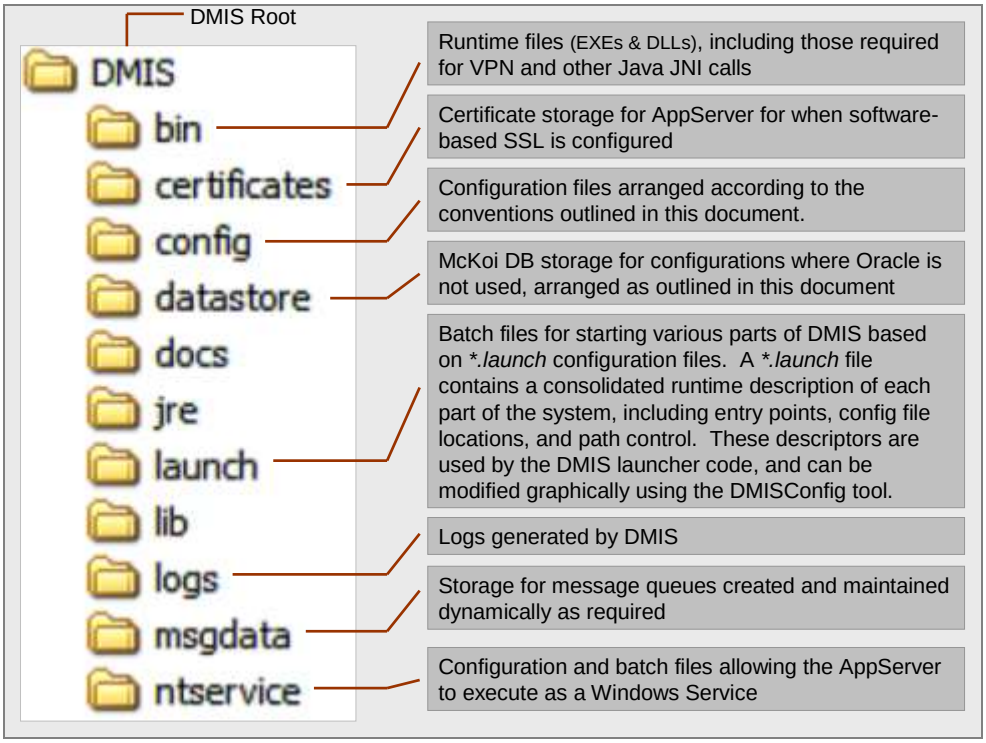
1.3. Audience

Audience is anyone on the DMIS team who is interested in designing, developing, or deploying DMIS services and applications.

2. Application Server Overview

2.1. Directory Structure

This version of the DMIS middleware uses some constructs that are new, particularly in the area of configuration management. The following diagram summarizes the various parts of the modern DMIS directory structure.



Application Server Directory Structure

Details on each of these child areas, as well as differences with respect to previous versions of DMIS, are discussed in this document.

2.2. Configuration

Much of the operation of the AppServer is determined by the way it is configured. The Distributed AppServer offers many new configuration and deployment features not found in previous incarnations.

2.2.1. Configuration Profiles

It is easy to establish configuration *profiles* for use in various scenarios. It may be useful, for example, to have a different configuration set for development, production, and testing purposes and have them coexist while being easily able to switch between them. In the directory structure, each profile appears as a subdirectory under the *config/* root.

Profiles can be selected on the command line (using the *–profile* switch combined with the name of the profile such as **"-profile test"**) when starting any part of DMIS. The *DefaultProfile.list* file can also be used to control the default resolution sequence when multiple profiles are installed in the *config/* area. This resolution data is used when the *–profile* switch is not otherwise specified and, in some cases, can provide for dynamic resolution that is very convenient.

```
development
production
test
```

Sample DefaultProfile.list file

This example will cause DMIS to look for profiles in the following order: *development*, *production*, and then *test*. The first one for which a profile exists (a child directory under *config/*) is used as the default profile. In most cases, the existing DefaultProfile.list works very well – the *development* profile is used in most cases and *production* is used when there is not development configuration. In other special situations, the default resolution can be easily changed.

Notice that the sample *DefaultProfile.list* file does not list the *shared* profile in its resolution order. The *shared* profile is a special area that is used to hold config files common to all other profiles, and is not normally referenced directly when launching DMIS. The DMIS ConfigurationManager module, used to programmatically access configuration files, handles all of the appropriate resolution of file locations and presents a simple way to access the files appropriate for the context that is executing.

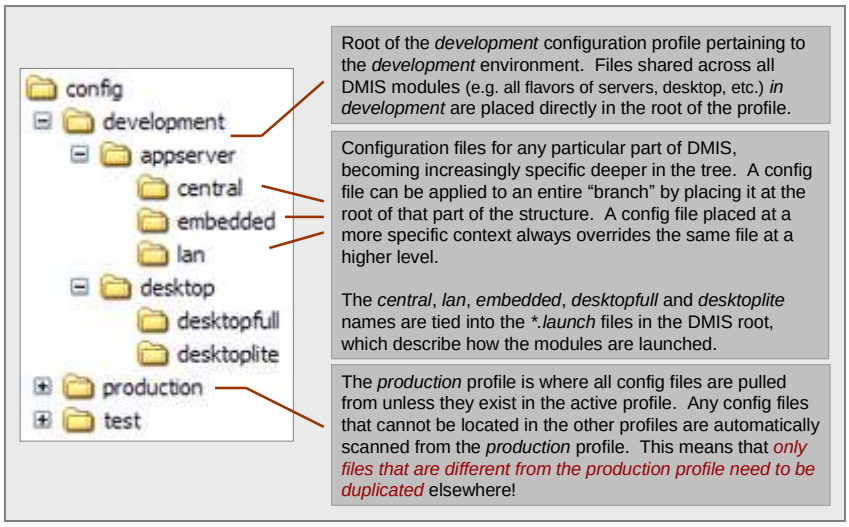
When the profile cannot be determined at runtime, either because *–profile* was not specified or no default profiles can be resolved, a runtime error occurs and an appropriate message is produced on the console.

2.2.2. Configuration Path Layout

2.2.2.1. Directory Structure

Within a given configuration profile, configuration files follow a predictable layout that resembles a Java package hierarchy. There are areas for each of the major DMIS components such as the DMIS Desktop, the CENTRAL AppServer, LAN Servers, and Embedded Servers.

The goal of the configuration structure is to eliminate or reduce unnecessary redundancy, and thus help keep files consistent across various deployment environments.

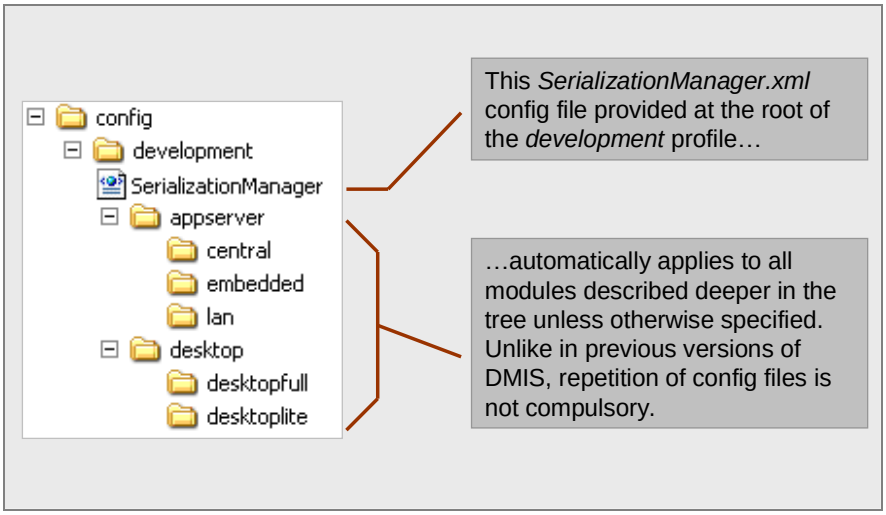


Configuration profile directory structure

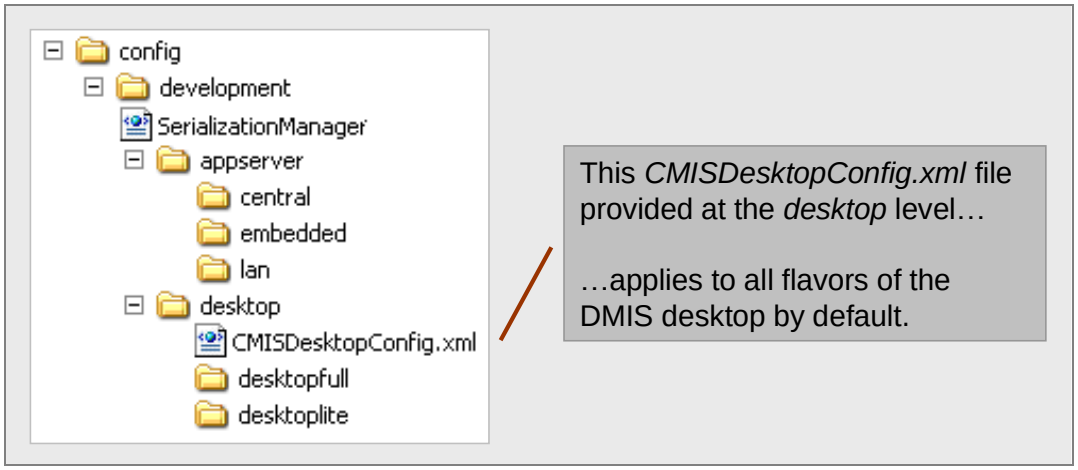
This arrangement allows for excellent flexibility without the need for undue redundancy in configuration files. All of the code in DMIS has been enhanced, working in concert to take advantage of the capabilities this scheme has to offer.

2.2.2.2. Configuration File Placement

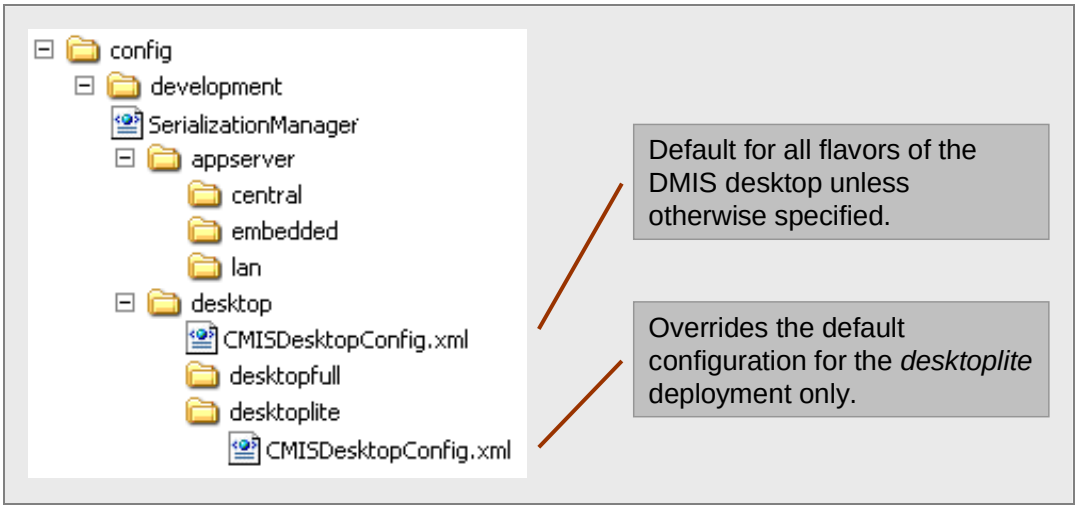
The configuration structure is arranged much like a Java package hierarchy – it starts broader and becomes increasingly specific as you drill down. A key feature exists to combat redundancy: *a config file at any level automatically applies to all levels underneath as well*. Therefore, a file that is almost always the same for all parts of DMIS (such as *SerializationManager.xml*) could be placed at the highest level in the profile so it applies across the board



The pattern continues all the way down the configuration hierarchy such that any given config file can be applied to any portion of DMIS.



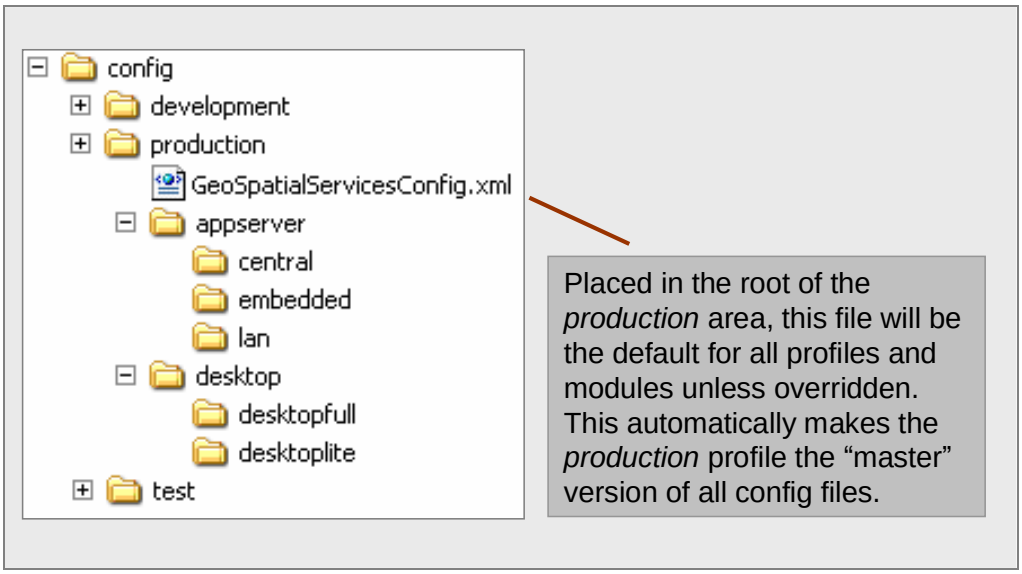
Overriding a configuration file for a specific part of the hierarchy is a simple matter of placing the desired version at the correct location in the tree.



2.2.2.3. Everything relative to the *'production'* Profile

The structure presented so far eases redundancy for files *within the same profile*. But DMIS history has shown that many files are actually the same across profiles as well (such as between production and development). Managing these common files across a number of profiles – even when redundancy *within each profile* is low – is unwieldy.

The *production* profile is a special profile that may contain all config files common to all profiles across the board. Assume, for example that the *GeoSpatialServicesConfig.xml* file is always the same across every conceivable situation:



The structure of the *production* profile is the same as the structure for the other profiles, and resolution is applied in the same fashion. Assume that the DMIS Central AppServer running under the *development* profile is looking for the *SerializationManager..xml* config file at runtime. The correct file is located by examining the following locations (in order):

First the development profile is scanned:
config/development/appserver/central
config/development/appserver
config/development

Then the 'production' area is scanned:
config/production/appserver/central
config/production/appserver
config/production/

Within the *shared* profile, any file can be overridden using the same approach outlined above for other profiles.

2.2.2.4. Determining the Configuration File Set for Deployment

For deployment purposes, or when only a subset of files is to be used, it is useful to determine the set of config files that are required for any given environment. Given the name of a profile (e.g. *production*, *development*, or *test*), and a module – in this example the Central AppServer - all related config files exist in these locations relative to the DMIS root:

config/production/appserver/central
config/production/appserver
config/production/

config/[profilename]/appserver/central

config/[profilename]/appserver
config/[profilename]

2.2.2.5. *.launch Files

A **.launch* file is a descriptor that contains all of the information required to launch a given part of DMIS. This information includes entry points, home directories, and configuration file locations.

```
<context>

   <name>Central Server</name>

   <module>
      <name>Central Server</name>
      <type>server</type>
      <entrypoint>org.dmis.appserver.ApplicationServerEntryPoint</entrypoint>
      <home>{$cfgloc}</home>

      <cfg>
         <base>file:config/{$profile(.)}</base>
         <aux>file:central/</aux>
         <output>file:central/</output>
      </cfg>
   </module>

</context>
```

Sample *.launch file

Notice that the format allows the use of symbols in order to maximize the flexibility of the format. There are a variety of symbols that can be used, but as they appear in this example:

- The **{$cfgloc}** symbol refers to the location of the **.launch* file itself in lieu of a hard-coded path or structure. In the context of this example, the "home path" (a.k.a. the working directory) for the module will be wherever this config file is located (usually in the DMIS root). In other situations, a different value might be used as appropriate. In either case, it is all consolidated within this one **.launch* file.

- The **{$parm.profile.path}** variable refers to the name (in the form of a relative path) of the profile that was specified on the command line (or resolved automatically through the *DefaultProfile.list* file).

Notice the relatively sophisticated and flexible manner in which the configuration path structure is declared. The structure itself actually somewhat arbitrary, and can include more levels than are illustrated here, but is defined here based on to the convention used for DMIS. The base configuration directory is declared as **file:config/{$parm.profile.path}**, which allows the profile-based **config/*profile*/** directory structure (e.g. *config/development*) to be used by the launcher code.

For this sample module, which is based on the one used by the CENTRAL AppServer, an auxiliary config path **file:central/** is defined as well. This causes configuration files to be pulled

from *config/development/central/* whenever possible, and *config/development* when a given file cannot otherwise be located. This is how configuration files can be shared across profiles (such as having a common *LoggerConfig.xml* for the desktop and the AppServers).

Note that **.launch* files can are human-readable but they can be edited graphically using the DMISConfig utility, which is described in this document.

2.2.2.6. Predefined *.launch Files

A series of predefined **.launch* files are provided in the launch/ subdirectory that describe the parameters of commonly invoked parts of DMIS:

- **LaunchCentralServer.launch** – The CENTRAL AppServer.
- **LaunchDesktopFull.launch** – Describes the DMIS desktop with full offline support. The offline support is accomplished through the use of an EMBEDDED AppServer, which is also configured and launched as appropriate.
- **LaunchDesktopLite.launch** – Describes the DMIS desktop, but only the GUI portion with no offline support (e.g. no EMBEDDED AppServer).
- **LaunchLANServer.launch** – A LAN AppServer.
- **LaunchServerDiag.launch** – The Server Diagnostic Console.
- **LaunchLogViewer.launch -** The Log Viewer utility.
- **LaunchActivationWizard.launch** – The application server Activation Wizard.
- **LaunchDMISConfig.launch** – Describes the DMISConfig utility, which can be used to indirectly edit **.launch* files and graphically modify other configuration settings.

Each of these can be used by feeding them into the Launcher class described next.

2.2.2.7. Launchers

The **.launch* files, which contain entry point and configuration information, are read by a *launcher class* when a part of DMIS is executed. The *org.dmis.launcher.Launcher* class is responsible for reading the given **.launch* file and starting DMIS using the correct parameters.

The launcher honors all of the configuration options (such as *–profile* for selecting a particular environment) and conventions described in this document.

```
C:\DMIS>java org.dmis.launcher.Launcher DesktopFull.launch –profile
test
```

2.2.3. AppServer Configuration Files

2.2.3.1. ApplicationServer.xml

The ApplicationServer.xml file is the primary configuration set for the DMIS application server. It contains basic identity information for the AppServer entity, authentication and message queuing parameters, and lists the modules to be run within the container.

```
<?xml version="1.0"?>
<applicationserver>
   <general>
   </general>

   <authenticator>classname</authenticator>

   <messaging>
      <queueparms>parms</queueparms>

      <client validversion=versionid1/>
      <client validversion=versionid2/>
      <client validversion=versionid3/>
   </messaging>

   <module name="Module 1" class="module1.classname"/>
   <module name="Module 2" class="module2.classname"/>
   <module name="Module 3" class="module3.classname">
      <parm name="name1" value="value1"/>
      <parm name="name2" value="value2/"/>
   </module>
</applicationserver>
```

Layout of ApplicationServer.xml configuration file

Each section has a particular meaning, and specific elements, as described in this section.

