Network Working Group
Request for Comments: 4992
Updates: 3981
Category: Standards Track

A. Newton
VeriSign, Inc.

August 2007

XML Pipelining with Chunks for the Internet Registry Information Service

Status of This Memo

Copyright Notice

Abstract

This document describes a simple TCP transfer protocol for the Internet Registry Information Service (IRIS). Data is transferred between clients and servers using chunks to achieve pipelining.

Table of Contents

1. Introduction

Using S-NAPTR [5], IRIS has the ability to define the use of multiple application transports (or transfer protocols) for different types of registry services, all at the discretion of the server operator. The TCP transfer protocol defined in this document is completely modular and may be used by any registry types.

This transfer protocol defines simple framing for sending XML in chunks so that XML fragments may be acted upon (or pipelined) before the reception of the entire XML instance. This document calls this XML pipelining with chunks (XPC) and its use with IRIS as IRIS-XPC.

XPC is for use with simple request and response interactions between clients and servers. Clients send a series of requests to a server in data blocks. The server will respond to each data block individually with a corresponding data block, but through the same connection. Request and response data blocks are sent using the TCP SEND function and received using the TCP RECEIVE function.

The lifecycle of an XPC session has the following phases:

1. A client establishes a TCP connection with a server.

2. The server sends a connection response block (CRB).

3. The client sends a request block (RQB). In this request, the client can set a "keep open" flag requesting that the server keep the XPC session open following the response to this request.

4. The server responds with a response block (RSB). In this response, the server can indicate to the client whether or not the XPC session will be closed.

5. If the XPC session is not to be terminated, then the lifecycle repeats from step 3.

6. The TCP connection is closed.

2. Document Terminology

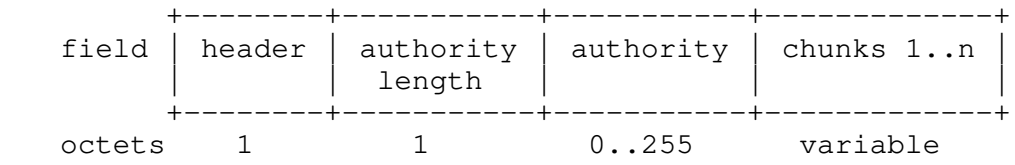
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [8].

Octet fields with numeric values are given according to the conventions in RFC 1166 [12]: the leftmost bit of the whole field is the most significant bit; when a multi-octet quantity is transmitted

the most significant octet is transmitted first. Bits signifying flags in an octet are numbered according to the conventions of RFC 1166 [12]: bit 0 is the most significant bit and bit 7 is the least significant bit. When a diagram describes a group of octets, the order of transmission for the octets starts from the left.

3. Request Block (RQB)

The format for the request block (RQB) is as follows:



Request Block

These fields have the following meanings:

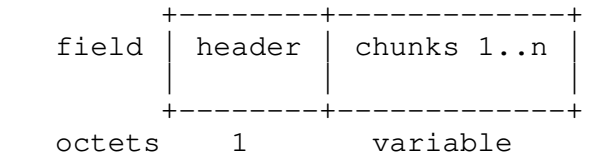
- header - as described in Section 5.
- authority length - the length of the authority field in this request block.
- authority - a string of octets describing the authority against which this request is to be executed. See [1] for the definition and description of an authority. The number of octets in this string MUST be no more and no less than the number specified by the authority length.
- chunks 1..n - the request data broken into chunks (Section 6).

4. Response Blocks

There are two types of blocks used by a server to respond to a client. The first type is a response block (RSB) defined in Section 4.1. It is used by a server to respond to request blocks (RQBs). The second type is a specialized version of a response block called a connection response block (CRB) defined in Section 4.2. It is sent by a server to a client when a connection is established to initiate protocol negotiation. Conceptually, a CRB is a type of RQB; they share the same format, but a CRB is constrained in conveying only specific information and is only sent at the beginning of the session lifecycle.

4.1. Response Block (RSB)

The format for the response block (RSB) is as follows:



Response Block

These fields have the following meanings:

- header - as described in Section 5.
- chunks 1..n - the response data broken into chunks (Section 6).

Servers SHOULD NOT send an RSB to a client until they have received the entire RQB. Servers that do begin sending an RSB before the reception of the entire RQB must consider that clients will not be expected to start processing the RSB until they have fully sent the RQB, and that the RSB may fill the client's TCP buffers.

4.2. Connection Response Block (CRB)

A connection response block (CRB) is a response block sent by a server to a client in response to the client initiating a session. A connection response block has the same format as a response block (RSB) (Section 4.1). The only difference is that it is constrained in one of two ways:

1. It contains only one chunk (see Section 6) containing version information (see Section 6.2) and the keep-open (KO) flag in the block header (see Section 5) has a value of 1 (meaning the connection is not closing). Servers MUST use this type of CRB to indicate service availability.

