DSoak Communication Protocols

1 OVERVIEW

The DSoak requires communications to take place among all of its processes, which include the Registry (R), Game Managers (GM's), Players (P's), Balloon Stores (BS's), Water Sources(WS's), and Umbrella Supplies (US's). For this document, the latter four types of processes are collective referred to as Game Processes. 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 pattern. The Hit protocol is a simply unreliable send; the shutdown protocol follows an unreliable multicast, and the Game Status and Umbrella Auction have unique patterns of their own.

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	GM, P, BS, WS, US	R	Request-Reply
Logout	To de-register processes efficiently	GM, P, BS, WS, US	R	Request-Reply
Alive	To make sure a registered process is still responsive	R	GM, P, BS, WS, US	Request-Reply
Dead Process	To let others know that a process have become unresponsive	R	GM	Request-Reply
Register Game Process	For the game manager to inform the registry of a new games	GM	R	Request-Reply
Game List	To get a list of games (in some state)	P	R	Request-Reply
Join Game	To all a game process to join an available game	P, BS, WS, US	GM	Request-Reply
Validate Process	For a GM to valid a process before allowing it to join	GM	R	Request-Reply
Game Status	To inform others of the status of a game, including which processes are still in the game	GM	R, P, BS, WS, US	Custom with TCP Stream
Buy Balloon	For a player to buy a balloon	P	BS	Request-Reply
Fill Balloon	For a player to filled an empty balloon	P	WS	Request-Reply
Throw Balloon	For a player to thrown a filled balloon at another player	P	GM	Request-Reply
Hit By Balloon	To notify a player that it was hit by a filled balloon from a specific player	GM	P	Request-Reply
Umbrella Auction	For an conducting umbrella auctions	US	P	Custom

Raise Umbrella	For a player to raise an umbrella	P	GM	Request-Reply
Umbrella Lowered	For a game manager to tell a player that its umbrella has been lowered	GM	P	Request-Reply
Leave Game	For a game process to voluntarily leave a game	P, BS, WS, US	GM	Request-Reply
Shutdown	To shutdown all processes	R	GM, P, BS, WS, US	Unreliable multicast

2 MESSAGES AND SHARED OBJECTS

Figures 1-6 show a class hierarchy of all the messages used in DSoak protocols and Tables 1-6 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.

Figure 1 – Abstract Message Classes

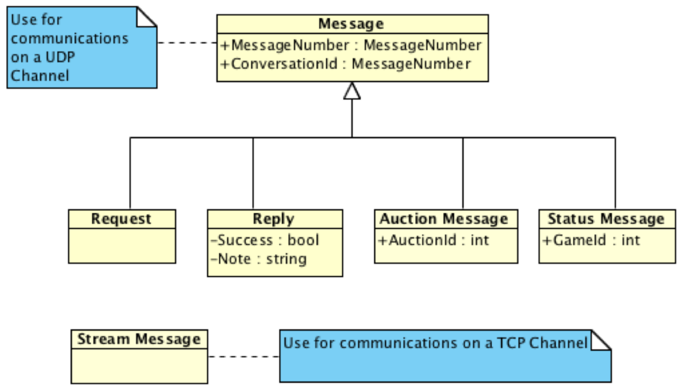

Use for communications on a UDP Channel
Message
+MessageNumber : MessageNumber
+ConversationId : MessageNumber
Request
Reply
-Success : bool
-Note : string
Auction Message
+AuctionId : int
Status Message
+GameId : int
Stream Message
Use for communications on a TCP Channel

