Sensys –

Communication Protocols

# Overview

Sensys requires communications to take place among processes of various types, including: *Sensis App* (A), *Sensis Server* (SS), *Sensor* (S), Customer (C), Medic (M). To facilitate communications through routers that are using Network Address Translation and firewalls, some communications will go through a *Proxy* (X).

For this document, A, S, C, and M are collective referred to as Local Processes (LP’s). LP’s may run on any host and any network. The other processes, namely SS, and X processes, must run on hosts with routable addresses, so LP’s can communication to them directly via IP end points. When an X process receives a message, it will valid the sender by making sure it is an alive process know to the Sensis Server, resolve the target process’s Id to an IP end point, and forward the message on to that end point. All LP’s processes will use local network broadcast for discovery and communicate via local IP endpoints.

Table 1 provides a high-level list of the all of the protocol that govern these communications. Most of the protocols follow a Request-Reply-Ack Through Proxy pattern to ensure reliability.

**Table 1 – Protocol List**

| Protocol | Purpose | Initiator | Other Processes | Pattern |
| --- | --- | --- | --- | --- |
| Login | To register processes so they can be authenticated and their public end points can be made know to the system. | A, C | SS | Request-Reply |
| Logout | To de-register processes efficiently | A, C | SS | Request-Reply |
| Heartbeat | To make sure a registered process is still responsive. To give additional location information for emergency use. | A | SS | Request-Reply |
| Sensor Detection | To automatically discover available sensors in a local network. | S | A | Custom |
| Sensor Handshake | To connect to an available sensor. | A | S | Request-Reply |
| Sensor Data Gathering (App) | For sensors to send data to app | S | A | Request-Reply |
| Sensor Data Gathering (Server) | For apps to send data to server | A | X, SS | Request-Reply Through a Proxy |
| Sending Warning Message | To send messages from server to app | SS | A, X | Request-Reply Through a Proxy |
| Initiate Emergency mode | To start the emergency mode | SS | A, X, P, C | Custom |
| End Sensor Communication | To efficiently release sensors | A | S | Unreliable Multicast |

# Messages and Shared Objects

Figures 1-4 show a class hierarchy of all the messages used in Sensis protocols and Tables 2-4

provides some additional details about each message class’s attributes and their meaning. All

messages will be serialized using JSON serialization with the class name (including inherited

names) and attribute names corresponding to Figures and Tables.

**Table 2 - Request Message Class Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Class name** | **Attribute, as it appears in the JSON Serialization** | **Attribute type** | **Meaning** |
| (all messages) | ConversationId | MessageNumber | A unique identifier for a conversation. The conversation’s id is same as the message number for the first message in the conversation. |
| MessageNr | MessageNumber | A unique identifier for a message. |
| **LoginRequest:#Messages** | **Used to start a Login conversation** | | |
| Identity | Identity | An object that holds the identity of the end user. |
| **LogoutRequest:#Messages** | **Used to start a Logout conversation** | | |
| **HeartbeatRequest:#Messages** | **Used to start a Heatbeat conversation** | | |
| GPSLocation | GPSLocation | An object that holds the location of the phone. |
| **AvailableSensorRequest:#Messages** | **Used to start a Sensor Detection conversation** | | |
| SensorID | int | An unique identifier for a type of sensor |
| SensorName | String | Name for this sensor |
| **SensorHandshakeRequest:#Messages** | **Used to start a Sensor Handshake conversation** | | |
| **SensorGatheringRequest:#Messages** | Used to start a Sensor Gathering conversation | | |
| SensorID | int | An unique identifier for a type of sensor |
| SensorData | SensorData | An object that holds the data of this sensor |
| LastReadingTime | long | The time of the last read |
| **MessageRequest:#Messages** | **Used to start a Message request, could be emergency or normal message** | | |
| MessageType | int | The type of message |
| MessageContent | String | The content of the message |
| **EndSensorsRequest:#Messages** | Used to tell sensors to end connections with this app | | |

Note: All concrete request message classes inherit from Message, and therefore include the ConversationId

and MessageNr attributes. Some message classes contain no other attributes. They are still listed in the table

below, but without attributes.