2.2.3.1.1. <general>

The <*general*> section contains settings that describe the AppServer at the highest level. It includes the <*context*> value to describe the role that the AppServer is playing in the hierarchy (EMBEDDED, INTERMEDIATE, or CENTRAL). It also contains the <*org*> element describing the organization responsible for the AppServer. In DMIS, this is usually the CogID of the responder organization.

All of these values form part of the RoutingID for the AppServer, which is used by the messaging subsystem to refer to this entity.

Important Note: *For security purposes, an AppServer must be activated before it can connect to any other part of the DMIS network. Activation is a process by which the DMIS network can validate that a given installation really deserves its identity. This prevents a rogue AppServer from "spoofing" another legitimately authorized entity. The Activation Wizard, which requires COG Administrator privileges to complete successfully, is used to active an AppServer.*

2.2.3.1.2. <authenticator>

The DMIS AppServer is required to authenticate incoming requests for various pieces of functionality. The <*authenticator*> elements informs the AppServer which class can be used for authentication purposes. The authenticator class is arbitrary but must implement the following interface:

```
interface org.dmis.appserver.auth.IAuthenticator {
   Object onAuthenticate(IServiceHelper sh,
                         String userId,
                         String password) throws Exception;
}
```

The IAuthenticator implementation is given credentials from the client (or another appropriate entity), and an IServiceHelper instance for access to AppServer functions. The implementation can return any related information (such as a peripheral data) as an object instance.

An authenticator for DMIS has been created and is configured as the default. This DMIS-specific class interprets the *userId* as *cogId/loginName*, and the password as a hashed value (so that plain text is never used for password transmissions). It returns a CMISOperator object as its result, which contains (among other things) the role descriptions and permission set for operator in question.

2.2.3.1.3. <messaging>

The <*messaging*> section contains basic configuration settings for the messaging subsystem, such as where messages will be stored when queued in a persisted state.

The <*queueparms*> value is used to pass parameters directly to the queuing implementation, using options that are described elsewhere in this document.

A set of <*client*> elements is used to list the client versions supported by this AppServer. Version numbers are validated as each client connects to the AppServer, and any clients whose version number does not appear in this list will be rejected. In DMIS, this rejection causes the DMIS Desktop to present the update web site where the operator can download an update.

2.2.3.1.4. <module>

The AppServer is essentially a message routing hub with a set of modules configured to run on top of it. Those modules provide the higher-level functionality normally associated with an AppServer, such as service management and clustering.

A <*module*> entry is used to install a given module on the AppServer. At a minimum, each module has a display name and the name of the class that comprises the module. Module implementations always implement the *IApplicationServerModule* interface, which is described in detail in this document. Modules can also be given parameters using the <*parm*> element, which lists a named parameter and an associated value as recognized by the module.

2.2.3.1.5. Sample ApplicationServer.xml Files

The following example illustrates how the ApplicationServer.xml file may appear for a CENTRAL AppServer installation.

```
<?xml version="1.0"?>
<applicationserver>
   <general>
      <context>CENTRAL</context>
      <cogid>0</cogid>
   </general>

   <authenticator>
      org.dmis.appserver.auth.DMISAuthenticator
   </authenticator>

   <messaging>
      <queueparms>path=msgqueue\{$cfgname}\</queueparms>

      <client validversion="loadsim10"/>
   </messaging>

   <module name="Cluster Manager"
           class="org.dmis.appserver.cluster.ClusterManager"/>

   <module name="Service Manager"
           class="org.dmis.appserver.servicemanager.ServiceManager"/>

   <module name="ThinRPC Module"
           class="org.dmis.appserver.module.thinrpc.ThinRPCModule">

      <parm name="port" value="3102"/>
   </module>

   <module name="Diagnostic Module"
           class="org.dmis.appserver.module.diag.DiagnosticModule"/>

   <module name="Hang Detection"
           class="org.dmis.appserver.module.hangdetect.HangDetectionModule"/>

   <module name="HTTP Server"
           class="org.dmis.appserver.module.http.HTTPServerModule">

      <parm name="port" value="8001"/>
      <parm name="root" value="webroot/"/>
   </module>
</applicationserver>
```

Notice that the **<context>** element is set to *CENTRAL*, and a number of modules are configured (including the Cluster Manager). An EMBEDDED server uses the same configuration format but has a much more limited set of features, as shown here:

```
<?xml version="1.0"?>
<applicationserver>
   <general>
      <context>EMBEDDED</context>
      <cogid>0</cogid>
   </general>

   <authenticator>
```

```
      org.dmis.appserver.auth.DMISAuthenticator
   </authenticator>

   <messaging>
      <queueparms>path=msgqueue\{$cfgname}\</queueparms>

      <client validversion="loadsim10"/>
   </messaging>

   <module name="Service Manager"
           class="org.dmis.appserver.servicemanager.ServiceManager"/>
</applicationserver>
```

Similarly, a LAN-based server has settings appropriate for an *INTERMEDIATE* server configuration.

2.3. Application Server Modules

2.3.1. Service Manager Module

2.3.1.1. Overview

Although the heart of a DMIS AppServer is the messaging subsystem, the Service Manager is perhaps the most visible component and the one developers deal with the most frequently. It is the container in which DMIS services are directly loaded, executed, and managed.

A service is a class that implements either the `IMessageService` or `IBackgroundService` interface, and is installed by posting a **.svc* deployment descriptor in the configuration directory. The Service Manager parses these descriptors, launches the appropriate services, and routes messages to them for processing as they arrive.

2.3.1.2. ServiceManager.xml

The *ServiceManager.xml* configuration file describes the core parameters for operation of the service container. This includes values related to message dispatching, error handling, and debugging support.

***Note**: The* ServiceManager.xml *file is <u>not</u> is modern parallel to the* RemoteServiceManagerConfig.xml *file used in previous versions of DMIS.* ServiceManager.xml *contains only the subset of that information required for container operation; settings related to specific services are contained in* *.svc *deployment descriptors.*

2.3.2. Clustering Module

2.3.2.1. Description

The *Clustering Module* allows multiple AppServers to be grouped together and, to a certain degree, act as a single logical unit. Clustering can offer many benefits including:

- Fail over in case any one member of the cluster goes offline,
- Scalability as processing can be divided across multiple machines

The DMIS clustering implementation offers additional benefits as well:

- Enhanced flexibility by allowing members of a cluster to be physically distant, even on distinct physical networks normally not visible to one another,
- Automatic direction of clustering-related traffic through the best (or any available) route,
- Shared memory and synchronization interfaces allowing clustered services to behave in a unified fashion regardless of network topology,
- Simple configuration and intuitive developer interfaces

A given server can participate in a cluster only if the *Cluster Manager* module is running. If this module is not installed, the AppServer will run in a stand-alone capacity regardless of the configuration of other surrounding AppServers.

2.3.2.2. Features

The Cluster Manager provides a host of services to DMIS Service Developers other module designers.

- Intercommunications between members of the cluster using high-priority features of the messaging subsystem,
- Operator tracking at a cluster level, making it possible to know who is connected to the cluster regardless of physical server relationships,
- Shared memory constructs such as Lists, Maps, and other object instances that allow service instances running on different members of the cluster to directly coordinate,
- Shared semaphore constructs that allows members of a cluster to synchronize ciritical pieces of logic across servers

When clustering is not enabled for a given AppServer, all of these features continue to operate, but as though the cluster has but a single member.

2.3.2.3. Configuration

The Cluster Manager does not require any configuration apart from installing the module in the *ApplicationServer.xml* file. The appropriate configuration entry is:

```
<module name="Cluster Manager"
        class="org.dmis.appserver.cluster.ClusterManager"/>
```

No other configuration settings are currently required since the Cluster Manager builds upon other existing AppServer facilities (in particular, the messaging subsystem) that are configured elsewhere.

2.3.2.4. Connectivity Among Cluster Members

In addition to running Cluster Manager on all of the AppServers to be clustered, the members must be visible to one another on the DMIS network so that information can flow between them. This is usually accomplished by configuring the *MessageRoutingHub.xml* to create direct physical connections between servers, though it is possible to cluster machines that are linked only indirectly.

Members of a cluster are connected through the same mechanisms as the rest of the DMIS network. As such, they receive the same benefits such as location independence and intelligent routing based one logical names. These features allow a great degree of flexibility regarding the physical layout of a DMIS AppServer cluster.

The Cluster Manager module detects the presence of peers by inspecting the *routing table*, described fully in this document, for entities that share RoutingIDs that match specific pattern to reveal a logical relationship.

2.3.3. ThinRPC Module

The ThinRPC Module provides a small, scalable method for clients to invoke functions on the AppServer. This is an alternative to the *CMISClientGateway* interface, which is stateful and designed for larger client applications (such as the DMIS desktop). The ThinRPC module can be installed by including the following entry in the *ApplicationServer.xml* config file:

```
<module name="ThinRPC Module"
        class="org.dmis.appserver.module.thinrpc.ThinRPCModule"/>
```

The ThinRPC interface is well-suited to web-based applications where many requests may be received in quick succession. The following class:

```
org.dmis.appserver.client.thinrpc.ThinRPCClient
```

can be used to easily perform authentication and send messages to an AppServer. The following code demonstrates how to use this facility.

```
CorrelatedResponseMsg invokeRemoteFunction(CoreContext coreCtx,
                                           String username,
                                           String password)
                              throws MessagingException {

  // Get a ThinRPCClient instance.
  ThinRPCClient rpcClient =
        new ThinRPCClient(coreCtx, "hostname", 3101)

  // Authenticate the given user.
  rpcClient.authenticate(username, password);

  // Create a message to be sent to the server.
  CorrelatedRequestMsg myRequest = new TIECorrelatedRequestMsg();
  myRequest.setContent("retrieveall");
```

```
    myRequest.setAttributeLong("cogId", cogId);
    myRequest.setAttributeBoolean("postedOnly", true);

    // Send the message and wait for the result before returning.
    return rpcClient.sendMessageAndWait(myRequest);
}
```

In the context of DMIS, the *username* is specified in **cog/loginname** format, and the *password* is a hashed value.

Notice that the send-and-wait paradigm is consistent with the same interfaces of the *CMISClientGateway* class. Instances of ThingRPCClient can be cached and reused. The user's credentials are revalidated at each method invocation.

2.3.4. Diagnostic Module

The Diagnostic Module enables metrics to be obtained in real time for the AppServer. This module must be running on order to use the ServerDiag utility against the server. It is enabled by including the following entry into the *ApplicationServer.xml* config file:

```
<module name="Diagnostic Module"
        class="org.dmis.appserver.module.diag.DiagnosticModule"/>
```

When the HTTP Server module is also installed, the Diagnostic Module can produce output in crude HTML form.

2.3.5. HTTP Server Module

The HTTP Server module is a very crude web server that is a byproduct of RPC research performed while developing the AppServer architecture. It is included since it is functional in spite of being quite limited. In this incarnation the web server handles GET requests, passes them to various others AppServer modules for interpretation, or provides a default file serving behavior is necessary. The appropriate configuration entry to enable this module is:

```
<module name="HTTP Server"
        class="org.dmis.appserver.module.http.HTTPServerModule">
   <parm name="port" value="port"/>
   <parm name="root" value="rootpath"/>
</module>
```

The *port* and *rootpath* parameters specify the HTTP port and path to the web root, respectively.

Heavy use of this prototype module is not expected, since it duplicates a functional subset of much more powerful and standardized implementations such as Tomcat.

2.3.6. Hang Detection Module

The Hang Detection module constantly reviews various critical processes running on the AppServer and reports any that are not responding as expected. The appropriate configuration entry is:

```
<module name="Hang Detection"
        class="org.dmis.appserver.module.hangdetect.HangDetectionModule"/>
```

Log output that is produced resembles the following redacted sample:

```
[Thu 1/20/05 15:01:25 EST]-[SEVERE] : One or more processes appear to be hung:
[Thu 1/20/05 15:01:25 EST]-[SEVERE] :    Multiplexed Message Distribution Thread
[Thu 1/20/05 15:01:25 EST]-[SEVERE] :    BaseSocketIONode Inbound Data Detector
[Thu 1/20/05 15:01:25 EST]-[SEVERE] :    Message I/O Poll Initiator
```

Expected and actual ping times are also provided as a guide.

3. Utility Programs

3.1. DMISConfig

3.1.1. Overview

The DMISConfig utility offers a method to configure key aspects of a DMIS AppServer visually rather than by editing XML files by hand. The entry point for this program is *org.dmis.launcher.LaunchDMISConfig*.

The various parts of DMIS are organized at the highest level as a series of **.launch* files. For each major functional piece, one of these master files controls the layout of configuration files as well as any startup parameters required for starting that operation. The launcher code used by DMIS to trigger DMIS modules reads these files directly and uses the information contained within to start the application or AppServer as appropriate.

The DMISConfig utility also reads these **.launch* files directly and allows the various options to which they point be edited graphically. The *File|Open* command presents a dialogue from which the **.launch* to be edited can be selected.

The DMISConfig GUI

When a **.launch* file is opened, the DMISConfig display changes to include tabs for each area that can be configured for that type of module. The configuration areas are somewhat different depending on the part of DMIS being edited (e.g. the desktop, various flavors of AppServer, etc).

3.1.2. General Settings

The General tab displays the highest level information for the given configuration. This includes a list of entry points for that DMIS entity, their types, and information on the locations of configuration data for them. A DMIS configuration can have multiple entry points, each with their own configuration set, such as a DMIS Desktop that supports offline mode by using both the GUI and an EMBEDDED AppServer.

The Launcher code starts each module on a separate thread and passes the appropriate parameters as configured for each entry point in the corresponding **.launch* file. In the code, this information is then used to construct the CoreContext instance that is passed around during execution to provide access to basic resources such as configuration files.

3.1.3. Connections

The Connections tab enables configuration of inbound and outbound connections to other entities on the DMIS network. The typical case involves connecting a DMIS Desktop client to an AppServer, though it may also be used to define interconnections between different AppServers (such as a LAN server's connection toe the CENTRAL installation).

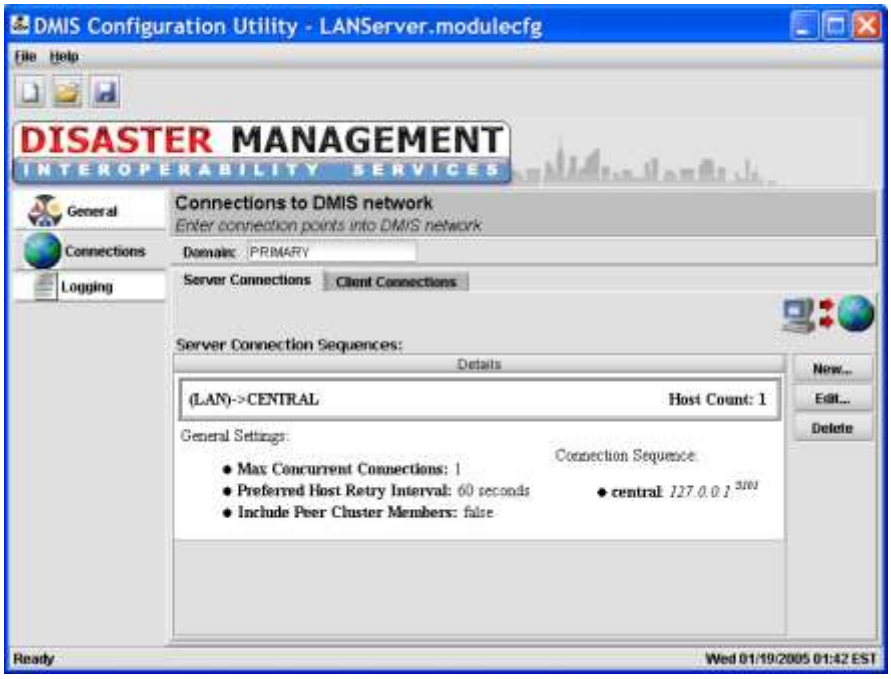
Two major types of connection channels can be defined: *outbound* channels and *inbound* channels. *Outbound channels* are used most commonly for connections from a DMIS desktop to an AppServer. *Inbound channels* are typically used by AppServers to accept connections from Desktop clients or other AppServers.

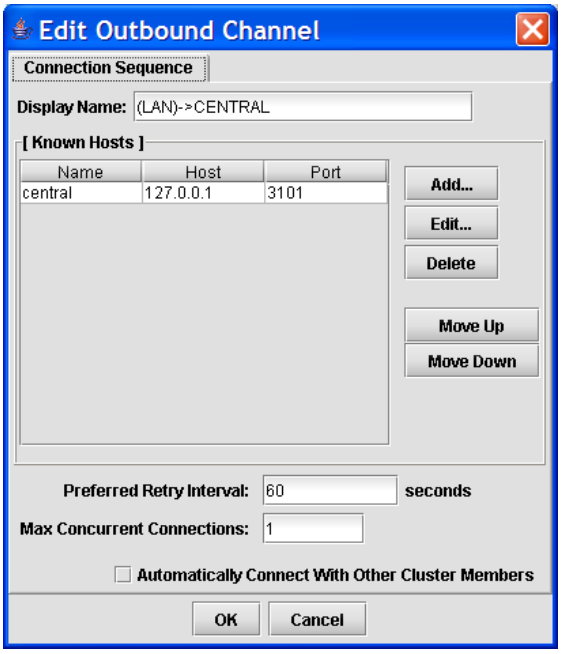
Note: It is very important in DMIS to distinguish between *physical connections* and *logical hierarchy*. The physical connection points defined using DMISConfig create a socket

connection into the DMIS network. The logical position of a given entity within the DMIS hierarchy, however, is determined by its RoutingID and not by the way it is physically connected into the network. Indeed, a primary feature of the messaging subsystem is its ability to correctly route at a logical level through a variety of physical constructs. Thus, it is possible to connect into the DM

3.1.3.1. Outbound Channels

In previous versions of DMIS, an outbound connection was defined as a *host name* and a *port number* in the *CMISServicesProxyConfig.xml* file. The DMIS client would initiate a connection to that location and monitor it, reestablishing the connection if it faltered for some reason.





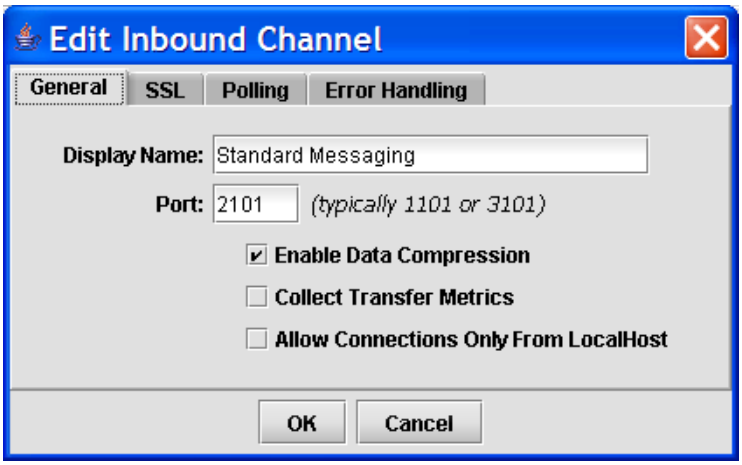
In this version of the DMIS Core Framework, an entity (such as the DMIS Desktop) can configure multiple connection points. These connection points have characteristics including the following:

- Connections can be prioritized in order of the preferred access points,
- In the event of a communications failure, any defined secondary connections are automatically used if the preferred ones are not available,
- When a preferred connection becomes available, it is used and any secondary connection is closed automatically.

