Workgroup: Datoriķi
Published: 9 March 2025
Author:
O. Putāns, Ed.
Latvijas Universitātes Eksakto Zinātņu un Tehnoloģiju Fakultātes Datoriku Studējošo Pašpārvalde

Datoriķu Saziņas Protokols (DSP)

Abstract

DSP is a communication protocol used for Datoriķdienas event as a form of communication. It is an application layer protocol used for communication between clients and servers. It is designed to be used in conjunction with the Internet Protocol Suite and is based on the principles of the Internet Protocol Suite. The protocol is designed to be used in conjunction with the Internet Protocol Suite, particularly as a session, presentation, and application layer protocol on top of TCP (Transmission Control Protocol) as the transport layer protocol. This document describes the protocol in the context of the Internet Protocol Suite, but the protocol can be used in other contexts as well. Inspiration for this document is taken from the IRC protocol in both concept and document structure.

Status of This Memo

This isn't an Internet Standards Track document and isn't actually associated with the Internet Engineering Task Force in any way. The protocol described in this document is to be only used in private settings where all parties are aware of the protocol and its reserverations.

This document is an active draft for describing the protocol and can change at any time. Recommendations and discussions are accepted at the time of publication.

Information about the current status of this document, any errata, and feedback is to be submitted by getting in touch with LU EZTF DSP members and associated persons.

This is our first standard-like document, pls don't be too harsh on us.

Table of Contents

1. Introduction

DSP is a communication protocol used for Datoriķdienas event as a form of communication. It is based on the principles of the Internet Protocol Suite and is designed to be used in conjunction with the Internet Protocol Suite. Delivery of all binary data between the client and the server is done using the DSP protocol. The DSP protocol is designed to be used in conjunction with the Internet Protocol Suite, particularly as a session, presentation, and application layer protocol on top of TCP (Transmission Control Protocol) [RFC9293] as the transport layer protocol. Although this document describes the protocol in the context of the Internet Protocol Suite, the protocol can be used in other contexts as well. Inspiration for this document is taken from the IRC protocol [RFC1459] in both concept and document structure.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Server

Using the DSP protocol, any connected clients can broadcast messages to all other connected clients via the server. The server is responsible for managing the connections between the clients and for broadcasting messages to all connected clients. The server is also responsible for managing the state of the clients. The server retains full control over the connections and can disconnect any client at any time. There exists exlusively a one-to-many relationship between the server and the clients.

1.3. Client

The client is responsible for sending messages to the server and receiving messages from the server. Clients are distinguished by their unique identifiers, which the clients can choose. See the section on the protocol for more information on how the clients are identified. In addition to the name, the server has knowledge of the client's host information, such as the IP

address and the port number. The server may assign a unique identifier to the client. which is public to all connected clients and can be used to identify the client in cases where the client's chosen name is not unique.

2. The DSP Specification

2.1. Overview

The protocol as described in this section is for both the server and client for communication between the two.

2.2. Character codes

The protocol uses UTF-8 character encoding as defined in RFC 2629 [RFC3629] for all content of sent messages.

2.3. Messages

Servers and clients in the duration of the connection are able to send messages to each other. These messages may or may not be broadcasted to all connected clients and may or may not be acknowledged by the server or the client. The messages may also warrant a response from the receiving party. Each message is a string of characters, encoded in UTF-8, and is always terminated by a zero byte ('\0', 0x00). The message is structured from the following parts:

1. Username
2. Message type
3. Message content
4. Message terminator

2.3.1. Username

The username is a string of alphabetic, numeric characters or underscore ('_' 0x5F) characters encoded in UTF-8. That means that the username is not allowed to contain any control characters, whitespace characters, or any other special characters. The username is always terminated by a space character (' ', 0x20).

The username **MUST** not be empty and **MUST** be no longer than 32 characters. The server **MAY** reserve specific usernames for specific purposes, such as usernames that are reserved for the server itself. In these cases, the server **MAY** modify the client's username or reject the message, returning an error message to the sender.

2.3.2. Message type

The message type is a single instruction string composed of only ASCII uppercase alphabetic characters (0x61 - 0x7A). The message type is always terminated by a space character (' ', 0x20) or the message terminator.