**Table 3 - Reply Message Class Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Class name** | **Attribute, as it appears in the JSON Serialization** | **Attribute type** | **Meaning** |
| (all reply messages) | ConversationId | MessageNumber | A unique identifier for a conversation. The conversation’s id is same as the message number for the first message in the conversation. |
| MessageNr | MessageNumber | A unique identifier for a message. |
| Note | String | An optional note, typically contain an error message is the request was not successful |
| Success | bool | True if the requested action was successfully completed; otherwise false |
| **LoginReply:#Messages** | **Used in Login conversation** | | |
| ProxyEndPoint | PublicEndPoint | Communication end point for a proxy. Any communications should be sent via a routing message to the proxy at this end point |
| **Reply:#Message** | **Used in a variety of protocols as a general reply** | | |
| **HeartbeatReply:#Messages** | **Used in a Heatbeat conversation** | | |
| **SensorHandshakeReply:#Messages** | **Used in a Sensor Handshake conversation** | | |
| **SensorGatheringReply:#Messages** | **Used in a Sensor Gathering conversation** | | |
| **MessageReply:#Messages** | **Used in a Message request, could be emergency or normal message** | | |

Note: All concrete reply message classes inherit from Message and Reply, and therefore include the

ConversationId, MessageNr, Success, and Note attributes.

**Table 4 - Shared Object Class Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Class name** | **Attribute, as it appears in the JSON Serialization** | **Attribute type** | **Meaning** |
| MessageNumber | ProcessId | int | A unique identify for the process that created this message number |
| SeqNumber | int | A number from a circular sequence that is unique within the context of the process. If MessageNumber B is created immediately following MessageNumber A on the same process, then A.SeqNumber + 1 = B.SeqNumber, until the Int32.Max. At this point, B.SeqNumber will set to 1. |
| Identity | UserID | String | ID of the user, that was given before. |
| Pin | int | Pin of the user, that was given before |
| PublicEndPoint | Host | String | A string representation of the host’s IP Address. It can be an IP Hostname as well. |
| Port | int | The end point’s port number |
| GPSLocation | latitude | double | The location of this phone in term of latitude according to GPS reading. |
| longitude | double | The location of this phone in term of longitude according to GPS reading. |
| timestamp | long | UTC time of this fix, in milliseconds since January 1, 1970. |
| SensorData | data | String | For the simulation this data is just a string of non-sense. Detail of this object will be implemented in the future. |

