The Aberoth Protocol

Delphi1024

June 9, 2018

1 Disclaimer

Please use the information contained in this document responsibly.
This document is based on personal research and might contain errors.

2 Data Type Conventions

Various data types are used in this document for clarifying the contents of the server messages and commands. These include basic types like single bytes as well as more complex ones like strings. The following section declares these data types:

byte A single eight-bit byte

short Two bytes in a row stored in big endian format (most significant byte first)

int Four bytes in a row stored in big endian format

String A String of characters. A String is stored by first storing the length of the String as a short and then following it up with the actual characters.

Command A Command is a sequence of bytes that cause the client to execute various actions. Each command starts with an identifier that identifies it. The length of the byte sequence is determined by the type of the command.

Command List A List of commands for the client. The list starts with a three byte value in big endian format specifying the length of the list. This field is followed by the actual commands.

Data A representative type for any number of bytes. This type is used when the content of the data is not relevant in the current context.

3 Protocol Structure

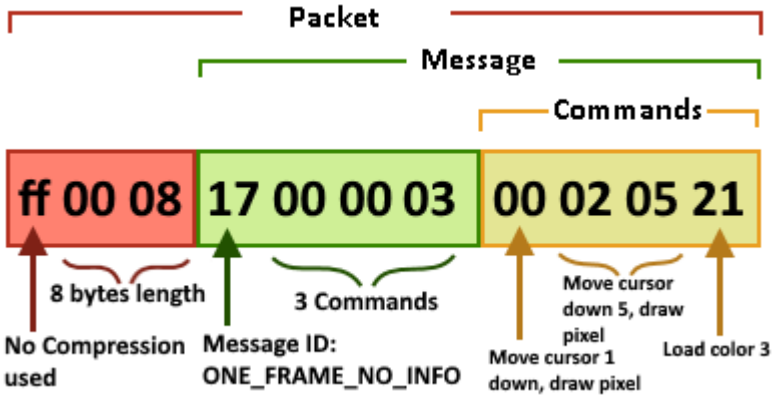
The basic protocol structure consists of packets which contain the server messages. Each packets begins with the compressionFlag followed by two bytes stating the length of the following data. Depending on the value of the compressionFlag, this data might be ZIP-compressed (using DEFLATE). If the compressionFlag has the value 0xff, the data is sent uncompressed otherwise ZIP-compression is used. In the last case, the compressionFlag is also used as part of the messageSize field.

Offset	Description	Data type
0x0000	messageSize MSB or compressionFlag	byte
0x0001	messageSize middle byte	byte
0x0002	messageSize LSB	byte
0x0004	message data	Data

The message data follows a similarly simple pattern: The first byte is used to identify the type of message, since each message has a unique Id. The different messages are covered in detail in a later part of this document.

Offset	Description	Data type
0x0000	Message identifier	byte
0x0001	Message content	Data

Here is an example of a packet that contains the ONE_FRAME_NO_INFO message with three commands:



3.1 Logging in

There is one exception to the above structure, which is the login handshake. To log in, the client sends all the required information to the server in form

of a single string. The string consists of multiple smaller strings which are seperated by the NULL (u+0000) character. The login string contains the following substrings, in the given order:

`playerName` The name of the account that is used to log in.

`password` The password of the account that is used to log in. The password is either sent in plain text, or the string "encryptMD5" is used, which causes the server to request a salted MD5 hash of the password. Neither option is particularly secure.

`screenDefinition` A numeric value determining the screen definition that should be used. The screen definition specifies the size of the main window before it is scaled. If this value is "1" then a widescreen resolution of 960x540 is used. A value of "0" or "-1" will set the resolution to 640x480. Any other value will cause the server to respond with a message explaining that the screen definition is invalid.

`scaleNumerator` The numerator of the scale factor that is used to scale the main window. The scale factor is represented as a fraction with the numerator and denominator saved separately.

`scaleDenominator` The denominator of the scale factor.

`fontSize` The selected font size.

`systemArchitecture` A string describing the User's system architecture. This includes the operating system and version, as well as the CPU architecture. While the Java client actually audits the system properties for this information, the HTML5 client sets this field to the string "Windows".

`javaEnvironment` A string describing the User's java environment. It consists of the Java version as well as the manufacturer of the Java virtual machine.

