Robocup Rescue Simulation - Communication Protocol

Cameron Skinner
Department of Computer Science
The University of Auckland

October 29, 2008

1 Introduction

This protocol describes the data exchange between the kernel, gis, simulators and agents in the Robocup Rescue Simulation software. The protocol has two layers, the transport layer and the content layer.

All numbers in both layers are encoded with the most significant byte first (big-endian).

2 Transport Layer

The transport layer is responsible for sending a *message* (i.e. a number of bytes) between simulator components. There are two transport mechanisms at present: UDP/IP or TCP/IP. The sending of a message may or may not block. Implementers are encouraged to provide both blocking and non-blocking implementations of the transport layer.

2.1 UDP

UDP packets may not be larger than 65536 bytes. Messages that are sent over UDP might be larger than this limit, so the UDP layer is wrapped in the *LongUDP* protocol.

LongUDP breaks the message into pieces and adds an 8 byte header to each piece. The size of the pieces is arbitrary but is guaranteed to be less than 65528. Each message is assigned a unique ID and each piece of the message is assigned a sequence number. Sequence numbers start from zero. All values are unsigned.

Note that UDP does not guarantee delivery so partial packets may result from the use of LongUDP. These should be discarded after a suitable timeout.

The 8 byte header is as shown in Table **??**.

Once all *total* pieces have been received the data segments can be concatenated in order of sequence number to reconstruct the original message.

2.2 TCP

The TCP protocol sends the data prefixed by a 32-bit integer specifying the length of the message.

3 Content Layer

The content layer describes the format of the messages that are exchanged between simulator components.

There are several types of data that can be encoded. Integers are encoded big-endian. Lists of integers are encoded by writing the length of the list as a 32-bit integer followed by the content of the list.

Properties are encoded as shown in Table **??**. Lists of properties are zero-terminated. Similarly, objects are encoded as shown in Table **??** and lists of objects are zero terminated. It is possible to encode only those properties that have changed, when appropriate. Implementors are encouraged to provide functions for writing all properties and for writing only those properties that have been modified.

An example of a list of one object is shown in Figure **??**.

Strings are encoded in ASCII and prefixed with the length of the string in bytes. Figure **??** has an example of this encoding.

The list of possible object types is shown in Table **??**. The list of property types is shown in Tables **??** and **??**.

3.1 Commands

Commands consist of a 32-bit header describing the type of the command, a 32-bit integer containing the size of the command in bytes followed by the content of the command. Commands can be concatenated into a zero-terminated list. An example of a list of two fictional commands is shown in Table **??**. When encoding objects, generally only those properties that have been changed will be written.

The list of possible command types and their header values is shown in Table **??** and is broken down into those that concern the GIS, simulators and agents.

3.1.1 GIS Commands

When the kernel starts it needs to connect to the GIS with a KG_CONNECT command. The GIS replies with either a GK_CONNECT_OK or GK_CONNECT_ERROR. The kernel replies with a KG_ACKNOWLEDGE if the connection was successful. The formats of these messages are shown in Table **??**

Byte Offset	Value	Meaning
0	0x0008	Magic number
2	ID	The ID of the message
4	Sequence number	The ID of this piece of the message
6	Total	The total number of pieces

Table 1: The LongUDP header

Field	Data type	Meaning
Header	int	Property type
Length	int	Size of property data in bytes
Value	int **OR** int list	Property value

Table 2: Property encoding format

Field	Data type	Meaning
Header	int	Object type
Length	int	Size of object data in bytes
ID	int	Object ID
Data	Property list	Properties of this object

Table 3: Object encoding format

Byte Offset	Value	Meaning
0	0xE8	Object header: TYPE_CIVILIAN
4	0x22	Length of object data (40 bytes)
8	0x10	ID of the object
12	0x06	Property header: POSITION
16	0x04	Length of property data (4 bytes)
20	0xAABBCCDD	Value of property
24	0x07	Property header: POSITION_HISTORY
28	0x0C	Length of property data (12 bytes)
32	0x02	Number of entries
36	0x11223344	Entry 1
40	0x55667788	Entry 2
44	0x00	Property header: NULL
48	0x00	Object header: NULL

Figure 1: An example object encoding

Byte Offset	Value	Meaning
0	0x0D	Length of the string
4	0x48	H
5	0x65	e
6	0x6c	l
7	0x6c	l
8	0x6f	o
9	0x20	
10	0x57	W
11	0x6f	o
12	0x72	r
13	0x6c	l
14	0x64	d
15	0x21	!