Figure 2 – Request Message Classes

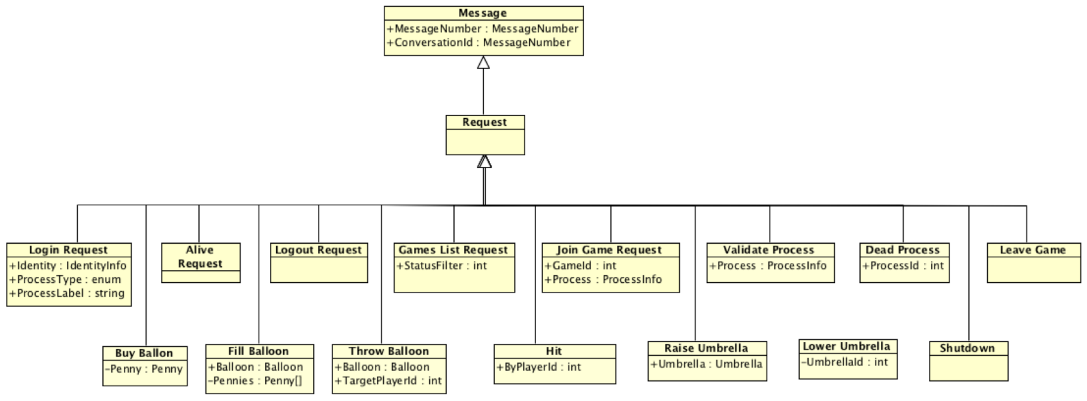


Figure 3 – Reply Message Classes

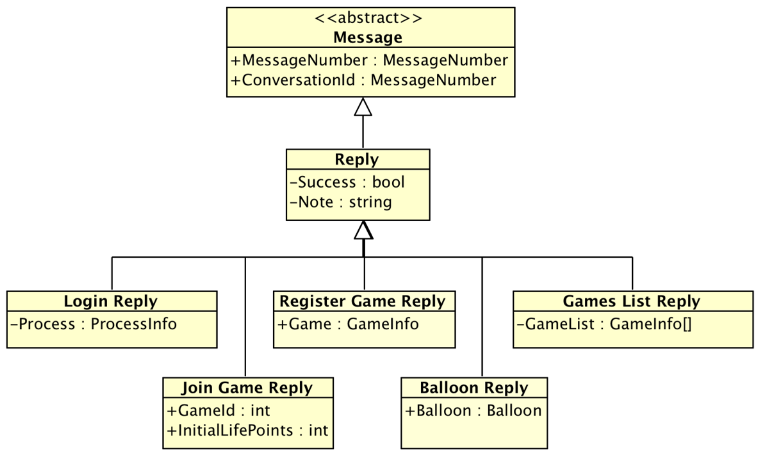


Figure 4 – Auction Message Classes

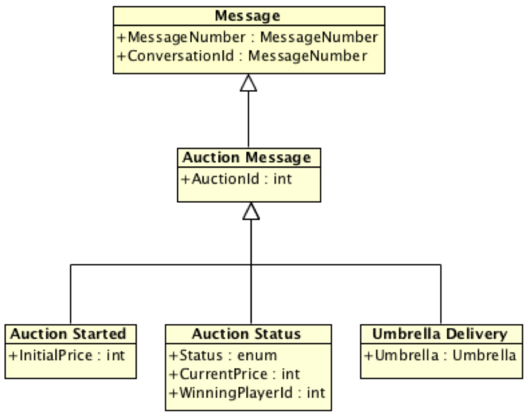


Figure 5 – Status and Stream Message Classes

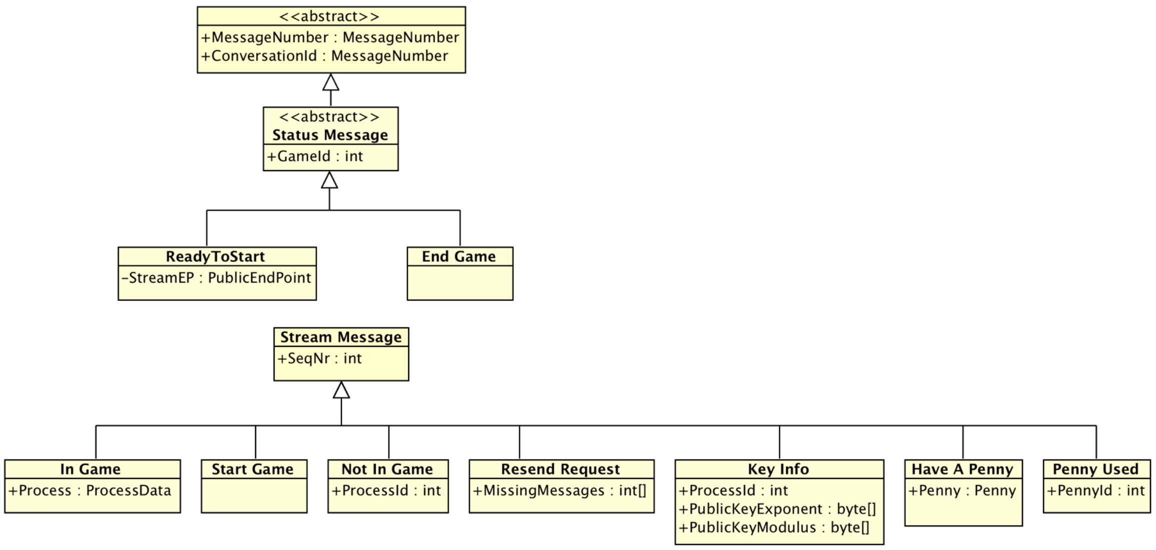


Figure 6 – Shared Objects

