



IPDR/SP Protocol Specification

Version 2.1

November, 2004

© 2004 IPDR, Inc.

Preface

Contacts

For general questions regarding this document and referrals to technical experts for detailed questions, please contact:

IPDR.org Editor-in-Chief:

Steve Cotton

scotton@ipdr.org

Abstract

This document specifies the IPDR Streaming Protocol and includes procedures, encodings, and message sets to accomplish this objective.

Change History

Version	Date	Name	Description
Initial Draft	11/11/03	Working Group Team	Initial Draft after harmonization
Review Draft 1	11/15/03	Working Group Team	Consolidation of Several Contributions
Review Draft 2	11/18/03	Working Group Team	Correction of Section Numbering
Review Draft 3	11/22/03	Working Group Team	Addition of Contributions
RD3.1	11/24/03	Working Group Team	Addition of Contributions
RD4	11/25/03	Working Group Team	Addition of Contributions
RD5	12/11/03	Cotton, Givoly, Meyer	Review of RD4, Action Items from 12/10/03 Conference Call
RD6	12/17/03	Weber	Resolution of GW Comments
Ballot Draft	1/2/04	Protocol Working Group	Sent to General Membership for Comment Prior to Steering Committee Approval
2.0	5/4/04	Protocol Group	Resolve Outstanding Issues Prior to Issue
2.0.1	11/4/04	Cotton, Kleinmann, Gotlib	1. Comprehensive cosmetic overhaul 2. Move XDR type encoding table to XDR specification, replace with reference in this document.
2.1	12/08/04	Kleinmann, Gotlib	1. Improved Template negotiation 2. Comprehensive cosmetic overhaul

Acknowledgements

Ty Roach, ACE*COMM

Greg Weber, Cisco Systems

Jeff Meyer, Hewlett Packard

Tal Givoly, Amdocs

Amit Kleinmann, Amdocs

Steve Cotton, Cotton Management Consulting

Table of Contents

1	INTRODUCTION	7
1.1	Specification of Requirements and Notation	7
1.2	Glossary	8
1.2.1	Terminology	8
1.2.2	Acronyms	10
1.3	References	11
1.4	Problem Statement	12
2	PROTOCOL OVERVIEW	13
2.1	Architecture	13
2.2	Configuration	14
2.2.1	Exporter configuration	15
2.2.2	Collector configuration	15
2.3	Data Types	15
2.3.1	Basic Types	15
2.3.2	Structures	15
2.4	Data Representation	15
2.5	Documents	15
2.6	Templates	16
2.6.1	Template Representation	17
2.6.2	Template Transmission and Negotiation	17
2.6.3	Changing Templates	17
2.7	Connection Establishment	18
2.8	Exporter Query Messages	19
2.9	Flow Control	19
2.10	Data Transfer	19
2.11	Sessions	20
2.11.1	Multiplexed Streams within the same single Session	20
2.11.2	Simultaneous multiple Sessions within a single connection	20
2.12	Reliability	21
2.12.1	Data acknowledgment	21
2.12.2	Collector high availability	21
2.13	Protocol Summary	23
2.13.1	IPDR/SP version discovery	27
2.13.2	Protocol Sequence	27

2.14	Backwards Compatibility	28
3	MESSAGE FORMAT	30
3.1	XDR equivalence to “ASCII art”	30
3.2	Common Header	31
4	MESSAGE DETAILS	33
4.1	Connection	33
4.1.1	CONNECT	34
4.1.2	CONNECT RESPONSE	35
4.1.3	DISCONNECT	36
4.2	Errors	36
4.2.1	ERROR	36
4.3	Flow Control	37
4.3.1	FLOW START	38
4.3.2	SESSION START	38
4.3.3	FLOW STOP	39
4.3.4	SESSION STOP	40
4.4	Template	40
4.4.1	TEMPLATE DATA	42
4.4.2	MODIFY TEMPLATE	43
4.4.3	MODIFY TEMPLATE RESPONSE	43
4.4.4	FINAL TEMPLATE DATA ACK	44
4.4.5	START NEGOTIATION	45
4.4.6	START NEGOTIATION REJECT	45
4.5	Data	45
4.5.1	DATA	46
4.5.2	DATA ACKNOWLEDGE	47
4.6	State Independent	47
4.6.1	GET SESSIONS	48
4.6.2	GET SESSIONS RESPONSE	48
4.6.3	GET TEMPLATES	49
4.6.4	GET TEMPLATES RESPONSE	50
4.6.5	KEEP ALIVE	51
5	UNDERLYING TRANSPORT	52
6	SERVICE DISCOVERY	53
6.1	UDP Protocol	53
6.2	Capability Files	54
7	SECURITY	56
8	COMPLETE IPDR/SP IDL DEFINITION	57