In addition, it is also possible to allow multiple connections to be active simultaneously, This feature demonstrates some of the peer-to-peer characteristics embedded within the new DMIS architecture.

3.1.3.2. Inbound Channels

Inbound channels allow a DMIS AppServer, or any other DMIS entity, to accept connections from other entities. Through DMISConfig, parameters for the channel can be established, including the use of data compression and SSL.

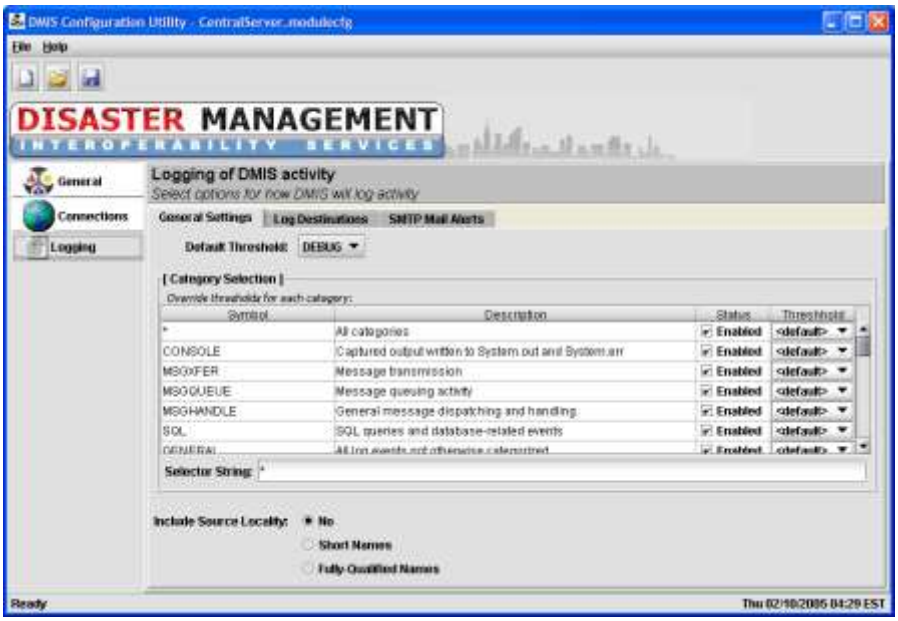


3.1.4. Logger Configuration

The logging facility in this version of DMIS is significantly more flexible, extensible, and powerful than in any previous version. A host of features are offered including:

- Granular categorization of log output at the module level and the ability to set different logger thresholds for each one,f
- Flexible direction of log output, including the ability to use symbols in filenames,
- SMTP-based logging that can send emails when certain kinds of log events arrive,
- Automatic capture and categorization of console (Java *System.out*) output

These options can be configured directly through DMISConfig.



Perhaps the most powerful feature of the logging facility is its ability to categorize and filter output at the module level. The *Default Log Level* controls the overall threshold determining when output is appended to the log, with possible values of **FATAL**, **ERROR**, **WARNING**, **INFO**, **DEBUG**, and **OFF**. Unless overridden, all log output will be handled while observing this setting.

The *Selector String* is where the real power of the logger is leveraged. It is a comma-delimited list of log categories and associated log levels. The category table is a graphical presentation of of the Selector String, and alterations to either one are immediately reflecting in both.

Valid logger category names include:

CONSOLE – Console output captured from *System.out* and *System.err*
DIAG – Output related to diagnostic commands as they are issued and processed
GENERAL – Output not otherwise categorized
INTERCOM – Log entries related to communication between cluster members
MSGHANDLE – Information from general message handling activities
MSGQUEUE – Activity related to message queue processing
MSGXFER – Output related to message transfer and transmission protocol states
SQL – Log of all database-oriented activity (connections, SQL statements, etc)
SVC – General output related to DMIS services running within the Service Manager
SVC:*nnn* – Output related to a specific service (with the symbolic name ***nnn***)

Log output is customized by the order and content of the *Selector String*. The asterisk (‘*’) refers symbolically to all defined log categories. Other categories are listed to override defaults or settings in categories mentioned previously in the string. The minus sign (‘-‘) disables logging for the given category. The following set of exampled is provided to clarify the use of this string:

Category String	Interpretation
*****	Log all categories at the default threshold
***,-SQL**	Log all categories except for SQL-related entries
***,-MSGXGER,SVC:TIE(DEBUG)**	Log all categories at the default threshold, except message transmission related entries. In addition filter all entries related to the TIE service at the DEBUG level.

Entries in the log output itself are tagged so that it is easy to see how the entries have been categorized, as illustrated (in **bold**) by this redacted sample:

```
[Thu 1/20/05 11:55:35 EST]-[CONSOLE:INFO]       : Default serialization format is 'binary'
[Thu 1/20/05 11:55:35 EST]-[GENERAL:INFO]       : Deploying message service [Alerts Service:
ALERTS]
[Thu 1/20/05 11:55:35 EST]-[SVC-ALERTS:DEBUG]   : Alerts Service started.
[Thu 1/20/05 11:55:35 EST]-[SVC-DOWNLOAD:DEBUG] : Distribution List Service started.
[Thu 1/20/05 11:55:35 EST]-[MSGHANDLE:INFO]     : Added to queue <....>
[Thu 1/20/05 11:55:35 EST]-[MSGHANDLE:INFO]     : Reached MsgHub.onMessageReceived()
[Thu 1/20/05 11:55:35 EST]-[MSGHANDLE:INFO]     : Reached DiagModule.onMessageReceived()
[Thu 1/20/05 11:55:35 EST]-[MSGHANDLE:INFO]     : Reached RSM.onMessageReceived()
[Thu 1/20/05 11:55:35 EST]-[SQL:INFO]           : SELECT  * FROM temp
[Thu 1/20/05 11:55:35 EST]-[SQL:INFO]           : SELECT * FROM Operator WHERE id = 1009
```

When any part of DMIS is started, information is send to the console in order to confirm the logging options that have been configured. In addition to providing the original filter string supplied in DMISConfig, its runtime interpretation is displayed as the set of log categories and the filter levels that will be used.

```
[-------------------------------------------------------------------]

 Log categories are configured as follows:

    <*>

    CONSOLE (DEBUG)      DIAG (DEBUG)          GENERAL (DEBUG)
    INTERCOM (DEBUG)     MSGHANDLE (DEBUG)     MSGQUEUE (DEBUG)
    MSGXFER (DEBUG)      SQL (DEBUG)           SVC (DEBUG)


[-------------------------------------------------------------------]
```

If the expected log output cannot be located, the first step in debugging the problem may be to verify that the logging options are set properly.

3.2. Activation Wizard

3.2.1. Overview

The DMIS messaging is extremely flexible, able to route messages at the logical level across a variety of physical topologies. An entity on the DMIS network declares it logical name, and the messaging subsystem handles the traffic flow.

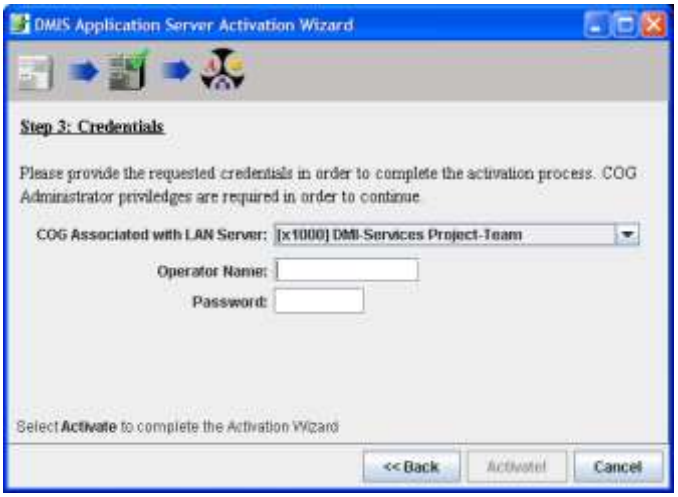
This flexibility can also create security concerns, since if one could impersonate a given entity then some of its traffic might be intercepted. Alternately, rogue DMIS installations could join clusters and subnetworks to which they do not rightfully belong.

To address these concerns, an Activation Wizard is provided to control the process of connecting Application Servers into the DMIS network. *All DMIS AppServers, except for ones embedded in the DMIS desktop, must be "activated" before they can be started.* Any attempt to start a DMIS AppServer that has not been properly activated results in an error condition at runtime. This feature ensures that there are controls on the way in which entities can connect to DMIS and identify themselves.

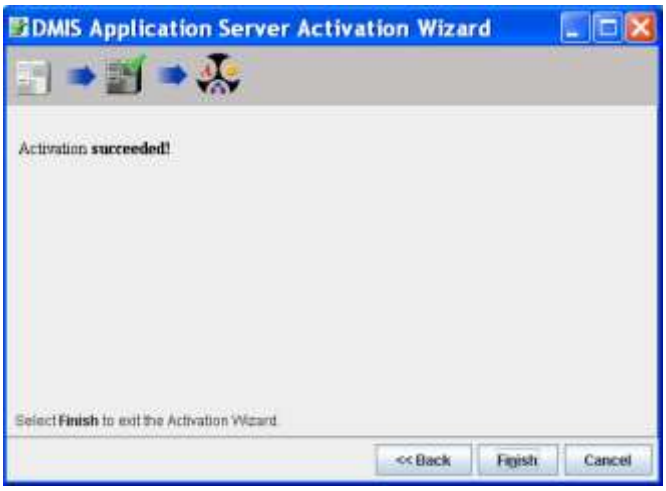
3.2.2. Using the Activation Wizard

The Activation Wizard can be started by using the Java entry point *org.dmiservices.launcher.LauncherActivationWizard*, which presents the following intuitive interface.

Activation must be performed separately for each *configuration profile* (discussed elsewhere in this document) and each type of AppServer to be used (e.g. a Central AppServer or a LAN AppServer). The wizard automatically determines which configuration sets apply to the particular DMIS installation and presents them here as a simple list.



The locus of the activation process is the establishment of an AppServer's identity on the DMIS network. In order to validate the identity of the server, appropriate credentials must be provided to signify that this AppServer truly represents the organization in question. An existing account with COG Administrator privileges is required in order to complete the activation process.



The activation information produced by this utility incorporates elements from the machine's DMIS installation and verified network identification, which are cross-checked as the AppServer starts. If any information does not match fully, the AppServer issues an error message and instructs the operator to run the Activation Wizard again. Because activation files are strong-encrypted and are tied to multiple values related to each installation, they cannot simply be copied from one DMIS instance to the next - or even between different configuration profiles on the same machine. Enabling a DMIS AppServer to connect to the DMIS network always requires the appropriate COG Administrator privileges at some point in the process.

3.3. ServerDiag

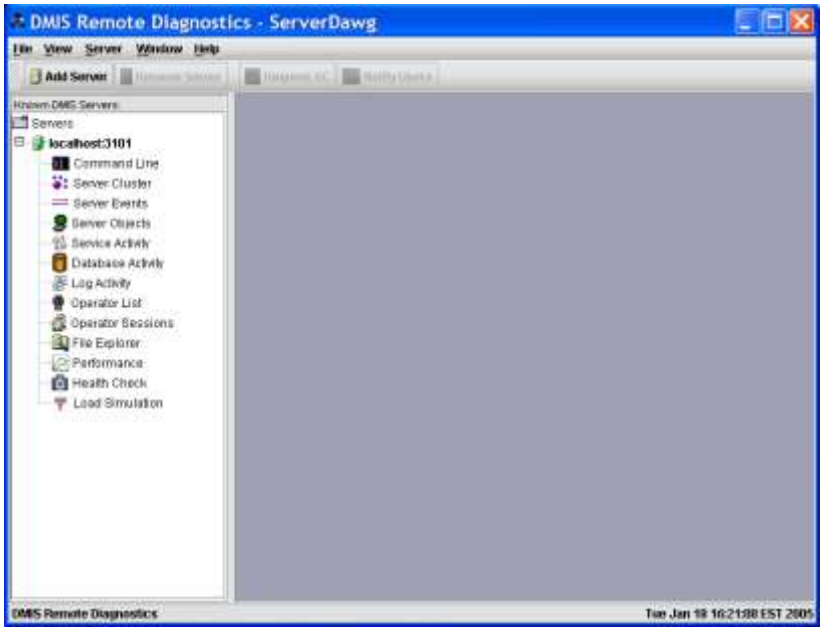
3.3.1. Overview

The ServerDiag tool offers a broad array of functions for checking availability, diagnosing, and debugging DIS AppServers. These features are packaged in an attractive interface that is intuitive and easy to use.

3.3.2. Starting ServerDiag

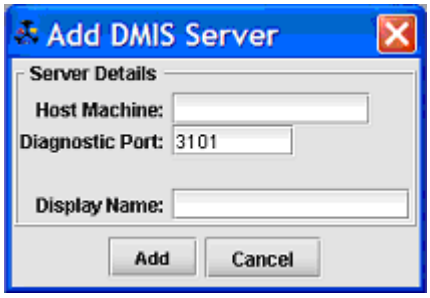
The Java entry points for ServerDiag is *org.dmis.launcher.LaunchServerDiag* and *org.dmis.launcher.LaunchServerDiagExpert*. Two flavors of ServerDiag are offered, the default mode and "expert" mode, which allow different degrees of functionality. The default mode is the version that is designed for use by LAN administrators when DMIS is installed at an organizational level.

When ServerDiag is launched, the main interface appears immediately.



The ServerDiag GUI

The tree on the left shows all of the AppServers this instance of ServerDiag is configured to see. New servers can be added to this tree by activating the *Add Server* button and supply a host name for the AppServer to be recognized. The port number defaults to 3101, which works in most situations, but can be overridden when circumstances warrant.

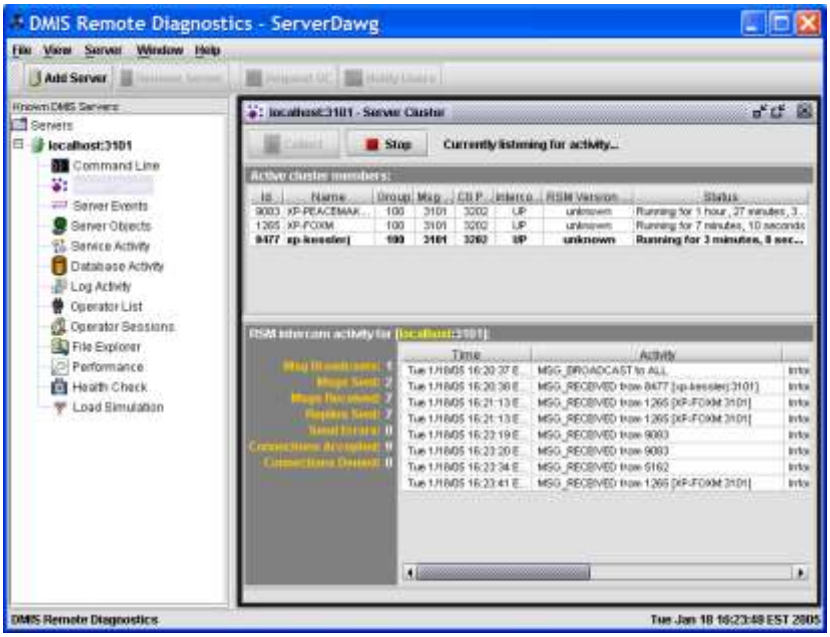


The server list is preserved across ServerDiag sessions.

Servers that are known to be online are displayed in **bold**, while other servers are displayed in a regular font. The right side of the display is reserved for each of the specialized diagnostic categories that are opened by double-clicking on one of the feature names.

3.3.3. Diagnostic Category: Server Cluster

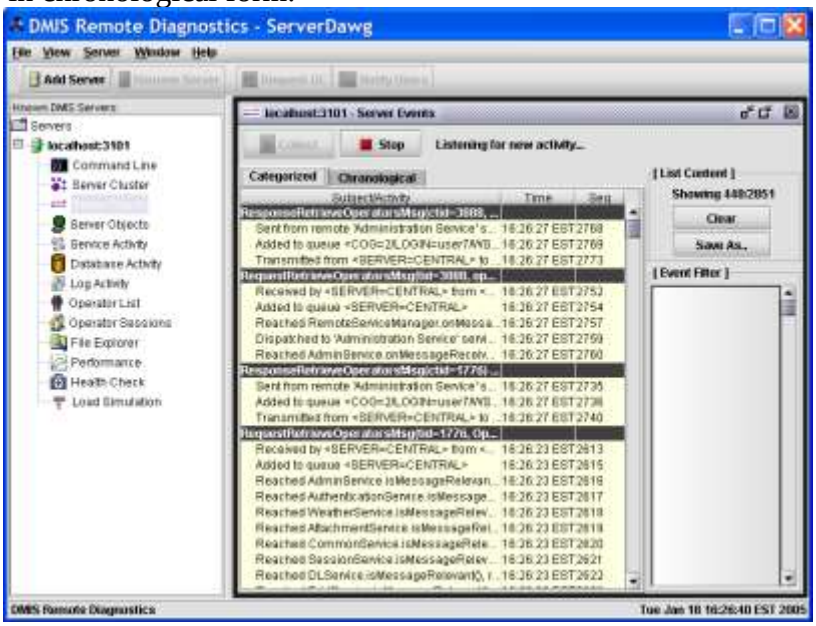
The Server Cluster category provides information regarding members in the current AppServer cluster (this cluster to which the selected AppServer belongs). This display includes information on other servers that belong to the cluster, as well as any intercom traffic between them.

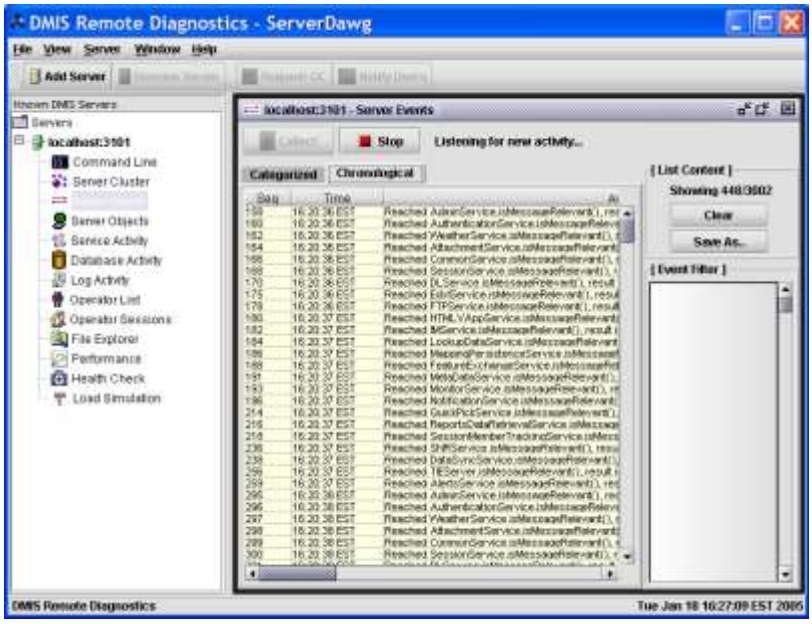


3.3.4. Diagnostic Category: Server Events

The Server Events category provides detailed information regarding various activities related to the AppServer. Events can be selected and filtered based on category, allowing certain types of events to be excluded so that any examination can be focused as needed.