`unknown` An unknown boolean ("true" or "false") string. It is set to "true" by the Java client and "false" by the HTML5 client.

`hardwareAcceleration` A boolean string determining if hardware acceleration is available on the User's machine.

`graphicsMemory` The amount of available graphics card memory. Since this value is meaningless when no hardware acceleration is selected or available, it is set to 0 in these cases.

`signature` A boolean string verifying that the used client has a valid signature. Both the Java and the HTML5 client set this value to "true" without checking though...

After the login string has been sent, the server will respond with a number that describes the status of the login attempt. The server will follow that number up with a null-terminated string that contains additional data. The following status values are accepted by the client:

- A value of 0 indicates that an error has occured during the login. The data string can contain some memorable and helpful lines such as "Error! Please and try again later" [sic].
- A value of 1 means that the login was successful. The client has to acknowledge this message by sending a 0 to the server. After the 0 was received, the server will start sending messages in the stardard protocol structure.
- A value of 2 means that the message inside the data string should be displayed to the player.
- A value of 3 means that the server is full and the player is queued up to enter.
- A value of 4 means that the server is full and too many players are already queued up.
- A value of 5 is sent if the value of the `password` field was "encryptMD5". This value requests the MD5 hash of the password to be sent to the server. The server also sends a hash salt in the data string that should be appended to the password before hashing. The resulting hash should be encoded in hexadecimal form and is send to the server. After the hash has been sent, the server will send a new status update.

The status values 2, 3 and 4 display a message box to the player. Dismissing the message will cause the client to check if the server has sent a new status update. If this is the case, the status value is interpreted as explained above. Otherwise the login attempt is canceled. If the status value is 0, the login attempt is immediately canceled and no message is displayed.

3.2 Special Player Names

Some values for the `playerName` field trigger a special behaviour by the server:

3.2.1 New Character Creation

When creating a new character, the client crafts a special string for the `playerName`, which contains some initial parameters for the new character. The string consists entirely of the ASCII characters 0-9. It starts with 27 pseudorandom numbers, presumably to create a unique identifier for the character before it is named. It is followed by the starting realm of the character, the initial hair color, skin color, shirt color and, finally, gender. The specific values for each are listed below.

Value	Gender
1	Random
2	Male
3	Female
other	Random

Value	Shirt Color
1	Random
2	Red
3	Orange
4	Yellow
5	Green
6	Blue
7	Purple
other	Random

Value	Skin Color
1	Random
2	Light
3	Medium
4	Dark
other	Random

Value	Hair Color
1	Random
2	Light
3	Medium
4	Dark
5	Very Dark
other	Random

Value	Starting Realm
1	White/Knowledge
2	Black/Power
3	Green/Thought
4	Red/Life
5	Purple/Change
6	Yellow/Creation
7	Cyan/Protection
8	Blue/Deceit
other	Random

4 Client and Server Messages

4.1 CREATE_WINDOW Message

The CREATE_WINDOW message is usually the first server message sent after successfully logging in. It specifies the dimensions of the main window as well as which event listeners to enable.

Offset	Description	Data type
0x0000	Constant value 0x14 (Identifier)	byte
0x0001	windowId	byte
0x0002	title	String
0x00nn	screenWidth	short
0x00nn + 2	screenHeight	short
0x00nn + 4	events	byte

When this message is received the screenWidth and screenHeight fields are taken as the basis of the main window dimension. To construct the main window, these fields are scaled with the scale factor that was set in the login message. While the screen dimensions may be set arbitrarily, the game will still only send rendering commands conforming to the screen definition set in the login message. This will cause the screen to be cropped when the dimensions are smaller than the required screen definiton and render artifakts on the blank space when the screen dimensions are too big.

The windowId field assigns an Id to the window. This Id is used to later associate user input events with the current play session.

The events field of this message decides which event listeners are enabled. A value of 0 means that only keyboard events are listened for. A value of 1 only listenes for mouse clicks and mouse movements. A value of 2 listens for keyboard events and mouse clicks, while a value of 3 also listenes for mouse movement. Finally any other value will cause the client to only listen for keyboard and mouse click events, but not for mouse movement.

4.2 ONE_FRAME_NO_INFO Message

This message containes a list of commands to render one frame. They are executed in the order they are received.