2. It contains only one chunk (see Section 6) containing a system error (see 'system-error' under Section 6.4) and the keep-open (KO) flag in the block header (see Section 5) has a value of 0 (meaning the server will close the connection immediately after sending the CRB). Servers MUST use this type of CRB when they can accept connections but cannot process requests.

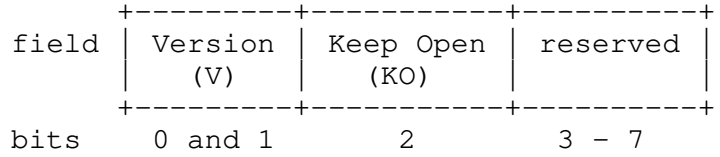
5. Block Header

Each data block starts with a one-octet header called the block header. This header has the same format for both request and response data blocks, though some of the bits in the header only have meaning in one type of data block. The bits are ordered according to the convention given in RFC 1166 [12], where bit 0 is the most significant bit and bit 7 is the least significant bit. Each bit in the block header has the following meaning:

- bits 0 and 1 - version (V field) - If 0 (both bits are zero), the protocol is the version defined in this document. Otherwise, the rest of the bits in the header and the block may be interpreted as another version. If a server receives a request for a version it does not support, it SHOULD follow the behavior described in Section 8.

- bit 2 - keep open (KO flag) - This flag is used to request that a connection stay open by a client and to indicate that a connection will stay open by a server, depending on the type of block. In a request block (RQB): a value of 1 indicates that a client is requesting that the server not close the TCP session, and a value of 0 indicates the client will expect their server to close the TCP session immediately after sending the corresponding response. In a response block (RSB) or a connection response block (CRB): a value of 1 indicates that the server expects the client to keep the TCP session open for the server to receive another request, and a value of 0 indicates that the server expects the client to close the TCP session immediately following this block.

- bits 3, 4, 5, 6, and 7 - reserved - These MUST be 0. If a server receives a request in which any of these bits is set to 1 and the server does not understand the purpose for the value, the server SHOULD follow the behavior described in Section 8.



Block Header

6. Chunks

Request and response blocks break down the request and response XML data into chunks. Request and response blocks MUST always have a minimum of 1 chunk. Each chunk has a one-octet descriptor. The first bit of the descriptor determines if the chunk is the last chunk in the block.

The bits of the chunk descriptor octet are ordered according to the convention given in RFC 1166 [12], where bit 0 is the most significant bit and bit 7 is the least significant bit. The bits of the chunk descriptor octet have the following meaning:

- bit 0 - last chunk (LC flag) - If 1, this chunk is the last chunk in the block.

- bit 1 - data complete (DC flag) - If 1, the data in this chunk represents the end of the data for the chunk type given. If this bit is never set to 1 in any chunk descriptor for chunks of the same type in a block, clients and servers MUST NOT assume the data will continue in another block. If the block transitions from one type of chunk to another without signaling completion of the data, clients and servers MUST assume that the remaining data will not be sent in a remaining chunk.

- bits 2, 3, and 4 - reserved - These MUST be 0.

- bits 5, 6, and 7 - chunk type (CT field) - determines the type of data carried in the chunk. These are the binary values for the chunk types:

  * 000 - no data or 'nd' type (see Section 6.1)

  * 001 - version information or 'vi' type (see Section 6.2)

  * 010 - size information or 'si' type (see Section 6.3)

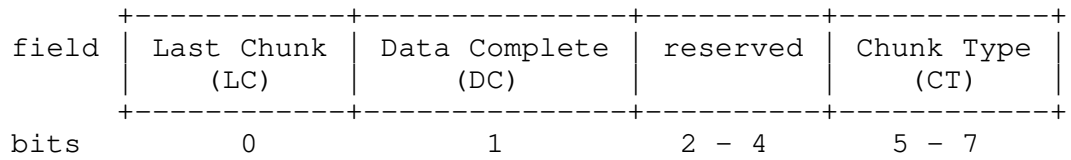
  * 011 - other information or 'oi' type (see Section 6.4)

  * 100 - SASL (Simple Authentication and Security Layer) data or 'sd' type (see Section 6.5)

  * 101 - authentication success information or 'as' type (see Section 6.6)

  * 110 - authentication failure information or 'af' type (see Section 6.7)

* 111 - application data or 'ad' type (see Section 6.8)



Chunk Descriptor

A block MAY have multiple types of chunks, but all chunks of the same type MUST be contiguous in a block and MUST be ordered in the block in the order in which their data is to be interpreted. Contiguous chunks must be ordered by type within a block in the following way:

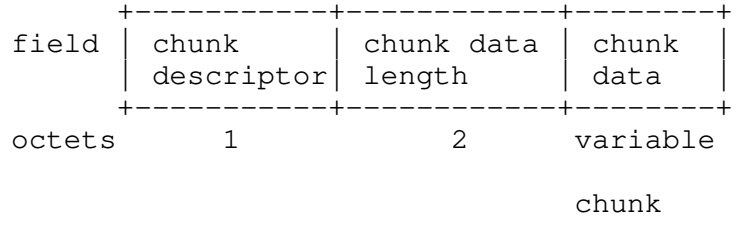
1. authentication-related chunks - either SASL data chunks (type 100), authentication success information chunks (type 101), or authentication failure information chunks (type 110), but not more than one type. During the setup of security mechanisms using these chunks, clients MUST NOT send subsequent requests until they have received either an authentication success or failure chunk.

