

Debugging Guide for Genuine Channels v2.5



Genrix

Genuine solutions for
successful development

This document describes approaches and means that simplify debugging and supporting distributed applications based on Genuine Channels. We will discuss how you can check the behavior of your distributed application and how you can quickly reveal the cause of occurred problems.

1. GENERAL DEBUGGING STRATEGIES	3
2. LOG FILES.....	3
3. WRITING LOG FILES	4
3.1. Logging Options	4
3.2. Logging Into a File	5
3.3. Logging Into Memory.....	6
3.4. Common Recommendations.....	6
3.5. Writing your own records.....	7
4. GENUINE LOG VIEWER	7
4.1. Command Line options	9
4.2. Downloading records from the server	9
5. ENTITIES	11
5.1. HostInformation	14
5.1.1. Lifetime	14
5.1.2. Available Information	14
5.2. Security	15
5.2.1. Lifetime	16
5.2.2. Available Information	17
5.3. Connections.....	17
5.3.1. Lifetime	18
5.3.2. Available information	18
5.4. Messages	19
5.4.1. Lifetime	20
5.4.2. Available Information	20
5.5. Invocations.....	26
5.5.1. Lifetime	27
5.5.2. Available Information	27
5.6. Broadcast Engine	28
5.6.1. Lifetime	31
5.6.2. Available Information	31

1. General Debugging Strategies

.NET Remoting is designed to be a versatile system and provides several methods for registering and obtaining objects. To reduce the number of possible problems, you are highly recommended to use the best practices described in the Beginner's Guide, Programming Guide and Knowledge Base. For example, you are not recommended to use SAO or CAO objects, sinks with thread-unsafe implementation. Marking the main method by the STAThreadAttribute attribute without necessity is certainly a bad idea as well.

When you encounter a problem, you usually have an exception with a stack trace and the situation in which it occurred. The full list of exceptions generated by Genuine Channels is available in the Programming Guide. Usually, the text of an exception is informative and tells you the cause of the problem.

After that, if neither the exception nor its description helps, it is a good idea to enable logging and try to reproduce the problem. Log files contain a complete list of events occurred during application execution and allow you to see the situation as a whole. You will see what connections were established, what invocations with what parameters and results were made and what Security Sessions were used.

In addition to log files, you might want to install a good sniffer to control what content is transmitted by your solution. We can recommend you Ethereal available at <http://www.ethereal.com>. It supports both TCP and HTTP protocols. In general, log files contain the transmitted content and provide better options for analysis.

2. Log Files

Usually, it is not convenient to control ongoing events at run time. .NET Remoting invocations are too quick and, on the other hand, it can take up to several hours to reproduce appropriate problems. Log Files allow us to track down all events occurred during execution for deferred analysis.

The Log File is a set of network, security, message and diagnostic records written by Genuine Channels. These events are written in the binary form to minimize expenses on writing and keeping the records. In this document, we will discuss how you can write log files, analyze them with the help of Genuine Log Viewer, convert them into XML format, add your own debugging records.

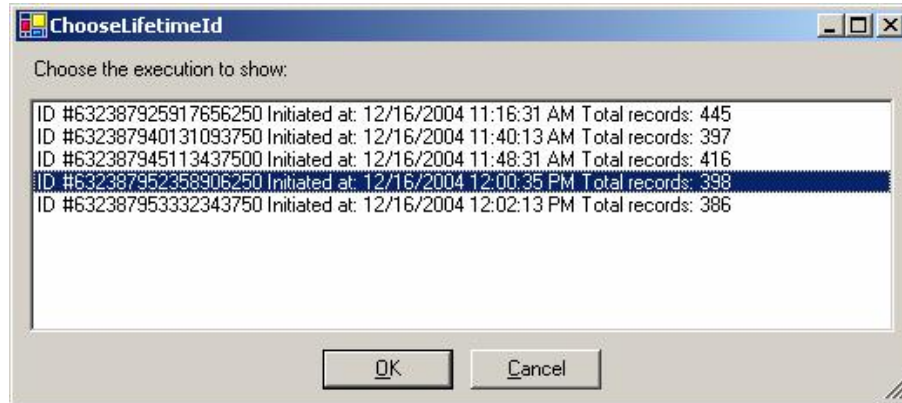
We will think of Genuine Channels in terms of a big finite-state machine. We will enumerate all states passed by Genuine Channels when a certain operation is performed. As soon as we encounter a problem, we can check on what stage the problem occurred and, as a result, we can figure out the list of possible causes of incorrect behavior.

Each state record is represented as a set of different fields depending on the event type. Each event can be either successful or unsuccessful. Unsuccessful events mean that an exception or something bad occurred before things come to a particular state. Unsuccessful events are usually displayed in red and contain exceptions and the descriptions of problems. Successful events mean that everything goes OK and things come to a particular state as expected.

Usually each event includes the DateTime when it occurred, the log category, the type of the event, the alerter that wrote the event, the exception associated with the event, the thread

identifier and the thread name where the event occurred and a number of other fields depending on the event type.

Records written by different application instances are united into record sequences. If you run the same application several times, the log file will contain different execution sequences. When you try to load such a file, you will be asked what sequence you want to view.



3. Writing Log Files

There are two supported approaches of writing event records. You can either put everything into a file or keep all records in memory and download them at run time. The second way is better if you do not have rights to create or write into a file or if you want to observe a situation during execution.

3.1. Logging Options

You can narrow down the set of written records by specifying what records you are interested in. To do this, you can specify a verbosity level for each category. Each log category is associated with a character. The verbosity level is specified by a digit. The zero level means that no records will be written. The first level allows records without binary content. The second level allows binary content to be included into the log. The third level makes it possible to include records containing personal or security information.

Here is a list of all possible categories.

Log Category	Letter and Default Level	Description
Connection	C1	This category contains connection notifications.
ChannelEvent	E1	This category contains notifications describing what Channel Events are raised.
MessageProcessing	M2	This category contains notifications related to message processing.
ImplementationWarning	I3	This category contains notifications generated by Genuine Channels in unexpected or unwanted situations.

Security	S1	This category contains security notifications.
BroadcastEngine	B2	This category contains notifications describing Broadcast Engine invocations.
Transport	T1	This category contains general transport notifications.
DXM	X1	This category contains notifications generated by Direct Exchange Manager.
HostInformation	H1	This category contains notifications related to HostInformation objects serving Security Sessions, Client Sessions and network information with regard to remote hosts.
AcceptingConnection	A1	This category contains notifications informing you about inbound connections and what local end points are served.
Debugging	D2	This category contains notifications generated by user code.
Version	V1	This category contains notifications revealing the environment information such as the Genuine Channels version, the CLR version, the version of OS and so on.
LowLevelTransport	L0	This category contains notifications generated by the low-level transport. These records simplify Genuine Channels debugging. By default, these records are not written.
StatisticCounters	N1	This category contains notifications generated by the Genuine Thread Pool and the Buffer Pool. These messages contain information about resource consumption.

For example, the string “C1E1M2I3S1B2T1X1H1A1D2V1L0N1” represents the default logging settings and can be obtained via the *GenuineLoggingServices.DefaultLoggingOptions* property. If you do not specify some categories, their default verbosity level will be used.

To enable low-level transport messages, you can enumerate all categories: “C1E1M2I3S1B2T1X1H1A1D2V1L2N1”; alternatively, you can simply specify that you need low-level messages with content: “L2”.

3.2. Logging Into a File

To enable logging into a file, execute the following line:

```
try
{
    Belikov.GenuineChannels.Logbook.GenuineLoggingServices.SetUpLoggingToFile(fileName, options);
}
catch
{
}
```

where *fileName* is a path plus the base part of the destination file name and *options* is a string or a null reference that determines the verbosity level for categories as it is described in the section above. If the specified file cannot be created or is already locked by another application, an exception will be raised.

Alternatively, if you use Genuine Channels compiled in the DEBUG mode, you can specify the ***EnableGlobalLoggingToFile="nameOfTheFile"*** channel parameter. Logging options can be specified by the ***LoggingParameters*** channel parameter.

The current date and the ***.genchlog*** extension are automatically added to the file name. For example, if you specify ***ServerLog*** as a file name, you will get ***ServerLog.2004-10-04.genchlog***, ***ServerLog.2004-10-05.genchlog*** and so on.

In general, you can inspect any file. In this case, you will lack information about connections, messages and HostInformations. You can always concatenate two files with the help of the copy command:

```
copy ServerLog.2004-10-04.genchlog+ServerLog.2004-10-05.genchlog+ ServerLog.2004-10-06.genchlog oneBigFile.genchlog
```

Please note that the current version of the Log Viewer parses absolutely all records while loading a file. As a result, loading of big files may take a significant amount of time and memory. A 300Mb file is usually loaded for 7-8 minutes.

3.3. Logging Into Memory

To enable logging into memory, execute the following line:

```
Belikov.GenuineChannels.Logbook.GenuineLoggingServices.SetUpLoggingToMemory(maximumSize, options);
```

If you use Genuine Channels compiled in the DEBUG mode, you can specify the ***EnableGlobalLoggingToMemory="maximumSize"*** channel parameter. Logging options can be specified by the ***LoggingParameters*** channel parameter.

The value of the ***maximumSize*** parameter determines how many bytes can be allocated for log records. As soon as the memory size occupied by log records exceeds this value, logging gets frozen. After a portion is downloaded and some space is available again, the logging is resumed.

You can enable or disable logging remotely.

3.4. Common Recommendations

1. To stop logging, use

```
Belikov.GenuineChannels.Logbook.GenuineLoggingServices.StopLogging();
```

If you use Genuine Log Viewer and it downloads content from a remote host, you can enable or disable logging via ***Connection > ToggleLogging***.

2. As mentioned earlier, each event has a thread ID and a thread name. By default, a thread has an empty name. Genuine Thread Pool always names its threads as "GC.GTP.#X" where X is the ordinal number. It is a good idea to assign names to threads if you want to understand in what thread an event happened. Use the following:

```
System.Threading.Thread.CurrentThread.Name = "MyThreadName";
```


3. The lion's share of each log file is occupied by binary content. If you specify "MII3B1D1", expenses on writing as well as the total file size will be considerably reduced. Though in this case, you will not be able to inspect the content of transmitted messages.
4. If you have only a part of a log file, consider that each record always starts from the AA 4B E7 78 byte sequence. Therefore, if you want to read it with the help of Genuine Log Viewer, cut off all bytes from the beginning until you find this sequence.
5. When Log Viewer downloads content from a remote host, the log file naturally contains records produced by the Log Viewer to Server interaction.
6. Try to avoid situations when two or more applications are trying to write records into the same file. FileWritingStream locks the file for writing, so none but the first application can write anything to the log. However, other applications will check whether the file is available every 15 seconds. If the first application exits, the second application will obtain the write access to the file and start writing its records. As a result, you will get several different record sequences in one log file. It is not a problem because they will not be mixed; you will just have to figure out what sequence you actually want to view.

3.5. Writing your own records

The current implementation of the logging system is experimental. It will be surely updated and improved; therefore, do not rely heavily on its functionality. However, if you are resolving a problem that is not easy to understand, you can write additional records to the log file.

```
Exception exception = . . .           // an exception you want to be associated with the record, or a null reference
HostInformation hostInformation = . . . ; // a HostInformation to be associated with the record
Stream stream = . . . ;                // binary content

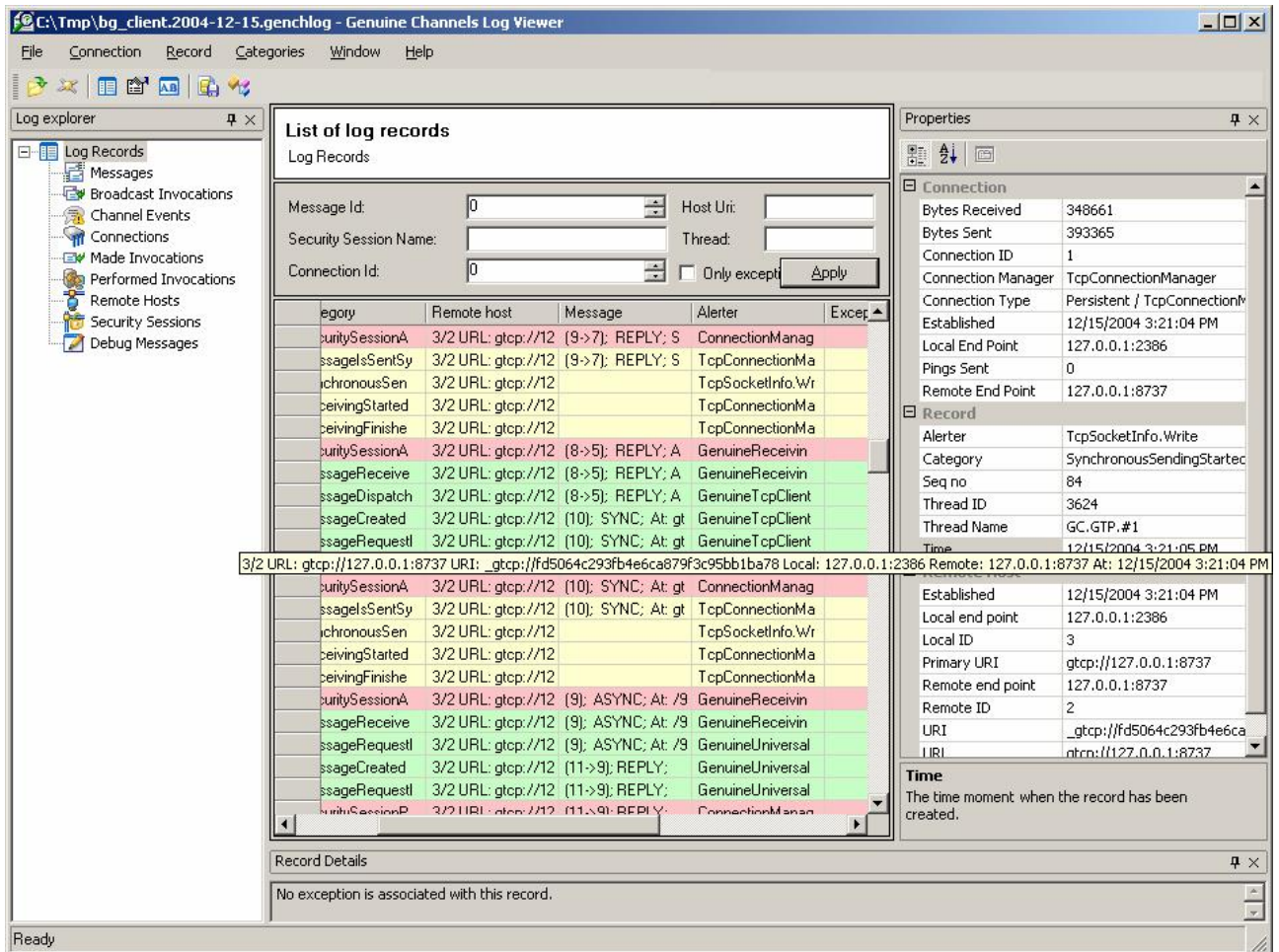
BinaryLogWriter binaryLogWriter = GenuineLoggingServices.BinaryLogWriter;
if ( binaryLogWriter != null && binaryLogWriter[LogCategory.Debugging] > 0 )
    binaryLogWriter.WriteEvent(LogCategory.Debugging, "The name of the place where you put the event",
        LogMessageType.Warning, exception, null, hostInformation, stream,
        GenuineUtility.CurrentThreadId, Thread.CurrentThread.Name,
        null, null, -1, 0, 0, 0, null, null, null, null,
        "My message with {0} several {1} parameters {2}.", "{", "}", ".");
```

As you see, you can store an exception, a HostInformation, a binary content respresented as a stream and a text message.