Offset	Description	Data type
0x0000	Constant value 0x17	byte
0x0001	List of rendering commands	Command List

4.3 ONE_FRAME_WITH_INFO Message

The ONE_FRAME_WITH_INFO message is similar to the ONE_FRAME_NO_INFO message except that it also tests the users connection.

Offset	Description	Data type
0x0000	Constant value 0x0c	byte
0x0001	List of rendering commands	Command List
0x00nn	PING_ID_BANDWIDTH_CHECK Command	Command

This message ends with a PING_ID_BANDWIDTH_CHECK command which causes the client to respond with a FRAME_RECEIVED message, sending a larger amount of data to the server, for connection testing. It is important to note that the PING_ID_BANDWIDTH_CHECK command is not part of the command list and is therefore not included in the command lists length field.

4.4 USER_INPUT Message

The USER_INPUT Message is used send user input to the server.

Offset	Description	Data type
0x0000	Constant value 0x0e	byte
0x0001	windowId	byte
0x0002	inputCommands	Command list

The windowId must be the same id that got associated with the current play session by the CREATE_WINDOW message.

The command list in this message consists entirely of MOUSE_INPUT and KEYBOARD_INPUT commands. While this list can contain any number of commands, in practice the client will send a USER_INPUT message as soon as the input occurs. This causes the list to always contain exactly one command.

4.5 FRAME_RECEIVED Message

The FRAME_RECEIVED Message is sent by the client in response to a PING_ID_BANDWIDTH_CHECK command.

Offset	Description	Data type
0x0000	Constant value 0x0d	byte
0x0001	trigger	Command
0x0005	lots of data	Data

The trigger field contains the PING_ID_BANDWIDTH_CHECK command that this message is responding to. The actual data sent consists of a sequence of pseudorandom numbers with a length given by the echoLength field of the PING_ID_BANDWIDTH_CHECK command.

4.6 CLIENT_STATUS Message

The CLIENT_STATUS Message is sent by the client every 30 seconds to show the server that the connection has not been terminated.

Offset	Description	Data type
0x0000	Constant value 0x16	byte
0x0001	status	short

The status field is usually set to 65000, which is interpreted as a heartbeat by the server. Effects of other values are not known.

5 Rendering Concepts

To better understand the commands used by the client and server, it is important to know some of the rendering concepts that are used.

5.1 The Main Window

The window is the main visible part of a play session. While it holds many parameters that influence the rendering process, all the actual rendering is done in the subwindows. The task of the window is therefore to manage these subwindows.

These are the properties of the main window:

windowId A unique Id assigned to the main window when it is created by the CREATE_WINDOW message. It is used to bind user inputs to this window.

subwindows This is the array of subwindows. All subwindows that are creates get stored in this array.

activeSubwindow The Id of the currently active subwindow. The active subwindow is the one that receives all rendering commands.

previousSubwindow The Id of the subwindow that was active before the last subwindow switch. This is required to make the SWITCH_BACK_TO_PREVIOUS_SUB_WINDOW command work.

width The width of the window, before any scale factors are applied.

height The height of the window, before any scale factors are applied.

scaleFactor The factor to scale the main window with. This can be set in the login dialog or the HTML5 client's options.

`cursorX` The x-coordinate of the drawing cursor position. This position is interpreted as being in the active subwindow's coordinate system. The cursor position is used by the DRAW_FILLED_RECT commands to determine where a rectangle should be drawn.

`curosrY` The y-coordinate of the drawing cursor position. This position is interpreted as being in the active subwindow's coordinate system.

`colorCache` An array that stores various colors used throughout the rendering process.

`resourceCache` This field is an associative array that contains all external resources used by the window. These resource include images and sound effects.

5.2 Subwindows

The subwindows are what receive the rendering commands. All subwindows are arranged in a stack by the main window to form a frame. The subwindows with higher `subwindowId`s will be at the top and can overlap lower subwindows. Each subwindow has the following properties:

`subwindowId` The unique Id of this subwindow. This id is used to reference this subwindow in various commands. The highest valid Id is 255.

`xPosition` The x-position of this subwindow in the main window's coordinates. The position of a subwindow can only be set once when it is created an can not be changed afterwards.

