Workgroup:	Internet Engineering Task Force
Internet-Draft:	draft-royer-phoenix-00
Published:	19 January 2025
Intended Status:	Informational
Expires:	23 July 2025
Author:	DM. Royer, Ed.
	RiverExplorer Games LLC

Phoenix: Lemonade Risen Again

Abstract

NOTE: This is just getting started, not ready for submission yet.

Email and MIME messages account for one the largest volumes of data on the internet. The transfer of these MIME message has not had a major updated in decades. Part of the reason is that it is very important data and altering it takes a great deal of care and planning.

This application transport can also transfer non-MIME data. It can be used as an XDR transport, or for opaue data (blobs of known or unknown data) transport.

Another major concern is security and authentication. This proposal allows for existing autnentication to continue to work.

This is a MIME message transport that can facilitate the transfer of any kind of MIME message. Including email, calendaring, and text, image, or multimedia MIME messages. It can transfer multipart and simple MIME messages.

The POP and IMAP protocols are overly chatty and now that the Internet can handle 8-bit transfers, there is no need for the overly complex text handling of messages.

This proposal includes a sample implementation. (https://github.com/RiverExplorer/Phoenix) Which also includes a gateway from this proposal to existing system. Thunderbid and Outlook plugins are part of the sample implementation.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 July 2025.

Copyright Notice

Table of Contents

1. Introduction

On the Internet, just about everything is a MIME object and there are many ways to transport MIME. This document specifies a new application level MIME transport mechanism and protocol. This document does not specify any new or changed MIME types.

Transporting MIME objects is generally done in one of two ways: (1) Broadcasting, (2) Polling. Both methods often require some form of authentication, registration, and selecting of the desired material. These selection processes are essentially a form of remote folder management. In some cases you can only select what is provided, and in others you have some or a lot of control over the remote folders.

In addition to other functions, this specification defines a remote and local folder management. This remote folder management is common with many type of very popular protocols. This design started by looking at the very popular IMAP and POP protocols.

An additional task is transporting the perhaps very large MIME objects. Some MIME objects are so large that some devices may default to looking at only at parts of the MIME object. An example is an email message with one or more very large attachments, where the device may default to not download the large attachment without a specific request from the user.

Some objects are transported as blocks of data with a known and fixed size. These are often transported with some kind of search, get, and put commands. In effect these are folder and file commands

Other MIME objects are transported in streams of data with an unspecified size, such as streaming music, audio, or video. This specification describes how to use existing protocols to facilitate the data streaming. And again, these are folder and file commands.

A MIME object can be a simple object, or it may contain many multipart sections of small to huge size. These sections can be viewed as files in the containing MIME object.

By implementing this specification application developers can use the techniques to manage local and remote files and folders. Remote email or files are the same thing in this specification. The sections of MIME object with multipart sections are viewed as files in the MIME object. You can interact with the entire folder, or just the files within it.

MIME objects have meta data, and they are called headers. Files and folders have meta data, and they are called file attributes. This specification does not mandate any meta data, it allows for a consistent transport of existing meta data.

File and folder meta data is a complex task that can involve access control lists and permissions. This specification defines a mechanism to transport this meta data, it does not define the meta data.

And this specification provides for the ability to define both protocol extensions and the creating of finer control for specific commands that may evolve over time.

This examples compares current folder and file manipulations to how it can be used in this protocol with email.

- You can search for file names. You can search email for: sender, subject, and more.
- You can search for file contents. You can search for email message contents.
- You can create, delete, and modify files. You can create, delete, and modify email messages.
- You can create, delete, and modify folders. You can create, delete, and modify email folders.

What this specification defines:

- How to use existing authentication implementations or use new ones.
- This specification describes a standard way to perform file operations that are remote to the application and agnostic to purpose of data being transported.
- Specifies a way to migrate from some existing protocols to Phoenix. Provides links to sample implementations.

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.

2. Terms and Definition used in this proposal

The following is a list of terms with their definitions as used in this specification.

AdminCmd
: A general term for any administrative command. Administrative and auditing operations. This list includes commands for authorized users to configure, query logs, errors, possibly user activity.