The display offers two distinct interpretations of recent events: *Categorized* and *Chronological*. The Categorized view sorts events into actions on specific objects, such as at a message level, ordered by the time at which each milestone was passed. The Chronological view is a flat display of events in chronological form.

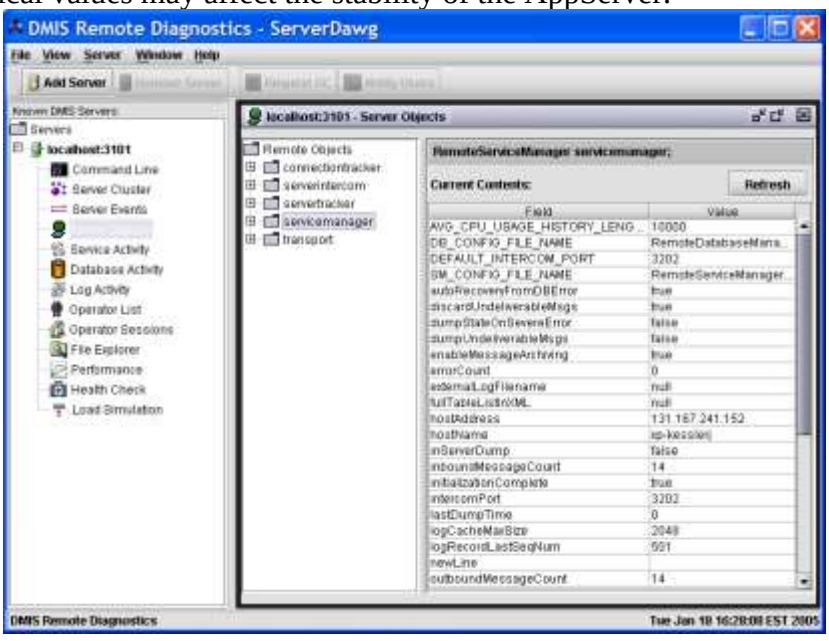




The contents of either display can be saved to a text file for a more detailed examination.

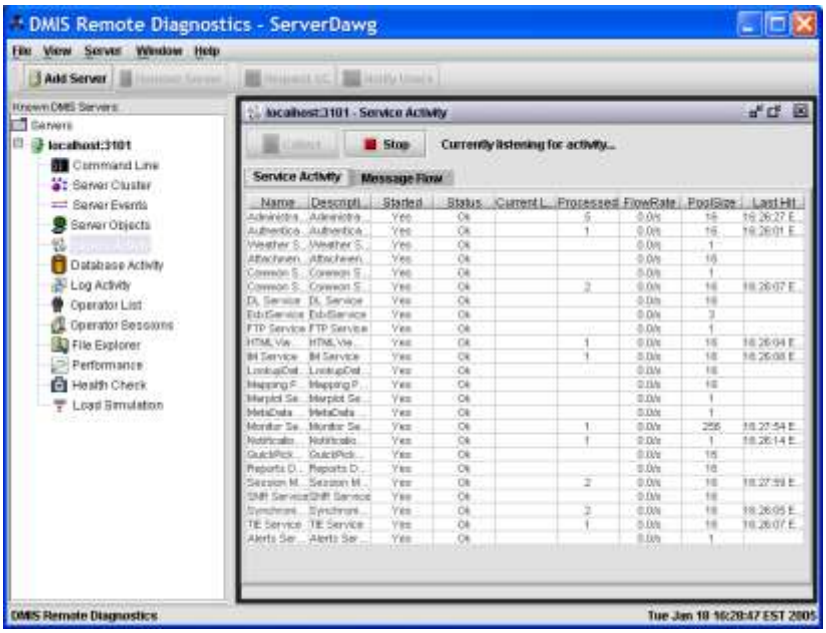
3.3.5. Diagnostic Category: Server Objects

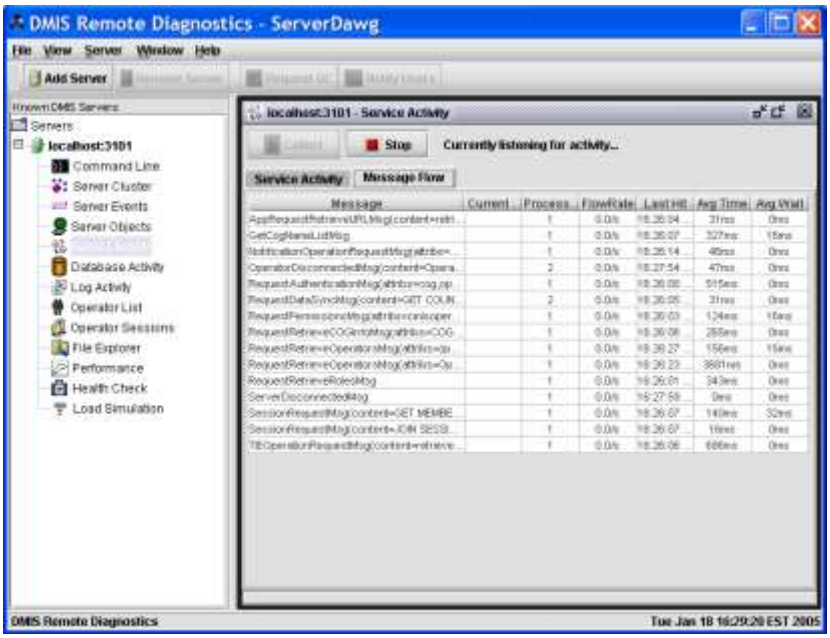
The Server Objects category provides insight into the internal state of critical objects on the AppServer. This advanced feature allows values for individual fields to be examined, and in some cases, altered dynamically. This feature must be used with extreme caution, since the alteration of critical values may affect the stability of the AppServer.



3.3.6. Diagnostic Category: Service Activity

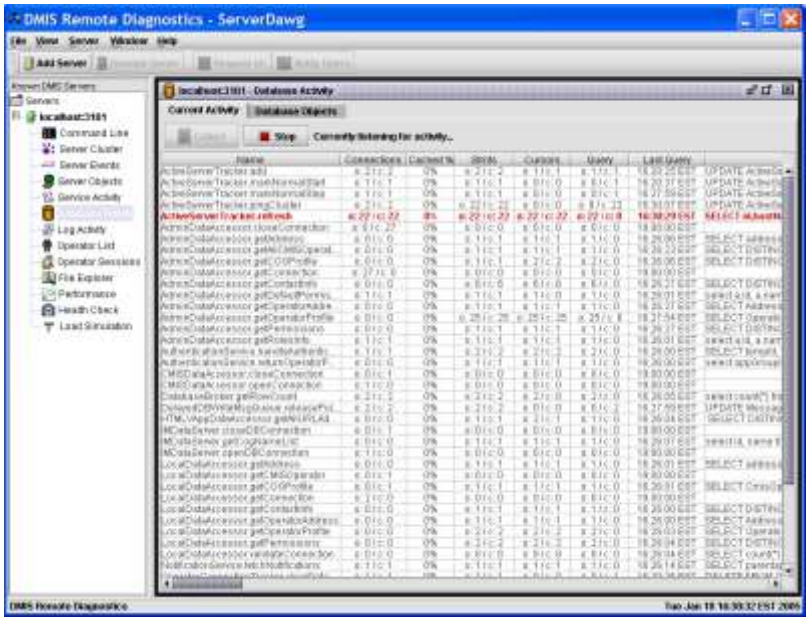
The Service Activity category displays current activity for each installed service, as well as activity sorted at an individual message level. A dynamic, color-oriented feature makes it easy to view activity in real time.



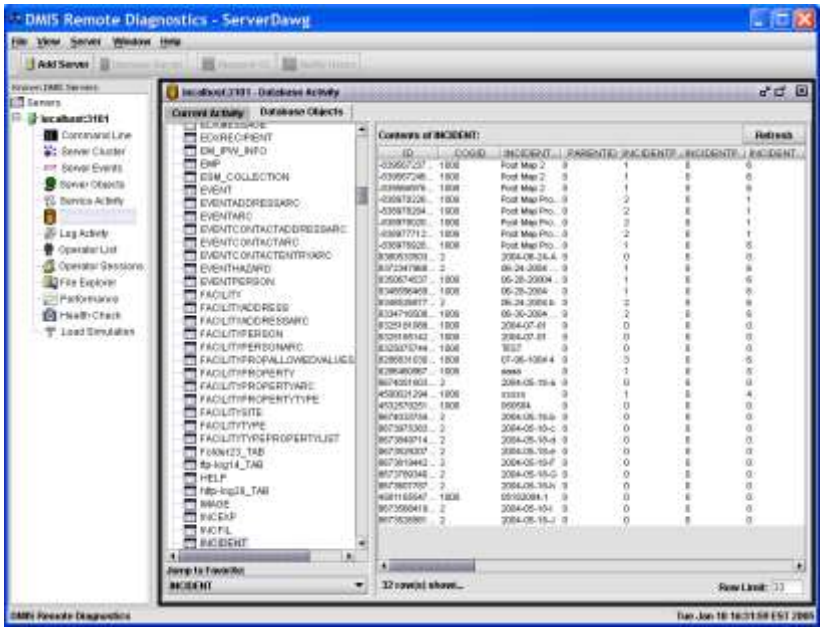


3.3.7. Diagnostic Category: Database Activity

The Database Activity category provides detailed information on all database-oriented activity. A count of statement, cursor, and query allocation is displayed with correlation to the Java method that performed the action. In addition, the text of the last executed SQL statement is provided. These pieces of information make it easy to isolate database-centered resource leaks.



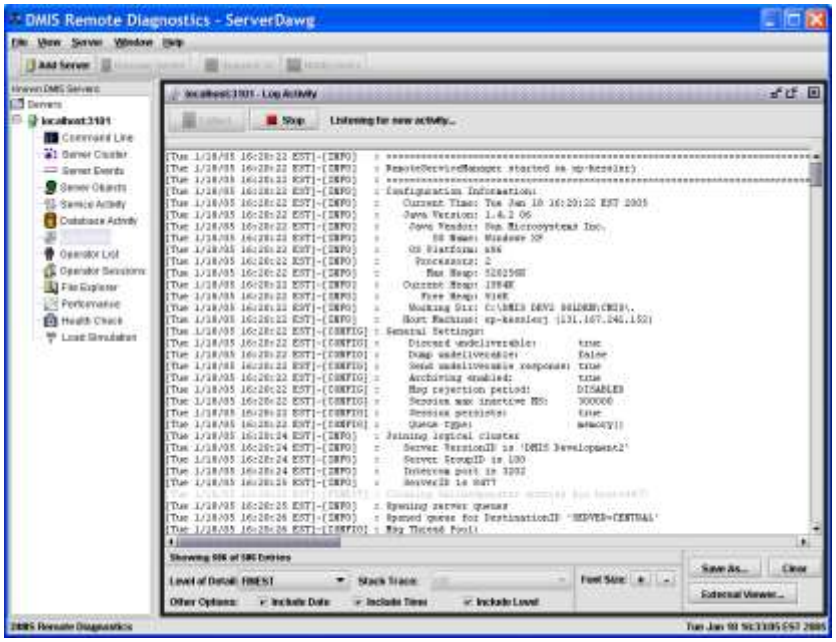
In addition to this details data, another tab is available for a simplistic overview of the contents of tables within the database. This facility is not meant to replace a full-featured utility (such as DBVisualizer), but may be helpful in some cases due to its integration with ServerDiag. One convenience, for example, is that the database connection information employed by the AppServer is used implicitly without requiring direct knowledge of connection information. In addition, crude database examinations can take place even in situations where tools such as DBVisualizer may not have direct access to the database (such as in production).



This relatively crude facility provides no means of changing values in the database.

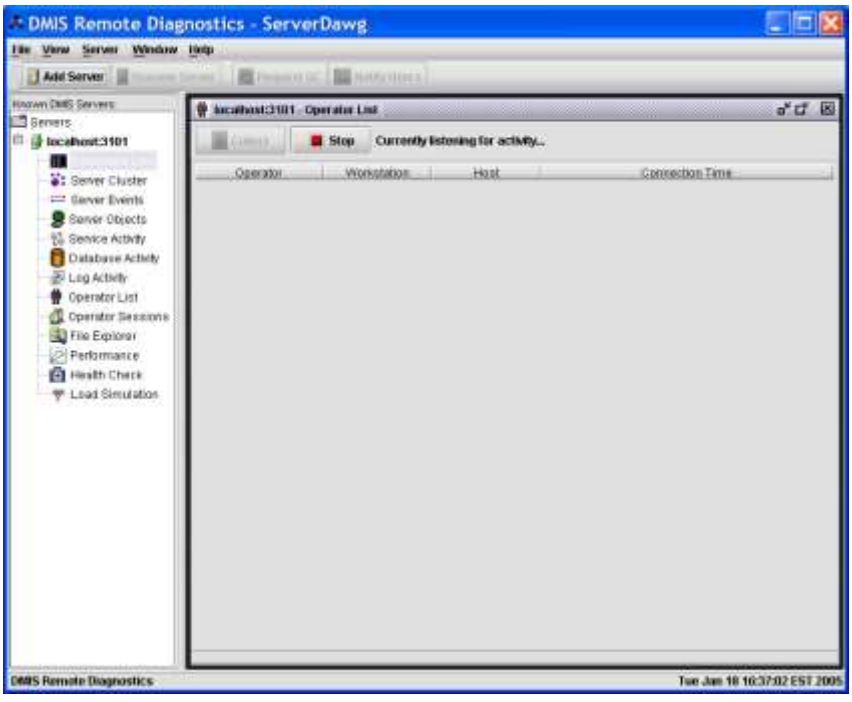
3.3.8. Diagnostic Category: Log Activity

The Log Activity category displays the contents of the AppServer log in real time. Log entries can be filtered and saved to an external file for a more detailed evaluation. A useful characteristic of this display is that the log is always shown at the FINEST granularity regardless of logger configuration on the server. That is, even if the *Logger.xml* configuration file is set to record only ERRORs and WARNINGs, this display will show details at the FINEST level and allow them to be filtered in real time through ServerDiag. This can be useful in capturing detailed information in situations where it would otherwise be lost.



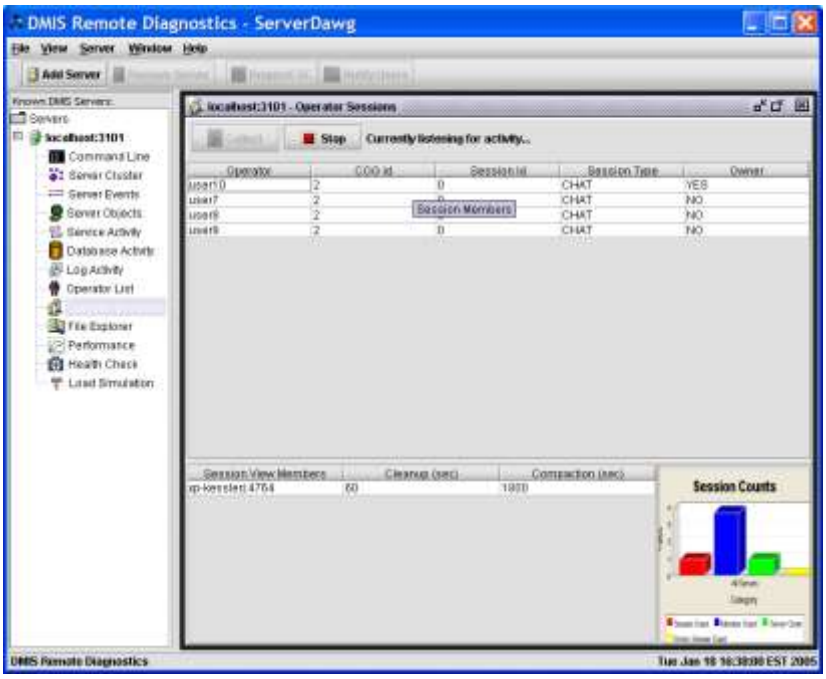
3.3.9. Diagnostic Category: Operator List

The Operator List category provides a flat list of all operators currently connected to the cluster. All operators connected to the selected server, plus any connected to other peer member in the same cluster, are shown.



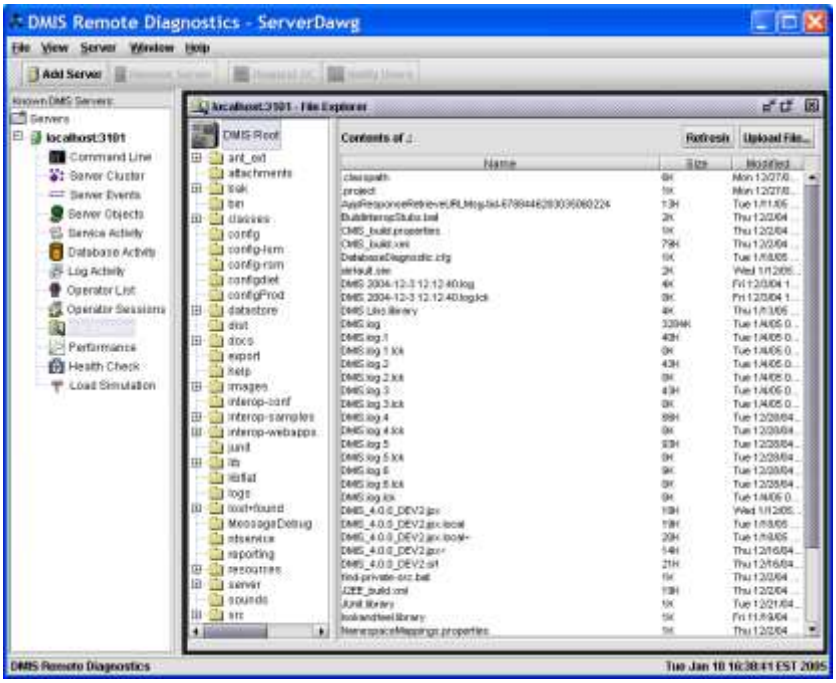
3.3.10. Diagnostic Category: Operator Sessions

The Operator Sessions category provides detailed information from the Session Manager regarding sessions that are currently being tracked. A list of all active sessions, as well as the operators participating in each one, is displayed.



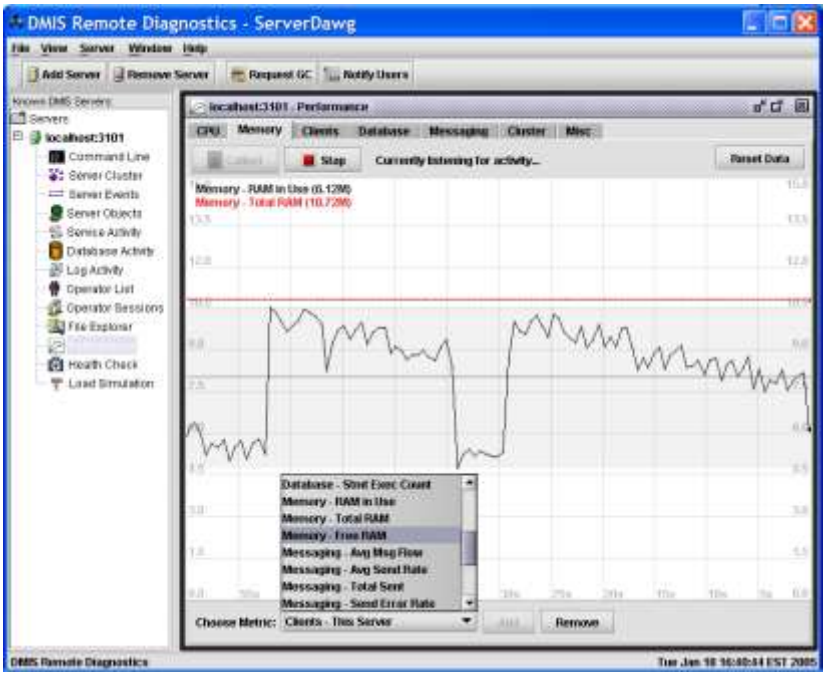
3.3.11. Diagnostic Category: File Explorer

The File Explorer display allows the user to view the DMIS directory structure and file contents on the AppServer. This can be useful for viewing past logs, configuration data, and diagnosing other issues. The display is limited to the directory structure under the DMIS installation; no other parts of the server's storage are visible. File can be double-clicked for download and viewing. In addition, a primitive File Upload facility is provided in order to allow movement of files up to the server, again only within the confines of the DMIS installation structure.