2. data chunks - either no data chunks (type 000) or application data chunks (type 111), but not both.

3. information chunks - either version information (type 001) or other information (type 011), but not both.

A block MUST have at least one type of the above chunks.

The format for a chunk is as follows:



These fields have the following meanings:

- chunk descriptor - as described above.
- chunk data length - the length of the data of the chunk.
- chunk data - the data of the chunk.

6.1. No Data Types

Servers and clients MUST ignore data in chunk types labeled no data. There is no requirement for these types of chunks to be zero length. A client MAY send "no data" to a server, and the server MUST respond with either a chunk of the same type or other information (Section 6.4).

6.2. Version Information Types

Chunks of this type contain XML conformant to the schema specified in [9] and MUST have the <versions> element as the root element.

In the context of IRIS-XPC, the protocol identifiers for these elements are as follows:

- <transferProtocol> - the value "iris.xpc1" to indicate the protocol specified in this document.
- <application> - the XML namespace identifier for IRIS [1].
- <dataModel> - the XML namespace identifier for IRIS registries.

In the context of IRIS-XPC, the authentication mechanism identifiers are the SASL mechanism names found in the IANA SASL mechanism registry defined by RFC 4422 [10].

This document defines no extension identifiers.

Clients MAY send a block with this type of chunk to a server. These chunks SHOULD be zero length, and servers MUST ignore any data in them. When a server receives a chunk of this type, it MUST respond with a chunk of this type. This interchange allows a client to query the version information of a server.

The octet sizes for the 'requestSizeOctets' and 'responseSizeOctets' attributes of the <tranferProtocol> element are defined in Section 6.3.

6.3. Size Information Types

Chunks of this type contain XML conformant to the schema specified in RFC 4991 [9] and MUST have the <size> element as the root element.

Octet counts provided by this information are defined as the sum of the count of all chunk data of a particular chunk type. For instance, if an XML instance is broken up into chunks of 20, 30, and 40 octets, the octet count would be 90 (20 + 30 + 40).

Clients MUST NOT send chunks of this type, and servers MAY close down a session using the procedure in Section 8 if a chunk of this type is received.

6.4. Other Information Types

Chunks of this type contain XML conformant to the schema specified in RFC 4991 [9] and MUST have the <other> element as the root element.

The values for the 'type' attribute of <other> are as follows:

'block-error' - indicates there was an error decoding a block. Servers SHOULD send a block error in the following cases:

1. When a request block is received containing a chunk of this type.

2. When a request block is received containing authentication success (see Section 6.6) or authentication failure (see Section 6.7) information.

3. When a request block is received containing size information (see Section 6.3).

4. When reserved bits in the request block are 1.

5. When a block has not been received in its entirety and the TCP session has been idle for a specific period of time (i.e., a data block has been received but no terminating chunk for the data block has been received). Two minutes is RECOMMENDED for this timeout value. Note, there is a difference between an idle condition due to the incomplete reception of a data block and an idle condition between request/response transactions associated with keeping the session open. For the latter, see Section 7.

'data-error' - indicates there was an error parsing data in chunks containing application or SASL data (e.g., XML is not valid in application data).

'system-error' - indicates that the receiver cannot process the request due to a condition not related to this protocol. Servers SHOULD send a system-error when they are capable of responding to requests but not capable of processing requests.

'authority-error' - indicates that the intended authority specified in the corresponding request is not served by the receiver. Servers SHOULD send an authority error when they receive a request directed to an authority other than those they serve.

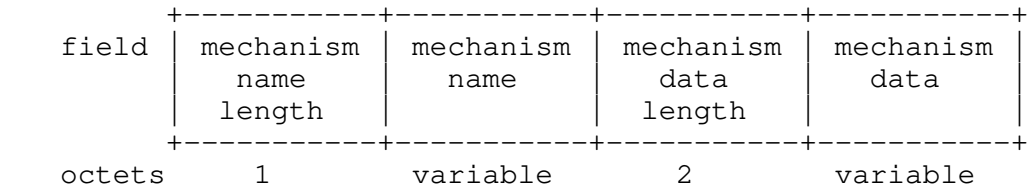
'idle-timeout' - indicates that an XPC session has been idle for too long. Usage of this value is defined in Section 7. Note, there is a difference between an idle condition due to the incomplete reception of a data block and an idle condition between request/response transactions associated with keeping the session open. For the former, see 'block-error' above.