Figure 2: An example string encoding

Byte Offset	Value	Meaning
0	0x99	Command type (fictional command “99”)
4	4	Length in bytes
8	1234	Data
12	0x99	Command type (another fictional “99” command)
16	8	Length in bytes
20	4321	Data
24	8765	Data
28	0	The list is zero-terminated

Table 4: A list of two fictional commands

Data type	Meaning	Notes
	KG_CONNECT	
int	Version	Unused
	GK_CONNECT_OK	
Object list	The objects in the world	
	GK_CONNECT_ERROR	
string	The reason for the error	
	KG_ACKNOWLEDGE	

Table 5: GIS commands

3.1.2 Viewer Commands

Viewers connect to the kernel with a VK_CONNECT command. The kernel replies with KV_CONNECT_OK or KV_CONNECT_ERROR and the viewer acknowledges a successful connection with VK_ACKNOWLEDGE. The message formats are shown in Table **??**.

3.1.3 Simulator Commands

Simulators connect to the kernel with an SK_CONNECT command. The kernel replies with KS_CONNECT_OK or KS_CONNECT_ERROR and the simulator acknowledges a successful connection with SK_ACKNOWLEDGE. Each timestep the simulators will receive a COMMANDS message (described in section **??**) and must reply with an SK_UPDATE. Once all simulators have replied the kernel will send an UPDATE message (also described in section **??**). The message formats are shown in Table **??**.

3.1.4 Agent Commands

Agents connect to the kernel with an AK_CONNECT command. The kernel replies with KA_CONNECT_OK or KA_CONNECT_ERROR and the agent acknowledges a successful connection with AK_ACKNOWLEDGE. The message formats are shown in Tables **??** and **??**.

3.1.5 Broadcast Commands

Each timestep the kernel collects all agent commands and broadcasts them to simulators, viewers and log files via a COMMANDS messages. Similarly, updates from simulators are broadcast to simulators, viewers and log files with an UPDATE message. These messages are described in Table **??**.

Data type	Meaning	Notes
	VK_CONNECT	
int	Version	Unused
	KV_CONNECT_OK	
Object list	The objects in the world	
	KV_CONNECT_ERROR	
string	The reason for the error	
	VK_ACKNOWLEDGE	

Table 6: Viewer commands

Data type	Meaning	Notes
	SK_CONNECT	
int	Version	Unused
	KS_CONNECT_OK	
int	The ID of the simulator	
Object list	The objects in the world	
	KS_CONNECT_ERROR	
string	The reason for the error	
	SK_ACKNOWLEDGE	
int	The ID of the simulator	Taken from KS_CONNECT_OK
	SK_UPDATE	
int	The ID of the simulator	Taken from KS_CONNECT_OK
int	Time	
Object list	Changed objects	

Table 7: Simulator commands

Data type	Meaning	Notes
	AK_CONNECT	
int	Temporary ID	Unique temporary ID
int	Version	Unused
int	Agent type	Logical OR of requested types
	KA_CONNECT_OK	
int	Temporary ID	Taken from AK_CONNECT
int	The real ID of the agent	Assigned by kernel
Object	Self	Object controlled by this agent
Object list	The objects in the world	
	KA_CONNECT_ERROR	
int	Temporary ID	Taken from AK_CONNECT
string	The reason for the error	
	AK_ACKNOWLEDGE	
int	Agent ID	Taken from KA_CONNECT_OK
	KA_SENSE	
int	Agent ID	
int	Time	
Object list	All changed objects	
	KA_HEAR	
int	Agent ID	
int	ID of sender	
int	Channel number	
int	Message length	
byte array	The message	

Table 8: Agent commands

Data type	Meaning	Notes
	AK_REST	
int	Agent ID	
	AK_MOVE	
int	Agent ID	
int list	Path to be moved	
	AK_EXTINGUISH	
int	Agent ID	
nozzle list	List of nozzles to be used	
	Nozzle format	
int	Target ID	
int	Target direction	Unused
int	Nozzle x coordinate	Unused
int	Nozzle y coordinate	Unused
int	Amount of water	
	AK_LOAD	
int	Agent ID	
int	Target ID	
	AK_UNLOAD	
int	Agent ID	
	AK_RESCUE	
int	Agent ID	
int	Target ID	
	AK_CLEAR	
int	Agent ID	
int	Target ID	
	AK_TELL	
int	Agent ID	
int	Target channel	$\in [0, 255]$
int	Message length	
byte array	Message body	
	AK_CHANNEL	
int	Agent ID	
int	Desired channels length	
byte array	Desired channels	