2.3.3. Message content

The message content is a string of characters encoded in UTF-8. The message content is always terminated by the message terminator. Note that this does mean that the message content is not allowed to contain the message terminator. In cases where the message content contains the message terminator, it should be escaped via other characters, for example, by encoding the entire message in base64 as defined in RFC 4648 [RFC4648]. While control characters are allowed in the message content, they may be subject to restrictions depending on the message type and server policies, which may include discarding such code points or the entire message. Additional restrictions may apply to the message content depending on the message type. For specific message types, see the section on the protocol. Note that the message content may be empty.

This protocol does not define a method for sending content that exceeds the length of the allowed message content. The server **MAY** handle longer messages than specified in this document, but in such cases, the other clients may not receive the full message or discard the message entirely.

2.3.4. Message terminator

The message terminator is always a zero byte ('\0', 0x00). The message terminator is used to indicate the end of the message. This is used by the sender to indicate to the receiver that the message is complete or that a new message is starting. See the discarding of invalid messages in the section on specific message types and their handling.

2.3.5. Message format in ABNF

Messages **MUST** be extracted from the incoming data stream by the connected party. An ABNF representation of the message format is included here. They are derived from the rules defined in RFC 3629 [RFC3629]. Additionally, a rule UTF8-ALPHANUM is defined as a UTF-8 character that has either a Numeric_Type or Alphabetic property as defined in the Unicode character database [CHARDB] .

```
message = username SP message-type [SP message-content] message-terminator
username = 1*32(UTF8-ALPHANUM | "_")
message-type = 1*20ALPHA
message-content = *1024UTF8-CHAR
message-terminator = %x00
```

Figure 1

2.4. Common client-server interaction flow

The following is a common interaction flow between the client and the server.

1. The client connects to the server.
2. The server sends a welcome message to the client.

3. The client sends a JOIN message to the server.
4. The server broadcasts a message to all connected clients that the client has joined.
5. The client sends a MESSAGE message to the server.
6. The server broadcasts the message to all connected clients.
7. The client sends a QUIT message to the server.
8. The server broadcasts a message to all connected clients that the client has left.
9. The server disconnects the client.

2.5. Challenge

Should a server believe that a client is not behaving as the server expects, the server **MAY** issue a challenge to the client. At this point, the server sends a CHALLENGE message to the client. The CHALLENGE message is a message type that is used to issue a proof of work challenge to the client as a penalty for the client's behavior. The server **MAY** disconnect the client if the client does not respond to the challenge within a reasonable time frame. The server **SHOULD NOT** disconnect the client if the client does not send a response to the challenge immediately, as the client may be busy processing previous messages. In this case, the server **SHOULD** respond to each message from the client with an error message until the client responds to the challenge.

The challenge is in the form of guessing an input phrase of a SHA-256 hash as defined in RFC 6234 [RFC6234]. The server sends the client a CHALLENGE message with the amount of leading zeroes that the guessed input phrase that is passed through the SHA-256 hash should have. Client has to guess, followed by the first part of the input phrase. The client has to respond with the full input phrase up to 255 bytes in length. The phrase consists of only printable ASCII characters (0x20 - 0x7E).

3. Message details

3.1. Username

When sending a message, the client **SHOULD** include the client's chosen username in the message. This username **SHOULD** not be changed during the duration of the connection. The server **MAY** discard messages that do not include the username. The username is used by clients to identify the sender of the message. The username **SHOULD NOT** be used by the server to identify the client, as clients can choose and change their usernames at any time, and have duplicates. The username **MUST NOT** contain any control characters, whitespace characters, or any other special characters. The username **MUST NOT** be empty.

The server **MAY** reserve specific usernames for specific purposes, such as usernames that are reserved for the server itself. In these cases, the server **MAY** modify the client's username or reject the message, returning an error message to the sender.

3.2. Message types

Below are the message types defined in the protocol. The message types are used to distinguish the purpose of the message and to allow the server to handle the message appropriately. The message types are always terminated by a space character (' ', 0x20) or the message terminator (in which case the message type is the last part of the message). The content of the message type is always composed of ASCII uppercase alphabetic characters (0x41 - 0x5A), being the same as the message type is defined in the protocol.