AuthCmd
: A general term for any authentication command. Authentication and authorization operations. These operations authenticate users and verity their authorization access.

CMD
: A specific protocol operation, or command. They are broken down into, AdminCmd, AuthCmd, FileCmd, and ProtoCmd.

Command, CMD
: Each packet contains a command. This command is also in the reply to the command. Not all commands have a reply. These are called a CMD or command.

FileCmd
: A general term for any file command. File and folder operations. This include creating, getting, modifying, deleting, moving, and renaming files.

Media Type
: Each MIME object has a media type that identifies the content of the object. This specification does not add, remove, or alter any MIME media type;

MIME
: This protocol transports MIME objects. This specification does not remove or alter any MIME objects;

Packet
: A packet is a blob of data that has a header (its length) followed by a Phoenix command with all of its values and parameters. Packets flow in both directions and asynchronously. Commands can be sent while still waiting for other replies. Each endpoint may send commands to the other endpoint without having to be prompted to send information.

Parameter
: Most commands have values that are associated with them. These values are called parameters. For example, the create folder command has the name of the new folder to be created as a parameter.

ProtoCmd
: A general term for all protocol commands. This also includes commands that do not fall into one of the other catagories described here in this definistions section.

SEQ, Command Sequence, CMD_SEQ
: Each command has a unique identifier, a sequence number. All replies to a command include the same sequence number as the original command. In this way replies can be matched up with their original command.

SSL
: For the purpose of this specification, SSL is interchangable with TLS. This document uses the term TLS. The sample implementation uses both SSL and TLS because the legacy UNIX, Linux, Windows, and OpenSSL code uses the term SSL as well as TLS.

TLS
: A way of securely transporting data over the Internet. See RFC-xxxx.

XDR
: RFC-4506 specifies a standard and compatible way to transfer binary information. This protocol uses XDR to transmit a command, its values and any parameters and replies. The MIME data, the payload, is transported as XDR opaque, and is unmodified.

3. Commands Overview

3.1. Administration Commands (AdminCmd)

3.1.1. Administration Commands Overview

Administrative command can be used to configure, audit, and manage the remote endpoint. Administrative command can be used to configure, audit, and manage user acess.

3.1.1.1. Capability Name

Implementations that support any ADMIN_CMD include ADMIN_CMD in the post authenication CAPABILITY list.

Implementations MUST NOT send a ADMIN_CMD capability in the pre authorization CAPABILITY list.

When a user has been authenticated, as part of the affermative reply, the ADMIN_CAPABILITY will be included in the reply when the user has administrative permissions.

The ADMIN_CAPABILITY reply will then be followed by the list of ADMIN commands the user is allowed to perform. For example, if a user only has permission to only view user lists, then only the USER_LIST ADMIN capibility will be provided.

3.1.1.2. Administration of users.

The following operations are defined for user administration.

Command and Capability Name	Brief Description.
USER_CREATE	Create a new user.
USER_DELETE	Delete a user.
USER_RENAME	Rename a user.

Command and Capability Name	Brief Description.
USER_LIST	List users and their capabilities.
USER_PERMISSIONS	Update user permissions.

Table 1

3.2. Authentication Commands

3.2.1. Authentication Commands Overview

TODO

3.3. Protocol

3.3.1. Basic File and Folder Operations

The file operations (FileOp) have protocol names. Here are their protocol names and a breif description.

Op Name	Brief Description.
FOLDER_CREATE	Create a new folder.
FOLDER_COPY	Copy a folder.
FOLDER_DELETE	Delete a folder.
FOLDER_RENAME	Rename a folder.
FOLDER_MOVE	Move a folder.
FOLDER_SHARE	Share a folder.
FOLDER_LIST	List folders and files.
FILE_GET	Get a known existing file.
FILE_CREATE	Create a new file.
FILE_MODIFY	Modify the contents of an existing file.
FILE_SHARE	Share a file.

Table 2

3.4. Protocol Commands

3.4.1. Protocol Overview

This protocol connects two endpoints over a network and facilitates the secure and authorized transfer of MIME objects.