Table 9: Agent commands (continued)

Data type	Meaning	Notes
	COMMANDS	
int	Time	
Command list	List of all commands	
	UPDATE	
int	Time	
Object list	All changed objects	

Table 10: Broadcast commands

Appendix A Tables of Constants

Type	Value
WORLD	0x01
ROAD	0x02
RIVER	0x03
NODE	0x04
RIVERNODE	0x05
BUILDING	0x20
REFUGE	0x21
FIRE_STATION	0x22
AMBULANCE_CENTER	0x23
POLICE_OFFICE	0x24
CIVILIAN	0x40
CAR	0x41
FIRE_BRIGADE	0x42
AMBULANCE_TEAM	0x43
POLICE_FORCE	0x44

Table 11: All possible object types

Type	Value
START_TIME	1
LONGITUDE	2
LATITUDE	3
WIND_FORCE	4
WIND_DIRECTION	5
HEAD	6
TAIL	7
LENGTH	8
ROAD_KIND	9
CARS_PASS_TO_HEAD	10
CARS_PASS_TO_TAIL	11
HUMANS_PASS_TO_HEAD	12
HUMANS_PASS_TO_TAIL	13
WIDTH	14
BLOCK	15
REPAIR_COST	16
MEDIAN_STRIP	17
LINES_TO_HEAD	18
LINES_TO_TAIL	19
WIDTH_FOR_WALKERS	20
SIGNAL	21
X	25
Y	26
FLOORS	28
BUILDING_ATTRIBUTES	29
IGNITION	30
FIERYNESS	31
BROKENNESS	32
BUILDING_CODE	34
BUILDING_AREA_GROUND	35
BUILDING_AREA_TOTAL	36
POSITION	38
POSITION_EXTRA	39
DIRECTION	40
STAMINA	42
HP	43
DAMAGE	44
BURIEDNESS	45
WATER_QUANTITY	46

Table 12: All property types that have integer data

Type	Value
SHORTCUT_TO_TURN	22
POCKET_TO_TURN_ACROSS	23
SIGNAL_TIMING	24
EDGES	27
ENTRANCES	33
POSITION_HISTORY	41

Table 13: All property types that have lists of integer data

Command	Header	Use
KG_CONNECT	0x10	Kernel connects to GIS
KG_ACKNOWLEDGE	0x11	Kernel acknowledges connection
GK_CONNECT_OK	x012	GIS accepts kernel connection
GK_CONNECT_ERROR	0x13	GIS rejects kernel connection
SK_CONNECT	0x20	Simulator connects to kernel
SK_ACKNOWLEDGE	0x21	Simulator acknowledges connection
SK_UPDATE	0x22	Simulator sends update to kernel
KS_CONNECT_OK	0x23	Kernel accepts simulator connection
KS_CONNECT_ERROR	0x24	Kernel rejects simulator connection
VK_CONNECT	0x30	Viewer connects to kernel
VK_ACKNOWLEDGE	0x31	Viewer acknowledges connection
KV_CONNECT_OK	0x32	Kernel accepts viewer connection
KV_CONNECT_ERROR	0x33	Kernel rejects viewer connection
AK_CONNECT	0x40	Agent connects to kernel
AK_ACKNOWLEDGE	0x41	Agent acknowledges connection
KA_CONNECT_OK	0x42	Kernel accepts agent connection
KA_CONNECT_ERROR	0x43	Kernel rejects agent connection
KA_SENSE	0x44	Kernel sends update to agent
KA_HEAR	0x45	Kernel sends an audio input to agent
UPDATE	0x50	Kernel broadcasts an update
COMMANDS	0x51	Kernel broadcasts agent commands
AK_REST	0x80	Agent does nothing
AK_MOVE	0x81	Agent moves
AK_LOAD	0x82	Agent loads a victim
AK_UNLOAD	0x83	Agent unloads a victim
AK_TELL	0x85	Agent sends a message
AK_EXTINGUISH	0x86	Agent extinguishes a fire
AK_RESCUE	0x88	Agent rescues a buried victim
AK_CLEAR	0x89	Agent clears a blocked road
AK_CHANNEL	0x90	Agent listens to channels

Table 14: All commands