`yPosition` The y-position of this subwindow in the window's coordinates.

`width` The width of the subwindow. It can only be set once when the subwindow is created.

`height` The height of the subwindow.

`rectangleWidth` The width of a rectangle that is rendered using one of the DRAW_FILLED_RECT commands.

`rectangleHeight` The height of a rectangle that is rendered using one of the DRAW_FILLED_RECT commands.

`roundingX` Rounding behaviour in x-direction. This field determines what happends when a scaling operation in x-direction does not result in an integer value. See the SET_FILLED_RECT_SIZE command for details.

`roundingY` Rounding behaviour in y-direction.

`textCache` An associative array containing all the text strings that are currently visible on this subwindow.

6 Commands

6.1 PING_ID_BANDWIDTH_CHECK Command

This Command requests the client to send a somewhat larger amount of data back to the server to check the connection and test the bandwidth of the user. The actual data send is just pseudorandom numbers.

Offset	Description	Data type
0x0000	Constant value 0x0b	byte
0x0001	unknown/unused	byte
0x0002	unknown/unused	byte
0x0003	echoLength	byte

The echoLength field determines how long the response to this command should be. A value of 1 means that the response should be 1 kiB (1024 Byte) in length. A value of 2 means the length should be 2 kiB (2048 Byte) and a value of 3 means 4 kiB (4096 Byte). Any other value causes the length to be 0 and an empty FRAME_RECEIVED message will be sent. The only exception to this is when the echoLength field itself is 0, in which case no respose will be sent. The unknown fields are probably a timestamp for when the command was sent from the server.

Interestingly, the server does not send this command properly. The mistake is, that the echoLength field is sent as the first field by the server, but read as the third by the client. This causes the client to read one of the unknown fields instead, which most of the time is not in the range required to trigger a meaningfull response. The Consequence of that error is, that the client almost never sends a proper FRAME_RECEIVED message to the client.

6.2 DRAW_FILLED_RECT_AT_Y_PLUS_ONE Command

This is one of the many commands that draw a rectangle on the screen. This variant moves the drawing cursor down one step and then fills a rectangle of the previously set size with the active subwindow's current color.

Offset	Description	Data type
0x0000	Constant value 0x00	byte

6.3 DRAW_FILLED_RECT_AT_X_PLUS_ONE Command

This command is similar to the previous one except the drawing cursor is moved one step to the right instead of down before filling the rectangle.

Offset	Description	Data type
0x0000	Constant value 0x03	byte

6.4 DRAW_FILLED_RECT_AT_DY Command

Another of the DRAW_FILLED_RECT variants this time with a variable step size. This command moves the drawing cursor down `delta` number of steps and then fills a rectangle the current color.

Offset	Description	Data type
0x0000	Constant value 0x02	byte
0x0001	delta	byte

6.5 DRAW_FILLED_RECT_AT_DX Command

The equivalent of the previous command for jumping to the right instead of down.

Offset	Description	Data type
0x0000	Constant value 0x04	byte
0x0001	delta	byte

6.6 DRAW_FILLED_RECT_AT_BLOCK_XY Command

This command sets the drawing cursor to the location specified by its `xPosition` and `yPosition` fields, then renders a rectangle at that location. The `xPosition` and `yPosition` fields are multiplied by the currently set rectangle dimensions before jumping.

Offset	Description	Data type
0x0000	Constant value 0x01	byte
0x0001	xPosition	byte
0x0002	yPosition	byte

6.7 DRAW_FILLED_RECT_AT_XY Command

This command is similar to the previous one, except that the `xPosition` and `yPosition` fields are not adjusted for the current rectangle size before jumping. This allows for rectangles to be rendered between pixels.

Offset	Description	Data type
0x0000	Constant value 0x05	byte
0x0001	xPosition	short
0x0002	yPosition	short

Note that the data type for the `xPosition` and `yPosition` fields is a short, instead of a byte like the previous command.

6.8 DRAW_FILLED_RECT_AT_Y_PLUS_ONE_REPEAT Command

This command is similar to the DRAW_FILLED_RECT_AT_Y_PLUS_ONE command, except that that the command is repeated multiple times.

Offset	Description	Data type
0x0000	Constant value 0x1b	byte
0x0001	repCount	byte

This command executes a DRAW_FILLED_RECT_AT_Y_PLUS_ONE command as many times as specified by the repCount field.