The endpoint that initiates the connection is called the client. The endpoint that is connected to, is called the server. The client is the protocol authority, and the server responds to client commands as configured or instructed by the client.

After the connection is successful and authenticated, ether endpoint may send commands to the other endpoint. When the server initiates an unsolicited command, it could be a any kind of notification or message for the client side application or the user. It could be reporting errors or updates to previous client initiated commands.

All commands initiated from the client have even numbered command sequence numbers. All commands initiated from the server have odd numbered command sequence numbers.

Some commands expect a command reply. Other commands do not expect a command reply. An example of a command that expects a reply is the ping command. An example of a command that does not expect a reply is the keep-alive command. Conceptually their are two kinds of commands: (a) Directive commands, and (b) Request commands. These are not specific protocol entities, these concepts will be used to describe the expected behavior when one of these are transmitted.

A directive type command expects the other endpoint to process the command and possibly reply with some results. An example could be: Send me an index of my emails in my InBox. The client would expect a result. Another example is a bye command, once sent, no reply is expected.

A request type command may or might not have any reply. For example, a keep-alive command is a request to not timeout and has no reply. And a send new email notifications command would expect zero or more replies and it would not require them, as they might not happen.

3.4.1.1. Packet Overview

All commands are sent in a packet. A packet has two parts, the packet header and the packet body.

The packet header has one value, the total length of the packet header, packet body, and payload sent as an unsigned 64-bit integer in network byte order. The length does not include its own length. It is the total length that follows the length value.

The packet body is divided into three parts: (1) Command sequence (SEQ), (2) The Command (CMD), and (3) The command specific data (Payload).

(1) The Command SEQ is a 32-bit unsigned integer sent in network byte order. This SEQ is an even number when initiated from the client, and an odd number when initiated from the server.

The first SEQ value sent from the client is zero (0) and is incremented by two each time.

The first SEQ value sent from the server is one (1) and is incremented by two each time.

In the event an endpoint command SEQ reaches its maximum value, then its numbering starts over at zero (0) for the client and one (1) for the server. An implementation must keep track of outstanding commands and not accidentally re-issue the same SEQ that may still get replies from the other endpoint.

(2) The command is a predefined enumerated 32-bit unsigned integer sent in network byte order. The value (in hex) 0xFFFFFFFF is reserved for extensions if the 32-bit range is exhausted.

(3) The payload has no predefined length, other what what is specified for the CMD in the packet. It could be zero to vary large in size. It could be opaque data, or it could be data that is XDR encoded. The contents are specific to the CMD in the packet.

3.4.1.1.1. Packet Reply Overview

All replies to a command are also a command packet. They contain the same command SEQ and command as the original packet. The endpoint recognizes it is a reply because (a) the command SEQ matches one that is waiting a reply. And (b) When the client gets an even numbered SEQ, it can only be a reply. And when the server gets an odd numbered SEQ, it can only be a reply.

Some commands have zero to many replies. Each of these multiple replies contains the same SEQ as the original command. An example, the client sends a request to be notified when new email arrives and uses command SEQ 20. Each time a new email arrives, a reply will be sent from the server with a command SEQ of 20. And over time, the client may get many with a SEQ of 20 as new emails arrive on the server.

4. Over the Wire Protocol Detail

This section specifies the details of what is transmitted over the network.

All protocol data transmitted between the endpoints is sent in network byte order.

All payload data transmitted between the endpoints is sent in original format. The payload consent is seen as an opaque blob of data within a command packet.

When a command packet is received by ether endpoint it: (1) Checks the command sequence number to determine if it is a reply or not. (2) If it is a reply, it looks at the command and dispatches it to the implementations commands reply code. (3) If is not a reply, it looks at the command and dispatches it to the implementations command code.

A command and all of its replies, use the same format as described here.

A packet has a 64-bit unsigned integer in network byte order that is set to the octet count of all of the data that follows this length value. The shortest packet is 16 octets in size, with a length value set to 8. With 8 octets for the length, and 8 octets for the packet.

Followed by a 32-bit unsigned integer in network byte order that is the command sequence number.