1. JOIN
2. QUIT
3. MESSAGE
4. CHALLENGE
5. RESCINDED
6. RESPONSE
7. ERROR

3.2.1. JOIN

The JOIN message type is used to inform the server that the client wishes to listen to messages broadcasted by other clients. The message content is not used, **SHOULD** be empty, and **MUST** be ignored by the server. An ABNF of the JOIN message is provided below.

When received by a server, the server **SHOULD** broadcast the JOIN message to all connected clients, including the client that sent the message.

Clients receiving a JOIN message **SHOULD** not respond to the message. It is up to the client to decide whether this message is displayed to the user or not.

```
JOIN = username SP "JOIN" message-terminator
```

Figure 2: JOIN message ABNF

3.2.2. QUIT

The QUIT message type is used to inform the server that the client wishes to stop listening to messages broadcasted by other clients. Note that the server may still send messages to the client after the QUIT message has been sent. The message content is not used, **SHOULD** be empty, and **MUST** be ignored by the server. An ABNF of a QUIT message is provided below.

Clients **SHOULD** ignore incoming QUIT messages.

When a client receives a QUIT message, the server indicates that no further messages will be sent to the client and the connection will be terminated.

```
QUIT = username SP "QUIT" message-terminator
```

Figure 3: QUIT message ABNF

3.2.3. MESSAGE

The MESSAGE message type is used to broadcast a message to all connected clients. The message content is the message that is to be sent to all connected clients. The message content **SHOULD NOT** be empty. It is up to the server implementation to decide how to handle empty messages. The server **MUST** broadcast the message content to all connected clients, including the client that sent the message.

The server **SHOULD** include the username of the client that sent the message in the broadcasted message and **MAY** modify the client's username before broadcasting the message. The server **MUST** not include the sender's hostname, IP address, or any other host information in the broadcasted message, unless the sender has explicitly included such information in the message content. The server **MAY** alter the message content before broadcasting it, but **SHOULD NOT** alter the message content in a way that changes the meaning of the message. The message structure is provided below.

Clients **SHOULD NOT** send messages with MESSAGE message type for which the message content exceeds 512 UTF-8 characters. The rest of the 512 UTF-8 characters are reserved for the server to let the server add additional information to the message before broadcasting it.

Any received messages of this type **MAY** exceed the 512 UTF-8 characters limit, and the client **MUST** handle such messages accordingly.

```
MESSAGE = username SP "MESSAGE" SP message-content message-terminator
```

Figure 4: MESSAGE message ABNF

3.2.4. CHALLENGE

The CHALLENGE message type is used by the server to issue a challenge to a client. The message content describes the parameters of the challenge. The message content consists of 2 parts: the number of leading zeroes of the target SHA-256 hash hexadecimal representation and the first part of the input phrase. Parameters are separated by a space character (' ', 0x20). This type of message **MAY** be issued by the server at any time as a means of penalizing the client for its behavior. The server **MAY** disconnect the client if the client does not respond to the challenge within a reasonable time frame. The message structure is provided below as ABNF.

Should the server decide that the client should be temporarily banned completely, the server **MAY** issue a CHALLENGE message with the number of leading zeroes set to 65 or more and the input phrase set to "timeout". This challenge is not possible to solve as the SHA-256 hash is always 64 characters long. The challenge **MAY** still be rescinded after a while by the Server.

```
CHALLENGE = "server" SP "CHALLENGE" SP number-of-zeroes SP input-phrase
message-terminator
number-of-zeroes = non-zero-digit *1DIGIT ; 1-99 leading zeroes (in regular use
it's only up to 64 as SHA-256 is 64 hex characters long)
input-phrase = *64(ALPHA / DIGIT) ; up to 64 alphanumeric chars
```

Figure 5: CHALLENGE message ABNF

3.2.5. RESPONSE

The RESPONSE message type is used to respond to a challenge issued by the server. The message content is the full phrase that the client has guessed. The message content **MUST NOT** be empty. The server **MUST** verify the response and respond accordingly. The start of the response phrase **MUST** match the input phrase that was sent in the CHALLENGE message. The server **MUST** not include the sender's hostname, IP address, or any other host information in the response message.

```
RESPONSE = username SP "RESPONSE" SP response-phrase message-terminator
response-phrase = *512(ALPHA / DIGIT) ; up to 512 alphanumeric chars
```

Figure 6: RESPONSE message ABNF

3.2.6. RESCINDED

The RESCINDED message type is used to inform the client that the server has rescinded the challenge or that the challenge has been fulfilled. The message content is not used, **SHOULD** be empty, and **MUST** be ignored by the client.