6.9 SET_FILLED_RECT_SIZE Command

The SET_FILLED_RECT_SIZE Command sets the size of the rectangles that are rendered by the DRAW_FILLED_RECT commands.

Offset	Description	Data type
0x0000	Constant value 0x19	byte
0x0001	width	short
0x0003	height	short
0x0005	rounding	byte

The width and height fields specify the width and height of the rectangles that should be rendered by subsequent DRAW_FILLED_RECT commands. Like most other dimensions, they are given without any scale factors applied. To get the actual dimensions of the rectangle, these fields have to be scaled by the scale factor from the login message.

The rounding field determines what should happend when applying the scale factor does not result in an integer value. This field is a bitfield, containing the rounding behaviour in x-direction as well as in y-direction. If the first bit (LSB) of the bitfield is set, scaling operation in x-direction should always round up to the nearest integer, otherwise they should be rounded down. The second bit specifies the rounding behaviour in the same way for scaling operations in y-direction.

6.10 DRAW_PIXEL Command

The DRAW_PIXEL command sets any pixel of the active subwindow to the subwindows current rendering color.

Offset	Description	Data type
0x0000	Constant value 0x12	byte
0x0001	xPosition	short
0x0003	yPosition	short

The pixel affected is given by the xPosition and yPosition fields.

6.11 COPY_AREA Command

The COPY_AREA Command copies the image data from one position on the active subwindow to another.

Offset	Description	Data type
0x0000	Constant value 0x15	byte
0x0001	sourceX	short
0x0003	sourceY	short
0x0005	width	short
0x0007	height	short
0x0009	destinationX	short
0x000b	destinationY	short
0x000d	destinationXSign	byte
0x000f	destinationYSign	byte

This command copies the area at position (sourceX, sourceY) with the dimensions specified by the width and height fields and pastes it at the position (destinationX, destinationY). The sign bits (destinationXSign and destinationYSign)specify wether the destination fields should be treated as negative values. A sign bit of 0 means the corresponding destination field should be used as is while a sign bit other then 0 means that it should be negated.

6.12 SET_COLOR Command

The SET_COLOR command sets the rendering color of the current subwindow.

Offset	Description	Data type
0x0000	Constant value 0x06	byte
0x0001	red	byte
0x0002	green	byte
0x0003	blue	byte

This command sets the rendering color of the active subwindow to the one supplied with the command. This causes all subsequent DRAW_FILLED_RECT commands to use this color.

6.13 SET_COLOR_WITH_ALPHA Command

The SET_COLOR_WITH_ALPHA command also sets the rendering color of the current subwindow, but has the capability to include an alpha (transparency) value in the color.

Offset	Description	Data type
0x0000	Constant value 0x1d	byte
0x0001	red	byte
0x0002	green	byte
0x0003	blue	byte
0x0003	alpha	byte

6.14 CACHE_CURRENT_COLOR Command

This command creates a new entry in the color cache for the current color of the active subwindow.

Offset	Description	Data type
0x0000	Constant value 0x07	byte
0x0001	cacheId	byte

The cacheId is used to reference the color in subsequent commands.

6.15 SET_COLOR_BASED_ON_CACHE Command

This command takes a color from the cache and modifies it slightly. This is the only command that makes use of the cached colors.

Offset	Description	Data type
0x0000	Constant value 0x1c	byte
0x0001	cacheId	byte
0x0002	colorDelta	byte

The cacheId field states which color from the color cache should be used as a basis for the color modification.

The colorDelta field is a bitfield specified how to modify the color. The red value of the color is modified by taking the three most significant bits of the delta field and subtracting 4 from them. The result is than added to the red value. The green value is modified the same way, but bits 3,4 and 5 are used instead. The blue value is modified by taking the two least significant bits and subtraction 2 from them. The result is then added to the blue value.

To load a color from the cache without changing it, the colorDelta field must have the value 0x92, which causes all the above calculations to add 0 to the color values.

6.16 SET_ON_SCREEN_TEXT Command

The SET_ON_SCREEN_TEXT command displays text on the active subwindow.