By default, the Genuine Log Viewer does not display debugging messages. You can enable them in the Category menu.

4. Genuine Log Viewer

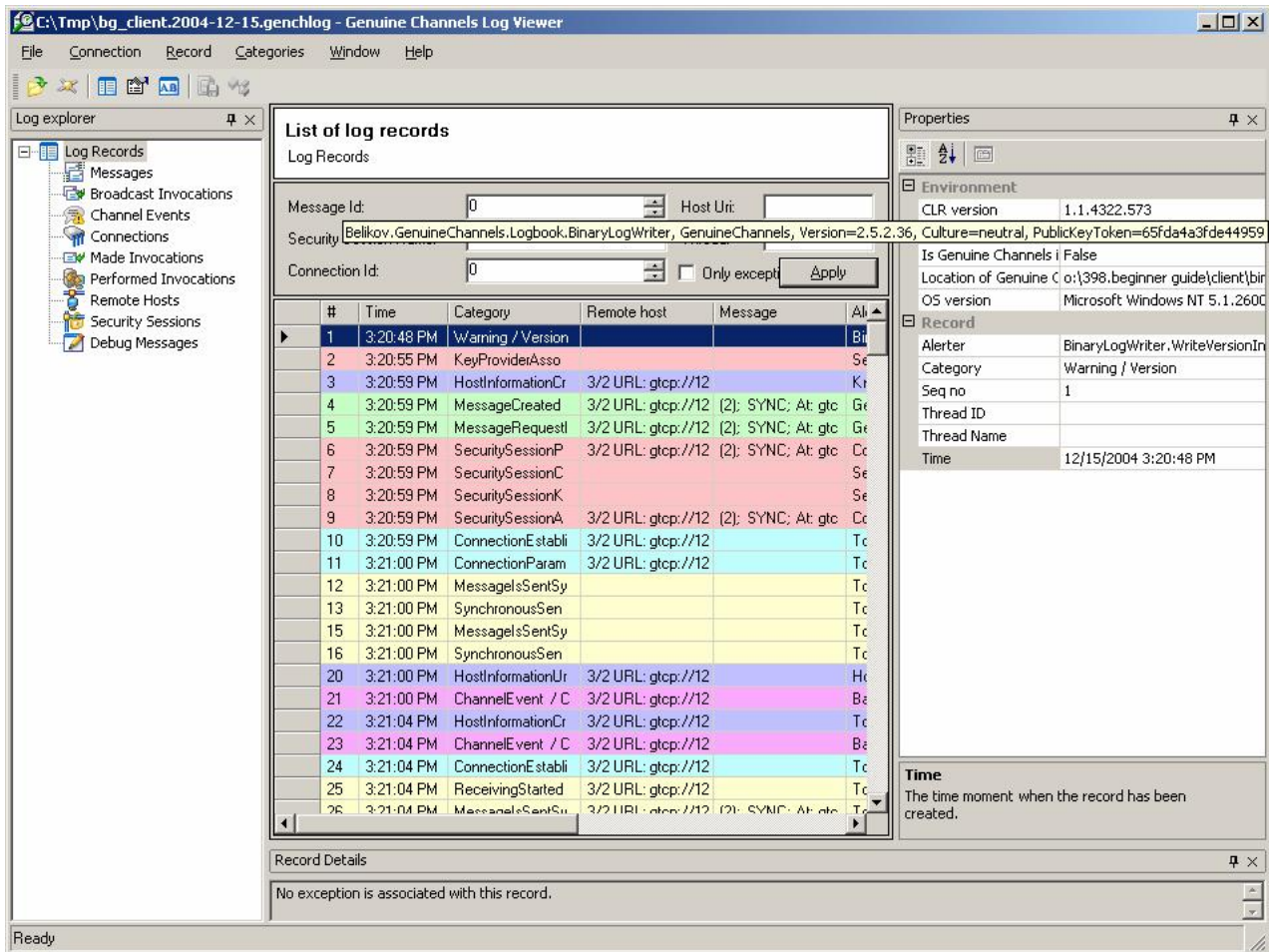
The main window consists of the grid control and three docking panels. The log explorer panel allows you to select entities. The properties panel displays all values associated with the currently selected record. The Record Details panel displays exceptions and brief information about the selected record.



The main window has a message filter that can be used to filter out unnecessary records. For example, the very first thing you might want to do is to check **Only exceptions** and click **Apply**. In this case, the window will contain only records with exceptions and you can walk through all occurred errors quickly.

The Genuine Log Viewer uses colors to separate records by categories. For example, all records connected with messages are green, security warnings are red, transport records are yellow and so on.

The very first record of each log file contains information about the used Genuine Channels, the version of CLR and the environment.



On the screenshot above, you can see that an application uses CLR version 1.1.4322.573, Genuine Channels 2.5.2.36 on Windows XP. The path to GenuineChannels.dll allows to understand where GenuineChannels.dll is located. Sometimes, this information is enough to resolve the problem.

4.1. Command Line options

After you install Genuine Channels, Genuine Log Viewer is associated with the *.genchlog* file extension. You can simply click such a file to open it Genuine Log Viewer. You can load a file into the Log Viewer at any moment by using **File > Load**.

To convert a log file into XML, use

```
"C:\Program Files\Genuine Channels\GenuineLogViewer\GenuineLogViewer.exe" -t filename.genchlog
```

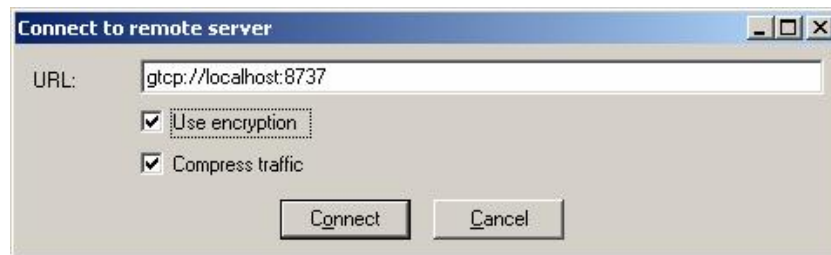
COMMAND LINE

The binary content is not available in the XML format.

4.2. Downloading records from the server

If you enable logging into memory by specifying, for example, *EnableGlobalLoggingToMemory="2500000"*, you can use the Genuine Log Viewer to connect to that server and download log records.

Click Connect and type the address of a remote host. Genuine Log Viewer supports GTCP, GHTTP and Shared Memory transports.

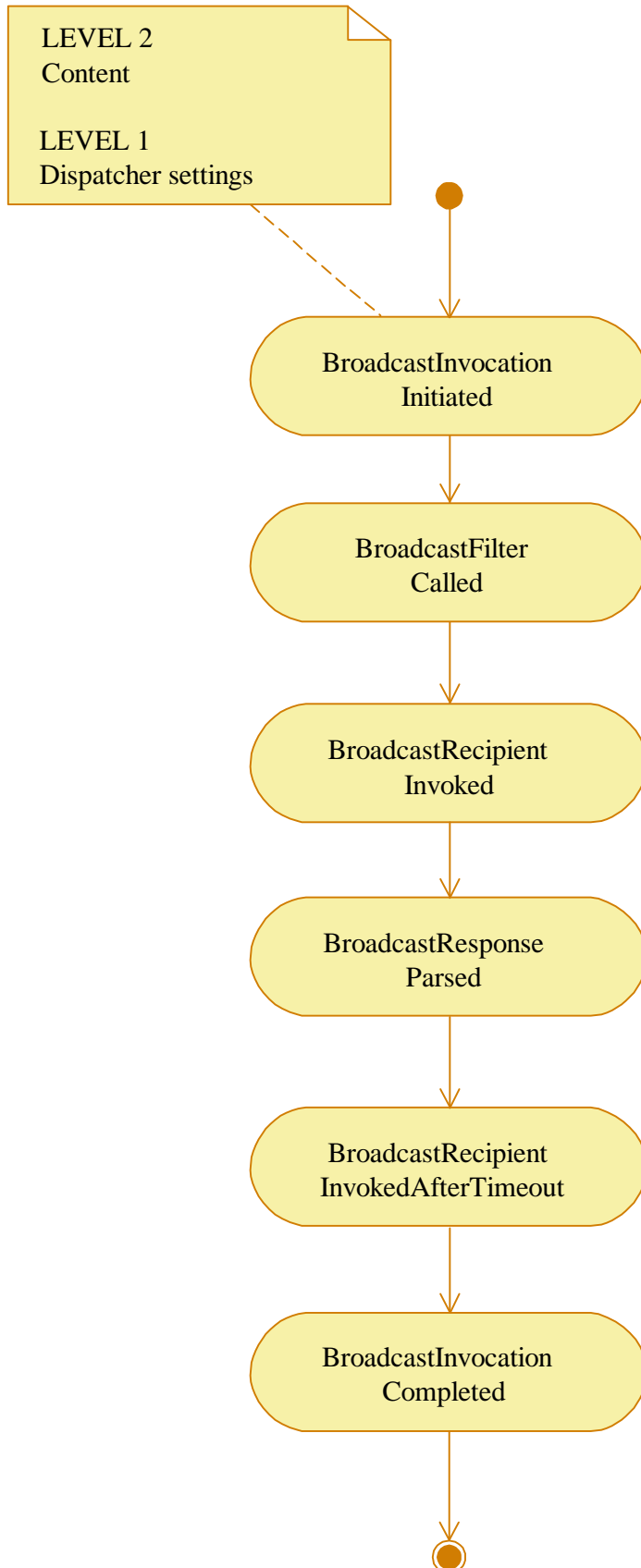


You can enable or disable logging on the remote host via *Connection > Toggle Logging*. To disconnect from the remote host, use *Connection > Disconnect*.

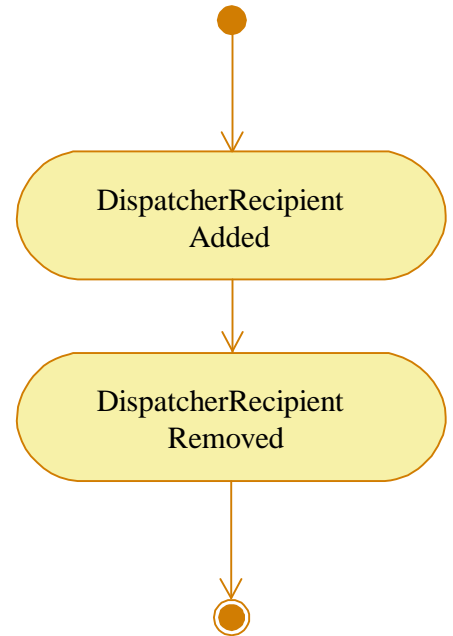
Log records are downloaded every 15 seconds.