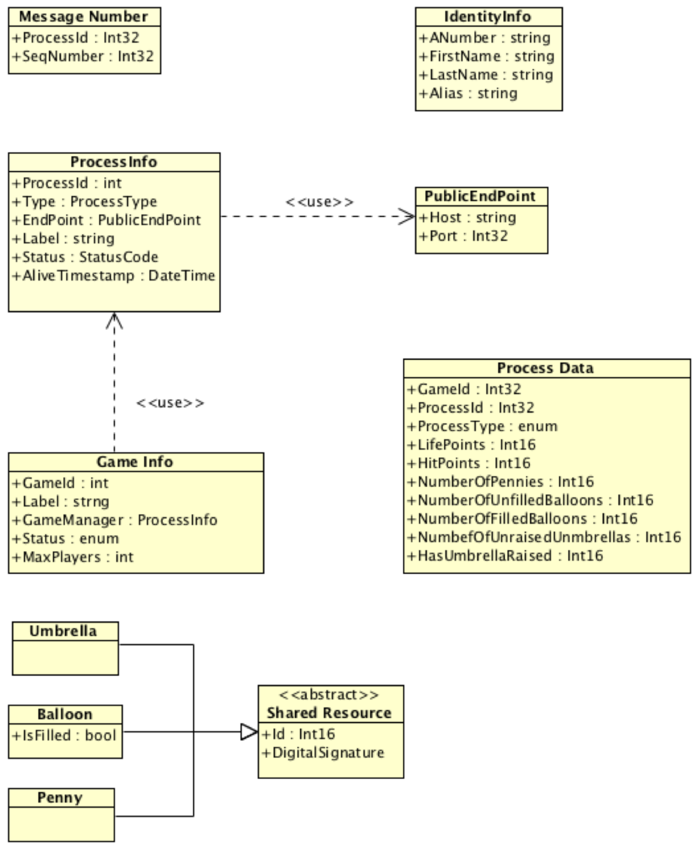


Table 2 – Request Message Class Descriptions

Note: All concrete request message classes inherit from Message and Request, and therefore include the ConversationId and MessageNr attributes. Some message classes contain no other attributes. They are still listed in the table below, but with attributes.