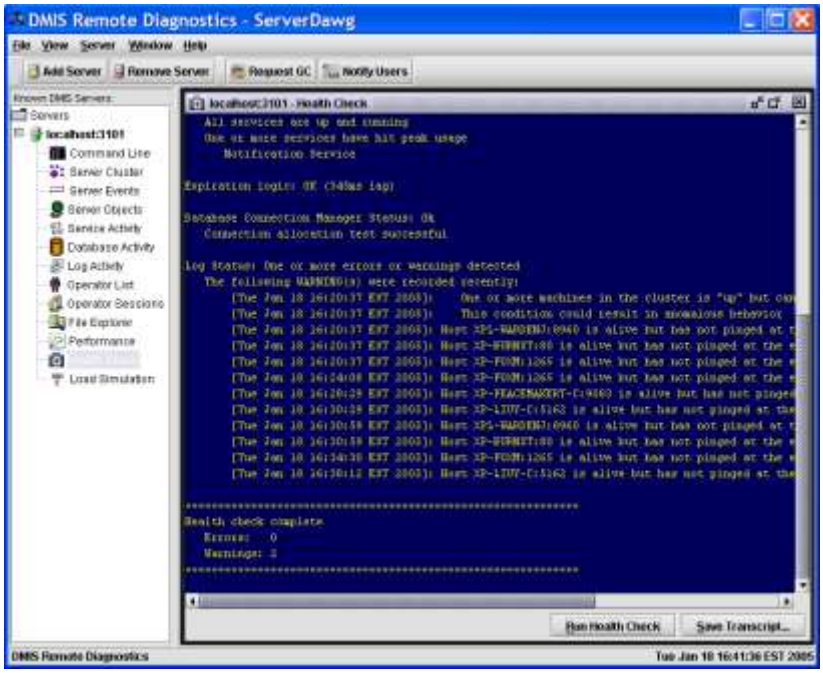
3.3.12. Diagnostic Category: Performance

The Performance category offers a sophisticated, yet intuitive, facility for measuring the performance of the given AppServer. Graphing is performed based on a variety of metrics, including memory usage, CPU load, and many other values. Metrics from different categories can also be shown on the same graph, making it possible to visualize relationships between performance variables. Information on the display is collected in real time from the AppServer.



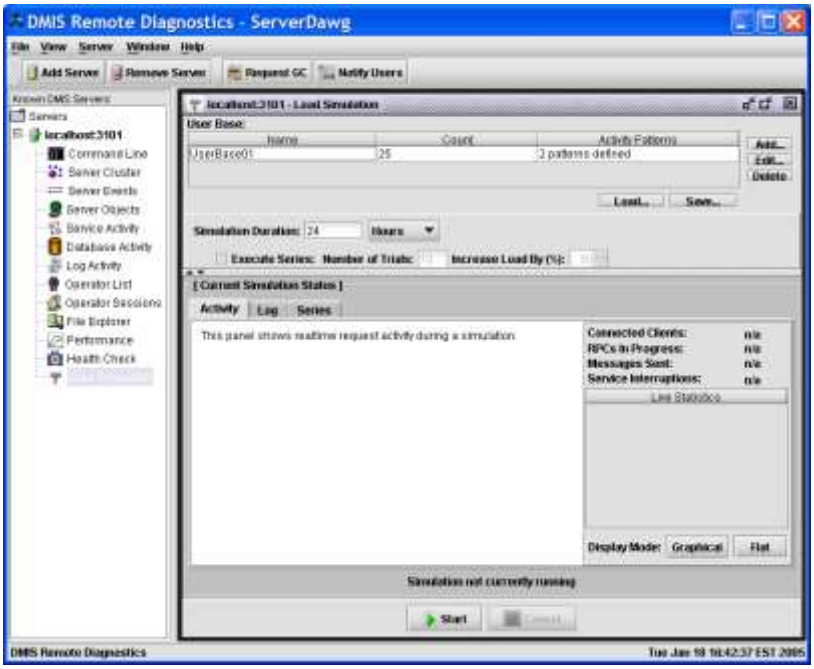
3.3.13. Diagnostic Category: Health Check

The Health Check display performs, on demand, a series of "sanity checks" on critical values within the AppServer and reports any potential anomalies that it finds. This can be useful in assessing whether the "mental state" of the server is stable.



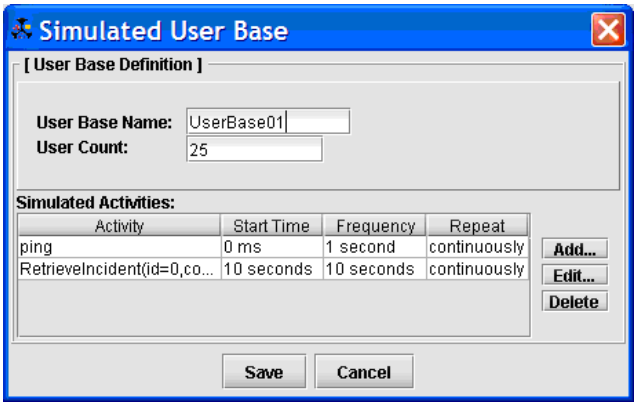
3.3.14. Diagnostic Category: Load Simulation

The Load Simulation category provides a powerful facility for generating a load against a given AppServer. Clients can be defined as 'user bases' of predetermined sizes, each with specific types of activities performed against the server and with a specified pattern. Simulation parameters can be saved for later recall and exchange with other individuals.



A user base is defined by selecting the *Add* button in the User Base section at the top part of the display. Each User Base is given a display name (to help distinguish it from any other defined User Base), an operator count, and a list of activity patterns.

The activities to be simulated for each user base are defined as one or more *activity patterns* by selecting the Add button in the Simulated Activities area at the bottom part of the dialog. When multiple activity patterns are defined, they are executed in parallel allowing complex and multi-layered behaviors to be created.

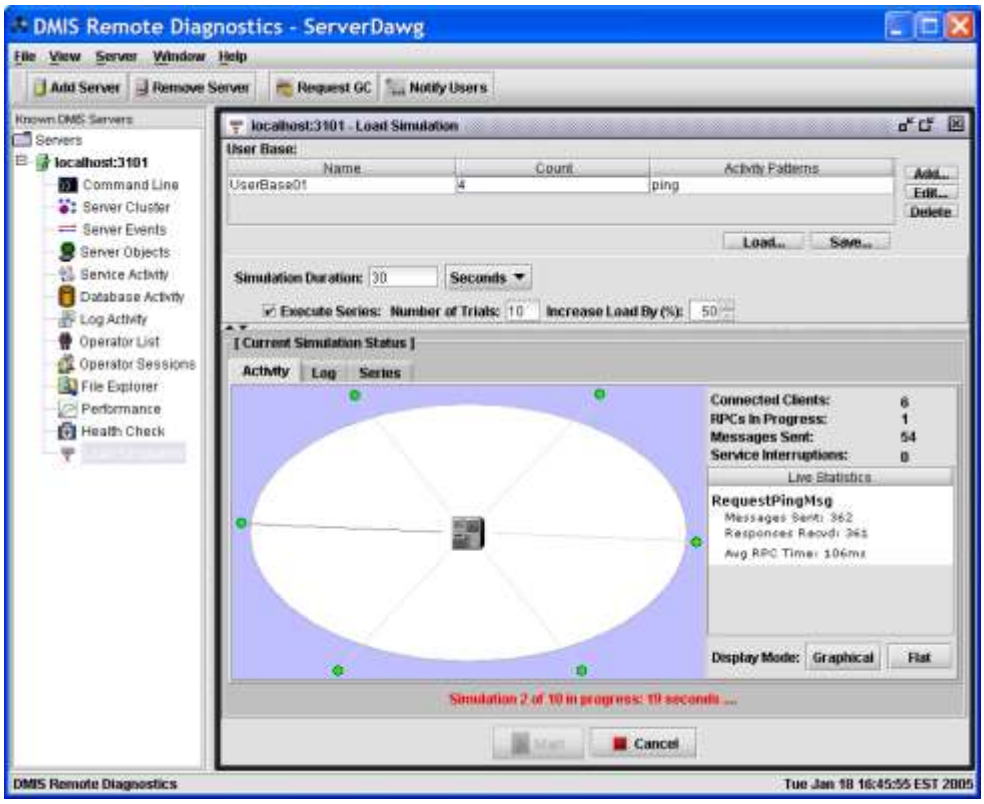


An activity pattern is a string of *predefined actions* delimited by the plus ('+') character. Each action may have one or more parameters that can be used to customize its behavior. The list of supported actions is provided in a dropdown list and can be appended to the currently displayed pattern by selecting the *Add* button as the desired pattern is highlight. The full activity syntax, along with any supported parameter names and defaults, is appended to the pattern string.

The entire sequence of actions can start anytime after the simulation begins, and can repeat at prescribed intervals. It is also possible to introduce variations into these intervals in order to create more lifelike load scenarios.

Once the load simulation is started, ServerDiag creates a client connection for each user in each defined user base, and initializes the activity patterns according to the User Bases to which they belong. The simulation includes an attractive animation depicting real-time activity between the simulated clients and the AppServer.

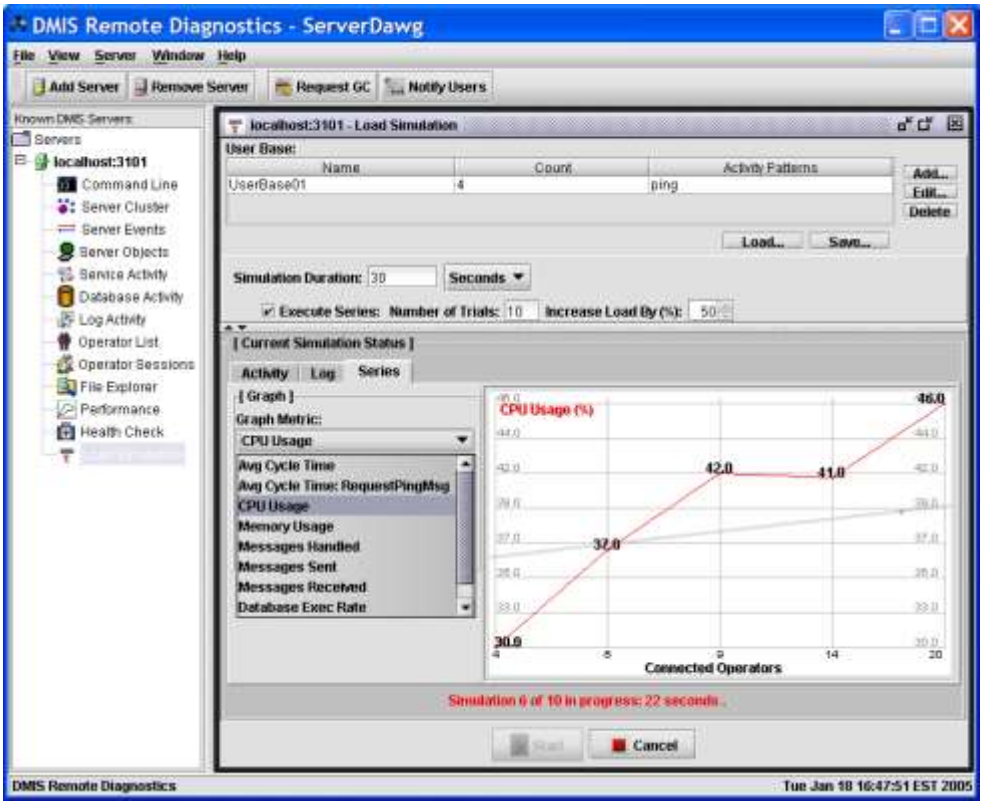
These simulated clients are not full DMIS Desktop clients, but are instead constructed at the messaging level in order to minimize resources and maximize the number of operators that can be simulated. Because the standard DMIS messaging subsystem is used, however, the simulated users (and their activities) appear real to the AppServer. Therefore, from the perspective of the AppServer, there is little distinction between a real world load and a simulated load.



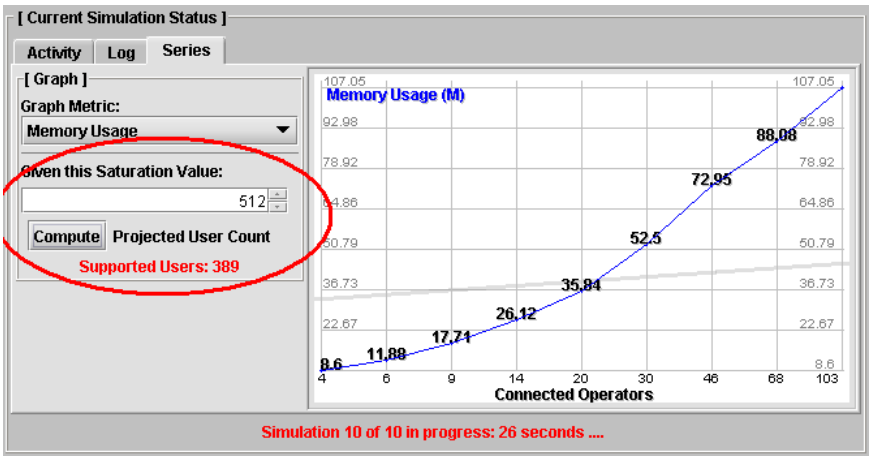
Note: There are practical limits to the number of users that can be simulated at any single workstation. Running the simulation on the same machine as the AppServer imposes further constraints on the accuracy of the simulation results, and these concerns become more pronounced as the simulated load increases. Therefore, high-volume tests are best performed on a machine dedicated to the load simulation, and perhaps even across multiple machines when the test load is higher than one machine can realistically produce. If the CPU usage on the ServerDiag machine is consistently high, memory usage is forcing the operating to page, or the ServerDiag GUI seems sluggish, the simulation load may be taxing the machine's capacity.

The Load Simulation facility also provides the ability to run a *series of simulations* in order to analyze performance trends as the load increases within the defined user bases and activity patterns. The number of trials can be set, along with a percentage by which the user counts should be increased after each successive execution.

As a simulation series executes, performance trends are graphed in real time on the *Series* tab of the Load Simulation display. Graphing can be performed against a number of different metrics in order to provide a more complete picture.



In addition to graphing trends for various metrics, the Load Simulator *can also perform crude projections* of how many users might be supported given the current activity scenario. The tool performs the projection using a "saturation value" provided by the user for any given metric, which refers to an arbitrary value at which the server should be considered (for the sake of the argument) fully loaded.

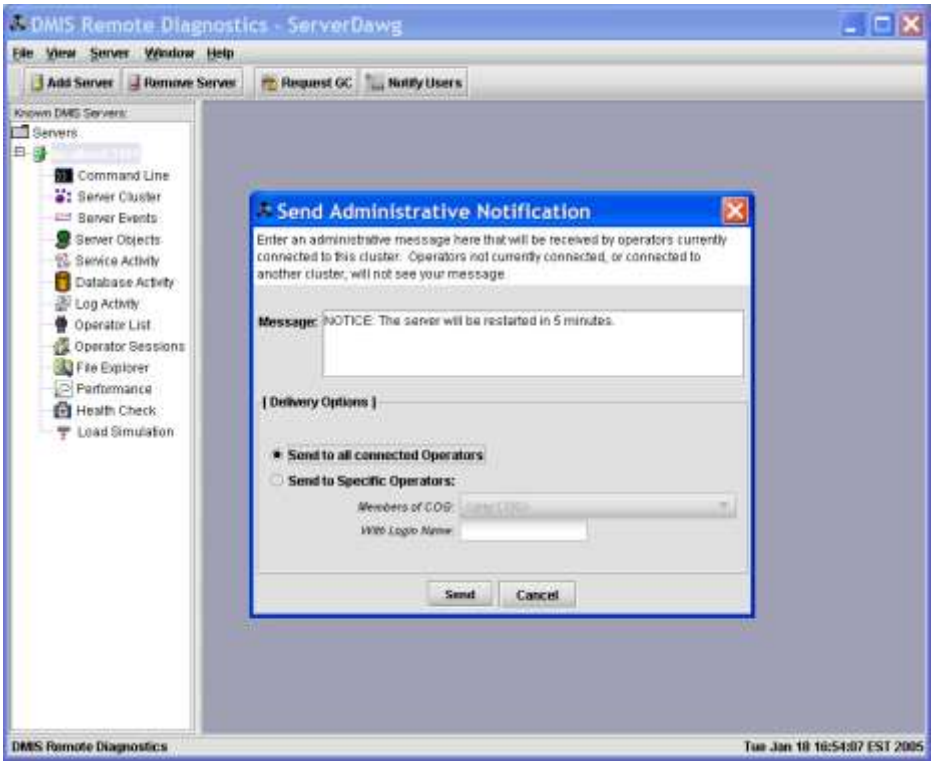


Note: The projected values are only estimates and may not reflect actual results. In fact, the projections tend to vary quite a bit depending upon which metric is used as its basis and the saturation value that is provided. The best estimates require realistic load scenarios and a lot of trials, but even then these values are only rough estimates.

The Load Simulation tool also provides a wealth of performance data in the Log tab at the end of each simulation. This information provides a high-level view of the AppServer's performance during the stress test.

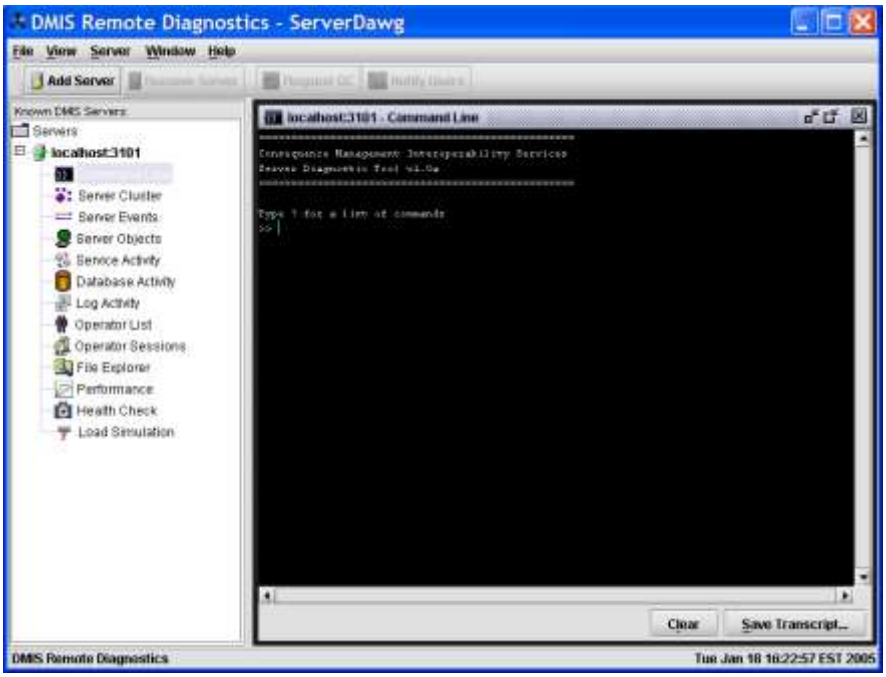
3.3.15. Administrative Notification Facility

ServerDiag provides a mechanism for send administrative messages to all DMIS operators that are currently online. This can be useful in situations, for example, were maintenance is planned for a given installation and any online operators must be informed.