1 Introduction

Service Elements are often required to export usage information to mediation and business support systems (BSS) to facilitate accounting. Though there are several existing mechanisms for usage information export, they are becoming inadequate to support the evolving business requirements from service providers.

For example, some of the export mechanisms are legacies of the Telco world. Typically usage information is stored in Service Elements as log files (e.g., CDR files), and exported to external systems in batches. These are reliable methods; however, they do not meet the real-time and high-performance requirements of today's rapidly evolving data networks.

RADIUS [7] is a widely deployed protocol that may be used for exporting usage information. However, it can only handle a few outstanding requests and has extensibility challenges due to its limited command and attribute address space and complexity of its vendor specific attributes (VSAs). A detailed analysis of limitations of RADIUS can be found in [8].

DIAMETER [8] is a new Authentication, Authorization, and Accounting (AAA) protocol that retains the basic RADIUS model, and eliminates several drawbacks in RADIUS. The current DIAMETER protocol and its extensions focus on Internet and wireless network access, and their support of accounting is closely associated with authentication/authorization events. DIAMETER is intended to solve many problems in the AAA area; by doing so, it does not adequately address some critical issues such as efficiency and performance in an accounting protocol.

There are also SNMP based mechanisms that generally require a large amount of processing and bandwidth resources.

Based on more detailed analysis along the lines mentioned above, the need for a reliable, fast, efficient and flexible accounting protocol exists. The IPDR Streaming Protocol is designed to address these critical requirements.

This document defines the Streaming Protocol that enables efficient and reliable delivery of any data, mainly Data Records from Service Elements to any systems, such as mediation systems and BSS/OSS. The protocol is developed to address the critical needs for exporting high volume of Data Records from the Service Element with efficient use of network, storage, and processing resources.

This document specifies the architecture of the protocol and the message format, which **MUST** be supported by all Streaming Protocol implementations.

1.1 Specification of Requirements and Notation

In this document, the keywords "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" are to be interpreted as described in BCP14 [11]. These keywords are not case sensitive in this document.

This document will use initial cap spelling for all keywords which are referenced. For instance the phrase Streaming Protocol is capitalized, to indicate that a concise definition of the meaning of this term is available in the terminology section.

The document will also utilize eXternal Data Representation (XDR)-based [14] Interactive Data Language (IDL) annotation when describing the structure of protocol messages.

1.2 Glossary

1.2.1 Terminology

Active Session

An Active Session is part of the Session during which the Exporter sends Data Records to the Collector. It begins after the Exporter sends a SESSION START message and ends when the Exporter sends a SESSION STOP message. A Session can include one or multiple Active Session(s).

Collection System

A system composed of several Collectors with the objective of receiving, reliably, a non-duplicate set of events, perhaps through the use of redundant Collectors and mechanisms to de-duplicate data

Collector Priority

A Collector is assigned a Priority value. Data Records SHOULD be delivered to the Collector in the flow ready state with the highest priority value (the primary Collector) within a Session.

Collector

A Collector is an implementation on the data receiving side of the Streaming Protocol. It is typically part of a BSS (e.g., billing, market analysis, fraud detection, etc.), or a mediation system. There could be more than one Collector connected to one Exporter to improve robustness of the usage information export system.

Configuration ID

An Identifier of a Template Set Configuration

Data Record

A Data Record is a collection of information gathered by the Service Element for various purposes, e.g., accounting. The structure of a Data Record is defined by a Template, and contains Fields.