Class Name	Attribute, as it appears in the JSON Serialization	Attribute Type	Meaning
(all request 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	Identity	Identity	An object that holds the identity of the end user.
AliveRequest:#Messages	--		
LogoutRequest:#Messages	--		
GameList:#Messages	StatusFilter	int	This is a selection filer, encoded as a bit mask,

			with the following meanings for the bits: NotInitialized=1, Initializing=2, Available=4, Starting=8, InProgress=16, Ending=32, Complete=64, Cancelled=128. Pass a -1 (all bit set) to get all games
JoinGameRequest:#Messages	GameId	int	The identifier of the game to join
	Process	ProcessInfo	Information about the player requesting to join the game. This information is what the Registry returns during login.
ValidateProcess:#Messages	Process	ProcessInfo	Information about a process that needs to be validated.
DeadProcess:#Messages	ProcessId	int	The identifier of the process discovered to be dead
LeaveGame:#Messages	--		
BuyBalloon:#Messages	Penny	Penny	A valid, unused penny
FillBalloon:#Messages	Balloon	Balloon	A valid, unfilled, unused balloon
	Pennies	Penny[]	A array of two valid, unused pennies
ThrowBalloon:#Messages	Balloon	Balloon	A valid, filled, unused balloon
	TargetPlayerId	int	An identifier of another player in the game
HitBalloon:#Messages	ByTargetPlayerId	int	The identifier of another player, specifically the one through balloon
RaiseUmbrella:#Messages	Umbrella	Umbrella	A valid, unused umbrella
LowerUmbrella:#Messages	UmbrellaId	int	The identifier of umbrella that was just lowered
Shutdown	--		

Table 3 – Reply Message Class Descriptions

Note: All concrete reply message classes inherit from Message and Reply, and therefore include the ConversationId, MessageNr, Success, and Note attributes. Some message classes contain no other attributes. They are still listed in the table below, but with attributes.

Class Name	**Attribute, as it appears in the JSON Serialization**	**Attribute Type**	**Meaning**
(all reply message)	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
Reply:#Message	--		
LoginReply:#Messages	Process	ProcessInfo	Information about the process that is logging in, including its process id and publically visible end point
GameListReply:#Message	GameList	GameInfo[]	A list of GameInfo objects, where the status of each the matches the status filter specified in the GameListRequest
JoinGameReply:#Message	GameId	int	The identifier from a game to join, found in a GameInfo object returned in the GameListReply
	InitialLifePoints	int	The initial life points for a player
BalloonReply:#Message	Balloon	Balloon	A valid balloon. It may be filled or unfilled depending on the context

Table 4 – Auction Message Class Descriptions

Note: All concrete reply message classes inherit from Message and AuctionMessage, and therefore include the ConversationId, MessageNr, and AuctionId attributes. Some message classes contain no other attributes. They are still listed in the table below, but with attributes.

Class Name	**Attribute, as it appears in the JSON Serialization**	**Attribute Type**	**Meaning**
(all auction message)	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.
	AuctionId	int	A unique identifier for an auction.
AuctionStarted:#Messages	InitialPrice	Int	A number that represent the initial price for the umbrella up for auction.
AuctionBid:#Messages	Bid	Penny[]	
AuctionStatus:#Messages	Status	Int	An status code with the following possible values: 1 – Price Lowered, 2 – Auction Ended and You Didn't Win, 3 – Auction Ended and You Won.
	Current Price	Int	If status code=1, then this is the new prices; otherwise it is ending price
	Highest Bid	Int	
	NumberHighestBidder	Int	
	WinningPlayerId	int	If status code<>1, then this is identifier of the player who won the auction. If there was no winner, then it will be 0.
UmbrellaDelivery:#Message	Umbrella	Umbrella	A valid, unused umbrella

Table 5 – Status Message Class Descriptions

Note: All concrete status message classes inherit from Message and StatusMessage, and therefore include the ConversationId and MessageNr attributes. Some message classes contain no other attributes. They are still listed in the table below, but with attributes.

Class Name	**Attribute, as it appears in the JSON Serialization**	**Attribute Type**	**Meaning**
(all status message)	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.
	GameId	int	A number that identifies the game the game manager is hosting and other processes are involved in. This number can used to help valid the message.
ReadyToStart:#Messages	StreamEP	PublicEndPoint	This is the end point of a TCP stream for this conversation. The receiving process needs to connect to this end point. After the connection is form and up to the End Game message, stream messages (see Table 6) will be transmitted on the new TCP connection.
EndGame:#Message	--		

Table 6 – Stream Message Class Descriptions

Note: All concrete stream message classes inherit from Stream Message (and not Message). Messages will be separated on the stream with and EOL character.