5. Entities

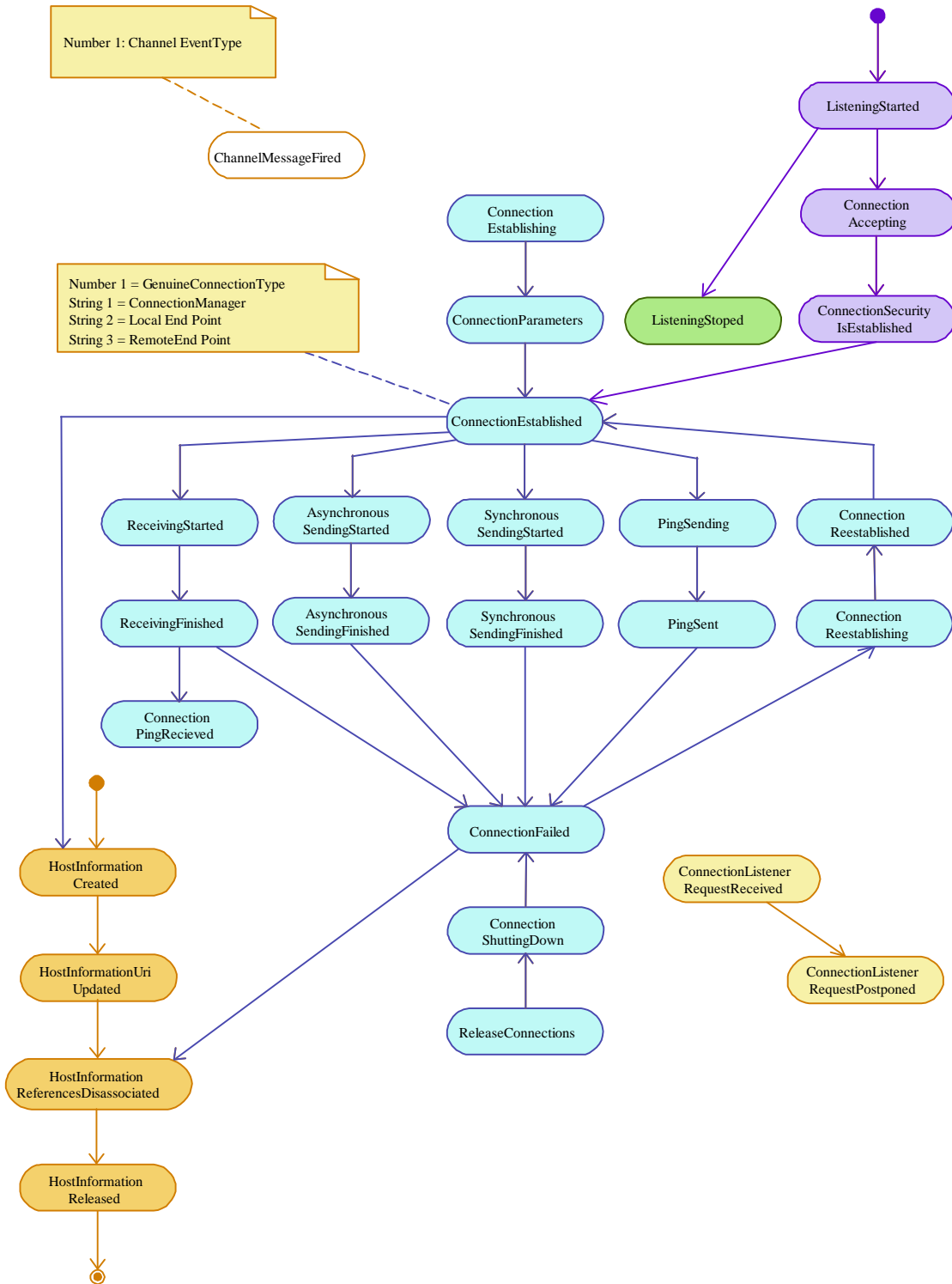
The following diagrams depict all steps passed by entities.



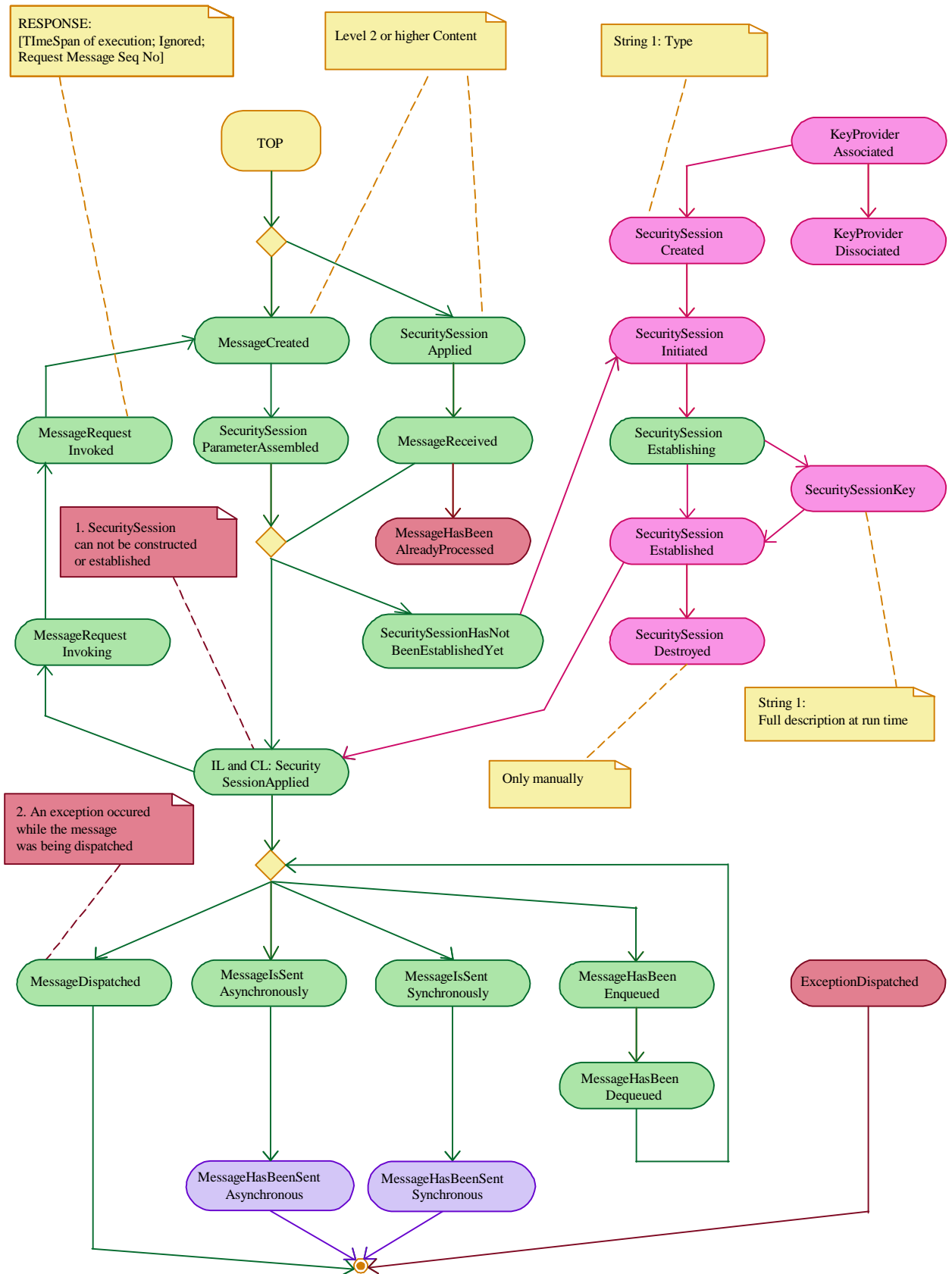
Broadcast Engine state diagram



Transport and HostInformation state diagram

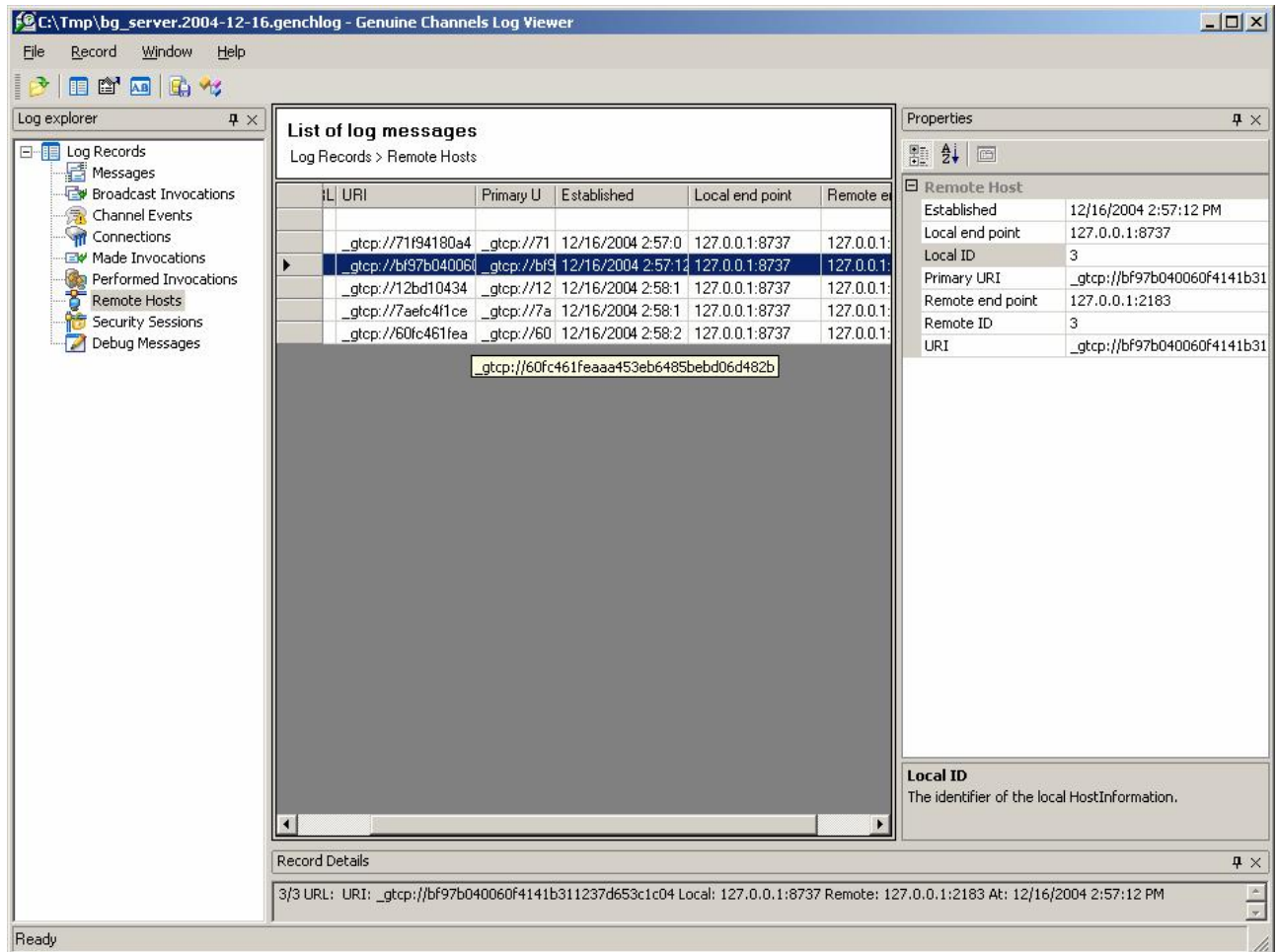


Security and Message Processing state diagram



5.1. HostInformation

Each HostInformation represents a remote host. Each remote host has a unique URI. The first HostInformation usually represents the local HostInformation that works as a container for Security Sessions. It should be ignored.



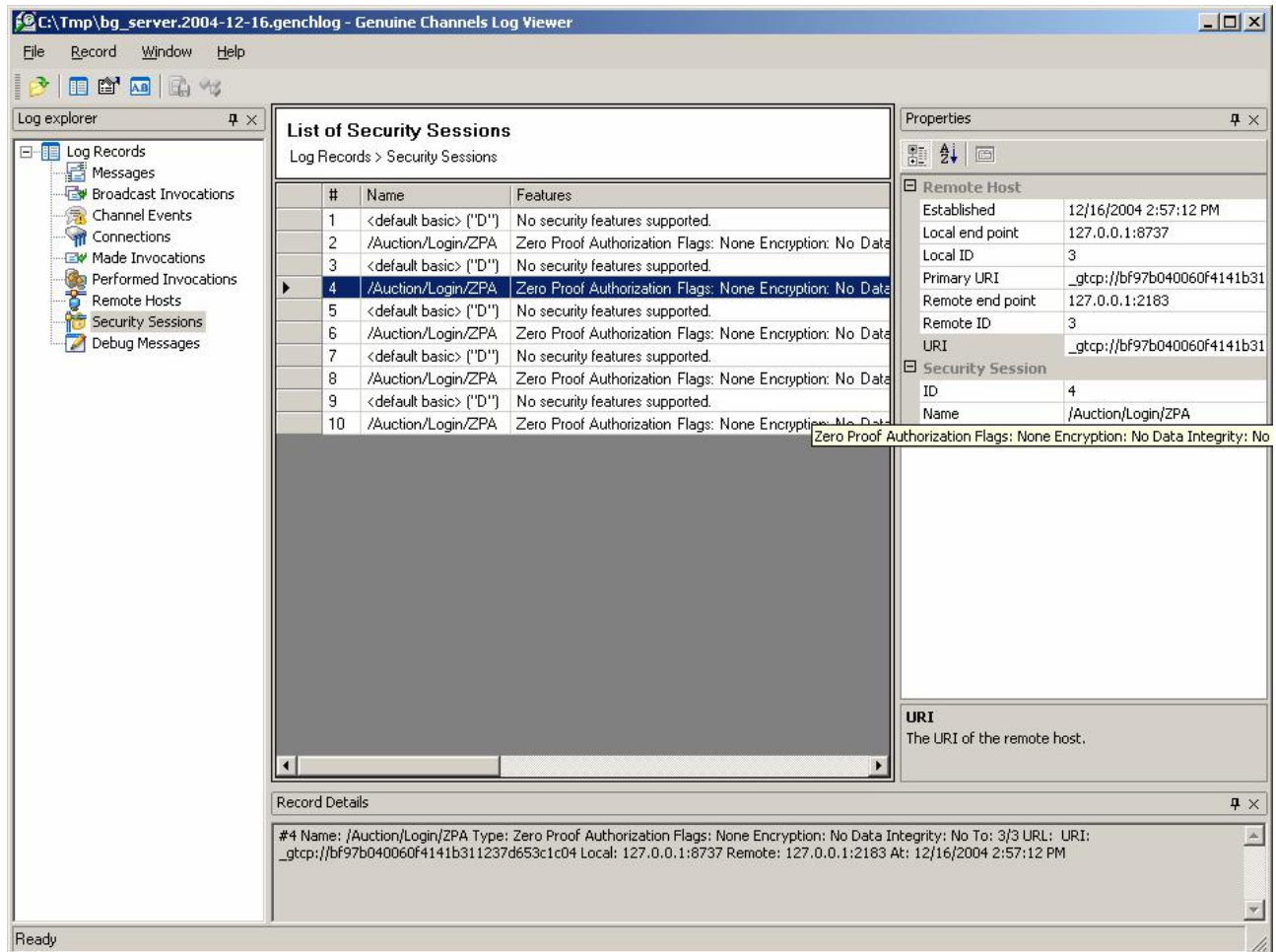
5.1.1. Lifetime

The client application usually writes only *HostInformationCreated* and *HostInformationUriUpdated* events. The server application does not usually write the *HostInformationUriUpdated* event.