Data Sequence Number

A Data Sequence Number is a sequence number, which is attached to all data messages to facilitate reliable and in-sequence delivery.

Document

A Document is a logical range of Data Records.

Document ID

A Document IDentifier.

Element

An Element is a formal declaration of a field in an IPDR Information Model or Service Definition. It is borrowed from the element term in XML-Schema.

Epoch Time

Epoch Time is the time elapsed since 1 January 1970 00:00:00. This is usually expressed in seconds.

Exporter

An Exporter is an implementation on the data producing side of the Streaming Protocol. It is typically integrated with the Service Element's software, enabling it to collect and send out Data Records to an interested consumer system using the protocol defined herein.

Field

A Field is a constituent of a Data Record. The formal term is Field instead of attribute or Element.

Identifier or ID

A means of referring to a specific instance of such items as a Field or a Type and distinguish among them. We also have "Field ID" and "Type ID" as a result.

Information Model

Information Model is a descriptive tool used to capture the set of managed information objects at a conceptual level, independent of any specific implementations or protocols used to transport the information.

Message

A Message is encoded according to rules specified by the Streaming Protocol and transmitted across the interface between an Exporter and Collector. It contains a common Streaming Protocol header and optionally control or user data payload.

Multiplexed Streams

Continuous flow of Data Records within a session where there are at least two Data Records that conforms to different Template Data.

Name

A Name is a means to refer to a specific Element and distinguish among them.

Service Definition

A Service Definition is an XML Schema representation of an Information Model which conforms to the IPDR.org Service Specification Guidelines [1].

Service Element

The logical entity in the IPDR Reference Model which senses usage of services by the Service Consumer and exports associated usage information to the IPDR Recorder. Physically, a Service Element can be any component of functionality from the IPDR High Level Model Network and Service Element (NSE) layer. These may include various components traditionally considered infrastructure network elements.

Session

A Session is a logical connection between an Exporter and one or multiple Collectors for the purpose of delivering Data Records. Multiple Sessions may be maintained concurrently in an Exporter or Collector; they are distinguished by Session IDs.

Stream

Continuous flow of Data Records (A series of Data Records that is placed one after another) from an Exporter toward a Collector within a certain Session. Each and every Data Records of a given stream conforms to the same Template Data.

Streaming Protocol

The Streaming Protocol maybe referred as Streaming, or the Protocol in this document. The Streaming Protocol is used at the interface(s) between an Exporter and one or multiple Collectors for the purpose of delivering Data Records.

Template

A Template is specification of the layout of fields within a Data Record. A Template defines the structure of any types of Data Record, and specifies the Data Type, meaning, and location of the fields in the record.

Template Set Configuration

A set of Templates (identified by certain Template IDs) with a certain order of template fields (identified by certain Field IDs) and associated negotiated enable/disable status.

Type

A Type is a constraint on the value and format of a Field, e.g., dateTime.

1.2.2 Acronyms

AAA	Authentication, Authorization and Accounting DIAMETER
ASCII	American Standard Code for Information Interchange
AVPs	Attribute-Value Pairs
BEEP	Blocks Extensible Exchange Protocol
BSS	Business Support Systems
CDR	Call detail Records

CRANE	Common Reliable Accounting for Network Elements
DSN	Data Sequence Number
HTTP	Hyper Text Transfer Protocol
IDL	Interactive Data Language
IPDR	Internet Protocol Detail Record
N	Negotiable
NE	Network Element
NN	Non-negotiable
NSE	Network and Service Element
OSS	Operation Support Systems
RADIUS	Remote Authentication Dial-In User Service
SCTP	Stream Control Transmission Protocol
SNMP	Simple Network Management Protocol
TCP	Transport Control Protocol
UUID	Universal Unique Identifier
VSA	Vendor-Specific Attributes
XDR	External Data Representation
XML	Extensive Markup Language