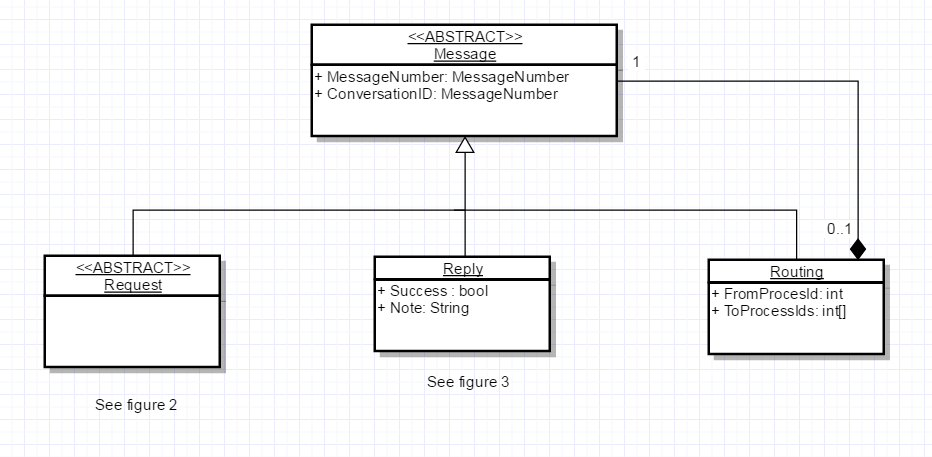


Figure - General Message Classes

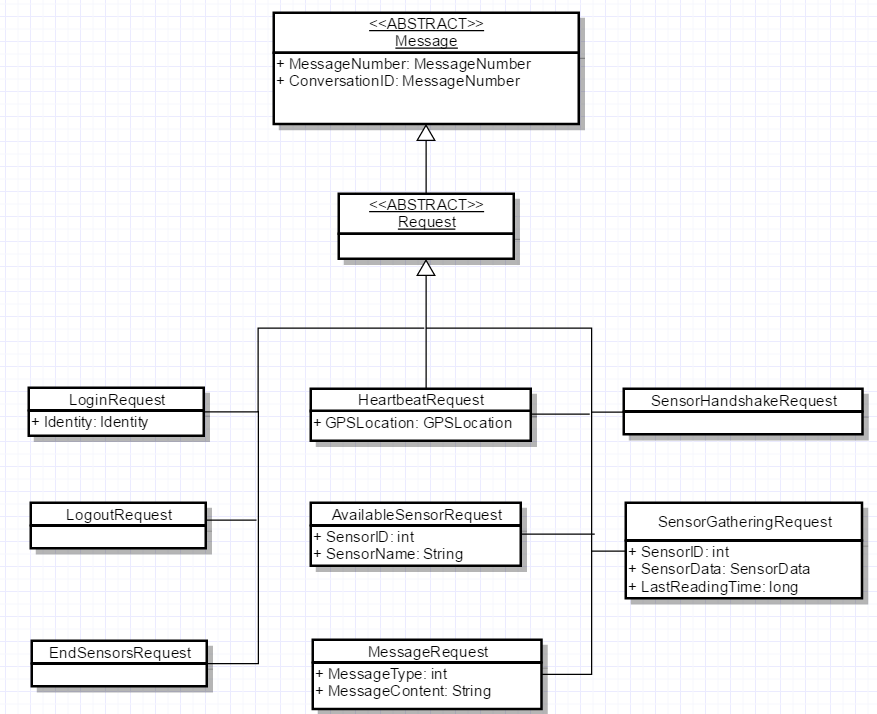


Figure - Request Message Classes

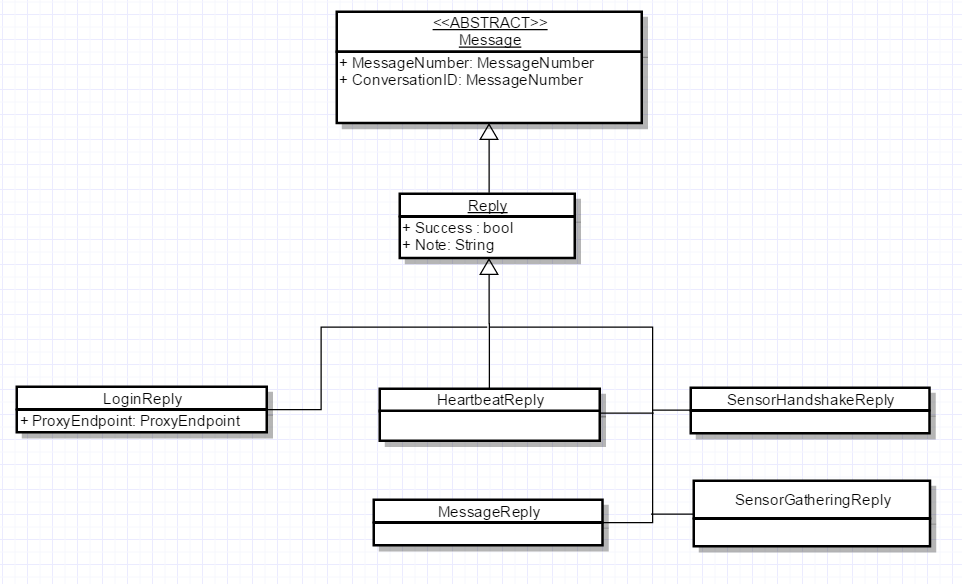


Figure - Reply Message Classes

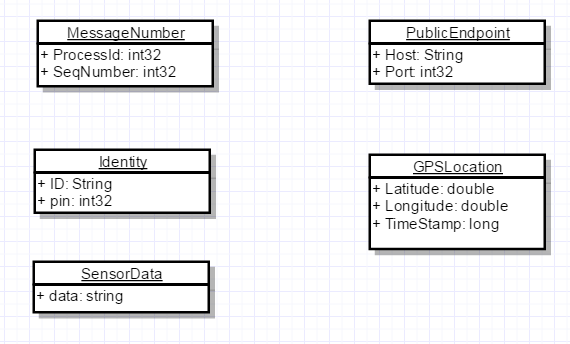


Figure - Shared Objects

# Communication Patterns

Most of the protocols in dSoak system follow one of two patterns: request-reply and unreliable