5.1.2. Available Information

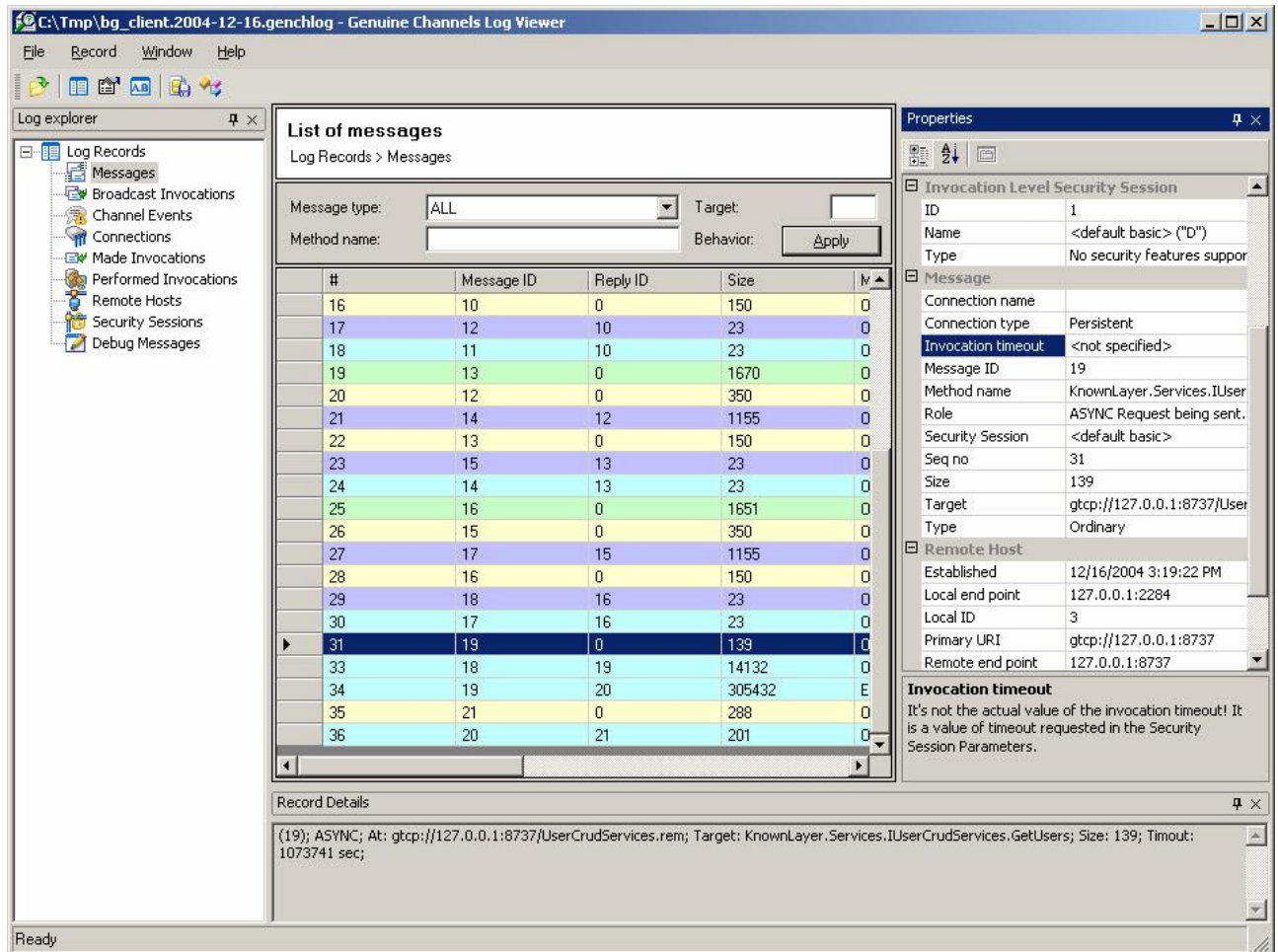
You have the URI, the URL and end points of the last established connection. The URL is available only if the connection is established by the current host. The URL is not available if the connection is accepted. *HostInformationReferencesDisassociated* and *HostInformationReleased* always have exceptions explaining the cause.

5.2. Security



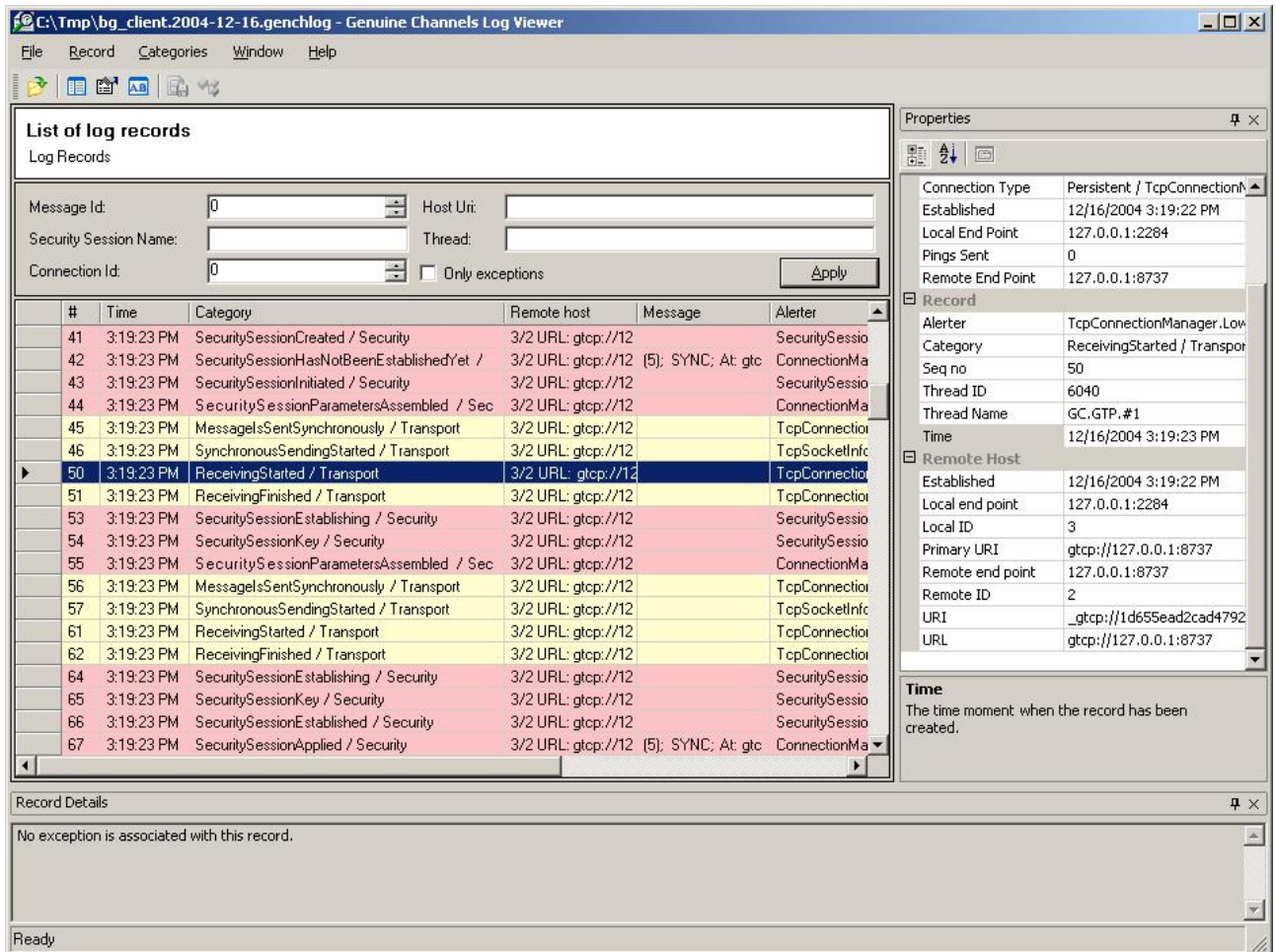
Each Security Session is established with a particular remote host. If you do not use Security Sessions, you will have only basic Security Sessions that do not provide any security functionality.

Security Session Parameters are written with each message. If you need to check what Security Session Parameters have been used during an invocation, open the corresponding request message.



5.2.1. Lifetime

In general, a set of states depends on the type of the Security Session and what role it plays. The following screenshot illustrates how a ZPA Client Security Session is established.



As you see, it passes almost all states and is finally applied to the message 5.

5.2.2. Available Information

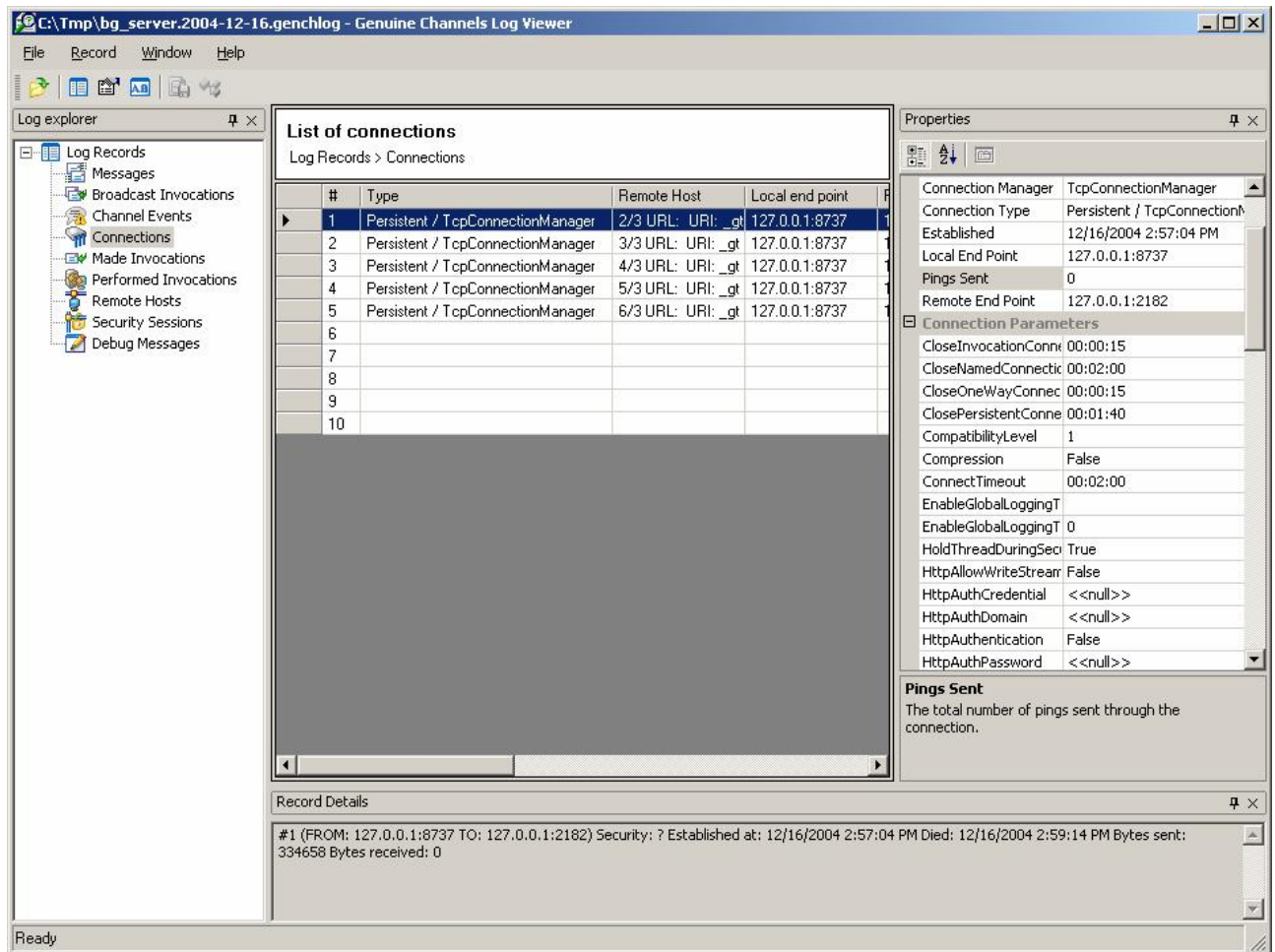
In general, you need to know only the remote host and the name of the Security Session. ZPA and SSPI Security Session provide additional information regarding the usage of encryption, integrity checking and impersonation. For example, SSPI Server Security Session displays “SSPI Package: NTLM Features: Impersonation Win Auth Type: NTLM IsAnonymous: False IsGuest: False IsSystem: False Name: DEVELOPMENT\Dima”. Each Security Session has a unique identifier.

If you use Security Sessions on the connection level, they will be displayed in the connection properties.

5.3. Connections

Log files contain two types of connections: stub connections and physical connections. Stub connections are usually displayed as long empty strings that mean that a physical connection to a specific remote host has been closed. Stub connections collect all messages that should be sent to that host if the connection is being reestablished. If a connection is reestablished, the new physical connection inherits all the properties of the stub connection.

For example, if you start five client applications and then close them, the server's log file will contain five empty lines representing stub connections.