Class Name	Attribute, as it appears in the JSON Serialization	Attribute Type	Meaning
(all stream messages)	SeqNr	int	A unique number from an incrementing sequence for each message
InGame:#StreamMessages	Process	ProcessData	Information about a process that is in a game
StartGame:# StreamMessages	--		
ResendRequest:# StreamMessages	MissingMessages	Int[]	Empty array resend a complete list of Processes in the Game; otherwise an array numbers of to send.
NotInGame:# StreamMessages	ProcessId	int	Identifier of a process no longer in a game
KeyInfo:# StreamMessages	ProcessId	Int	Identifier of the process which own the key
	PublicKeyExponent	byte[]	Exponent part of the public key
	PublicKeyModulus	Byte[]	Modulus part of the public key
HaveAPenny:# StreamMessages	Penny	Penny	A penny being given the receiving process
PennyUsed::# StreamMessages	PennyId	int	Identifier of a penny that has been used somewhere

Table 7 – Shared Object Class Descriptions

Instances of this classes are parts of messages and therefore their serialization will be embedded in a message serializations. JSON encapsulates them inside of brackets, but does not include the class names.

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	ANumber	string	The end user's A#
	Alias	string	An alias for the end user. This string will be displayed in various places and made available to other processes
	FirstName	string	The end user's real first name
	LastName	string	The end user's real last name
ProcessInfo	ProcessId	int	A unique identify for the process in the context of a registry, i.e., installation of DSoak.
	EndPoint	PublicEndPoint	The process's publically visible end point.
	Label	String	A label that can be displayed with process information.

	Status	int	The last known state of the process. See C# class definition for value of the enumeration.
	Type	int	An indication of the process's type, e.g. Player, GameManager, Balloon Store, etc. See C# class definition for value of the enumeration.
	DigitalSignature	Byte Array	Not used yet
	Wins	int	The number of Wins this process has had since starting up. Only used for Player processes
	Draws	int	The number of Draws this process has had since starting up. Only used for Player processes
	Losses	int	The number of losses this process has had since starting up. Only used for Player processes
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
GameInfo	GameId	int	The identifier for game
	Label	string	A label for the game that can be used for displayed purposes
	GameManager	ProcessInfo	Process information about the game manager hosting the game. This include public end point for the game manager
	Status	int	The status of the game: NotInitialized=1, Initializing=2, Available=4, Starting=8, InProgress=16, Ending=32, Complete=64, Cancelled=128
	MaxPlayers	int	The maximum number of players allowed to join the game.
Umbrella	Id	int	A unique identifier
	DigitalSignature	Byte[]	A byte array contains a simple digital signature
Balloon	Id	int	A unique identifier
	DigitalSignature	Byte[]	A byte array contains a simple digital signature
	IsFilled	Bool	True if the balloon is filled with water
Penny	Id	Int	A unique identifier
	DigitalSignature	Byte[]	A byte array contains a simple digital signature

3 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.

3.1 REQUEST-REPLY

Figure 7 – Request and Reply Communication Pattern

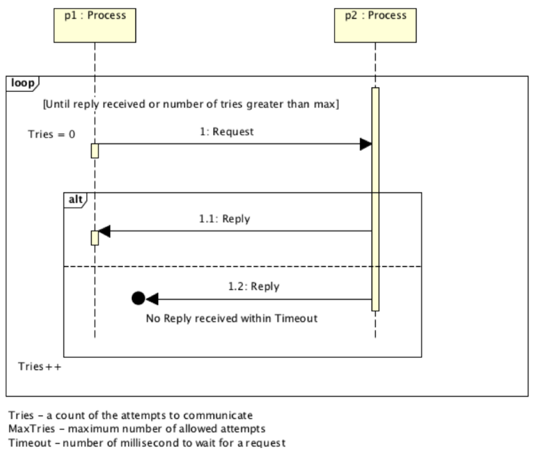

p1 : Process
p2 : Process
loop
[Until reply received or number of tries greater than max]
Tries = 0
1: Request
alt
1.1: Reply
1.2: Reply
No Reply received within Timeout
Tries++
Tries – a count of the attempts to communicate
MaxTries – maximum number of allowed attempts
Timeout – number of millisecond to wait for a request

Figure 7 show a basic sequence diagram for a request-reply pattern with the following substitutable 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 that 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.