Offset	Description	Data type
0x0000	Constant value 0x08	byte
0x0001	textId	short
0x0003	xPosition	short
0x0005	yPosition	short
0x0007	lineShift	byte
0x0008	style	byte
0x0009	font	byte
0x000a	textContent	String

This command renders the text in the textContent field at the position given by the xPosition, yPosition and lineShift fields. The xPosition and yPosition fields set the point where the text is anchored on the subwindow and the lineShift field indicates how many lines the first line of text should be rendered above the anchor point. The lineShift field does not have to match the actual line count of the text.

The font field lets the server decide which font to use. In practice, this field chooses between three identical fonts on the Java client and is ignored completely on the HTML5 client.

Setting the textContent field to the empty string will remove the text from the active subwindow.

The style field specifies the style that the text should be rendered in. A value of 0 indicates plain text. A value of 1 renderes the text in *italics*, while a value of 2 causes the text to be rendered **bold**. Finally a value of 3 causes the text to be rendered in a smaller font with no special styling applied to it.

The textId field is used to reference the text in a later MOVE_ON_SCREEN_TEXT Command. Some values of textId also carry a special meaning with them:

- A value of 0 causes all text on the active subwindow to be cleared, with the exception of broadcast messages.
- A value of 2 will cause the text to be displayed in the bottom-left corner of the active subwindow. The text will also always be rendered white, regardless of the subwindow's current color. This textId is used to display the players stats.
- A value of 3 will cause the text to be displayed in the bottom-left corner of the active subwindow. The text will also always be rendered yellow, regardless of the subwindow's current color. This textId is used to display hint messages.
- A value of 4 indicates a broadcast message. It is always displayed at the top in the center of the active subwindow and can not be cleared.
- A value of 65001 indicates a status text. This text will always be rendered white and will always be displayed in the top-left.

- A value of 65535 will lock the yPosition field to be the center of the active subwindow. The xPosition field is unaffected.

6.17 MOVE_ON_SCREEN_TEXT Command

The MOVE_ON_SCREEN_TEXT command will update the position of text on the active subwindow.

Offset	Description	Data type
0x0000	Constant value 0x09	byte
0x0001	textId	short
0x0003	xPosition	short
0x0005	yPosition	short
0x0007	lineShift	byte

This command is basically the same as the SET_ON_SCREEN_TEXT command, but instead of supplying a string, it reuses the one with the given textId.

6.18 SUB_WINDOW commands

The SUB_WINDOW commands are commands that manage the main windows subwindows. The commands have a similar structure in that they all start with the constant 0x18, followed by a value that identifies the subcommand. Finally, they all contain the Id of the subwindow that they operate on.

6.18.1 CREATE_SUB_WINDOW Command

The CREATE_SUB_WINDOW command creates a new subwindow and adds it to the main window's subwindow list. The active subwindow will also be set to this new subwindow.

Offset	Description	Data type
0x0000	Constant value 0x18	byte
0x0001	Constant value 0x00	byte
0x0002	subwindowId	byte
0x0003	xPosition	short
0x0005	yPosition	short
0x0007	width	short
0x0009	height	short

This command creates a subwindow with the id subwindowId and adds it to the main windows list of subwindows. If a subwindow with the given id already exists, it is discarded the same way as by a DESTROY_SUB_WINDOW command.

The xPosition and yPosition fields determine the offset of the subwindow to the main windows origin. The width and height fields similary specify the dimensions of the new subwindow.

6.18.2 SWITCH_TO_SUB_WINDOW Command

This command changes the active subwindow to the one specified by the subwindowId in this command.

Offset	Description	Data type
0x0000	Constant value 0x18	byte
0x0001	Constant value 0x01	byte
0x0002	subwindowId	byte

After this command has been processed, the subwindow with the specified subwindowId will be active. The Id of the previously active subwindow is saved in case of a subsequent SWITCH_BACK_TO_PREVIOUS_SUB_WINDOW command.

6.18.3 SWITCH_BACK_TO_PREVIOUS_SUB_WINDOW Command

This command returns to the subwindow that was active before the last SWITCH_TO_SUB_WINDOW command.