5.3.1. Lifetime

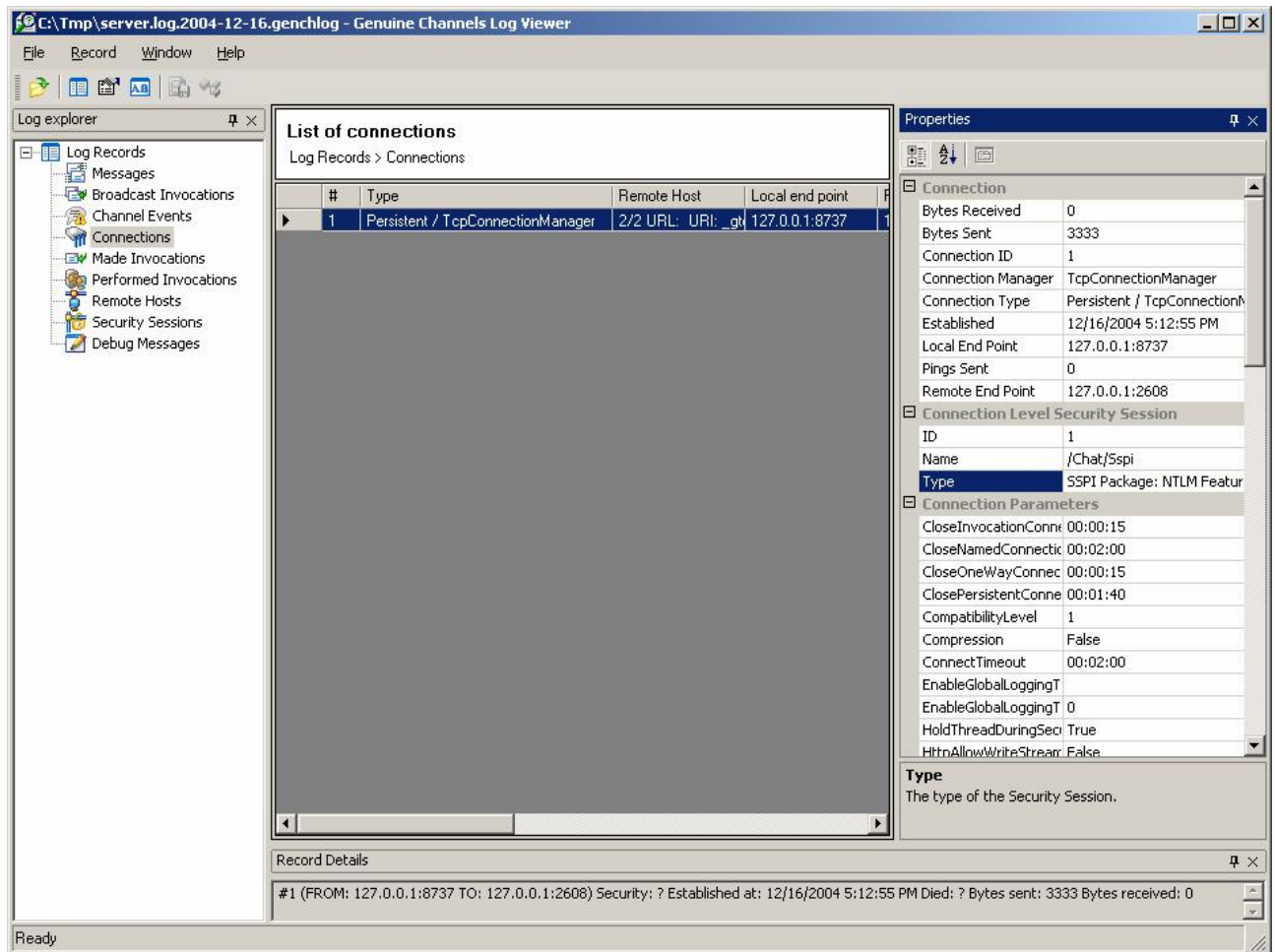
Connections very seldom follow the rules stated on the state diagram. Many connection events are skipped since version 2.5.1 because such actions as sending or receiving content, pings, messages are always duplicated by message, transport and security records. In some places, logging stuff just created too many records so it was hard to read log files. As a result, connection entities usually produce only initialization and termination notifications and some events that are not duplicated by other entities.

You can always enable low-level transport notifications (“L3”) that notify you about operations with sockets and web requests. In this case you will get socket.Receive, socket.BeginReceive, socket.Send and such kind of operations as separate records.

5.3.2. Available information

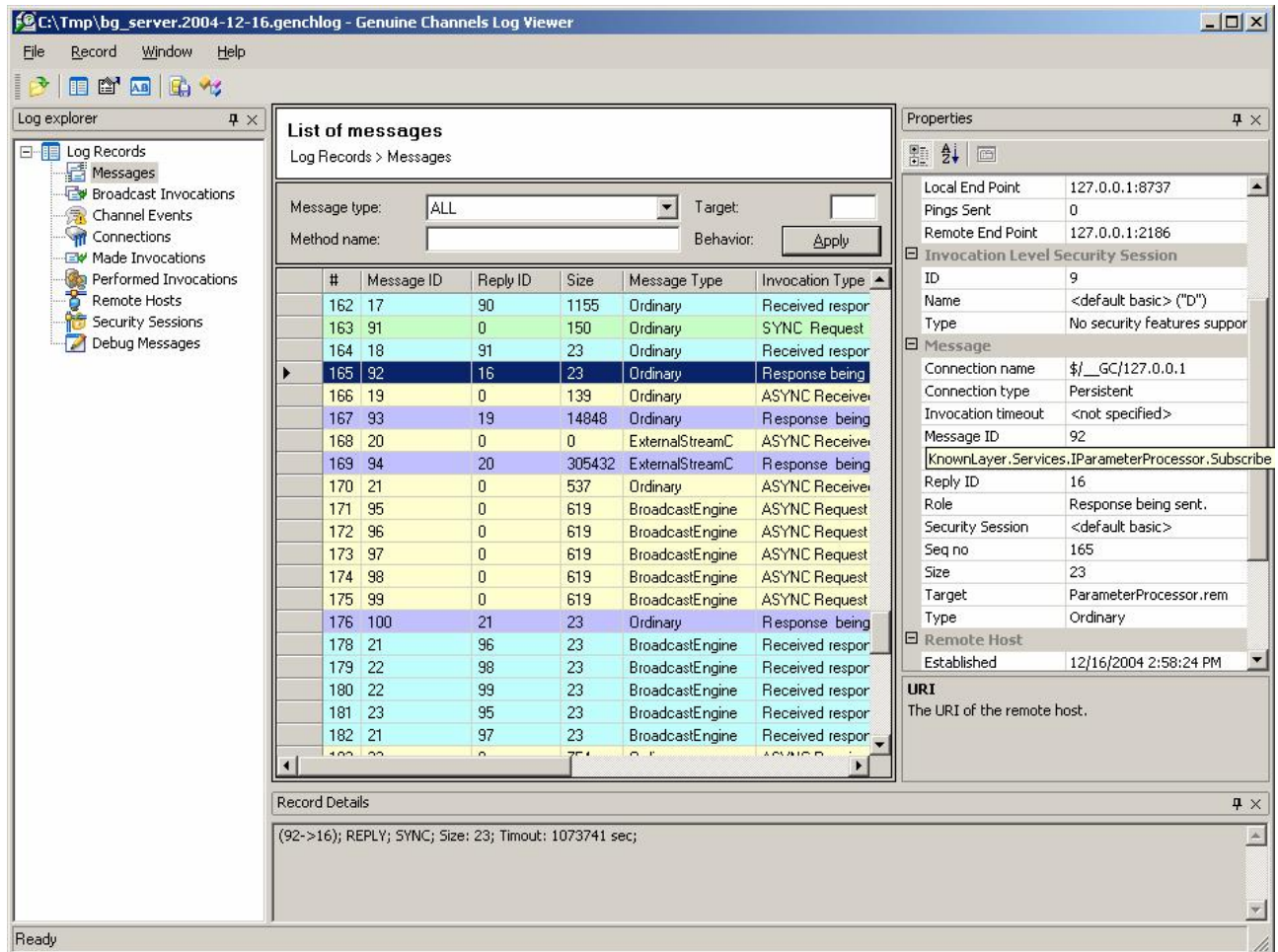
Transport Context parameters are written and available every time a connection is established or accepted. Genuine Log Viewer calculates network activity based on available information about messages. As a result, the number of sent or received bytes does not have any actual relation to the real traffic!

If you use Security Sessions on the connection level, they will be displayed in the connection properties.



5.4. Messages

The transport layer supports different types of messages. On the screenshot below you can see *Ordinary*, *BroadcastEngine* and *ExternalStreamConsumer* messages.



Ordinary messages are generated by .NET Remoting. The Broadcast Engine marks its messages as **BroadcastEngine**. Besides **Ordinary** and **BroadcastEngine**, you can find **ExternalStreamConsumer** (DXM), **TrueBroadcast** (IP Multicasting), **Custom** (pings and connection checkins).

Ping messages sometimes look like ordinary .NET Remoting invocations with ReplyID = -3.

5.4.1. Lifetime

Ordinary messages nearly always behave exactly as it is displayed on the diagram. Some other types of messages do not always follow the rules. Messages created by the Broadcast Engine are discussed in section 5.7.

5.4.2. Available Information

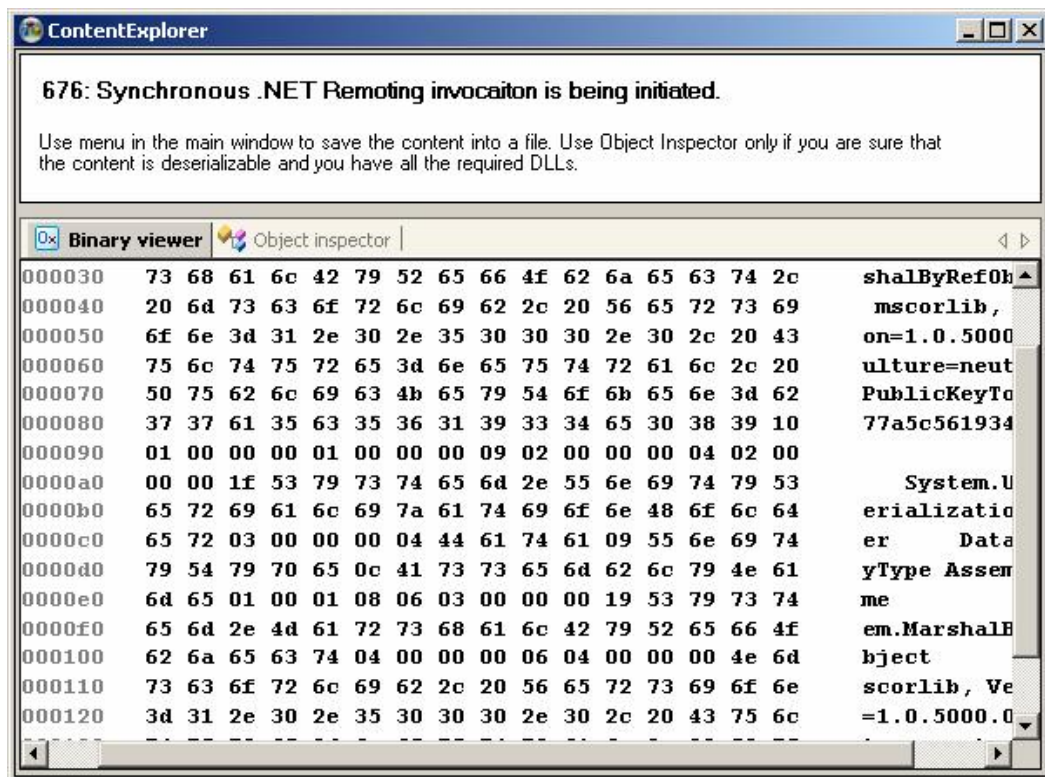
The logging system places high emphasis on messages and their content. Probably, you will be interested only in message size, message target, message method name, message role and message content. Genuine Log Viewer allows you to deserialize .NET Remoting messages into graphs if you have necessary DLL and EXE files.

Available message properties

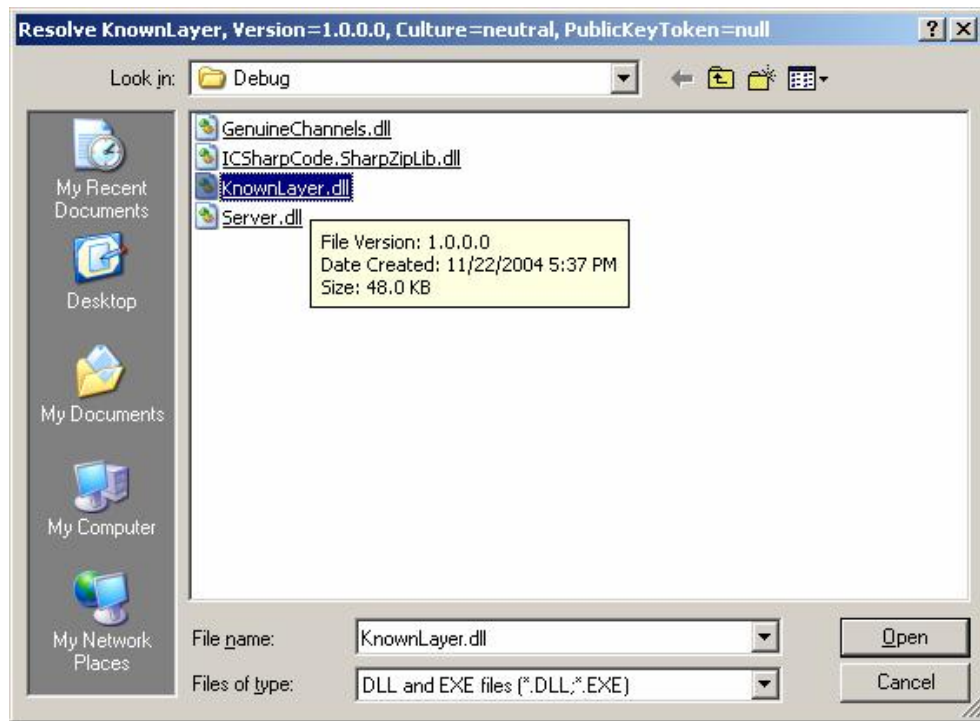
Name	Sample and description
Size	350 (bytes) The size of the message in bytes. This property always displays the actual size of the message. Compression, encryption or integrity checking does not change the value of this parameter.
Target	/8e4c7c80_6e70_4ef2_81fd_2064329dce71/AvrdXUdCNhgqmYD9KKaPaFJD_1.rem UserCrudServices.rem The target of the invocation represents a remote entry point associated with a specific MBR object or a DXM consumer that will handle the message.
Method Name	KnownLayer.Services.ISecurityServices.Login System.MarshalByRefObject.CreateObjRef The name of the method to be invoked on the target object. Method Names always equal to Target Names if you send DXM messages.
Type	Ordinary ExternalStreamConsumer TrueBroadcast The type of the message.
Invocation Level Security Session	<default basic> ("D") /Auction/Login/ZPA The Invocation Level Security Session is displayed in a separate section. See the connection (or invocation) properties for Connection Level Security Session.
Message ID	65 It is the logical identifier of the invocation assigned by the host that initiated the invocation. This identifier is not unique, but it remains the same when a message is transmitted to another host. The Message ID/Host URI pair is unique. Message ID allows you to find the corresponding messages in log files generated by client and server applications.
Reply ID	20 This number represents the identifier of the initial request invocation. This property is zero for all invocation requests and it is not zero for all invocation responses. For example, ReplyID=65 means that it is a reply to MessageID=65. Reply identifiers are not unique. The Reply ID/Host URI pair is not unique either since each invocation can cause several responses.
Seq No	103 This property represents the unique identifier of each message. The ordinal numbers of the same message do not match in log files produced by client and server applications.
Role	Response being sent. ASync Received request. Sync Request being sent. This property displays the role of the currently selected message. Responses are usually always ASync, though it depends on transport layer implementation.