Clients MUST NOT send chunks of this type, and servers MAY close down a session using the procedure in Section 8 if a chunk of this type is received.

6.5. SASL Types

The SASL chunk type allows clients and servers to exchange SASL data.

The format for the data of this type of chunk is as follows:



SASL Authentication

These fields have the following meaning:

- mechanism name length - the length of the SASL mechanism name.
- mechanism name - the name of the SASL mechanism as registered in the IANA SASL mechanism registry defined by [10].
- mechanism data length - the length of the SASL data.
- mechanism data - the data used for SASL.

These fields MUST NOT span multiple chunks. Therefore, it should be noted that SASL data length exceeding the length of the chunk minus the length of SASL profile name minus one is an error.

Depending on the nature of the SASL mechanism being used, SASL data is sent from clients to servers and from servers to clients and may require multiple request/response transactions to complete. However, once a SASL exchange is complete and a server can determine authentication status, the server MUST send either authentication success information (see Section 6.6) or authentication failure information (see Section 6.7).

When used as an initial challenge response for SASL mechanisms that support such a feature, the mechanism data length may be set to a decimal value of 65,535 to indicate an absent initial response. A value of 0 indicates an empty initial response.

6.6. Authentication Success Information Types

Chunks of this type contain XML conformant to the schema specified in RFC 4991 [9] and MUST have the <authenticationSuccess> element as the root element.

This type of chunk is only sent from a server to a client. If a client sends it to a server, this will result in a block error (see 'block-error' in Section 6.4). The usage of this chunk type is defined in Section 6.5. A server MAY close down a session due to reception of this type of chunk using the procedure in Section 8.

SASL mechanisms may use the <data> child element to pass back arbitrary binary data as base 64 binary. The absence of this element indicates the absence of such data, where as the presence of the element with no content indicates an empty data set.

6.7. Authentication Failure Information Types

Chunks of this type contain XML conformant to the schema specified in RFC 4991 [9] and MUST have the <authenticationFailure> element as the root element.

This type of chunk is only sent from a server to a client. If a client sends it to a server, this will result in a block error (see 'block-error' in Section 6.4). The usage of this chunk type is defined in Section 6.5. A server MAY close down a session due to reception of this type of chunk using the procedure in Section 8.

6.8. Application Data Types

These chunks contain application data. For IRIS, these are IRIS [1] XML instances.

7. Idle Sessions

If a server needs to close a connection due to it being idle, it SHOULD do the following:

1. Send an unsolicited response block containing an idle timeout error (see 'idle-timeout' in Section 6.4) with the keep-open (KO) flag in the block header (Section 5) set to a value of 0.

2. Close the TCP connection.

8. Closing Sessions Due to an Error

If a server is to close a session due to an error, it SHOULD do the following:

1. Send a response block containing either a block-error or data-error (see Section 6.4) or version information (see Section 6.2) with the keep-open (KO) flag in the block header (Section 5) set to a value of 0.

2. Close the TCP connection.

9. Use over TLS

XPC may be tunneled over TLS [4] by establishing a TLS session immediately after a TCP session is opened and before any blocks are sent. This type of session is known as XPCS.

When using TLS, a convention must be established to allow a client to authenticate the validity of a server. XPCS uses the same convention as described by IRIS-BEEP [2].

TLS enables authentication and confidentiality.

Implementers should note that while XPC and XPCS have separate URI scheme names and S-NAPTR application protocol labels, both are identified with the same <transferProtocol> value in version information chunks (see Section 6.2).

10. Update to RFC 3981

Section 6.2 of RFC 3981 [1] (IRIS-CORE) states that IRIS-BEEP [2] is the default transport for IRIS. This document revises RFC 3981 and specifies IRIS-XPC as the default transport for IRIS. The TCP well-known port registration is specified in Section 13.5.

11. IRIS Transport Mapping Definitions

This section lists the definitions required by IRIS [1] for transport mappings.

11.1. URI Scheme

See Section 13.1 and Section 13.2.

11.2. Application Protocol Label

See Section 13.3 and Section 13.4.

12. Internationalization Considerations

XML processors are obliged to recognize both UTF-8 and UTF-16 [3] encodings. Use of the XML defined by [9] MUST NOT use any other character encodings other than UTF-8 or UTF-16.

13. IANA Considerations

13.1. XPC URI Scheme Registration

URL scheme name: iris.xpc

Status: permanent

URL scheme syntax: defined in [1].

Character encoding considerations: as defined in RFC 3986 [6].

Intended usage: identifies IRIS XML using chunks over TCP

Applications using this scheme: defined in IRIS [1].

Interoperability considerations: n/a

Security Considerations: defined in Section 14.

Relevant Publications: IRIS [1].

Contact Information: Andrew Newton <andy@hxr.us>

Author/Change controller: the IESG

13.2. XPCS URI Scheme Registration

URL scheme name: iris.xpcs

Status: permanent

URL scheme syntax: defined in [1].