Clients **MUST NOT** send RESCINDED messages to the server.

```
RESCINDED = username SP "RESCINDED" message-terminator
```

Figure 7: RESCINDED message ABNF

3.2.7. ERROR

The ERROR message type is used to inform the client that an error has occurred. The message content is the error message that is to be sent to the client. The message content **SHOULD NOT** be empty. The server **MUST** send the error message to the client that sent the message that caused the error. The server **SHOULD** include the username of the client that sent the message that caused the error in the error message. The server **MAY** modify the client's username before sending the error message. The server **MUST** not include the sender's hostname, IP address, or any other host information in the error message, unless the error message is related to the host information, however, great care should be taken to avoid leaking sensitive information.

The error message **SHOULD** be in a human-readable format and **MAY** include additional information about the error, such as the error code or a description of the error. The error message **SHOULD** be in English, but **MAY** be in any language if so desired by the server implementation. The server **MUST** not include any control characters in the error message, as they may be subject to restrictions depending on the client implementation. The server **MUST** not include the message terminator in the error message, as it may cause the client to interpret the error message as multiple messages.

```
ERROR = username SP "ERROR" SP error-message message-terminator
error-message = *512UTF8-CHAR
```

Figure 8: ERROR message ABNF

3.2.8. Invalid/other messages

All messages that do not conform to the above defined message types **SHOULD** be discarded by the server and responded to with an ERROR message, unless the server has extended the protocol with additional message types and handles such messages accordingly. This protocol does not prohibit any implementation-specific message types.

An invalid message **MUST NOT** be broadcasted to other clients or cause the connection to be terminated.

3.3. Connection termination

Shortly prior to termination of the connection, the semantics of which are implementation specific, the server **MAY** send a string of characters indicating the termination of the connection which may or may not be a reference to a certain fictional entertainment franchise in which potentially possessed metal constructs residing in colourful shells resembling a fox, a bear, a chicken, a rabbit in a children's pizzeria, and a very purple character are found in.

4. IANA Considerations

No IANA actions required.

5. Security Considerations

As this document describes a communications protocol, any readers should be aware of the security implications of using the protocol. The protocol is lightweight and does not include any security features. It is not recommended to use the protocol in any environments where confidentiality, data integrity, or peer entity authentication.

Should the protocol be implemented using UNIX sockets, or any other form of inter-process communication, special care should be taken to ensure that the protocol isn't vulnerable to attacks such as buffer overflows.

The protocol is lightweight and can pose risks of Denial of Service attacks to the server or the clients. It should be noted that servers should be configured with appropriate security measures to at least mitigate and limit consequences of such attacks, such as rate limiting, connection limiting, and monitoring. Clients should be aware that the server has full control over the connections and can disconnect any client at any time, potentially impacting data integrity and availability.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC1459] Oikarinen, J. and D. Reed, "Internet Relay Chat Protocol", RFC 1459, DOI 10.17487/RFC1459, May 1993, <https://www.rfc-editor.org/info/rfc1459>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[CHARDB] Whistler, K.W., "Unicode Character Database", Technical report Unicode Character Database, 2024, <http://www.unicode.org/reports/tr44/>.

Contributors

Thanks to all of the contributors.

O. Putāns
LU EZTF DSP

R. Kauliņš
LU EZTF DSP

Author's Address

Oskars Putāns (editor)
Latvijas Universitātes Eksakto Zinātņu un
Tehnoloģiju Fakultātes Datoriku Studējošo
Pašpārvalde
Raiņa Bulvāris 19
Rīga, LV-1050
Latvia