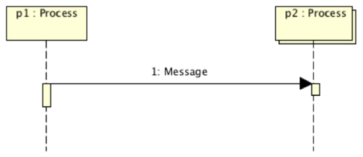
The Tries variable is an internal count of the attempts.

3.2 Unreliable Multicast

Figure 8 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 8 – Unreliable Multicast Communication Pattern



4 Communication Protocols

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

4.1 Login (Request-Reply)

Used by any process (except the Registry) to login to the system

Message Sequence:

Initiating Process → (LoginRequest) → Registry
Registry → (LoginReply) → Initiating Process

Semantics 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 Registry will try to authenticate the identity in the LoginRequest against the list of recognized end users.
- If the Registry 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 Registry 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 registry will resend a successful LoginReply. The registry uses the publically visible end point of the process to determine whether it also has information on that process.

4.2 Logout (Request-Reply)

Used by any process (except the Registry) to logout to the system

Message Sequence:

> Initiating Process → (LogoutRequest) → Registry
> Registry → (Reply) → Initiating Process

Semantics and Behaviors:

- Any process (except the Registry) can logout at any time by sending the registry a LogoutRequest message from its main communication channel (the one that it used to log in).
- When the registry 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 registry send back a Reply message with a Success=false and the error message in the Note field.

4.3 Alive (Request-Reply)

Used by Registry to determine is a process is still alive

Message Sequence:

> Registry → (AliveRequest) → Receiving Process
> Receiving Process → (Reply) → Registry

Semantics and Behaviors:

- The registry periodically sends an Alive Request all processes that currently logged in.
- Each Alive Request represents a new conversation and therefore contains a unique conversation id
- The receiving process must response as quickly as possible to the AliveRequest with a Reply message. The Reply must contain the same conversation id that was in the AliveRequest. The Success field of the Reply message must be set to true.
- If the registry does not receive the Reply in with the timeout limit, it will try sending the same AliveRequest again and a third time if necessary. If the receiving process still has not responded, then the Registry is conclude that the receiving processing is dead and will automatically log it out.
- When the registry logs a process out, it will initiate a Dead Process conversation with the game manager that may have been working with that process.

4.4 Dead Process (Request-Reply)

Used by the registry to notify game managers that a process in its game is unresponsive and therefore should be removed from the game.

Message Sequence:

Registry → (Dead Process) → Game Manager
Game Manager → (Reply) → Registry

Semantics and Behaviors:

- When registry logs a process out, either a consequence of a successful logout conversation or a failed alive conversation, it determines if the process was a part of a game. If it was, then the registry initiates a dead process conversation with the game manager of that game by sending it a DeadProcess message. The ProcessId field in the DeadProcess message is the identifier of the recently logged out process.
- When the game manage receives the DeadProcess message, it removes the specified process from the game and sends back a Reply message. The Success field will be true if the process was in fact in the game; otherwise it will be false.

4.5 Game List (Request-Reply)

Used by a player to discover games.