1.3 References

- [1] IPDR.org v3.5 Document Map Overview.
- [2] RFC 3423 (CRANE protocol specification).
- [3] CRANE documentation, source and distribution contributed to IPDR
(http://www.ipdr.org/members/protocols/contribs/crane/CRANE_distribution_2003-04-15T21-36-38Z.zip).
- [4] IPDR.org Protocol Encoding:XDR 3.5.1 IPDR Organization, Inc., November, 2004
- [5] IPDR.org Specifications:Service Specification Design Guide 3.5, IPDR Organization, Inc.
- [6] The Fundamental Nature of Standards: Technical Perspective by Ken Krechmer
(available at <http://www.csrstds.com/fundtec.html>). Published in IEEE Communications Magazine, Vol. 38, No. 6, p. 70, June, 2000.
- [7] Remote Authentication Dial In User Service (RADIUS), RFC 2865, June 2000, Rigney, C., Willens, S., Rubens, A. and W. Simpson.
- [8] DIAMETER Base Protocol, RFC 3588, September 2003, Calhoun, P., et. al.
- [9] DIAMETER Framework Document, draft-calhoun-diameter-framework-09.txt.(available at <http://www.watersprings.org/pub/id/draft-calhoun-diameter-framework-09.txt>), Calhoun, P., et. al.
- [10] Stream Control Transmission Protocol", RFC 2960, October 2000, Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson.
- [11] Key words for use in RFCs to Indicate Requirement Levels, BCP14, RFC 2119, March 1997, Bradner, S.
- [12] Security Architecture for the Internet Protocol, RFC 2401, November 1998, Kent, S. and R. Atkinson.
- [13] The TLS Protocol, Version 1.0, RFC 2246, January 1999, Dierks, T. and C. Allen
- [14] XDR: External Data Representation Standard, RFC 1832, August 1995, R. Srinivasan.
- [15] The Blocks Extensible Exchange Protocol Core, RFC 3080, March 2001, M. Rose
- [16] XML-Schema Part 2: Datatypes, W3C, May 2001, <http://www.w3.org/TR/xmlschema-2/>

1.4 Problem Statement

Industry members, as well as the IPDR.org members themselves, perceive the main issue addressed by IPDR.org as the standardization of a billing-grade protocol to replace the variety of protocols currently used to export usage data from network and Service Elements. Every day that goes by without a Streaming Protocol of such nature in the arsenal of protocols turns people away from IPDR.org to solve their needs.

Target applications for this Streaming Protocol would be Service Elements such as:

- Soft switches/VoIP Gateways
- Web/Proxy servers
- Application servers
- Firewalls
- Content Delivery Networks (CDN)
- Streaming media servers
- Game servers
- Passive/active Probes
- Edge access devices (routers, CSU/DSU, CMs, etc).
- Location-based wireless services

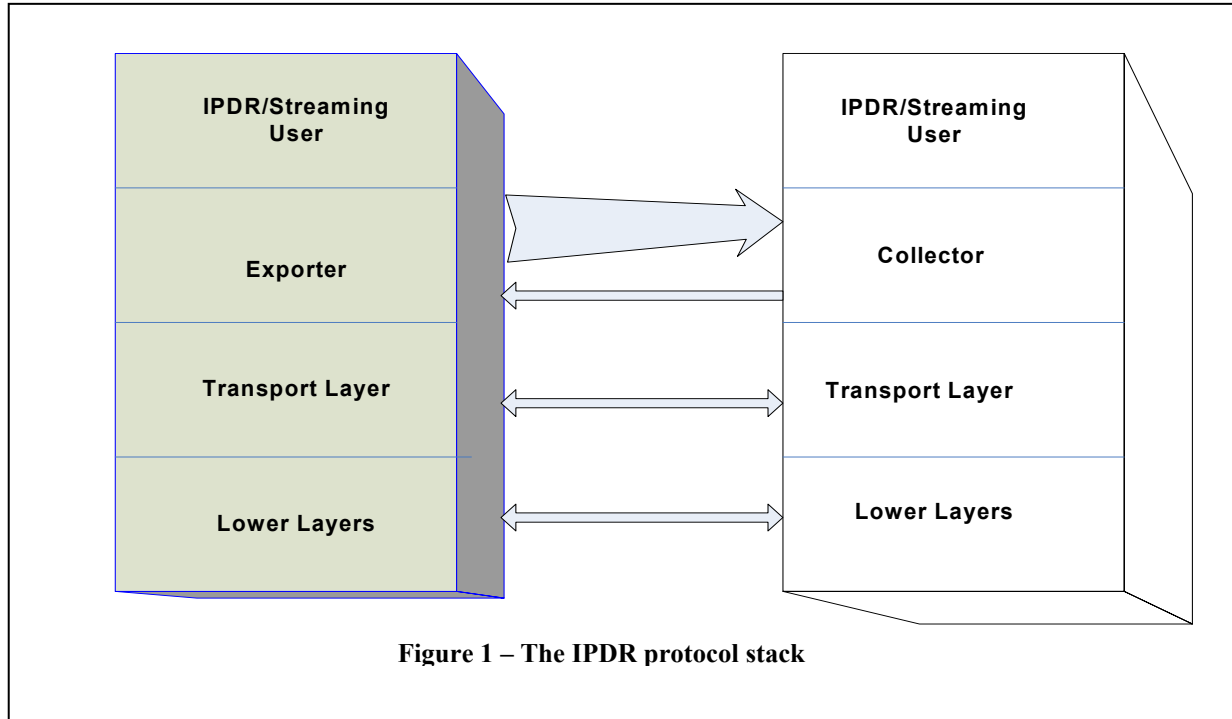
What would it take to make such a protocol gain adoption rapidly?