	Anyway, the first word (SYNC/ASYNC) makes sense only if you inspect an invocation request that is sent to a remote host.
Connection Type	Persistent The type of the connection pattern specified by the Security Session Parameters applied to the message.
Connection Name	<empty> This property represents the name of the connection requested by the Security Session Parameters.
Invocation Timeout	<not specified> This property represents the invocation timeout requested by the Security Session Parameters.

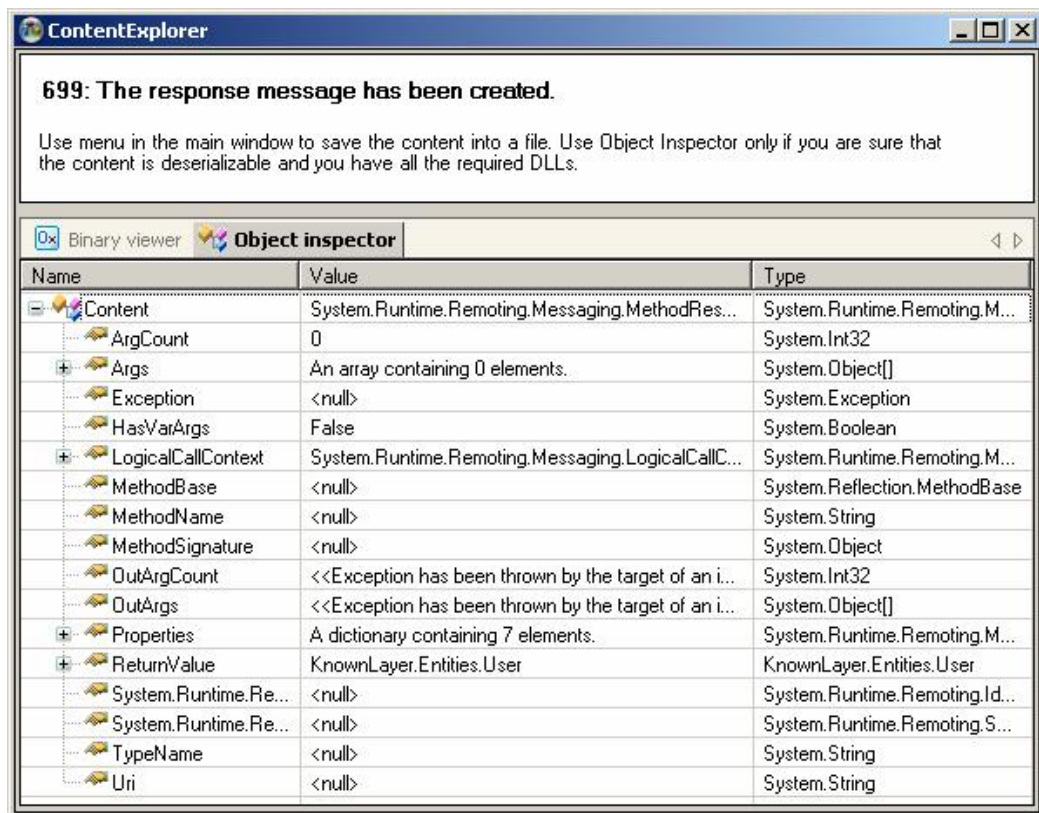
To examine the content of a message, double click it.



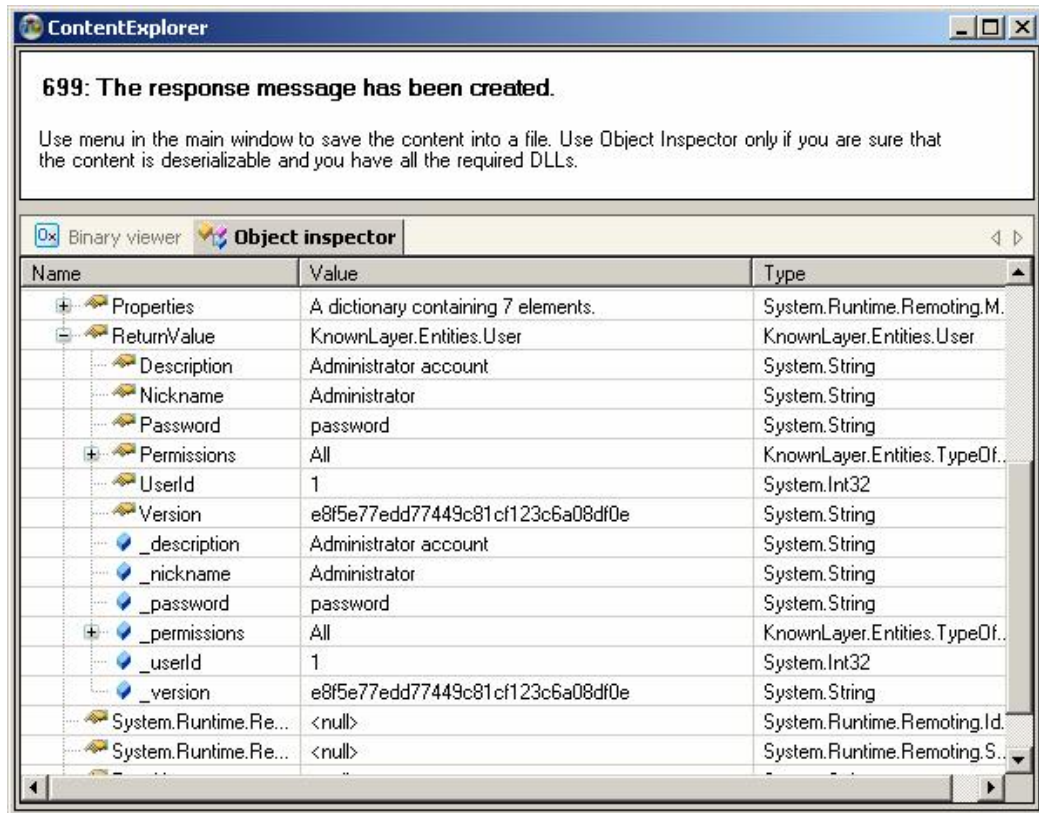
The binary form is usually useful only if you want to understand what has been sent during a DXM invocation. Having clicked *Object inspector*, you usually get the following window:



On the title bar you can see what assembly is requested. You need to find and open the assembly that can be used to deserialize the message content.

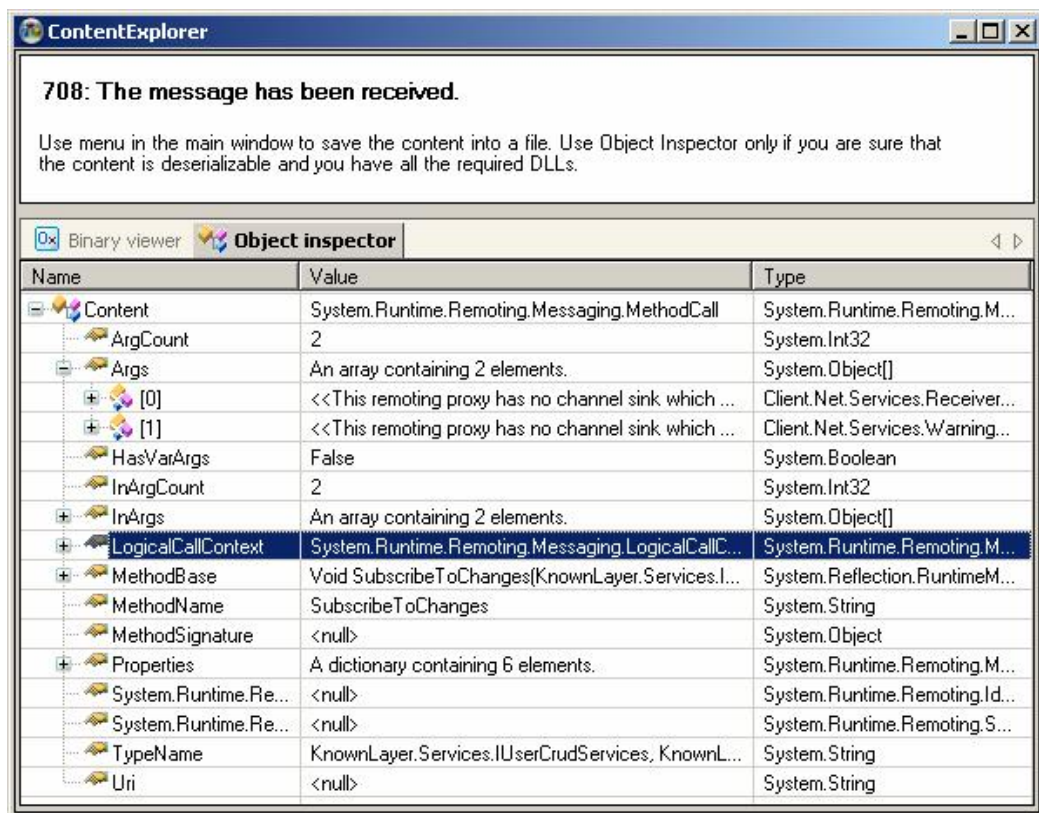


As you see, this is a response message (System.Runtime.Remoting.Messaging.MethodRes...) and it contains a ReturnValue. You can unwrap it and inspect inner members.

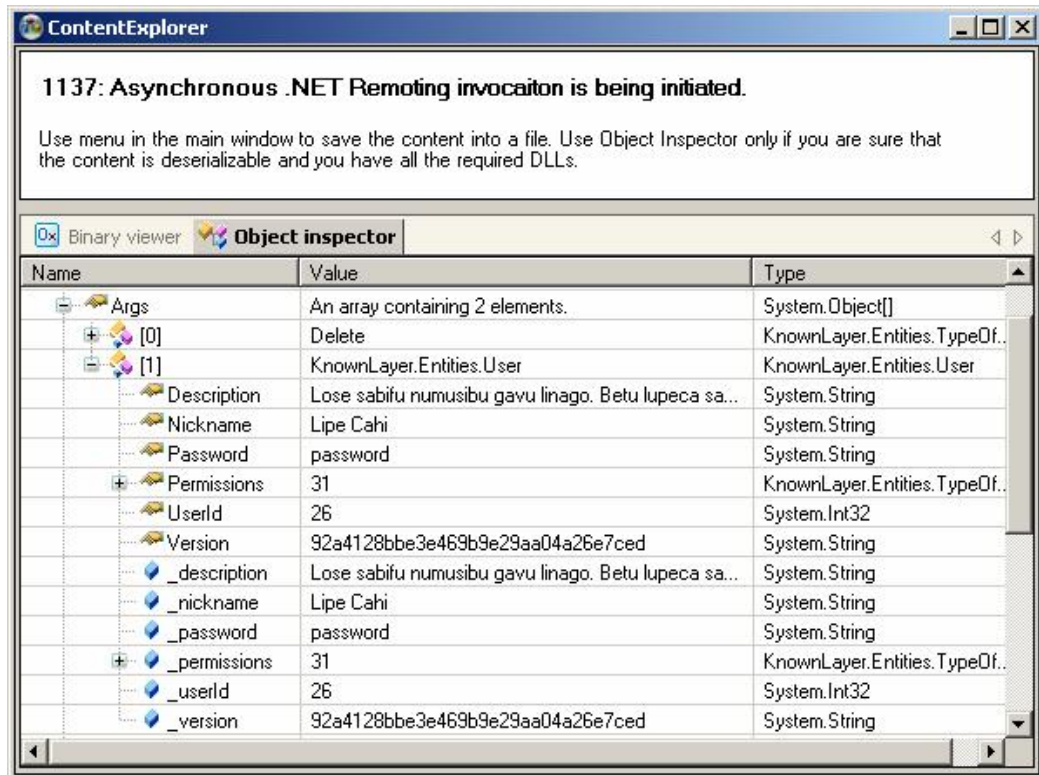


Method calls contain all arguments and definitions of targets and methods.

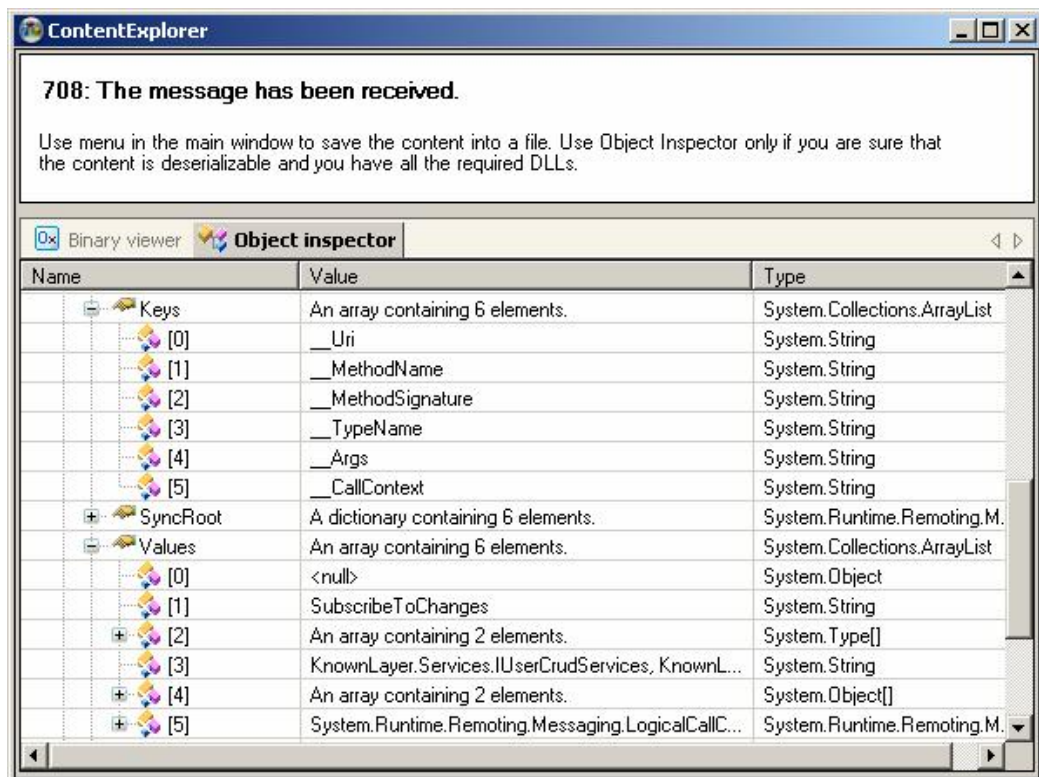
Let us take a closer look. On the next screenshot you can see that a remote host has sent two parameters (MBR objects) and invoked our [void SubscribeToChanges(KnownLayer...] method on the [KnownLayer.Services.IUserCrudServices, Known...] object.