Character encoding considerations: as defined in RFC 3986 [6].

Intended usage: identifies IRIS XML using chunks over TLS

Applications using this scheme: defined in IRIS [1].

Interoperability considerations: n/a

Security Considerations: defined in Section 14.

Relevant Publications: IRIS [1].

Contact Information: Andrew Newton <andy@hxr.us>

Author/Change controller: the IESG

13.3. S-NAPTR XPC Registration

Application Protocol Label (see [5]): iris.xpc

Intended usage: identifies an IRIS server using XPC

Interoperability considerations: n/a

Security Considerations: defined in Section 14.

Relevant Publications: IRIS [1].

Contact Information: Andrew Newton <andy@hxr.us>

Author/Change controller: the IESG

13.4. S-NAPTR XPCS Registration

Application Protocol Label (see [5]): iris.xpcs

Intended usage: identifies an IRIS server using secure XPCS

Interoperability considerations: n/a

Security Considerations: defined in Section 14.

Relevant Publications: IRIS [1].

Contact Information: Andrew Newton <andy@hxr.us>

Author/Change controller: the IESG

13.5. Well-Known TCP Port Registration for XPC

Protocol Number: TCP

TCP Port Number: 713

Message Formats, Types, Opcodes, and Sequences: defined in Section 4.2, Section 3, and Section 4.1.

Functions: defined in IRIS [1].

Use of Broadcast/Multicast: none

Proposed Name: IRIS over XPC

Short name: iris.xpc

Contact Information: Andrew Newton <andy@hxr.us>

13.6. Well-Known TCP Port Registration for XPCS

Protocol Number: TCP

TCP Port Number: 714

Message Formats, Types, Opcodes, and Sequences: defined in Sections 9, 4.2, 3, and 4.1.

Functions: defined in IRIS [1].

Use of Broadcast/Multicast: none

Proposed Name: IRIS over XPCS

Short name: iris.xpcs

Contact Information: Andrew Newton <andy@hxr.us>

14. Security Considerations

Implementers should be fully aware of the security considerations given by IRIS [1] and TLS [4]. With respect to server authentication with the use of TLS, see Section 6 of IRIS-BEEP [2].

14.1. Security Mechanisms

Clients SHOULD be prepared to use the following security mechanisms in the following manner:

- SASL/DIGEST-MD5 - for user authentication without the need of session encryption.
- SASL/OTP - for user authentication without the need of session encryption.
- TLS using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher - for encryption.
- TLS using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher with client-side certificates - for encryption and user authentication.
- TLS using the TLS_RSA_WITH_AES_128_CBC_SHA cipher - for encryption. See [7].
- TLS using the TLS_RSA_WITH_AES_128_CBC_SHA cipher with client-side certificates - for encryption and user authentication. See [7].
- TLS using the TLS_RSA_WITH_AES_256_CBC_SHA cipher - for encryption. See [7].
- TLS using the TLS_RSA_WITH_AES_256_CBC_SHA cipher with client-side certificates - for encryption and user authentication. See [7].

Anonymous client access SHOULD be considered in one of two methods:

1. When no authentication has been used.
2. Using the SASL anonymous profile: SASL/ANONYMOUS

As specified by SASL/PLAIN, clients MUST NOT use the SASL/PLAIN mechanism without first encrypting the TCP session (e.g., such as with TLS). Clients MUST implement SASL/PLAIN and TLS using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher.

14.2. SASL Compliance

The following list details the compliance of IRIS-XPC for use with SASL, as specified by RFC 4422 [10], Section 4.

1. The SASL service name to be used by IRIS-XPC is "iris-xpc".

2. Section 6.2 describes the negotiation facility used to determine the available security mechanisms. This facility may be used both before the initiation of SASL exchanges and after the installation of security mechanisms.

3.

   a) Section 6.5 describes the mechanism to initiate authentication exchanges.

   b) Section 6.5 describes the mechanism to transfer server challenges and client responses.

   c) Section 6.6 and Section 6.7 describe the mechanisms to indicate the outcome of an authentication exchange. Section 6.6 describes how additional data may be carried with this message.

4. Non-empty authorization identity strings used within IRIS-XPC MUST be normalized according to RFC 4013 [11]. The semantics of the non-empty authorization identity strings is server dependent, and clients MUST use the values for these strings as given by configuration or the user.

5. Clients or servers wishing to abort an ongoing authentication exchange MUST close the connection.

6. After new security layers are negotiated, they take effect on the first octet following the authentication success (as) (Section 6.6) chunk sent by the server and on the first octet sent after receipt of the authentication success (as) chunk sent by the client.

7. IRIS-XPC can be used with both TLS and SASL. When used in combination, TLS MUST always be applied before any SASL mechanism.

8. IRIS-XPC does not support multiple SASL authentications. However, if TLS is being used in combination with SASL, TLS authentication MUST occur before any SASL authentication.