Message Sequence:

Player → (GameListRequest) → Registry
Registry → (GameListReply) → Player

Semantics and Behaviors:

- A player initiates the conversation by sending a GameListRequest message to the registry. The StatusFilter field of the message must be a non-zero bitmask (integer) value. Each bit in the bitmask correspond to a desired status. See GameInfo.PossibleStatusCodes for a list of the bit values and their meanings.
- Any game with one for the desired statuses will be returned to the player in a GameListReport.
- A StatusFilter of 0 returns no games
- A StatusFilter of -1 returns all games

4.6 Join Game (Request-Reply)

Used by a game process to join an available game

Message Sequence:

Game Process → (JoinGameRequest) → Game Manager
Game Manager → (JoinGameReply) → Game Process

Semantics and Behaviors:

- A game process initiate the conversation by sending a JoinGameRequest message to a game manager. The JoinGameRequest message must contain the game id of an available game currently hosted by a game manager, plus the game process's information. The process information is what was returned by the registry during login.
- When the game manager receives a JoinGameRequest, it will initiate a validate process conversation with registry to make sure that the game process logged before replying back on the join request.
- If the game process is valid and the specified game is available on game manager, the game manager will be reply back to the game process with a JoinGameReply message with Success=true. That message will contain the game id (the same as the one in the JoinGameRequest message) and the game processes initial life points. The latter item only pertains to players.
- If the game manager cannot allow the game process to join the game for any reason, then the game manager will send by a JoinGameReply with Success=false and a reason in the Note field.

4.7 Validate Process (Request-Reply)

Used by any process (except the registry) to validate another process, which to check to see if it is logged in.

Message Sequence:

> Initiating Process → (ValidateProcess) → Registry
> Registry → (Reply) → Initiating Process

Semantics and Behaviors:

- A process will initiate the conversation by sending a ValidateProcess message to the Registry with Process field set to the id of the target process that needs to be checked.
- If the target process is logged in, the registry will return a Reply with Success=true.
- If the target process is not logged in, the registry will return a Reply with Success=false.

4.8 Game Status (Custom)

Used by game managers to communicate game status information to the registry and other game processes.

Message Sequence:

> *On main UDP communication channel*
> Game Manager → (ReadyToStart) → Game Process
>
> *New TCP communication channel*
> Game Process → connects to specified TCP port → Game Manager
>
> *TCP Channel: When the game starts*
> Game Manager → Start Game → Game Process

TCP Channel: As needed either before or after start
Game Manager → InGame → Game Process
Game Manager → KeyInfo → Game Process
Game Manager → HaveAPenny → Game Process
Game Manager → NotInGame → Game Process
Game Manager → PennyUsed → Game Process
Game Process → ResendRequest → Game Manager
Game Process → PennyUsed → Game Manager

On main UDP communication channel
Game Manager → (EndGame) → Receiving Process

Semantics and Behaviors:

- The game manager will initiate the conversation by a) listening on a TCP channel and b) sending a ReadyToStart message to another process that contains the port number for that channel.
- When the receiving process gets a ReadyToStart message, it needs to connect a TCP channel to the port specified in the ReadyToStart message.
- The game process needs to keep the conversation and TCP channel open until an EndGame message is received.
- Each message send by the game manager on the TCP will contain a sequence number. The game process can request one or more messages to be resent by sending a list of sequence numbers to the game manager on the TCP channel. The game manage will resend each specified message, plus all messages after the last requested message.

4.9 BUY BALLOON (REQUEST-REPLY)

Used by a player to buy a balloon from a balloon store

Message Sequence:

Player → (BuyBalloonRequest) → Balloon Store
Balloon Store → (BalloonReply) → Player

Semantics and Behaviors:

- A player will initiate the conversation by sending a BuyBalloonRequest message to a balloon store that is part of the same game. The BuyBalloonRequest message must contain a valid, unused penny.
- If the balloon store has an available balloon, the player is valid, and penny is valid and unused, the balloon store will take a balloon out of its inventory and send it back to the player in a BalloonReply with Success=true. The balloon store will also send a PennyUsed message to the game manager as part of the GameStatus conversation.
- Otherwise, the balloon store will send back a BalloonReply with Success=false and a reason in the Note field.