When we delete a user in the Beginner's Guide sample, it sends the full user information:



You can unwrap the Properties member and analyze the .NET Remoting stuff.



In general, the list of made invocations gives you the full list of happened events including lease management and SAO/CAO/ObjRef work. Sometimes, it greatly simplifies debugging.

All loaded assemblies are cached, so you will not be asked to select the same assembly again. You can reset the list of loaded assemblies via **Record > Reset cached assemblies**.

You can always save the dump of the message via **Record > Save to file**.

The deserialization of DXM messages is not supported by the current implementation. You can view the content of DXM messages in the binary representation or save it into a file.

5.5. Invocations

Genuine Log Viewer collects all invocations into **Made Invocations** and **Performed Invocations** folders. **Made Invocations** include invocations sent to a remote host. In other words, if you send an invocation request and receive a response, it is a **Made Invocation**.

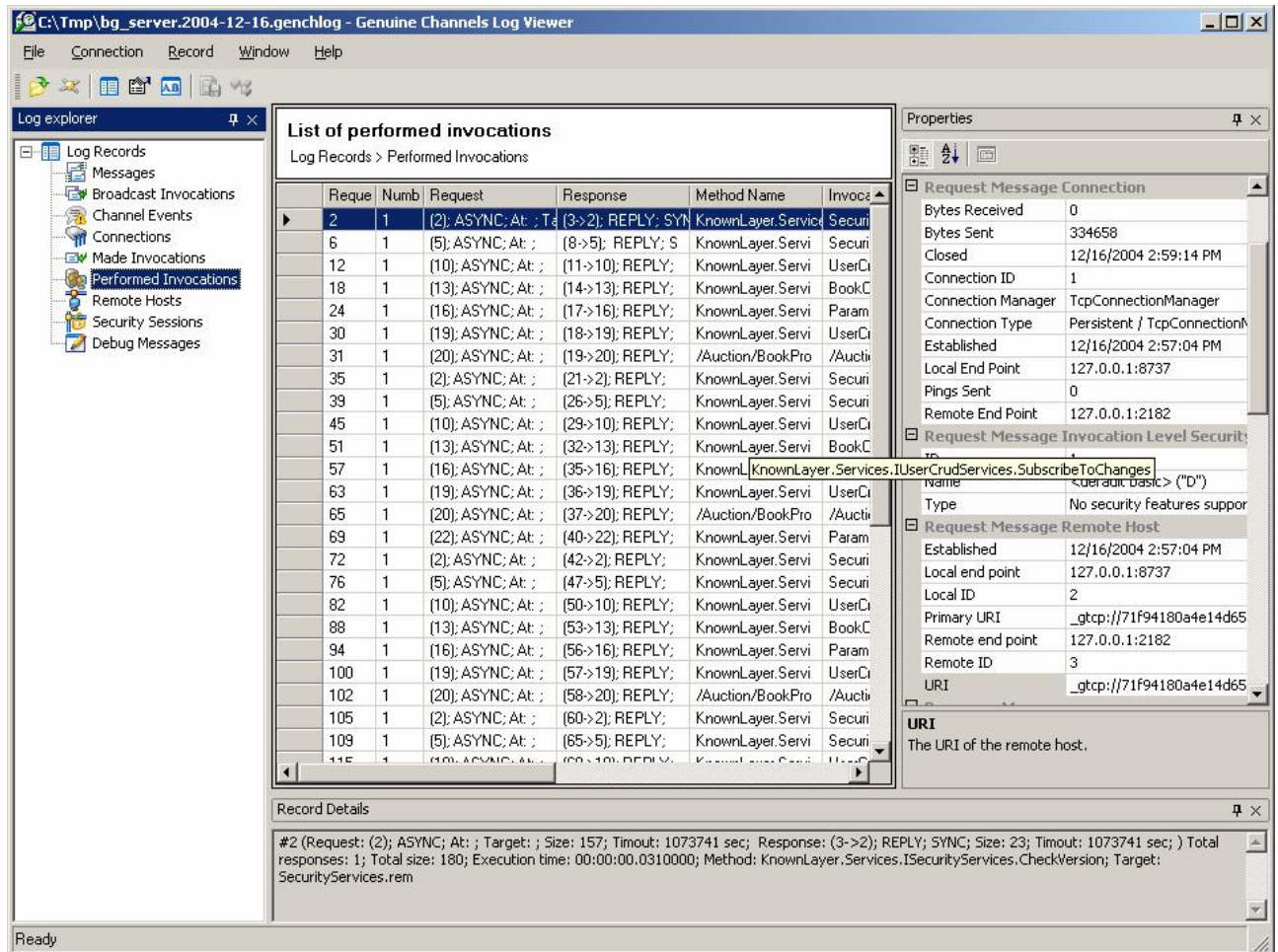
The screenshot shows the 'Genuine Channels Log Viewer' application. The 'Log explorer' on the left shows a tree view with 'Log Records' expanded, containing 'Messages', 'Broadcast Invocations', 'Channel Events', 'Connections', 'Made Invocations' (selected), 'Performed Invocations', 'Remote Hosts', 'Security Sessions', and 'Debug Messages'. The main pane displays a 'List of made invocations' table with columns: Request, Number, Request, Response, Method Name, and Inv. The table lists several invocations, with the 58th invocation selected. The 'Properties' pane on the right shows details for the selected invocation, including 'Request Message Connection' and 'Request Message Remote Host'. The 'Record Details' pane at the bottom shows the full log entry for the selected invocation.

Request	Number	Request	Response	Method Name	Inv
15	1	(10): SYNC; At: /8	(12->10): REPLY;	System.MarshalB	/84
19	1	(12): SYNC; At: /8	(14->12): REPLY;	System.MarshalB	/84
21	1	(13): SYNC; At: /8	(15->13): REPLY;	System.MarshalB	/84
25	1	(15): SYNC; At: /8	(17->15): REPLY;	System.MarshalB	/84
27	1	(16): SYNC; At: /8	(18->16): REPLY;	System.MarshalB	/84
40	1	(24): SYNC; At: /2	(8->24): REPLY;	System.MarshalB	/21
42	1	(25): SYNC; At: /2	(9->25): REPLY;	System.MarshalB	/21
46	1	(27): SYNC; At: /2	(11->27): REPLY;	System.MarshalB	/21
48	1	(28): SYNC; At: /2	(12->28): REPLY;	System.MarshalB	/21
52	1	(30): SYNC; At: /2	(14->30): REPLY;	System.MarshalB	/21
54	1	(31): SYNC; At: /2	(15->31): REPLY;	System.MarshalB	/21
58	1	(33): SYNC; At: /2	(17->33): REPLY;	System.MarshalB	/21
77	1	(45): SYNC; At: /8	(8->45): REPLY;	System.MarshalB	/8t
79	1	(46): SYNC; At: /8	(9->46): REPLY;	System.MarshalB	/8t
83	1	(48): SYNC; At: /8	(11->48): REPLY;	System.MarshalB	/8t
85	1	(49): SYNC; At: /8	(12->49): REPLY;	System.MarshalB	/8t
89	1	(51): SYNC; At: /8	(14->51): REPLY;	System.MarshalB	/8t
91	1	(52): SYNC; At: /8	(15->52): REPLY;	System.MarshalB	/8t
95	1	(54): SYNC; At: /8	(17->54): REPLY;	System.MarshalB	/8t
97	1	(55): SYNC; At: /8	(18->55): REPLY;	System.MarshalB	/8t
110	1	(63): SYNC; At: /8	(8->63): REPLY;	System.MarshalB	/8t
112	1	(64): SYNC; At: /8	(9->64): REPLY;	System.MarshalB	/8t
116	1	(66): SYNC; At: /8	(11->66): REPLY;	System.MarshalB	/8t
118	1	(67): SYNC; At: /8	(12->67): REPLY;	System.MarshalB	/8t

Record Details

```
#58 (Request: (33): SYNC; At: /2152f7a5_cd97_4f4a_b787_7296bf5bb93b/grVsOUIn2RM_jj3XAVb6Ns8if_4.rem; Target: System.MarshalByRefObject.CreateObjRef; Size: 350; Timeout: 1073741 sec; Response: (17->33): REPLY; ASYNC; Size: 1155; Timeout: 1073741 sec; ) Total responses: 1; Total size: 1505; Execution time: 00:00:00.0160000; Method: System.MarshalByRefObject.CreateObjRef; Target: /2152f7a5_cd97_4f4a_b787_7296bf5bb93b/grVsOUIn2RM_jj3XAVb6Ns8if_4.rem
```

Performed Invocations are invocations executed on the local host. In other words, when we receive an invocation request, execute it and send back an invocation response.



5.5.1. Lifetime

When you need to track down a made or performed invocation, open the **Messages** folder and find the corresponding incoming and outgoing messages. After that, you can use filtering to find all events related to a particular message.

5.5.2. Available Information

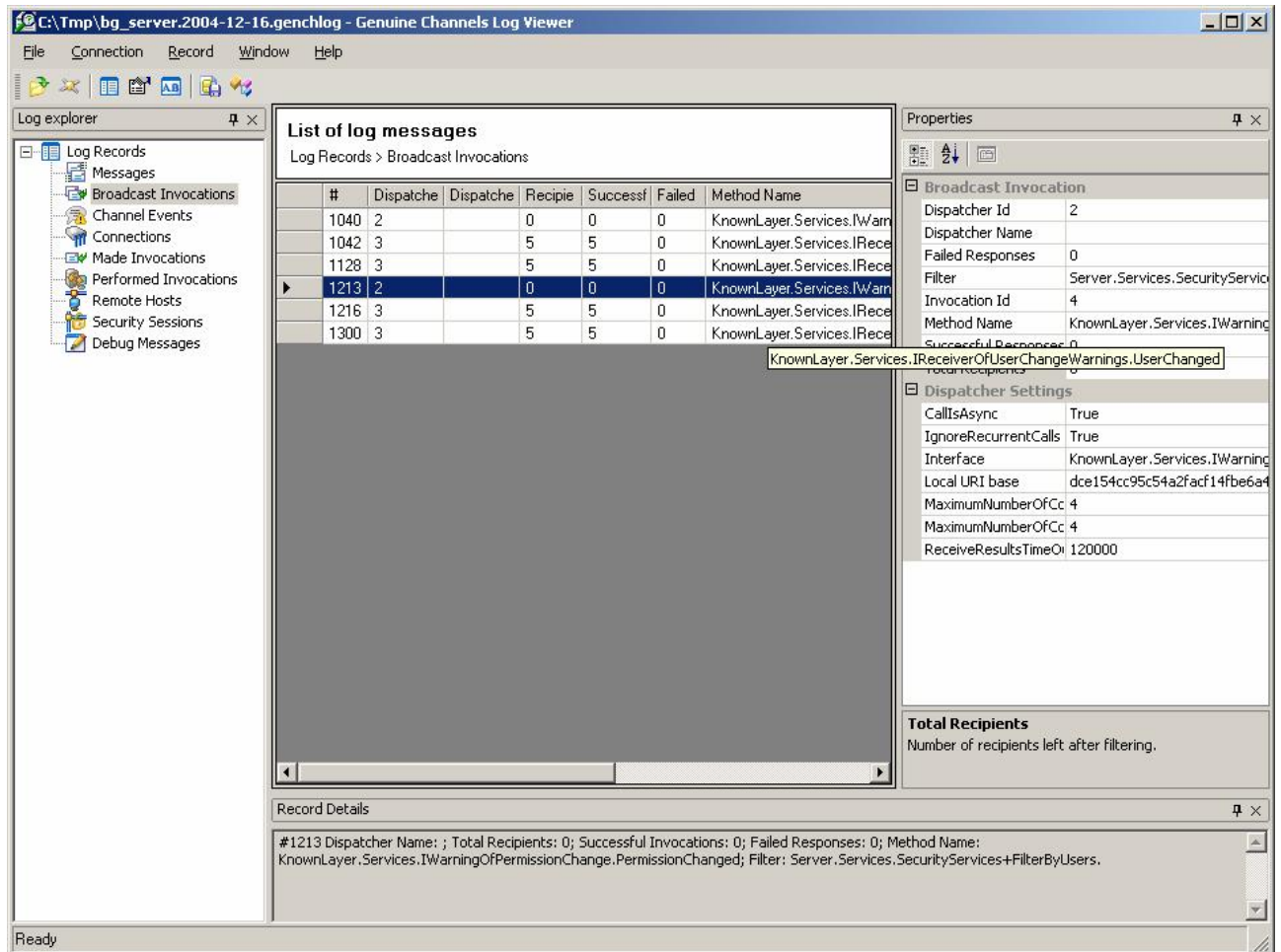
The properties of an invocation include its request message, response message and statistical information. The statistical information contains the size of the request message, the size of the response message(s), the total size of the invocation and its execution time. The execution time of made invocations includes the time spent on sending the request message and receiving the response message (a network round trip). The execution time of performed invocations does not include a network round trip and displays the pure time of execution.