15. References

15.1. Normative References

[1] Newton, A. and M. Sanz, "IRIS: The Internet Registry Information Service (IRIS) Core Protocol", RFC 3981, January 2005.

[2] Newton, A. and M. Sanz, "Using the Internet Registry Information Service over the Blocks Extensible Exchange Protocol", RFC 3983, January 2005.

[3] The Unicode Consortium, "The Unicode Standard, Version 3", ISBN 0-201-61633-5, 2000, <The Unicode Standard, Version 3>.

[4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[5] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

[6] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[7] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[8] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[9] Newton, A., "A Common Schema for Internet Registry Information Service Transfer Protocols", RFC 4991, August 2007.

[10] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[11] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

15.2. Informative References

[12] Kirkpatrick, S., Stahl, M., and M. Recker, "Internet numbers", RFC 1166, July 1990.

Appendix A. Examples

This section gives examples of IRIS-XPC sessions. Lines beginning with "C:" denote data sent by the client to the server, and lines beginning with "S:" denote data sent by the server to the client. Following the "C:" or "S:", the line contains either octet values in hexadecimal notation with comments or XML fragments. No line contains both octet values with comments and XML fragments. Comments are contained within parentheses.

It should also be noted that flag values of "yes" and "no" reflect binary values 1 and 0.

The following example demonstrates an IRIS client issuing two requests in one XPC session. In the first request, the client is requesting status information for "example.com". This request and its response are transferred with one chunk. In the second request, the client is requesting status information for "milo.example.com", "felix.example.com", and "hobbes.example.com". This request and its response are transferred with three chunks.

```
S:           (connection response block)
S: 0x20      (block header: V=0,KO=yes)
S:           (chunk 1)
S: 0xC1      (LC=yes,DC=yes,CT=vi)
S: 0x01 0xBF (chunk length=447)
S:           (Version Information)
S: <?xml version="1.0"?>
S: <versions xmlns="urn:ietf:params:xml:ns:iris-transport">
S:   <transferProtocol protocolId="iris.xpc1"
S:     authenticationIds="PLAIN EXTERNAL">
S:     <application protocolId="urn:ietf:params:xml:ns:iris1"
S:       extensionIds="http://example.com/SIMPLEBAG">
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dchk1"/>
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dreg1"/>
S:     </application>
S:   </transferProtocol>
S: </versions>

C:           (request block)
C: 0x20      (block header: V=0,KO=yes)
C: 0x0B      (authority length=11)
C:           (authority="example.com")
C: 0x65 0x78 0x61 0x6D 0x70 0x6C 0x65 0x23 0x63 0x6F 0x6D
C:           (chunk 1)
C: 0xC7      (LC=yes,DC=yes,CT=ad)
C: 0x01 0x53 (chunk length=339)
C:           (IRIS XML request)
```

```
C: <request xmlns="urn:ietf:params:xml:ns:iris1"
C:   xsi:schemaLocation="urn:ietf:params:xml:ns:iris1 iris.xsd" >
C:   <searchSet>
C:     <lookupEntity
C:       registryType="urn:ietf:params:xml:ns:dchk1"
C:       entityClass="domain-name"
C:       entityName="example.com" />
C:   </searchSet>
C: </request>

S:           (response block)
S: 0x20      (block header: V=0,KO=yes)
S:           (chunk 1)
S: 0xC7      (LC=yes,DC=yes,CT=ad)
S: 0x01 0xE0 (chunk length=480)
S:           (IRIS XML response)
S: <iris:response xmlns:iris="urn:ietf:params:xml:ns:iris1">
S:   <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name" entityName="example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
S:   </iris:resultSet>
S: </iris:response>

C:           (request block)
C: 0x00      (block header: V=0,KO=no)
C: 0x0B      (authority length=11)
C:           (authority="example.com")
C: 0x65 0x78 0x61 0x6D 0x70 0x6C 0x65 0x23 0x63 0x6F 0x6D
C:           (chunk 1)
C: 0x07      (LC=no,DC=no,CT=ad)
C: 0x01 0x4E (chunk length=339)
C:           (IRIS XML request)
C: <request xmlns="urn:ietf:params:xml:ns:iris1"
C:  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:  xsi:schemaLocation="urn:ietf:params:xml:ns:iris1 iris.xsd" >
C:   <searchSet>
C:    <lookupEntity
C:      registryType="urn:ietf:params:xml:ns:dchk1"
C:      entityClass="domain-name"
```

```
C:      entityName="milo.example.com" />
C:  </searchSet>
C:           (chunk 2)
C: 0x07      (LC=no,DC=no,CT=ad)
C: 0x00 0xA9 (chunk length=169)
C:           (IRIS XML request)
C:  <searchSet>
C:    <lookupEntity
C:      registryType="urn:ietf:params:xml:ns:dchk1"
C:      entityClass="domain-name"
C:      entityName="felix.example.com" />
C:  </searchSet>
C:           (chunk 3)
C: 0xC7      (LC=yes,DC=yes,CT=ad)
C: 0x00 0xB5 (chunk length=181)
C:           (IRIS XML request)
C:  <searchSet>
C:    <lookupEntity
C:      registryType="urn:ietf:params:xml:ns:dchk1"
C:      entityClass="domain-name"
C:      entityName="hobbes.example.com" />
C:  </searchSet>
C:</request>

S:           (response block)
S: 0x00      (block header: V=0,KO=no)
S:           (chunk 1)
S: 0x07      (LC=no,DC=no,CT=ad)
S: 0x01 0xDA (chunk length=474)
S:           (IRIS XML response)
S: <iris:response xmlns:iris="urn:ietf:params:xml:ns:iris1">
S:   <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name" entityName="milo.example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>milo.example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
S:   </iris:resultSet>
S:           (chunk 2)
S: 0x07      (LC=no,DC=no,CT=ad)
S: 0x01 0xA2 (chunk length=418)
S:           (IRIS XML response)
```

```
S:  <iris:resultSet>
S:    <iris:answer>
S:      <domain authority="example.com" registryType="dchk1"
S:        entityClass="domain-name" entityName="felix.example.com-1"
S:        temporaryReference="true"
S:        xmlns="urn:ietf:params:xml:ns:dchk1">
S:        <domainName>felix.example.com</domainName>
S:        <status>
S:          <assignedAndActive/>
S:        </status>
S:      </domain>
S:    </iris:answer>
S:  </iris:resultSet>
S:           (chunk 3)
S: 0xC7      (LC=yes,DC=yes,CT=ad)
S: 0x01 0xB5 (chunk length=437)
S:           (IRIS XML response)
S:  <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name"
S:  entityName="hobbes.example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>hobbes.example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
S:   </iris:resultSet>
S: </iris:response>
```