4.10 Fill Balloon (Request-Reply)

Used by a player to fill a balloon with a water source

Message Sequence:

> Player → (FillBalloonRequest) → Water Source
> Registry → (BalloonReply) → Player

Semantics and Behaviors:

- A player will initiate the conversation by sending a FillBalloonRequest message to a balloon store that is part of the same game. The FillBalloonRequest message must contain a valid, unfilled balloon and two valid, unused pennies (in an array).
- If the water source has available water, the player is valid, and pennies are valid and unused, then the water source will take a unit of water out of its inventory, fill the balloon, and send it back to the player in a BalloonReply with Success=true.
- Otherwise, the water store will send back a BalloonReply with Success=false and a reason in the Note field.

4.11 Throw Balloon (Request-Reply)

Used by a player throw a balloon at another player

Message Sequence:

> Player → (ThrowBalloonRequest) → Game Manager
> Game Manager → (Reply) → Player

Semantics and Behaviors:

- A player will initiate the conversation by sending a ThrowBalloonRequest message to a game manager that is hosting the game it is in. The ThrowBalloonRequest message must contain a valid, filled balloon and a target player's id.
- If the player is valid, target player id is valid, and the balloon is valid, filled, the game manager will do two things: a) send back a Reply with Success=true and b) initiate a hit conversation with the target player.
- Otherwise, the game manager will send back a Reply with Success=false and a reason in the Note field.

4.12 Hit By Balloon (Request-Reply)

Used by a game manager to notify a player that it has been hit by a balloon

Message Sequence:

> Game Manager → (Hit Notification) → Player
> Player → (Reply) → Game Manager

Semantics and Behaviors:

- A game manager will initiate the conversation by sending a Hit Notification message to a player. The Hit Notification will contain the id of the player who throw the balloon.
- The player should deduct one life point when it receives the message.
- Game managers, balloon stores, water sources, and umbrella suppliers will not engage in conversations will players whose life points have gone to 0.
- The player could initiate a Leave Game conversation if its life points go to zero.

4.13 Umbrella Auction (Custom) – Optional Extra Credit

Used by an umbrella supplier to notify a player of a new auction.

Message Sequence:

At the beginning of a new auction
Umbrella Supplier → (Auction Started) → Player

Periodically
Umbrella Supplier → (Auction Status) → Player

Periodically
Umbrella Supplier → (Umbrella Delivery) → Player

Semantics and Behaviors:

- When an umbrella Supplier start an auction, it will notify the all of the players in the game by sending a Auction Started message.
- Then, periodically while the auction is still in progress, it will send an Auction Status message.
- Finally, after the auction ends, it will send an Umbrella Delivery message to the winner, if there was one.

4.14 Raise Umbrella (Request-Reply) – Optional Extra Credit

Used by a player to raise an umbrella

Message Sequence:

Player → (Raise Umbrella) → Game Manager
Game Manager → (Reply) → Player

Semantics and Behaviors:

- The Raise Umbrella message must contain a valid, unused umbrella
- If the raising is successful, then the Reply.Success will be True; otherwise, it will be false.

4.15 Umbrella Lowered (Request-Reply) – Optional Extra Credit

Used by a notify the player that it's umbrella has been lowered

Message Sequence:

Game Manager → (Lower Umbrella) → Player
Player → (Reply) → Game Manager

Semantics and Behaviors:

- If the player doesn't respond, it will be removed from the game.

4.16 Leave Game (Request-Reply)

Used by a game process to voluntarily leave a game

Message Sequence:

Game Process → (LeaveGameRequest) → Game Manager
Game Manager → (Reply) → Game Process

Semantics and Behaviors:

- A game process initiate the conversation by sending a LeaveGameRequest message to a game manager. The game manager must be the manager of the game to which the game process current belongs.
- If the game process is part of the manager's game, the game manager will be reply back to the game process with a Reply message with Success=true.
- Otherwise, the game manager will send back a Reply with Success=false and a reason in the Note field.

4.17 Shutdown (Unreliable Multicast)

Used by the registry to shut down all other process

Message Sequence:

Registry → (Shutdown) → ReceivingProcess

Semantics and Behaviors:

- On shutdown, the registry sends a Shutdown message to all process that are currently logged in.
- When a receiving process gets a Shutdown message and it is from the registry, it needs to stop gracefully.