multicast. These are described here and referenced in Section 4.

## Request-Reply

Figure 5 show a basic sequence diagram for a Request-Reply pattern with the following concepts:

* an initiating process p1
* a receiving process p2
* a message Request that p1 sends to p2 to start the conversation
* a message Reply that p2 returns after receiving the request
* a MaxTries parameter limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply
* Timeout parameter specifies how long the initiator will wait for a reply
* The Tries variable is an internal count of the attempts.

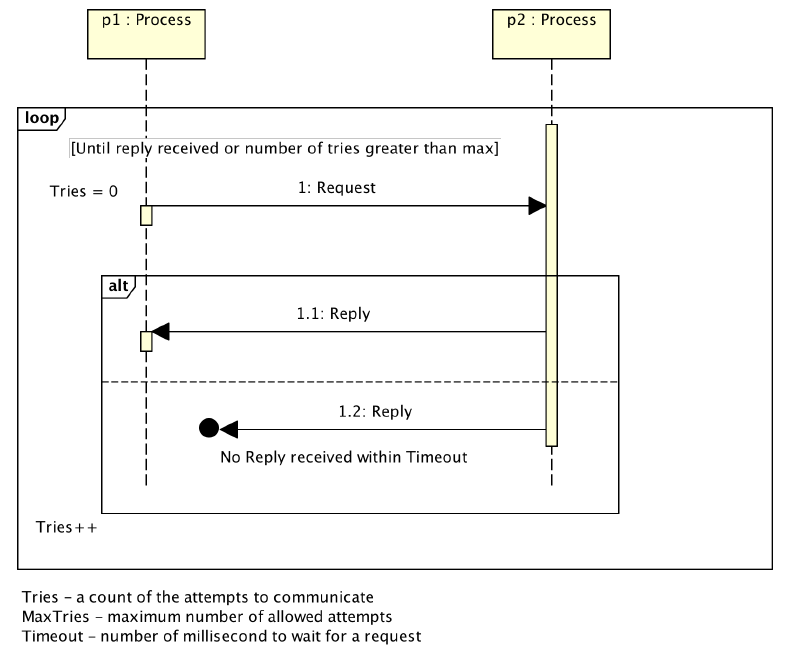


Figure - Request-reply pattern

## Request-reply through a proxy

Figure 6 show a sequence diagram for the Request-Reply Through Proxy pattern with the following concepts:

* an initiating process p1
* a proxy process x
* a receiving process p2
* a message m1 that p1 places in a Routing w1 and sends to x, which forwards w1 to p2. This starts the conversation.
* p2 returns message m2 by placing m2 in a Routing w2 and sending to x, which forwards w2 to p1.
* a MaxTries parameter limits the number of attempts that will be made before the conversation is aborted if not successfully completed with a reply
* Timeout parameter that specific how long the initiator will wait for a reply