Offset	Description	Data type
0x0000	Constant value 0x18	byte
0x0001	Constant value 0x02	byte
0x0002	subwindowId (Not used)	byte

This command makes the subwindow that was active before the last SWITCH_TO_SUB_WINDOW command the active subwindow. However, like it saves the Id of the currently active subwindow, so the next SWITCH_BACK_TO_PREVIOUS_SUB_WINDOW command will undo this command. Therefore, consecutive SWITCH_BACK_TO_PREVIOUS_SUB_WINDOW messages will only switch back and forth between the last two active subwindows.

6.18.4 DESTROY_SUB_WINDOW Command

This command removes a subwindow from the rendering pipeline and releases all system resources associated with it.

Offset	Description	Data type
0x0000	Constant value 0x18	byte
0x0001	Constant value 0x03	byte
0x0002	subwindowId	byte

The DESTROY_SUB_WINDOW command removes the subwindow with the given Id from the rendering process.

While it is possible to destroy the currently active subwindow, the results of doing so may be unpredictable. The active subwindow always has a second reference pointing to it, which prevents it from being properly removed. Depending on the client (Java or HTML5), this can result in undefined behaviour.

6.19 USE_GLOBAL_RESOURCE Commands

These commands are used for everything that deals with additional resources, specifically images and sound effects. They come in six variants:

6.19.1 RESOURCE_TYPE_IMAGE_NO_DATA

This variant renders a previously cached image.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x00	byte
0x0004	xPosition	short
0x0006	yPosition	short

This command renders the image specified by the resourceId at the given position.

6.19.2 RESOURCE_TYPE_PNG

This variant receives and renders a PNG encoded image.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x01	byte
0x0004	xPosition	short
0x0006	yPosition	short
0x0008	dataLength	short
0x000a	PNG file data	Data

This command receives a PNG file and saves it to the image cache using the given resourceId. It then renders the image at the position specified by the xPosition and yPosition fields.

6.19.3 RESOURCE_TYPE_IMAGE_RAW

Like the previous variant, this one receives an image, but this time it is sent as raw RGB values.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x05	byte
0x0004	xPosition	short
0x0006	yPosition	short
0x0008	dataLength	short
0x000a	Raw RGBA data	Data

This command's fields work exactly the same way as in the RESOURCE_TYPE_PNG subcommand. The RGBA data is formated in the following way:

Offset	Description	Data type
0x0000	Image width	byte
0x0001	Image height	byte
0x0002	Red 1	byte
0x0003	Green 1	byte
0x0004	Blue 1	byte
0x0005	Alpha 1	byte
...	...	...
0x000m	Red n	byte
0x000m	Green n	byte
0x000m	Blue n	byte
0x000m	Alpha n	byte

6.19.4 RESOURCE_TYPE_SOUND_EFFECT_NO_DATA

The RESOURCE_TYPE_SOUND_EFFECT_NO_DATA subcommand plays a sound effect that has previously been cached.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x02	byte
0x0004	pan	byte
0x0005	volume	byte
0x0006	instanceId	short

This command plays the sound effect with the given resourceId, applying the volume and pan position. The actual volume of the sound is calculated by linearly mapping the possible values of the volume field (0-255) to the range from -80dB to +80dB. The interpretation of the pan field differs slightly between the HTML5 client and the Java client. The Java client interpretes a value of 0 as 80% left and a value of 255 as 80% right and interpolates linearly between

those values. The HTML5 client uses the JS sound capabilities to position the sound at a point on a semicircle around the listener. A value of 0 means the sound is positioned 100% to the left and a value of 255 means 100% to the right.

The instanceId is a secondary Id that is used to reference different instances of the sound during playback. It is not related to the resourceId. The instanceId allows the same sound effect to be played more than once at a time.

6.19.5 RESOURCE_TYPE_SOUND_EFFECT

The RESOURCE_TYPE_SOUND_EFFECT subcommand receives a sound effect and plays it.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x03	byte
0x0004	pan	byte
0x0005	volume	byte
0x0006	instanceId	short
0x0008	dataLength	short
0x000a	Sound effect name	Data

The pan position, volume and resourceId fields work exactly the same as in the previous subcommand.

The sound effect name is the name of the file that the sound effect is loaded from. The way sound effects are loaded depends on the client implementation. The Java client first checks wether the sound file has already been downloaded in a previous session, in which case it loads that file from the local hard drive. The downloaded sounds are saved in the 'resource'-folder in the same directory that the client is stored in. If the client is unable to find the sound on the players hard drive, it proceeds to download the file with the corresponding name from the online resource folder located at www.aberoth.com/resource/. All sound effect files used by the Java client are in WAVE (.wav) format, but are encoded by XORing every byte with 0xb5 (181 decimal).