- It would have to be a final version of a protocol ratified by IPDR.org.
- It would have to have a production-grade implementation (not merely reference implementations).
- The production-grade implementation would have to be embeddable in a variety of network/Service Element platforms with relative ease (low risk and fast implementation time).
- It should be simple enough so people could implement a compliant version on their own.
- Endorsed by 1-2 leading network equipment manufacturers.
- The protocol MUST address a set of requirements (informally specified below):
 - Reliable data delivery with intrinsic fault-tolerance, enables auditing and duplicate elimination.
 - Flexible - supports arbitrary concurrent export of data structures.
 - Efficient implementation possible:
 - o Network link efficiently utilized (overhead/payload and compact payload)
 - o Memory footprint efficient (buffers could be small, copies could be prevented)
 - o Avoids unnecessary copies and conversions in network/Service Element
 - Scalable:
 - o Can handle on the order of 30,000 records/second/processor (assuming 1Ghz processor).
 - o Can operate across a disruptive WAN links
 - Securable to avoid tampering and eavesdropping.
 - Streaming real-time – IPDR already has a document oriented IPDR/File Transfer protocol, what's missing is a Streaming Protocol.

2 Protocol Overview

2.1 Architecture

The IPDR/SP is an application running over a reliable transport layer protocol.



The transport layer protocol is responsible for delivering IPDR/SP Messages between Exporters and Collectors. The transport layer **MUST** support the following capabilities:

1. Reliable, in-sequence message delivery.
2. Connection oriented.

The transport layer may support:

1. Authentication.
2. Bundling of multiple messages into a single datagram.

Possible transport layer protocols may be TCP or SCTP [10]. TCP supports the minimal requirements for IPDR/SP, but lacks some desirable capabilities that are available in SCTP, these include:

1. Session level authentication.
2. Message based data delivery (as opposed to stream based).
3. Fast connection failure detection.

Another possible candidate transport protocol is BEEP (RFC3080) [15]. BEEP sits between low-level transport protocols such as TCP or SCTP and higher-level application protocols such as HTTP.

Reliable delivery of Data Records is achieved through both the transport layer level and the IPDR/SP level. The transport layer acknowledgements are used to ensure reliable delivery of data packets and detection of lost Exporters, the IPDR/SP acknowledges IPDR/SP Messages after they have been processed and the accounting information has been placed in persistent storage.

Being a reliable protocol for delivering Data Records, traffic flowing from an Exporter to a Collector is mostly Data Records. There are also bi-directional control message exchanges, though they only comprise a small portion of the traffic.

Streams of Data Records from a given Exporter have a large amount of internal consistency. That is, the accounting events all represent instances of a small set of accounting record types. IPDR/SP, like the IPDR/XDR [4] format utilizes the concept of Templates in order to eliminate the transmission of redundant information such as Field Identifiers and typing information on a per accounting record basis.