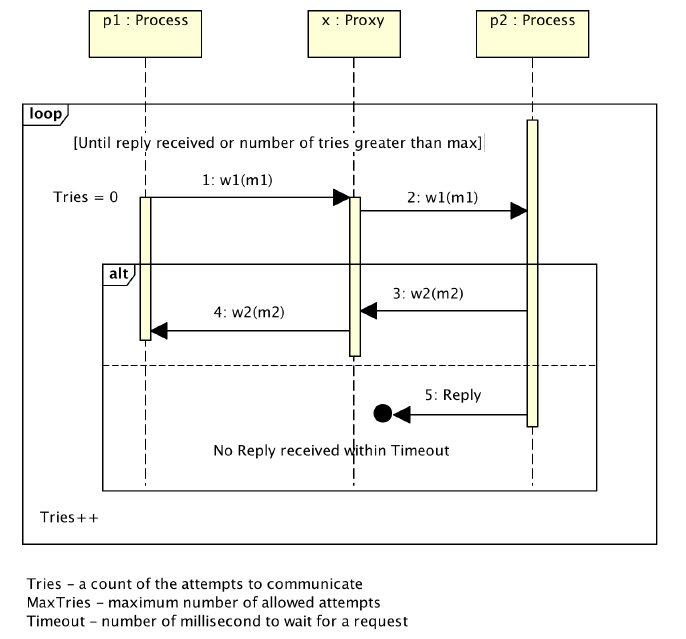


Figure - Request-reply Through a Proxy

## Unreliable Multicast

Figure 7 show an unreliable multicast pattern with the following substitutable concepts:

• an initiating process (p1)

• a receiving process (p2)

• a message that p1 sends to p2

# 

Figure - Unreliable Multicast

# Communication Protocols

Below are detailed descriptions of the conversations outlined in Table 1.

## Login (Request-reply)

Used by App to login to the system

Message sequence:

A 🡪 (LoginRequest) 🡪 S

S 🡪 (LoginReply) 🡪 A

Semantic and behaviors:

* The ProcessId in the LoginRequest’s MessageNr and ConversationId can be 0.
* The LoginRequest must include an Identity object that contains the end user’s real identity.
* The Server will try to authenticate the identity in the LoginRequest against the list of recognized end users.
* If the Server can log the process in, it will return a LoginReply with a unique ProcessId for the new process and its publically visible end point. Also, it will set the process’s status to “Registered”. The LoginReply.ProcessInfo will contain the ProcessId, public end point, and new status. The Success field will also be set to true.
* If the Server cannot log the process in, it will return a LoginReply with Success=false. Also, the LoginReply.Note will contain a message about why the login failed.
* If a process that is already logged in again, the server will send a failure LoginReply. Only one user with that credential can log into the system at one time.

## Logout (Request-Reply)

Used by App to logout of the system

Message Sequence:

A 🡪 (LogoutRequest) 🡪 S

S 🡪 (Reply) 🡪 A

Semantics and Behaviors:

* App can logout at any time by sending the server a LogoutRequest message from its main communication channel (the one that it used to log in).
* When the server receives a LogoutRequest message, it will log the process out and send back a Reply message with Success=true. If there was an error of any kind, then the server send back a Reply message with a Success=false and the error message in the Note field.

## Heartbeat (Request-Reply)

Used by App to tell server it is still alive, also update its location for emergency mode.

Message Sequence:

A 🡪 (HeartbeatRequest) 🡪 S

S 🡪 (Reply) 🡪 A

Semantics and Behaviors:

* The app periodically sends a Heartbeat Request the server, packing the location of the app inside.
* Each Heartbeat Request represents a new conversation and therefore contains a unique conversation id
* The receiving process must response as quickly as possible to the Heartbeat Request with a Reply message. The Reply must contain the same conversation id that was in the HeartbeatRequest. The Success field of the Reply message must be set to true.
* If the app does not receive the Reply in with the timeout limit, it will try sending the same HeartbeatRequest again and a third time if necessary. If the receiving process still has not responded, then the app is conclude that the receiving processing is dead and will automatically prompt user and exit the app.
* If the server does not receive a Heartbeat request after 10 minutes, it will log the user out.

## Sensor Handshake (Request-Reply)

Used by App to register itself as the receiver of sensor data.

Message Sequence:

A 🡪 (HandshakeRequest) 🡪 SS

SS 🡪 (Reply) 🡪 A

Semantics and Behaviors:

* The app send a Handshake request to the sensor it detected to ask for that sensor to start sending data to the app.
* The receiving process, if has not registered to any other app must add the IPEndpoint of this app as the destination for the next sensor data package, then send a Reply back to the app, with Success=true. The Sensor then process to stop broadcasting SensorDetection messages.
* If another app has already been the destination, then the sensor immediately send a Reply back to the app with Success=False and Note as the error message.