The screenshot shows the 'Genuine Channels Log Viewer' application. The 'Log explorer' pane on the left shows a tree view with 'Log Records' expanded, and 'Made Invocations' selected. The main pane displays a 'List of made invocations' table with columns: Name, Invocation Target, Request, Respons, Total Siz, and Execution Time Sp. The table lists various invocations, including 'MarshalByR' and 'MarshalB' with targets like '_gtcp://71f94180a4e14d65a629d32aaf88ff25/WarningOfPermissionChange.rem'. The 'Properties' pane on the right shows details for the selected invocation, including 'Request Message Connection' (Bytes Received: 0, Bytes Sent: 334658) and 'Request Message Remote Host' (Established: 12/16/2004 2:57:04 PM, Local end point: 127.0.0.1:8737, Remote end point: 127.0.0.1:2182). The 'Record Details' pane at the bottom shows the raw log entry for the selected record.

Name	Invocation Target	Request	Respons	Total Siz	Execution Time Sp
MarshalByR	_gtcp://71f94180a4e14d65a629d32aaf88ff25/WarningOfPermissionChange.rem	350	1154	1504	00:00:00.0310000
MarshalB	_gtcp://71f94180a4e14d65a629d32aaf88ff25/WarningOfPermissionChange.rem	150	23	173	00:00:00
MarshalB	/84f24ac0_8468_	350	1155	1505	00:00:00
MarshalB	/84f24ac0_8468_	150	23	173	00:00:00
MarshalB	/84f24ac0_8468_	350	1155	1505	00:00:00
MarshalB	/84f24ac0_8468_	150	23	173	00:00:00
MarshalB	/84f24ac0_8468_	350	1155	1505	00:00:00
MarshalB	/84f24ac0_8468_	150	23	173	00:00:00
MarshalB	_gtcp://bf97b0400_	350	1154	1504	00:00:00.0320000
MarshalB	_gtcp://bf97b0400_	150	23	173	00:00:00
MarshalB	/2152f7a5_cd97_	350	1155	1505	00:00:00.0000000
MarshalB	/2152f7a5_cd97_	150	23	173	00:00:00
MarshalB	/2152f7a5_cd97_	350	1155	1505	00:00:00
MarshalB	/2152f7a5_cd97_	150	23	173	00:00:00
MarshalB	/2152f7a5_cd97_	350	1155	1505	00:00:00.0160000
MarshalB	/2152f7a5_cd97_	150	23	173	00:00:00
MarshalB	_gtcp://12bd1043_	350	1154	1504	00:00:00.0310000
MarshalB	_gtcp://12bd1043_	150	23	173	00:00:00
MarshalB	/8bb3cc5d_a3f0_	350	1155	1505	00:00:00
MarshalB	/8bb3cc5d_a3f0_	150	23	173	00:00:00
MarshalB	/8bb3cc5d_a3f0_	350	1155	1505	00:00:00
MarshalB	/8bb3cc5d_a3f0_	150	23	173	00:00:00
MarshalB	/8bb3cc5d_a3f0_	350	1155	1505	00:00:00
MarshalB	/8bb3cc5d_a3f0_	150	23	173	00:00:00
MarshalB	/8bb3cc5d_a3f0_	350	1154	1504	00:00:00.0000000

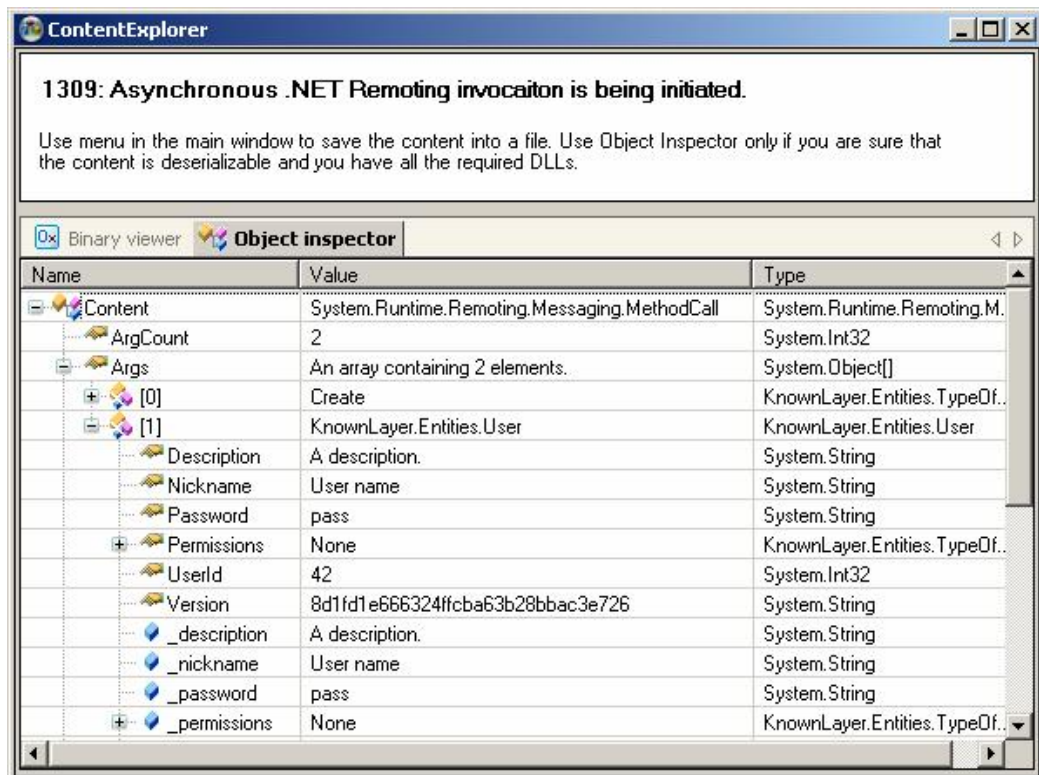
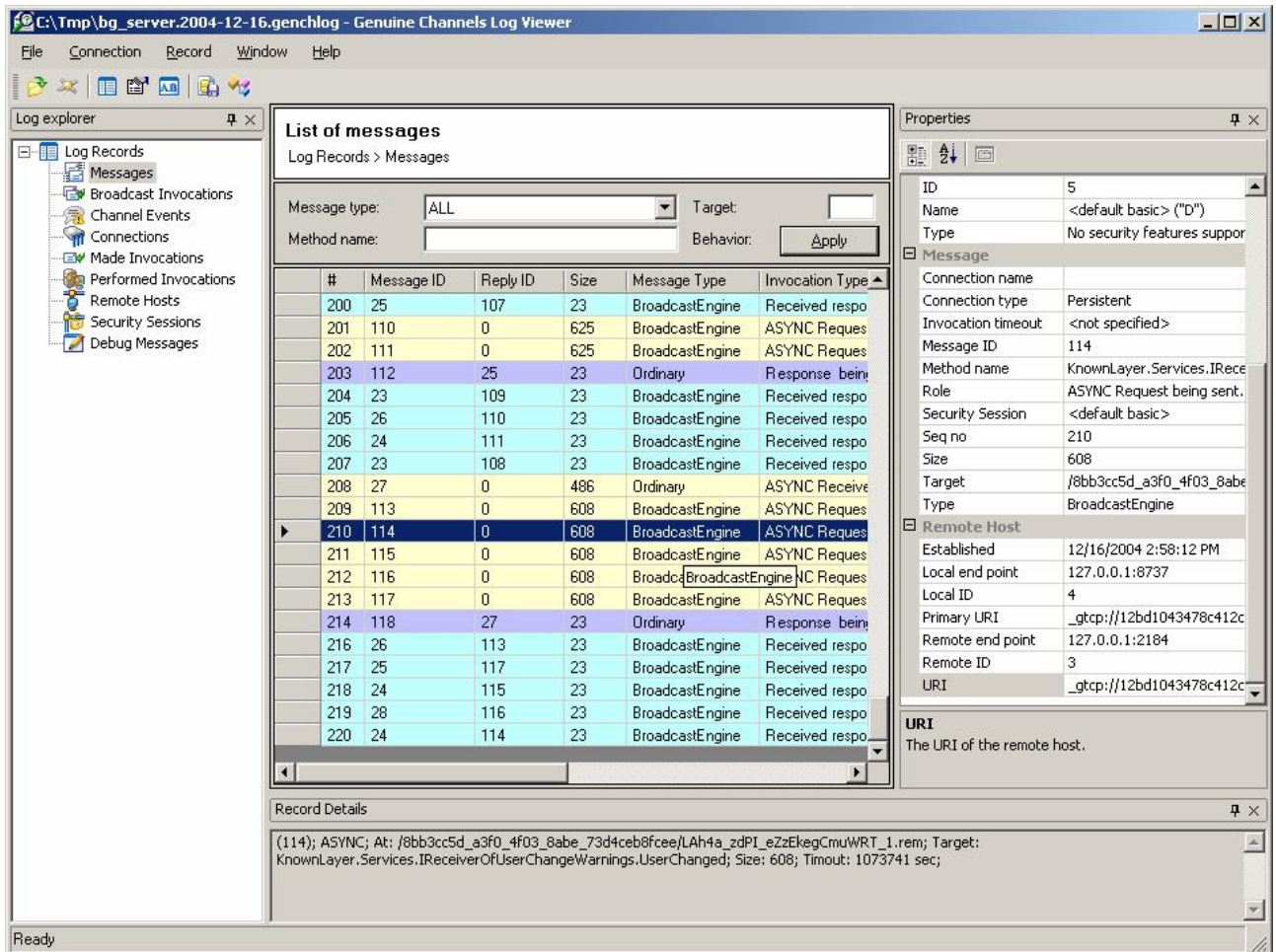
5.6. Broadcast Engine

The **Broadcast Invocations** folder contains only general statistical information and can be useful only when you want to check the number of recipient that successfully received a particular message.



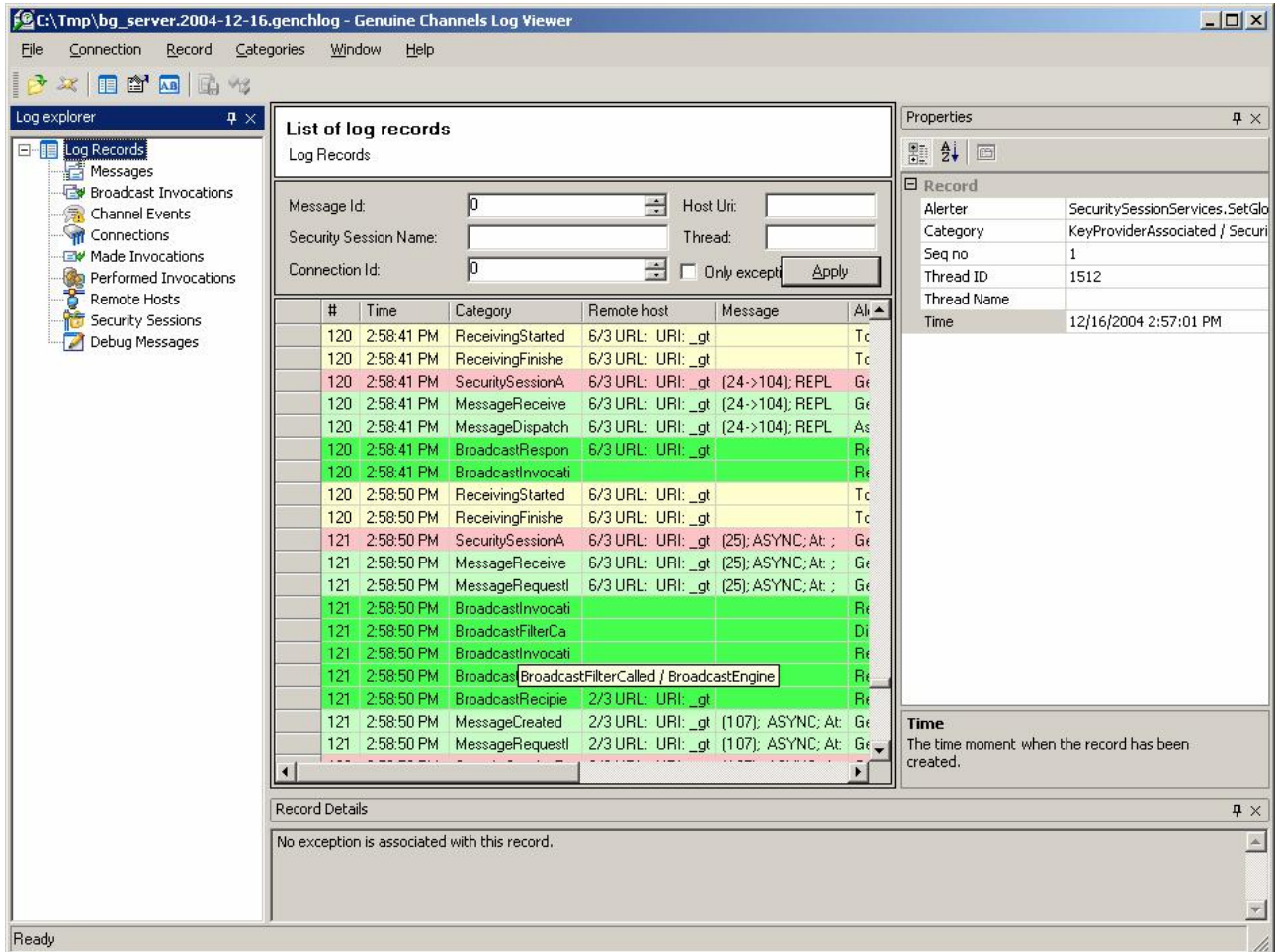
If you need to find out what data has been sent or received during a particular broadcasting, work with messages as it is described in section 5.4.

For example, if you want to find out what content has been sent to those five clients, open the **Messages** folder and look for the message you want to find:



5.6.1. Lifetime

Broadcast records are displayed in bright green. The broadcast engine and dispatchers always follow the state diagram.



5.6.2. Available Information

You have the name of the used filter, the method name, the number of invoked recipients and the number of received responses. You can inspect the content of sent and/or received requests and responses like in the case of the regular .NET Remoting invocation.

The name of the dispatcher can be set via *Dispatcher.DbgDispatcherName*. The identifier of the dispatcher is available during execution via the *Dispatcher.DbgDispatcherId* property.