The HTML5 client will always access the online resource folder. However, instead of WAVE files, it searches for OGG or MP3 files depending on the capabilities of the browser. When loading a file, the HTML5 client appends the appropriate file suffix (.ogg or .mp3) to the filename and loads that file from the www.aberoth.com/resource. These files are not encoded in any special way, making them playable in the browser:
http://www.aberoth.com/resource/2bcb7e7590adadf0f7fd885d82b21c1e0.mp3

6.19.6 RESOURCE_TYPE_STOP_SOUND_EFFECT

The RESOURCE_TYPE_STOP_SOUND_EFFECT subcommand stops a currently playing sound effect.

Offset	Description	Data type
0x0000	Constant value 0x1a	byte
0x0001	resourceId	short
0x0003	Constant value 0x06	byte
0x0004	pan (Not used)	byte
0x0005	volume (Not used)	byte
0x0006	instanceId	short

This command stops the sound with the given resource and instance Id.

6.20 MOUSE_INPUT Command

The MOUSE_INPUT Command represents a mouse action by the player, such as clicking or moving the cursor.

Offset	Description	Data type
0x0000	Constant value 0x13	byte
0x0001	subwindowId	byte
0x0002	mouseButton	byte
0x0003	action	byte
0x0004	xPosition	short
0x0006	yPosition	short

The subwindowId field contains the Id of the subwindow that the mouse event occured in. This will always be set to the highest Id of the subwindows that the mouse event is contained in.

The mouseButton field specifies which button on the mouse has been pressed, if any.

The action field determines the kind of mouse interaction that the player performed. A value of 0 means that a mouse button has been pressed, a value of 1 means a button has been released, a value of 2 means that the mouse cursor has been moved and a value of 3 means that the mouse has been dragged (moved with a button held down).

The xPosition and yPosition field specify the coordinate of the mouse event relative to the origin of the subwindow in which it occured. The fields are normalized, meaning they have to be divided by the scalefactor that was set in the login message, before they can be sent to the server.

6.21 KEYBOARD_INPUT Command

The KEYBOARD_INPUT command represents a key press.

Offset	Description	Data type
0x0000	Constant value 0x10	byte
0x0001	keyCode	byte
0x0002	action	byte

The keyCode field contains a unique Id representing the key for this key event. The Ids are the same as the ones used by the java.awt.event.KeyEvent class.

The action field specifies the type of interaction.

- A action value of 0 means that a key was pressed.
- A action value of 1 means that a previously held key was released.
- A action value of 2 means that a key was typed. This event is usually fired when the player is typing a text.
- A action value of 11 signals that a special action is performed which is unreleated to a key. The possible special actions are listed below.
- A action value of 12 signals that the character typed should go into a special buffer on the server which is used to store in-game speech.

There exist some special values for the keyCode field that are available when using action 11. They are put in place to make the mobile version of the HTML5 client easier to use, but could theoretically be sent from any client:

- A keyCode value of 1 means that the player character should start attacking, similar as if the 'f'-key is held down.
- A keyCode value of 2 means that the player character should stop attacking.
- A keyCode value of 3 signals that from now on all mouse clicks should be interpreted as right clicks, similar as if the 'drop' checkbox is selected.
- A keyCode value of 4 means that the mouse click interpretation should go back to normale.
- A keyCode value of 5 signals that the player is about to say something.
- A keyCode value of 6 signals that the player character should speak the current contents of the speech buffer.

6.22 DATA_INPUT Command

This command is unused by any client. It is unknown what it does.

Offset	Description	Data type
0x0000	Constant value 0x11	byte
0x0000	data	String

This command can be send as part of a USER_INPUT message. However, it is not know what effects the value of data has, if any.

6.23 LOAD_COLOR_FROM_CACHE Commands

All commands with an identifier not listed in any of the previous commands load a color from the cache.

Offset	Description	Data type
0x0000	cacheId	byte

After subtracting 30 from the cacheId field, the color at the corresponding index of the color cache is set as the active color.