## Sensor Data Gathering – App ( Request-Reply)

Used by Sensor to send its data to the app.

Message Sequence:

SS 🡪 (SensorGatheringRequest) 🡪 A

A 🡪 (Reply) 🡪 SS

Semantics and Behaviors:

* Sensor send a SensorGatheringRequest to app, packing the sensor data inside.
* The receiving process immediately send a Reply back to the sensor, with Success=true.
* If the sensor does not receive the Reply in with the timeout limit, it will try sending the same SensorGatheringRequest again and a third time if necessary. If the receiving process still has not responded, then the Sensor is conclude that the receiving processing is dead and will automatically delete the destination app and turn off the sensor (user has to manually turn it on again to start broadcasting its SensorDetectionMessage).

## Sensor Data Gathering – Server ( Request-Reply Through a proxy)

Used by the app to send sensor data to the server

Message Sequence:

A 🡪 Routing(SensorGatheringRequest) 🡪 X 🡪 Routing(SensorGatheringRequest) 🡪 S

S 🡪 Routing(Reply) 🡪 X 🡪 Routing(Reply) 🡪 A

Semantics and Behaviors:

* App sends a SensorGatheringRequest to server, packing the sensor data inside.
* The receiving process immediately send a Reply back to the app, with Success=true.
* If the app does not receive the Reply in with the timeout limit, it will try sending the same SensorGatheringRequest again and a third time if necessary. If the receiving process still has not responded, then the App is conclude that the receiving processing is dead and will automatically delete the destination app go back to login screen.

## Sending Warning Message ( Request-Reply Through a proxy)

Used by the server to send warning message to the app

Message Sequence:

A 🡪 Routing(MessageRequest) 🡪 X 🡪 Routing(MessageRequest) 🡪 S

S 🡪 Routing(Reply) 🡪 X 🡪 Routing(Reply) 🡪 A

Semantics and Behaviors:

* Server sends a MessageRequest to the app, with type is WarningMessage and Content is the content of the message.
* The receiving process immediately send a Reply back to the server, with Success=true.
* If the server does not receive the Reply in with the timeout limit, it will try sending the same SensorGatheringRequest again and a third time if necessary. If the receiving process still has not responded, then the server is conclude that the receiving processing is dead and will log the user out.

## Initiate Emergency Mode ( Custom)

Used by Server to initiate an emergency case.

Message Sequence:

S 🡪 (MessageRequest) x3 for 5s after each message 🡪 A

(possibly dealing with human factor here by calling medics)

A 🡪 (MessageRequest) 🡪 S

S 🡪 (Reply) 🡪 A

Semantics and Behaviors:

* Server sends three Message Requests to the app, each message will be send 5 seconds after the last message, using server’s clock. The Message Request’s type is Emergency and Content is the emergency content.
* Server open an emergency state on this user and notify medical personnel (not implemented, just simulated in the simulation) to take care of the situation.
* The emergency state only ends after the server receive a Message Request from app saying the customer is okay, with type is EndingEmergency and Content has either customer id or medic’s code.
* When receiving EndingEmergency message, the server immediately send a reply to the app with Success = true.

## Sensor Detection ( Custom)

Used by Sensor to broadcast its availability to any app listening.

Message Sequence:

SS 🡪 (SensorDetectionMessage) 🡪 Broadcast address

Semantics and Behaviors:

* Sensor send a SensorDetectionMessage to the broadcast address (255.255.255.255 for the zero network).
* This message is send periodically every 1 minute until the sensor receive a HandshakeRequest.
* This message protocol only starts when the sensor is turned on to give user a sense of physically controlling the whole process.

## End Sensor Communication (Unreliable Multicast)

Used by the app to release all sensors

Message Sequence:

A 🡪 (EndSensorsRequest) 🡪 all Sensors

Semantics and Behaviors:

* On closing, the app sends a EndSensorsRequest message to all sensors that are currently handshaking with it.
* When a receiving process gets a EndSensorsRequest message and it will turn of the sensor.