3.3.16. The ServerDiag Command Line

ServerDiag provides a command-line feature that allows diagnostic commands to be entered in a console-like environment. Many of the features offered by the GUI are also available here as commands, and some commands have no counterpart in the GUI.



The following section describes the usage, and command set, o\of this facility in more detail.

3.3.16.1. Usage

Commands are executed by typing them into the command line window. They are sent to the AppServer and results are displayed in the window immediately following the text of the request. Commands that are supported include:

```
clstmt                  - clears the current detailed SQL history
csi [type id] [id]      - clears session information for all or a specific session
dbc                     - returns database connection history info
dbi                     - returns state info from DatabaseManager
dbstmt                  - returns detailed info on recently issued SQL
gci                     - returns info on the server cluster
gsi [dirty]             - returns info on server sessions
envi                    - returns general server environmental data
evi                     - returns ExpirationVector diagnostics
shell [cmd]             - runs the given shell command on the server
gt [table]              - returns contents of the given remote table
hb                      - returns heartbeat status from various modules
ici                     - returns info from the server intercom
logtext [lines]         - current contents of the log
notify [rtgid] [text]   - sends a text message to all operators matching the RoutingID
notifyall [text]        - sends a text message to all connected operators
oplist                  - returns a list of connected operators
qi [queuename]          - returns info on defined queues (or a given queue)
query [query]           - executes the given database query on the remote db
restartsvcs             - restarts all services on the server
sci [S/E] [interval]    - starts/stops session compaction with given interval
shutdown [rtid]         - requests immediate shutdown of given clients
smci [S/E] [interval]   - starts/stops session member cleanup with given interval
startsvc [svcid]        - starts the service with the given ID
stopsvc [svcid]         - stops the service with the given ID
svci                    - returns info on running services
svrdump [x] | [new]     - returns list of ServerDumps, gets existing dump, or creates new dump
threadlist              - returns list of threads running on the server
tlgi                    - returns general info from the transport layer
```

```
tlci [rtid]              - returns connection info from the transport layer
```

Each of the command is now discussed in more detail.

3.3.16.2. Commands

3.3.16.2.1. help/?

Displays the same quick-reference that is shown when the tool is first started.

3.3.16.2.2. clstmt

Clears the existing history of SQL statements that is being kept by the AppServer. This can be used (in conjunction with the **dbstmt** command) to easily determine the impact that any given activity has on the database. To see the database details of any given action, issue **clstmt**, execute the action in the desktop, then issue **dbstmt**. The resulting list will *largely consist* of all SQL executed as a result of that action (though some other unrelated queries may appear in there as well due to the multi-process nature of the AppServer).

3.3.16.3. dbc

Shows detailed information on database connection/stmt/cursor usage by the AppServer, including connections currently open, and tracking (at the method level) of where database activity has occurred throughout the lifespan of the server instance. Output looks similar to the following (with some information truncated due to space constraints):

```
>> dbc
Activity By Method:

   Source Locality                                          Connections      Cached  Stmts            Cursors          Queries Cmds    Last SQL Executed
   ======================================================== ================ ======= ================ ================ ======= ======= ================>
   ActiveServerTracker.add                                  [o:1    c:1    ] 0%      [o:1    c:1    ] [o:1    c:1    ] 1       0       t-21.3s:  SELECT…
   ActiveServerTracker.pingCluster                          [o:3    c:3    ] 67%     [o:3    c:3    ] [o:0    c:0    ] 0       3       t-14.0s:  UPDATE…
   ActiveServerTracker.updateServerInstancesFromDB          [o:3    c:3    ] 67%     [o:3    c:2    ] [o:3    c:2    ] 3       0       t-0.8s:   SELECT…
   AdminDataAccessor.getCMISOperator                        [o:0    c:0    ] 100%    [o:15   c:15   ] [o:15   c:15   ] 15      0       t-19.2s:  SELECT…
   CMISDatabaseLogHandler$1.onActionInvoked                 [o:1    c:1    ] 0%      [o:1    c:0    ] [o:0    c:0    ] 0       1       t-18.9s:  DELETE…
   CMISDatabaseLogHandler.writeMsg                          [o:1    c:0    ] 0%      [o:1    c:0    ] [o:0    c:0    ] 0       0
   DelayedDBWriteMsgQueue.releasePulledMsgs                 [o:22   c:22   ] 45%     [o:22   c:22   ] [o:0    c:0    ] 0       22      t-15.8s:  UPDATE…
   OperatorConnectionTracker.clearDataForHost               [o:1    c:1    ] 100%    [o:1    c:1    ] [o:0    c:0    ] 0       1       t-20.9s:  DELETE…
   OperatorConnectionTracker.refresh                        [o:1    c:1    ] 0%      [o:2    c:2    ] [o:2    c:1    ] 2       0       t-20.6s:  SELECT…
   RemoteServiceManager.run                                 [o:1    c:1    ] 100%    [o:1    c:1    ] [o:0    c:0    ] 0       1       t-14.0s:  select…
   RemoteServiceManager.waitForGoodDBConnection             [o:1    c:1    ] 0%      [o:1    c:1    ] [o:0    c:0    ] 0       1       t-22.0s:  select…
   SessionPersistenceBroker.removeDeadMembers               [o:2    c:2    ] 100%    [o:2    c:2    ] [o:2    c:2    ] 2       0       t-9.1s:   select…
   SessionPersistenceBroker.removeDeadSessions              [o:2    c:2    ] 50%     [o:4    c:4    ] [o:0    c:0    ] 0       4       t-9.1s:   Update…
   SessionPersistenceBroker.setOperatorsToInactive          [o:1    c:1    ] 0%      [o:1    c:1    ] [o:0    c:0    ] 0       1       t-19.1s:  update…
   SessionUtilities.getOperatorId                           [o:0    c:0    ] 100%    [o:3    c:3    ] [o:3    c:3    ] 3       0       t-19.2s:  select…
   SNRDataServer.createDBConnection                         [o:16   c:0    ] 0%      [o:0    c:0    ] [o:0    c:0    ] 0       0

Current Connections:

   Allocated Connections: Locality                          ID    Age          Open  Stmts Created AutoCommit Catalog    ThreadID
   ======================================================== ===== ============ ===== ============= ========== ========== ==========
   CMISDatabaseLogHandler:writeMsg                          1015  19531ms      true  1             true                  CMISDatabaseLogHandler
   SNRDataServer:createDBConnection                         1045  15452ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1031  17312ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1050  14327ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1034  16952ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1022  18640ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1037  16578ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1025  18125ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1049  14531ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1028  17672ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1048  14734ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1044  15672ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1040  16187ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1051  14250ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1046  15156ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1043  15890ms      true  0             false                 main
   SNRDataServer:createDBConnection                         1047  14937ms      true  0             false                 main

   Allocated Statements: Locality                           ID    ConnID Age          ThreadID                       Cursors Created Cmds Execd Last SQL
   ======================================================== ===== ====== ============ ============================== ================ ========== ==========>
   CMISDatabaseLogHandler:writeMsg                          1021  1015   19547ms      CMISDatabaseLogHandler         0                59         INSERT IN…

   Allocated ResultSets: Locality                           ID    StmtID ConnID Age          ThreadID                       SQL Executed
   ======================================================== ===== ====== ====== ============ ============================== ============================
>>
```

3.3.16.4. dbi

Shows high-level information on database usage by the AppServer, including a the total number of connections made, and the current connection “flow rate”. Sample output:

```
>> dbi
```

```
  Driver:            oracle.jdbc.driver.OracleDriver
  URL:               jdbc:oracle:thin:@131.167.241.14:1521:CMISA
  Login:             cmis
  Password:          ****
  Total connections: 259 made since startup
  Connection rate:   ~0.67/second
  Preallocation:     true
  Min pool size:     4
  Max pool size:     64
  Current pool size: 4
  Peak pool size:    8
  Cached connect:    239 time(s) [92%]
  Validation SQL:
  Allocation test:   Ok
>>
```

3.3.16.5. dbstmt

Displays the current history of SQL statements that is being kept by the AppServer. This can be used to more easily diagnose the database activity generated by the AppServer. Information provided includes a details list of SQL statements executed, when they were executed (relative to the current time), which method in the AppServer codebase issued the statement, and how long the statement took to execute. Sample output (with some truncation due to space constraints):

```
>> dbstmt
SQL History - Last 5 Minute(s)

   Issued     Method Name                                          Exec Time  Query
   ========== ================================================== ========== ====================>
   t-0.2s:    ActiveServerTracker:updateServerInstancesFromDB    16ms       SELECT id,hostName,lastStartTime,       lastStopTime…
   t-9.4s:    SessionPersistenceBroker:removeDeadSessions        0ms        Update CMISSession set isActive=0 WHERE id NOT IN (S…
   t-9.4s:    SessionPersistenceBroker:removeDeadSessions        0ms        delete from CMISSession where isActive=0 AND id NOT IN…
   t-9.5s:    SessionPersistenceBroker:removeDeadMembers         16ms       select operatorId,cogId,owner,sessionId, sessionCatId…
   t-12.4s:   ActiveServerTracker:pingCluster                    0ms        UPDATE ActiveServerArray   SET lastHeartbeatPing=102209…
   t-12.4s:   RemoteServiceManager:run                           0ms        select 1 from dual
   t-15.2s:   ActiveServerTracker:updateServerInstancesFromDB    15ms       SELECT id,hostName,lastStartTime,       lastStopTime,las…
   t-19.4s:   SessionPersistenceBroker:removeDeadSessions        0ms        Update CMISSession set isActive=0 WHERE id NOT IN (SELE…
   t-19.4s:   SessionPersistenceBroker:removeDeadSessions        0ms        delete from CMISSession where isActive=0 AND id NOT IN…
   t-19.5s:   SessionPersistenceBroker:removeDeadMembers         0ms        select operatorId,cogId,owner,sessionId, sessionCatId…
   t-27.5s:   ActiveServerTracker:pingCluster                    0ms        UPDATE ActiveServerArray   SET lastHeartbeatPing=10220950…
   t-27.5s:   RemoteServiceManager:run                           0ms        select 1 from dual

>>
```

3.3.16.6. gci

Returns a list of the machines that comprise the “active server array”. This is a list of all the CMIS AppServers running against the same database as the server to which ServerDiag is connected. Sample output:

```
>> gci

Active:
   ID   CMIS AppServer   M/I Ports Status
   ==== ================ ========= ================================================
   8099 WP-KesslerJ2     2101/2202 Active since 2002-05-22 11:08:10.0
   4502 WP-WardenJ-c     2101/2202 Active since 2002-05-22 10:48:58.0
   3529 WP-WEIGEL-C      2101/2202 Active since 2002-05-22 11:41:09.0

Inactive:
   ID   CMIS AppServer   M/I Ports Status
   ==== ================ ========= ================================================
   9312 ns-bso-dev1      2101/2202 Inactive
>>
```

3.3.16.7. envi

Returns basic environmental information from the AppServer Virtual Machine. Sample output:

```
>> envi
  System time:       Wed May 22 15:50:07 EDT 2002
  Database time:     Wed May 22 15:50:07 EDT 2002
  Startup time:      Wed May 22 15:08:00 EDT 2002
  Java version:      1.3.0_02
  Java vendor:       Sun Microsystems Inc.
  OS name:           Windows 2000
  OS platform:       x86
  Current heap:      7712K
  Free heap:         1634K
  Working Dir:       E:\CMIS_Release1\CMIS\.
>>
```

3.3.16.8. evi

Returns detailed status information on the ExpirationVector agent. This is a key component used for time-sensitive operations throughout the AppServer.

3.3.16.9. gt [table]

A shorthand version of the **query** command, this command queries the given table and returns its entire contents in a cleanly formatted block. The query is performed against the same database that the AppServer is using.

```
>> gt Operator

   ID                   PERSONID             PERSONCOGID USERNAME
   ==================== ==================== =========== =============
   0                    0                    0           none
   1                    1                    2           user1
   2                    2                    2           user21
   3                    3                    2           user3
   4                    4                    2           user4
   5                    5                    2           user5
   6                    6                    2           user6
   7                    7                    2           user7
   8                    8                    2           user8
   9                    9                    2           user9
   10                   10                   2           user10
   11                   11                   2           user11
   12                   12                   2           user12
   13                   13                   2           user13
   14                   14                   2           user14
   15                   15                   2           user15
   16                   16                   2           user16
   17                   17                   2           user17
   18                   18                   2           user18
   19                   19                   2           user19
   20                   20                   2           user20
>>
```

In this instance, the command is same as issuing this equivalent: **query select * from Operator**.

3.3.16.10. hb

Returns “heartbeat” information for various important processes running on the AppServer. This is used to determine whether those processes are still alive and processed at the expected rates. Sample output:

```
>> hb

   Process Name                                        Last Pulsate ThreadID
   =================================================== ============ ==========
   ActiveServerArray Cleanup Logic                     14812ms      6915075
   AppServer Pinger                                    5312ms       674272
   DatabaseManager Preallocator                        15ms         560748
   ExpirationVector Async Agent                        15ms         3003101
   genUniqueID() Pregenerator                          15ms         6194169
   Logger Publishing Thread                            23624ms      471035
   MemoryMsgQueueEventWriter Flusher Thread            15ms         3655181
   Queue Multiplex Distributor                         15ms         1560607
   Server Intercom Connection Listener                 843ms        6637295
   Transport Layer Connection Listener                 15ms         3161056
   Transport Layer I/O Dispatcher                      15ms         5913118
>>
```

3.3.16.11. ici

Returns detailed information on the status of the Server Intercom module. The Server Intercom is a mechanism used to allow different AppServers to communicate and work together as a cluster. Sample output:

```
>> ici
  ServerID:            8099
  Intercom port:       2202
  Alive:               true
  Messages broadcast:  6
  Messages sent:       20
  Message send errors: 0
  Inbound connections: 16
  Messages received:   23
  Replies sent:        23
  Denied connections:  0

  Recent activity:

    Activity       When                     Server                         Message
    ============== ======================== ============================== ======================================
    MSG_BROADCAST  Wed 5/22/02 15:31:03 EDT ALL-this                       InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:31:05 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:36:46 EDT 1204 [WP-BURNST-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:37:44 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:42:43 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:43:30 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorDisconnectedMsg
    MSG_BROADCAST  Wed 5/22/02 15:45:26 EDT ALL-this                       InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:45:51 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:46:30 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:49:59 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:52:56 EDT 3448 [wp-sojkaj-c:2101]        RequestServerListReloadMsg
    MSG_RECEIVED   Wed 5/22/02 15:52:57 EDT 3448 [wp-sojkaj-c:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:55:31 EDT 1204 [WP-BURNST-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:56:20 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:56:50 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:57:20 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 15:59:26 EDT 1204 [WP-BURNST-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:00:03 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:00:11 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:06:48 EDT 1204 [WP-BURNST-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:09:04 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:16:31 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorConnectedMsg
    MSG_BROADCAST  Wed 5/22/02 16:16:35 EDT ALL-this                       InformOperatorConnectedMsg
    MSG_BROADCAST  Wed 5/22/02 16:16:48 EDT ALL-this                       InformOperatorDisconnectedMsg
    MSG_BROADCAST  Wed 5/22/02 16:18:23 EDT ALL-this                       InformOperatorConnectedMsg
    MSG_BROADCAST  Wed 5/22/02 16:20:17 EDT ALL-this                       InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:22:22 EDT 1204 [WP-BURNST-C:2101]        InformOperatorConnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:23:17 EDT 1204 [WP-BURNST-C:2101]        InformOperatorDisconnectedMsg
    MSG_RECEIVED   Wed 5/22/02 16:23:51 EDT 3529 [WP-WEIGEL-C:2101]        InformOperatorDisconnectedMsg
>>
```

3.3.16.12. logtext [lines]

Returns a subset of contents from the current log. If no line count is given, the most recent 50 lines are returned. Otherwise, the given number of lines are returned, up to a maximum of 2048. Sample output:

```
>> logtext 35

[Log text: Most recent 35 events]
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Supported desktop software:
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Version '1.0_02_18_02cFULL'
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Version 'loadsim10'
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Starting socket layer
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Port is 2101
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Pipe size is 32 I/O initiator thread(s)
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Polling interval ranges from 20ms/client to 100ms/client
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Polling interval ranges from 100ms/client to 3000ms/client
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Clients become IDLE after after 60000ms/client inactivity
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Polling interval maxes out after 300000ms/client inactivity
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Errors before disconnection is disabled
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Bad connections, reconnect disabled for 5000ms/client
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Data compression ENABLED
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Socket read timeout is 600000ms
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Messaging I/O: Accepting all incoming connections
[Wed 5/22/02 16:28:32 EDT]-[INFO]       : Starting all Services
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Administration Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Authentication Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Attachment Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Common Session Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'DL Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'HTMLViewer Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'IM Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Mapping Persistence Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Monitor Service' started
[Wed 5/22/02 16:28:32 EDT]-[INFO]       :    Message Service 'Reports Data Retrieval' started
[Wed 5/22/02 16:28:33 EDT]-[INFO]       :    Message Service 'Session Cleanup Service' started
[Wed 5/22/02 16:28:33 EDT]-[INFO]       :    Message Service 'Session Member Tracking Service' started
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Message Service 'SNR Service' started
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Message Service 'Synchronization Service' started
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Message Service 'TIE Service' started
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Starting transport layer
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Opening intercom channel
[Wed 5/22/02 16:28:38 EDT]-[INFO]       :    Starting cluster heartbeat
[Wed 5/22/02 16:28:38 EDT]-[INFO]       : ServiceManager is running...

>>
```

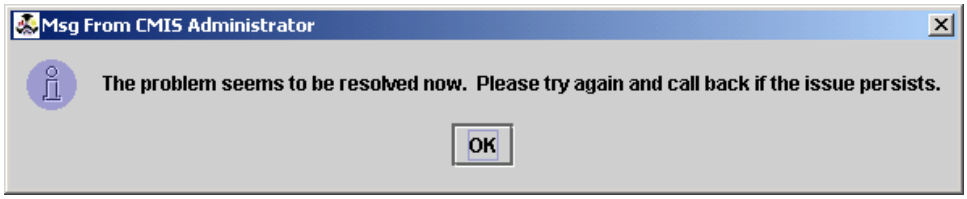
NOTE: Log output is always returned by ServerDiag at the FINEST level no matter how the current log level setting is configured. It is independent of the configuration setting. This is meant to defeat situations where problems cannot be debugged due to insufficient log level settings on the AppServer.

3.3.16.13. notify [rtgid] [text]

Sends an "admin alert message" to all *currently connected operators that match the given RoutingID*. In this example, an administrative message is sent to the workstation supporting *jkessler*:

```
>> oplist

Operator               Workstation Server
====================== =========== ==========================
1000/jkessler          5688        8099  [WP-KesslerJ2:2101]
2/user16               5617        4502  [WP-WardenJ-c:2101]

>> alertsome WS=5688 The problem seems to be resolved now.  Please try again and call
back if the issue persists.
Ok
>>
```

Every matching client (connected through the CMIS desktop software) will see a dialog box like this:



This command presumes that the issuer knows how to formulate a CMIS RoutingID to specify the appropriate targets for the message. A RoutingID of *COG=nnnn*, for example, will send the notification to everyone connected who is part of the given COG. A RoutingID of *COG=nnnn/login=yyyy* will go to a specific individual regardless of where his is logged in.