Example 1

In the following example, an IRIS client requests domain status information for "milo.example.com", "felix.example.com", and "hobbes.example.com" in one request. The request is sent with one chunk; however, the answer is returned in three chunks.

```
S:           (connection response block)
S: 0x20      (block header: V=0,KO=yes)
S:           (chunk 1)
S: 0xC1      (LC=yes,DC=yes,CT=vi)
S: 0x01 0xBF (chunk length=447)
S:           (Version Information)
S: <?xml version="1.0"?>
S: <versions xmlns="urn:ietf:params:xml:ns:iris-transport">
```

```
S:   <transferProtocol protocolId="iris.xpc1"
S:     authenticationIds="PLAIN EXTERNAL">
S:     <application protocolId="urn:ietf:params:xml:ns:iris1"
S:       extensionIds="http://example.com/SIMPLEBAG">
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dchk1"/>
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dreg1"/>
S:     </application>
S:   </transferProtocol>
S: </versions>

C:           (request block)
C: 0x00      (block header: V=0,KO=no)
C: 0x0B      (authority length=11)
C:           (authority="example.com")
C: 0x65 0x78 0x61 0x6D 0x70 0x6C 0x65 0x23 0x63 0x6F 0x6D
C:           (chunk 1)
C: 0xC7      (LC=yes,DC=yes,CT=ad)
C: 0x02 0xAB (chunk length=683)
C:           (IRIS XML request)
C: <request xmlns="urn:ietf:params:xml:ns:iris1"
C:   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:   xsi:schemaLocation="urn:ietf:params:xml:ns:iris1 iris.xsd" >
C:    <searchSet>
C:     <lookupEntity
C:       registryType="urn:ietf:params:xml:ns:dchk1"
C:       entityClass="domain-name"
C:       entityName="milo.example.com" />
C:   </searchSet>
C:   <searchSet>
C:     <lookupEntity
C:       registryType="urn:ietf:params:xml:ns:dchk1"
C:       entityClass="domain-name"
C:       entityName="felix.example.com" />
C:   </searchSet>
C:   <searchSet>
C:     <lookupEntity
C:       registryType="urn:ietf:params:xml:ns:dchk1"
C:       entityClass="domain-name"
C:       entityName="hobbes.example.com" />
C:   </searchSet>
C: </request>

S:           (response block)
S: 0x00      (block header: V=0,KO=no)
S:           (chunk 1)
S: 0x07      (LC=no,DC=no,CT=ad)
S: 0x01 0xDA (chunk length=474)
S:           (IRIS XML response)
```

```
S: <iris:response xmlns:iris="urn:ietf:params:xml:ns:iris1">
S:   <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name" entityName="milo.example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>milo.example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
S:   </iris:resultSet>
S:            (chunk 2)
S: 0x07       (LC=no,DC=no,CT=ad)
S: 0x01 0xA2  (chunk length=418)
S:            (IRIS XML response)
S:  <iris:resultSet>
S:    <iris:answer>
S:      <domain authority="example.com" registryType="dchk1"
S:        entityClass="domain-name" entityName="felix.example.com-1"
S:        temporaryReference="true"
S:        xmlns="urn:ietf:params:xml:ns:dchk1">
S:        <domainName>felix.example.com</domainName>
S:        <status>
S:          <assignedAndActive/>
S:        </status>
S:      </domain>
S:    </iris:answer>
S:  </iris:resultSet>
S:            (chunk 3)
S: 0xC7       (LC=yes,DC=yes,CT=ad)
S: 0x01 0xB5  (chunk length=437)
S:            (IRIS XML response)
S:  <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name"
S:  entityName="hobbes.example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>hobbes.example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
```

```
S:   </iris:resultSet>
S: </iris:response>
```