Followed by a 32-bit unsigned integer in network byte order that is the command.

Followed by zero or more octets of payload data.

There is no space, padding, or line endings between the parts of the packet. The payload is sent without any modification and is not encoded or transformed in any way. A packet is shown here vertically only to aid in readability.

```
+-----------------------------------------------------------------+
| ...64-bit.unsigned.integer.length ..............................|
+-----------------------------------------------------------------+
| ...32-bit.unsigned.command.SEQ...|
+----------------------------------+
| ...32-bit.unsigned.command.CMD...|
+----------------------------------+
| payload.....
```

The payload size and format varies for each command. The details of the payload content, and the format of that content, is described in each specific CMD section.

An implementation can send, receive, and dispatch packets within its implementation by looking at the length, SEQ, and CMD, then passing the payload to code that can handle that payload.

- Read in a 64-bit value. - Convert the value from network byte order, to host byte order. This is the total length of the data that follows. - Read in length octets into the packet payload. - Get the 32-bit value in the payload, it is the SEQ in network byte order. - Convert the SEQ from network byte order, to host byte order. - Get another 32-bit value in the payload, it is the CMD in network byte order. - Convert the CMD from network byte order, to host byte order. - Dispatch the CMD with SEQ and all of the data that follows to implementation

The following is pseudo code that explains how processing incoming XDR data can be handled:

```
// Where:
//  uint64_t, is a 64-bit unsigned integer.
//  uint32_t, is a 32-bit unsigned integer.
//  uint8_t *, is a pointer to 8-bit data.
//  XDR, is an XDR object.
//
//  CmdPacket, is an object that represents all commands
//  and replies.
//
//  NOTE: See the sample implementation.
//
uint64_t   NetLength;
uint64_t   PacketLength;
uint8_t *  Data;
```

```
uint8_t *  DataPointer;
XDR        Xdr;
CmdPacket  Packet;

// Read the length and convert to host byte order.
//
read(FromClientSocket, &NetLength, sizeof(uint64_t));
PacketLength = ntohll(NetLength);

// Allocate PacketLength data, and read it.
//
Data = new uint8_t[PacketLength]
DataPointer = Data;

// Initialize the XDR deserializer.
//
xdrmem_create(&Xdr, Data, PacketLength, XDR_DECODE);

// Decode the received data into a Packet.
//
if (xdr_CmdPacket(&Xdr, &Packet)) {

  // If the lowest bit is set, it is an odd number.
  //
  if (Packet.Sequence & 0x01) {
    SequenceIsEvenNumber = false;
  } else {
    SequenceIsEvenNumber = true;
  }

  // The client sends even numbered sequences, and the server
  // sends the same even numbers sequence in the reply to
  // the command.
  //
  // If a client gets an odd numbered sequence, it is a command
  // from the other endpoint.
  //
  // The server sends odd numbered sequences, and the client
  // sends the same odd numbers sequence in the reply to
  // the command.
  //
  // If a server gets an even numbered sequence, it is a
  // command from theother endpoint.
  //
  if (WeAreTheClient) {
    if (SequenceIsEvenNumber) {
      DispatchReply(Packet);
    } else {
      DispatchCommandFromOtherEndpoint(Packet);
    }
  } else {
    if (SequenceIsEvenNumber) {
      DispatchCommandFromOtherEndpoint(Packet);
    } else {
      DispatchReply(Packet);
    }
  }
```

```
}
```

5. IANA Considerations

This memo includes no request to IANA. [CHECK]

6. Security Considerations

This document should not affect the security of the Internet. [CHECK]

7. References

7.1. 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>.

7.2. Informative References

[exampleRefMin] Surname [REPLACE], Initials [REPLACE]., "Title [REPLACE]", 2006.

[exampleRefOrg] Organization [REPLACE], "Title [REPLACE]", 1984, <http://www.example.com/>.

Appendix A. Administrative Enumerated Binary Values

Phoenix is a binary protocol. Each value is sent as an unsigned 32-bit integer in xdr format.

The values for the commands are arbitrary and were assigned as created. There is no plan or origination to the numbers. There is no priority or superiority to any value. The table is sorted by name, not value.