3.3.16.14. notifyall [text]

Sends an "admin alert message" to all *currently connected operators*. A typical usage might be this:

```
>> alertall The server will be going down shortly.  Please restart your CMIS session.
Ok
>>
```

Every client connected through the CMIS desktop software will see a dialog box like this:

3.3.16.15. oplist

Returns a list of currently connected operators, the workstation they are connected from, and the AppServer (one of the servers in the active server array) that is servicing them.

```
>> oplist

Operator               Workstation Server
====================== =========== ==========================
1000/jkessler          5688        8099  [WP-KesslerJ2:2101]
2/user16               5617        4502  [WP-WardenJ-c:2101]

>>
```

3.3.16.16. qi [queuename]

Returns detailed information on message queues. If **qi** is issued without a parameter, a listing of currently defined queues is provided along with status information for each (which is truncated due to space constraints):

```
>> qi
  Queue                                          Type                         Active Msgs Added     Chained   Expired
  ============================================== ============================ =========== ========= ========= =========
  COG=1000/LOGIN=jkessler                        FileBasedMsgQueue            1           1         0         0
  COG=1000/LOGIN=jkessler/WS=5688                FileBasedMsgQueue            4           4         0         0
```

```
  COG=1000/LOGIN=jkessler/WS=5688/SUBQUEUE=AUTH      FileBasedMsgQueue            0         1         0         0
  SERVER=CENTRAL                                     FileBasedMsgQueue            5         5         0         0
>>
```

If a specific queue name is given, details for that one queue are returned (which will contain a list of specific messages in the queue, with the characteristics of each, if there are any):

```
>> qi SERVER=CENTRAL

  Name:           SERVER=CENTRAL
  Class:          FileBasedMsgQueue
  Current time:   Wed May 22 16:59:21 EDT 2002

  Total msgs added:  5
  Total chained:     0
  Total expired:     0
  Total pulled:      5
  Total rejected:    0
  Total processed:   5
  Total remarked:    0
  Last relevant action: Processed
  Last relevant msg:    GetCogNameListMsg(tid=8336)

    No messages currently in queue
>>
```

3.3.16.17. query [sql]

A simple command that allows arbitrary queries against the database and returns the results in a cleanly formatted block. The query is performed against the same database that the AppServer is using.

```
>> query SELECT * FROM Operator

   ID                   PERSONID             PERSONCOGID USERNAME
   ==================== ==================== =========== =============
   0                    0                    0           none
   1                    1                    2           user1
   2                    2                    2           user21
   3                    3                    2           user3
   4                    4                    2           user4
   5                    5                    2           user5
   6                    6                    2           user6
   7                    7                    2           user7
   8                    8                    2           user8
   9                    9                    2           user9
   10                   10                   2           user10
   11                   11                   2           user11
   12                   12                   2           user12
   13                   13                   2           user13
   14                   14                   2           user14
   15                   15                   2           user15
   16                   16                   2           user16
   17                   17                   2           user17
   18                   18                   2           user18
   19                   19                   2           user19
   20                   20                   2           user20
>>
```

The query is sent directly through JDBC with no interim parsing, and the results are display with column names the correspond to the metadata from the query that was issued.

3.3.16.18. restartsvcs

Restarts all installed services on the AppServer. The AppServer itself is not recycled – just the CMISServices that are running within it. The stop( ) method is called on all services, and the boot-up process for services is followed again. All statistics for the services (see the **svci** command) are reset as well.

3.3.16.19. shell [cmd]

Runs a shell command on the AppServer, similar to running the same command at the command prompt in the server room.

Note: This command, because it operates on the AppServer, can be dangerous and should be used with extreme caution!

3.3.16.20. shutdown [rtid]

Sends an "immediate shutdown request" message to all *currently connected* operators that match the given RoutingID. In this example, the desktop running on the workstation supporting *jkessler* is closed remotely:

```
>> oplist

Operator               Workstation Server
====================== =========== ==========================
1000/jkessler          5688        8099  [WP-KesslerJ2:2101]
2/user16               5617        4502  [WP-WardenJ-c:2101]

>> shutdown WS=5688 Ok
Ok
>> oplist

Operator               Workstation Server
====================== =========== ==========================
2/user16               5617        4502  [WP-WardenJ-c:2101]
>>
```

This command presumes that the issuer knows how to formulate a CMIS RoutingID to specify the appropriate targets for the message. A RoutingID of *COG=nnnn*, for example, will send the request to everyone connected who is part of the given COG. A RoutingID of *COG=nnnn/login=yyyy* will go to a specific individual regardless of where his is logged in.

3.3.16.21. startsvc [svcid]

Starts a service that is in a stopped state. The service is identified by its *service id*, which can be obtained by issuing an *svci* command.

3.3.16.22. stopsvc [svcid]

Stops a service that is current running. The service is identified by its *service id*, which can be obtained by issuing an *svci* command.

3.3.16.23. svci

Returns detailed information on services that are currently running on the AppServer, and performance data for messages that have been processed. Sample output:

```
>> svci
  Service                   Started  Status Hits   Avg/Msg    Avg Wait   PoolSz Busy   Peak
  ========================= ======== ====== ====== ========== ========== ====== ====== ======
  Administration Service    true     OK     0      0ms        0ms        16     0      0
  Attachment Service        true     OK     0      0ms        0ms        16     0      0
  Authentication Service    true     OK     0      0ms        0ms        16     0      0
  Common Session Service    true     OK     0      0ms        0ms        16     0      0
  DL Service                true     OK     0      0ms        0ms        16     0      0
  HTMLViewer Service        true     OK     0      0ms        0ms        16     0      0
  IM Service                true     OK     0      0ms        0ms        16     0      0
  Mapping Persistence Servi true     OK     0      0ms        0ms        16     0      0
  Monitor Service           true     OK     0      0ms        0ms        16     0      0
  Reports Data Retrieval    true     OK     0      0ms        0ms        16     0      0
  Session Cleanup Service   true     OK     0      0ms        0ms        1      0      0
  Session Member Tracking S true     OK     1      109ms      0ms        16     0      1
  SNR Service               true     OK     0      0ms        0ms        16     0      0
  Synchronization Service   true     OK     0      0ms        0ms        16     0      0
  TIE Service               true     OK     0      0ms        0ms        16     0      0

  Message Class                          Hits     Avg Time   Avg Wait
  ====================================== ======== ========== ==========
  GetCogNameListMsg                      1        141ms      0ms
  OperatorConnectedMsg                   2        124ms      0ms
  OperatorDisconnectedMsg                2        726ms      0ms
  RequestAuthenticationMsg               1        469ms      0ms
  RequestDataSyncMsg                     1        141ms      0ms
  SessionRequestMsg                      2        172ms      0ms

Message Thread Pool:

  Current pool size:  5
  Min pool size:      5
  Max pool size:      32
  Peak pool size:     5
  Free thread count:  5
  Busy thread count:  0
  Peak busy count:    1
  Total requests:     5
  Avg. wait time:     0.0ms/request
  Times blocked:      0
  Currently blocked:  0 (max=0)
  Thread timeout:     60000ms
  Thread priority:    MAX_PRIORITY

  Thread Name                            Dispatches Most Recent Current State
  ====================================== ========== =========== ====================
  Message Dispatch ThreadPool:1          0          0ms         Awaiting Dispatch
  Message Dispatch ThreadPool:2          0          0ms         Awaiting Dispatch
  Message Dispatch ThreadPool:3          0          0ms         Awaiting Dispatch
  Message Dispatch ThreadPool:4          0          0ms         Awaiting Dispatch
  Message Dispatch ThreadPool:5          5          1200587ms   Awaiting Dispatch
>>
```

3.3.16.24. svrdump [x] | [new]

Allows the operator to examine dumps that have been taken automatically by the AppServer as the result of a SEVERE error in the log. If issued without a parameter, this command returns a list of dumps that are available on the server. This output looks like the following:

```
>> svrdump
The following dumps have been taken:
1. Dump @ 2002-04-22 04h 48m 23s 0039ms.txt
2. Dump @ 2002-04-22 04h 48m 30s 0523ms.txt

>>
```

A specific dump can be examined more closely by issuing **svrdump [x]**, where **x** is the dump number as it appears in the numbered listing shown in the output above. The process of

retrieving a full dump file may take a minute or two, but this delay is normal and should be expected. The **svrdump new** command causes the server to generate a new dump, which will subsequently appear in the **svrdump** list demonstrated above.

3.3.16.25. tlci

Returns information from the transport layer on operators that are currently connected. These metrics describe the physical attributes of the connection in great detail. Some of the following output has been truncated due to space constraints:

```
>> tlci


OperatorConnectionTracker:
  Login                COG      Workstation ServerID
  =================== ======== =========== ============================
  user16               2        5617        4502 [WP-WardenJ-c:2101]
  jkessler             1000     5688        8099 [WP-KesslerJ2:2101]
  user1                2        5688        7918 [WP-NATHANIELD-C:2101]


  Connected Clients List:

  RoutingID                              Version                PollCount PollRate   Bytes Sent Bytes Recd Errors LastMsgIO
  ====================================== ====================== ========= ========== ========== ========== ====== ============
  COG=1000/LOGIN=jkessler/WS=5688        MsgIOProtocolVersion100 2876      0.8/s      2.9K       3.7K       0      178417ms
>>
```

3.3.16.26. tlgi

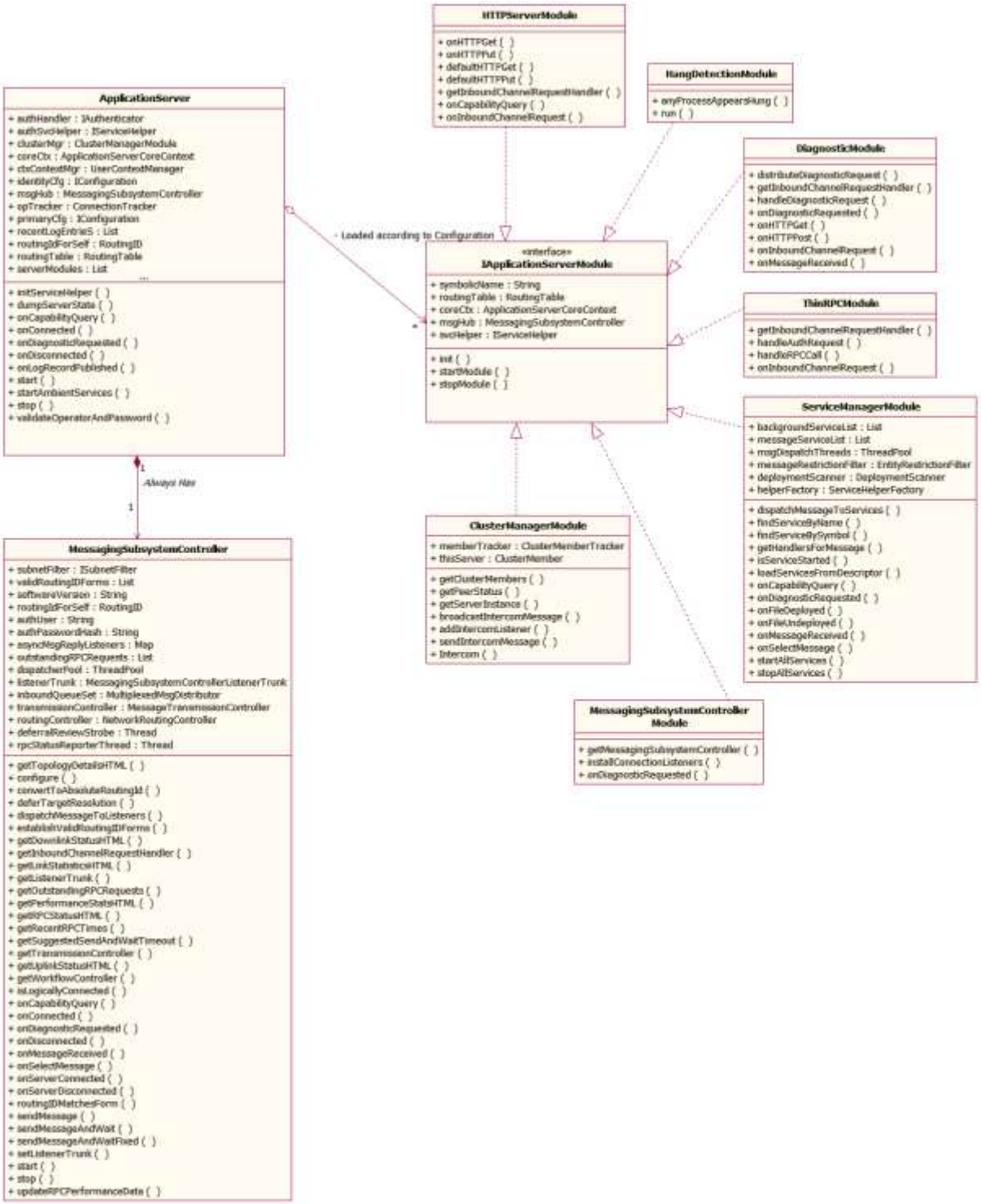
Returns general metrics from the transport layer, which is the portion of the AppServer that deals with message transmission to/from connection operators running the CMIS desktop.

```
>> tlgi
Transport Manager:
  Alive:                    true
  Current client count:     1
  Min connection interval:  2000
  Message I/O port:         2101
  Message I/O pipe size:    32
  Min polling interval:     20
  Max ACTIVE interval:      100
  Max IDLE interval:        3000
  Clients become IDLE At:   60000ms
  IDLE poll maxout period:  300000ms
  Errors before disconnect: 0
  Bad client disable period: 5000
  Disconnected clients:     0
  Disallowed IPs:           0
  Disallowed clients:       0
  Recent connections:
        127.0.0.1 (127.0.0.1)

  Overall poll count:       4061
  Overall poll rate:        5.6/s

Polling ThreadPool:

  Current pool size:  1
  Min pool size:      1
  Max pool size:      32
  Peak pool size:     1
  Free thread count:  1
  Busy thread count:  0
  Peak busy count:    1
  Total requests:     1862
  Avg. wait time:     0.0ms/request
  Times blocked:      0
  Currently blocked:  0 (max=0)
  Thread timeout:     60000ms
  Thread priority:    NORM_PRIORITY

  Thread Name                            Dispatches Most Recent Current State
  ====================================== ========== =========== ===================
  Socket I/O Polling ThreadPool:1        1862       78ms        Awaiting Dispatch

  Message Class                          Total Sent Avg Bytes Avg Time   Send Errs  Total Recd Avg Bytes Avg Time   Rcv Errors
  ====================================== ========== ========= ========== ========== ========== ========= ========== ==========
  GetCogNameListMsg                      0          -         -          -          1          217       15ms       0
  RequestAuthenticationMsg               0          -         -          -          1          434       15ms       0
  RequestDataSyncMsg                     0          -         -          -          1          341       0ms        0
  RequestRetrieveOperatorsMsg            0          -         -          -          1          324       15ms       0
  ResponseAuthenticationMsg              1          1169      47ms       0          0          -         -          -
  ResponseDataSyncMsg                    1          435       16ms       0          0          -         -          -
  ResponseGetCogNameListMsg              1          638       15ms       0          0          -         -          -
  SessionChangeEventMsg                  1          951       32ms       0          0          -         -          -
  SessionRequestMsg                      0          -         -          -          2          648       8ms        0
  SessionResponseMsg                     2          739       24ms       0          0          -         -          -
  TIECorrelatedRequestMsg                0          -         -          -          1          312       0ms        0
>>
```