Example 2

In the following example, an IRIS client sends a request containing SASL/PLAIN authentication data and a domain status check for "example.com". The server responds with authentication success information and the domain status of "example.com". Note that the client requests that the connection stay open for further requests, but the server does not honor this request.

```
S:           (connection response block)
S: 0x20      (block header: V=0,KO=yes)
S:           (chunk 1)
S: 0xC1      (LC=yes,DC=yes,CT=vi)
S: 0x01 0xBF (chunk length=447)
S:           (Version Information)
S: <?xml version="1.0"?>
S: <versions xmlns="urn:ietf:params:xml:ns:iris-transport">
S:   <transferProtocol protocolId="iris.xpc1"
S:     authenticationIds="PLAIN EXTERNAL">
S:     <application protocolId="urn:ietf:params:xml:ns:iris1"
S:       extensionIds="http://example.com/SIMPLEBAG">
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dchk1"/>
S:       <dataModel protocolId="urn:ietf:params:xml:ns:dreg1"/>
S:     </application>
S:   </transferProtocol>
S: </versions>

C:           (request block)
C: 0x00      (block header: V=0,KO=no)
C: 0x0B      (authority length=11)
C:           (authority="example.com")
C: 0x65 0x78 0x61 0x6D 0x70 0x6C 0x65 0x23 0x63 0x6F 0x6D
C:           (chunk 1)
C: 0x44      (LC=no,DC=yes,CT=sd)
C: 0x00 0x11 (chunk length=11)
C:           (SASL data)
C: 0x05      (mechanism length=5)
C:           (mechanism name="PLAIN")
C: 0x50 0x4C 0x41 0x49 0x43
C: 0x00 0x0A (sasl PLAIN data length=10)
C:           (sasl PLAIN data: authcid="bob")
C:           (sasl PLAIN data: authzid=NULL)
C:           (sasl PLAIN data: password="kEw1")
C: 0x62 0x6F 0x62 0x20 0x00 0x20 0x6B 0x45 0x77 0x31
C:           (chunk 2)
```

```
C: 0xC7      (LC=yes,DC=yes,CT=ad)
C: 0x01 0x53 (chunk length=339)
C:           (IRIS XML request)
C: <request xmlns="urn:ietf:params:xml:ns:iris1"
C:   xsi:schemaLocation="urn:ietf:params:xml:ns:iris1 iris.xsd" >
C:   <searchSet>
C:     <lookupEntity
C:       registryType="urn:ietf:params:xml:ns:dchk1"
C:       entityClass="domain-name"
C:       entityName="example.com" />
C:   </searchSet>
C: </request>

S:           (response block)
S: 0x00      (block header: V=0,KO=no)
S:           (chunk 1)
S: 0x45      (LC=no,DC=yes,CT=as)
S: 0x00 0xD0 (chunk length=208)
S:           (authentication success response)
S: <?xml version="1.0"?>
S: <authenticationSuccess
S:   xmlns="urn:ietf:params:xml:ns:iris-transport">
S:   <description language="en">
S:     user 'bob' authenticates via password
S:   </description>
S: </authenticationSuccess>
S:           (chunk 2)
S: 0xC7      (LC=yes,DC=yes,CT=ad)
S: 0x01 0xE0 (chunk length=480)
S:           (IRIS XML response)
S: <iris:response xmlns:iris="urn:ietf:params:xml:ns:iris1">
S:   <iris:resultSet>
S:     <iris:answer>
S:       <domain authority="example.com" registryType="dchk1"
S:         entityClass="domain-name" entityName="example.com-1"
S:         temporaryReference="true"
S:         xmlns="urn:ietf:params:xml:ns:dchk1">
S:         <domainName>example.com</domainName>
S:         <status>
S:           <assignedAndActive/>
S:         </status>
S:       </domain>
S:     </iris:answer>
S:   </iris:resultSet>
S: </iris:response>
```

Example 3

Appendix B. Contributors

Substantive contributions to this document have been provided by the members of the IETF's CRISP Working Group, especially Robert Martin-Legene, Milena Caires, and David Blacka.

Author's Address

Andrew L. Newton
VeriSign, Inc.
21345 Ridgetop Circle
Sterling, VA 20166
USA

Phone: +1 703 948 3382
EMail: andy@hxr.us
URI: http://www.verisignlabs.com/

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.