The values are not unique. They are only unique within the context in which they are used.

Some of these values are reused for other commands. For example USER_CREATE is both an (a) AUTH capability reply informing the user that they have permission to create a user with the (b) USER_CREATE command.

Some values may be reused if they are parameter arguments to other commands. For example xxxxxx.

Decimal Value	Command / Capability Name	Brief Description.
x	USER_CERT	Manage a users certificate.
x	USER_CREATE	When sent in a capability reply USER_CREATE informs the user that they have permission to create users. When sent as a command the USER_CREATE instructs the other endpoint to create a named user.
x	USER_DELETE	Delete a user.
x	USER_LIST	List users and their capabilities.
x	USER_PERMISSIONS	Update user permissions.
x	USER_RENAME	Rename a user.
x	USER_RESET	Used to coordinate resetting a users authentication information.
4294967296	Reserved for future expansion.	4294967296 has a hex value of: 0xffffffff

Table 3

Appendix B. Authentication Enumerated Binary Values

Phoenix is a binary protocol. Each value is sent as an unsigned 32-bit integer in xdr format.

The values for the commands are arbitrary and were assigned as created. There is no plan or origination to the numbers. There is no priority or superiority to any value. The table is sorted by name, not value.

The values are not unique. They are only unique within the context in which they are used.

Some of these values are reused for other commands. For example USER_CREATE is both an (a) AUTH capability reply informing the user that they have permission to create a user with the (b) USER_CREATE command.

Some values may be reused if they are parameter arguments to other commands. For example xxxxxx.

Decimal Value	Command / Capability Name	Brief Description.
x	AUTH_TODO	xxx.

Decimal Value	Command / Capability Name	Brief Description.
xxx	AUTH_xxx	xxx.
4294967296	Reserved for future expansion.	4294967296 has a hex value of: 0xffffffff

Table 4

Appendix C. File and Folder Enumerated Binary Values

Phoenix is a binary protocol. Each value is sent as an unsigned 32-bit integer in xdr format.

The values for the commands are arbitrary and were assigned as created. There is no plan or origination to the numbers. There is no priority or superiority to any value. The table is sorted by name, not value.

The values are not unique. They are only unique within the context in which they are used.

Some of these values are reused for other commands. For example USER_CREATE is both an (a) AUTH capability reply informing the user that they have permission to create a user with the (b) USER_CREATE command.

Some values may be reused if they are parameter arguments to other commands. For example xxxxxx.

Decimal Value	Command / Capability Name	Brief Description.
x	FILE_TODO	xxx.
xxx	FILE_xxx	xxx.
4294967296	Reserved for future expansion.	4294967296 has a hex value of: 0xffffffff

Table 5

Appendix D. Protocol Enumerated Binary Values

Phoenix is a binary protocol. Each value is sent as an unsigned 32-bit integer in xdr format.

The values for the commands are arbitrary and were assigned as created. There is no plan or origination to the numbers. There is no priority or superiority to any value. The table is sorted by name, not value.

The values are not unique. They are only unique within the context in which they are used.

Some of these values are reused for other commands. For example USER_CREATE is both an (a) AUTH capability reply informing the user that they have permission to create a user with the (b) USER_CREATE command.

Some values may be reused if they are parameter arguments to other commands. For example xxxxxx.

Decimal Value	Command / Capability Name	Brief Description.
x	PROTO_TODO	xxx.
xxx	PROTO_xxx	xxx.
4294967296	Reserved for future expansion.	4294967296 has a hex value of: 0xffffffff

Table 6

Acknowledgements

This template uses extracts from templates written by Pekka Savola, Elwyn Davies and Henrik Levkowetz. [REPLACE]

Contributors

Thanks to all of the contributors. [REPLACE]

Author's Address

Doug Royer (EDITOR)
RiverExplorer Games LLC
848 N. Rainbow Blvd #1120
Las Vegas, Nevada 89107
United States of America
Phone: 1+714-989-6135
Email: DouglasRoyer@gmail.com
URI: https://RiverExplorer.games