4. UML Diagrams

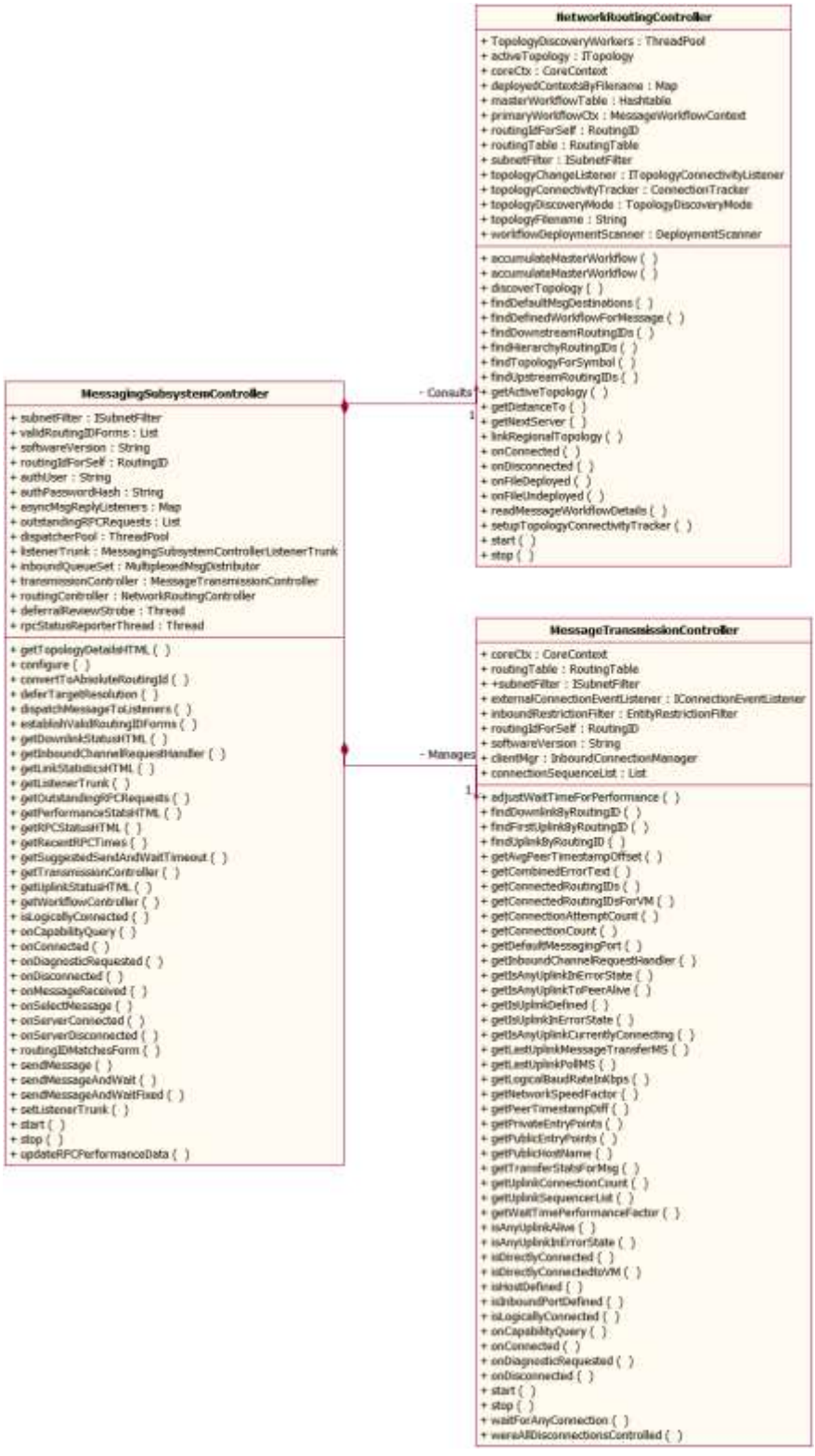
4.1. Application Server

4.1.1. Overview

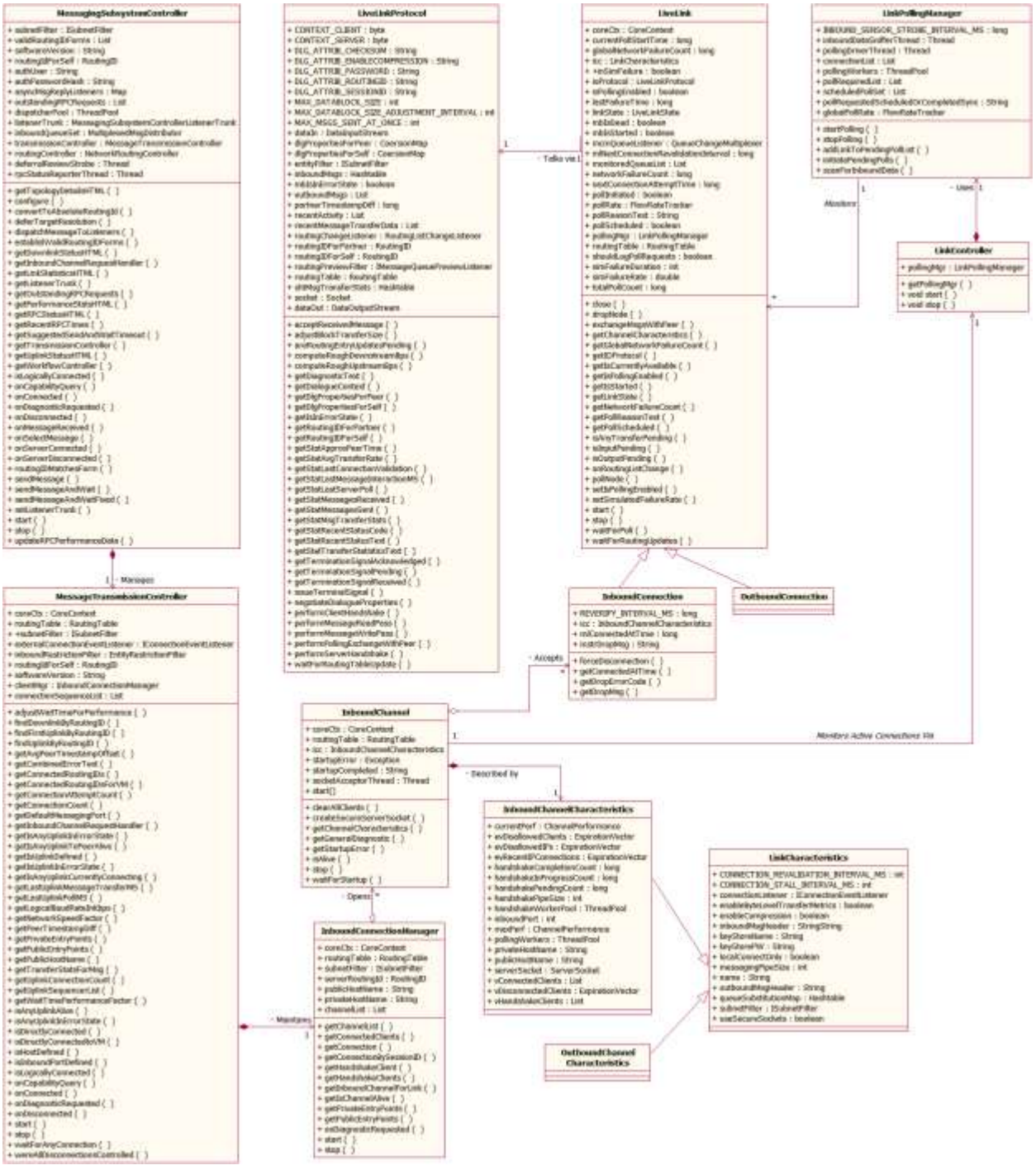


4.2. Messaging Subsystem

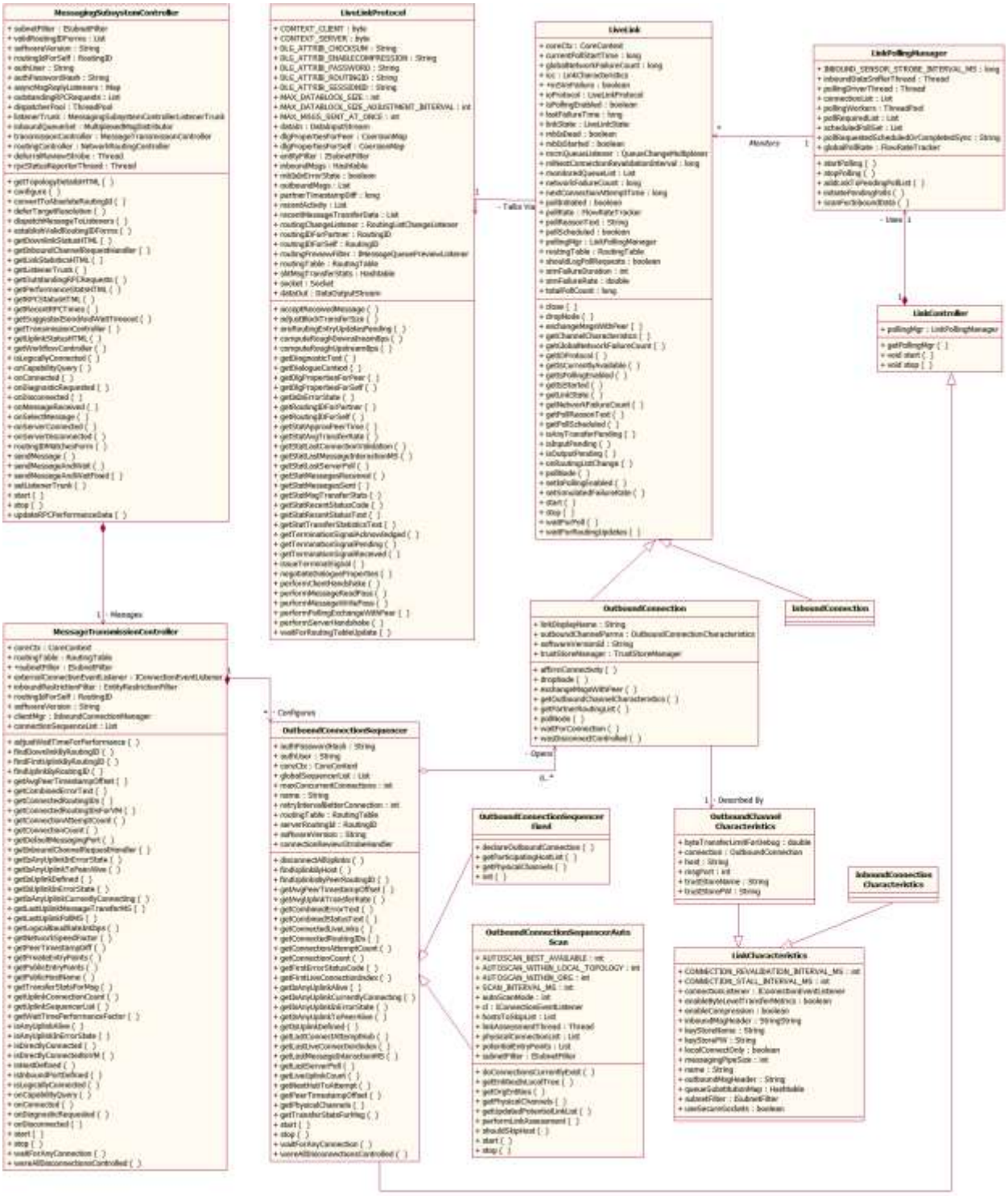
4.2.1. Overview



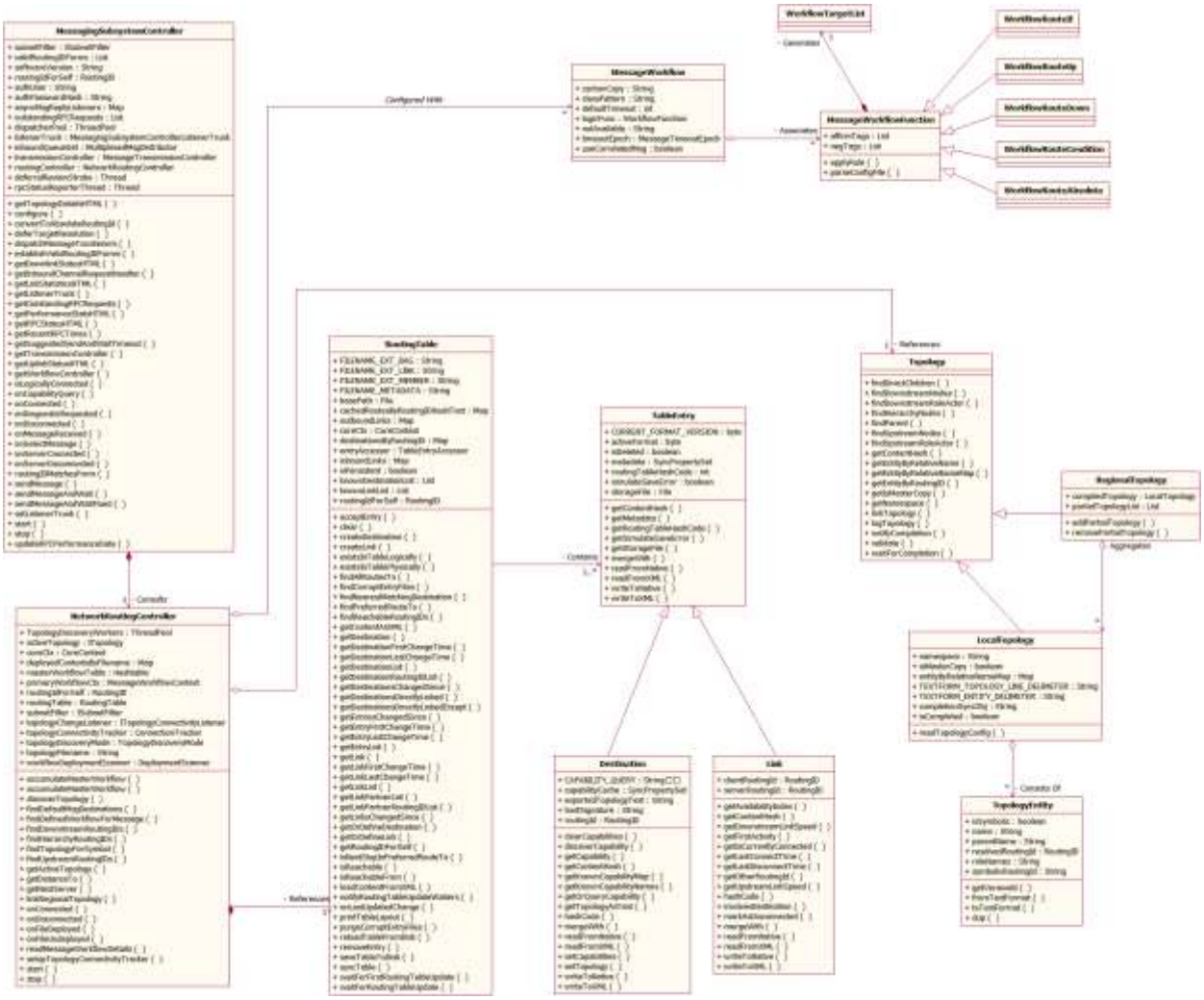
4.2.2. Inbound Connections



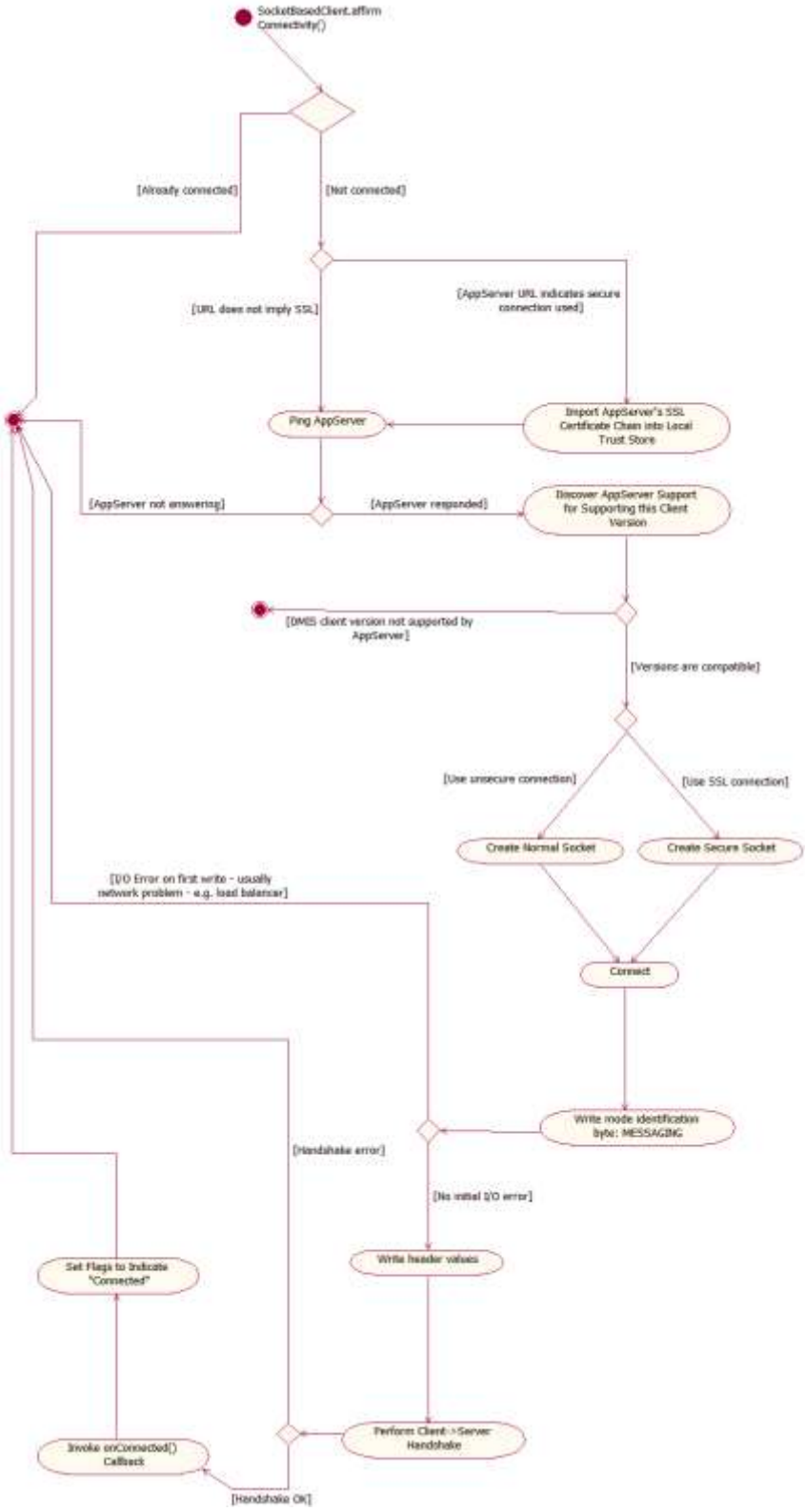
4.2.3. Outbound Connections



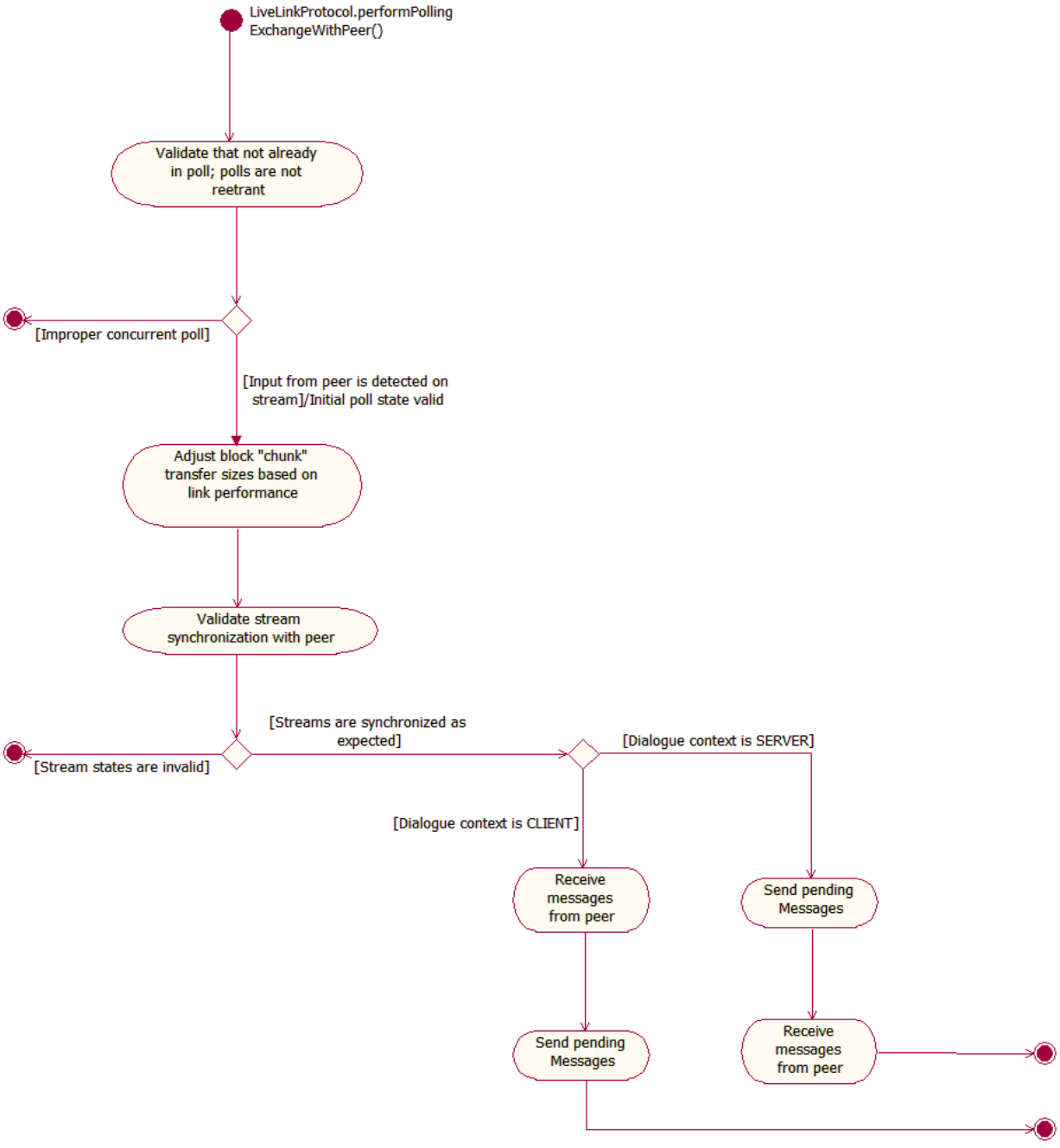
4.2.4. Inbound Connections



4.2.5. Socket Level Connection Process



4.2.6. Poll-Level Message Exchange



This Page Intentionally Left Blank

This Page Intentionally Left Blank