IPDR/SP also incorporates IPDR/Service Definitions, based on XML-Schema, by reference. IPDR/Service Definitions describe in detail the properties of the various accounting records and their Fields. IPDR/SP Templates, like IPDR/XDR Templates identify the Uniform Resource Identifiers (URI) of the schema which describes a given accounting record structure and identifies each Field using its qualified name, as defined in the referenced XML-Schema or in subordinate imported schemas.

An IPDR/SP Exporter may be associated with multiple Collectors. For any given stream of accounting records (termed a "Session"), a single active Collector will be targeted with those Data Records. However, in the event of a detected failure, if an alternate Collector is available, that stream will be redirected to the alternate Collector.

If an IPDR/SP Exporter is managing multiple Sessions these may designate different Collectors as their primary consumer.

A Session may exist between an Exporter and one or more Collectors. A Collector Priority is assigned to each Collector per Session. The Collector Priority reflects the Exporter's preference regarding which Collector should receive Data Records (and optionally negotiate with, on the Template data set). The assignment of the Collector Priority should consider factors such as geographical distance, communication cost, and Collector loading, etc. It is also possible for several Collectors to have the same priority. In this case, the Exporter could randomly choose one of them as the primary Collector to:

1. (Optionally) perform Template data negotiation.
2. Deliver Data Records.

2.2 Configuration

The means by which an Exporter and a Collector are configured are not defined within the scope of this specification

The configuration includes the connection initiation policy (whether the Collector or the Exporter can initiate the connection).

2.2.1 Exporter configuration

The Exporter is responsible for configuring network addresses of all Collectors belonging to a Session as well as the Collector Priority for the Session (It is possible for several Collectors to have the same priority).

2.2.2 Collector configuration

The Collector is responsible for configuring network addresses of all the Exporters it expects to open connection with.

2.3 Data Types

2.3.1 Basic Types

The basic Types and derived Types are encoded in an identical manner as specified for XDR encoding when packaging IPDRDocs in files.

See the IPDR/Protocol/Encoding/XDR specification for definitions of Types.

2.3.2 Structures

Support for structures shall be a negotiable feature. The IPDR/XDR encoding format for IPDR 3.5 does, however, describe a model of binary encoding of structures which may be applied to IPDR/SP.

2.4 Data Representation

All data MUST use an encoding of big-endian.

IDL shall be used to describe data and message formats as well as the use of XDR with the following exceptions:

1. Indefinite length (IPDR/XDR docs [at least])
2. No 32-bit alignment padding

2.5 Documents

The IPDR/SP allows indicating logical boundaries of Data Records. A logical range of Data Records is called a Document.

When a Session is initially established from an Exporter to a Collector, a unique Document Id is assigned by the Exporter in the SESSION START Message.

An Exporter can indicate logical boundaries in the stream (e.g. indicating a new logical document of records), by closing and then reestablishing a Session with the Collector using a new unique Document Id.

The sequence numbering of the Data Records SHOULD begin at 0 each time a new Document ID is assigned to a Session

2.6 Templates

The IPDR/SP enables efficient delivery of Data Records. This is achieved by negotiating a set of Templates for a Session before actual Data Records are delivered. A Template defines the structure of a DATA Message payload by describing the data Type, meaning, and location of the Fields in the payload. By agreeing on Templates, Collectors understand how to process DATA Messages received from an Exporter. As a result, an Exporter only needs to deliver actual data attributes (Fields) without attaching any descriptors of the data; this reduces the volume of information sent over communication links.

A Template is an ordered list of Field Identifiers. A Field Identifier is the specification of a Field in the Template. It specifies an accounting item that a Service Element MAY collect and export. Each Field specifies the Type of the Field.

A Template references an IPDR Service Definition [5] document, where a more complete definition of the Template is included.

The Template is optionally negotiated upon Session initiation between the Exporter and the Collector. This allows the Exporter to avoid sending Fields that the Collector is not interested in.

The IPDR/SP supports usage of several Templates concurrently (for different types of records). Fields contained in a Template could be enabled or disabled. An enabled Field implies that the outgoing Data Record will contain the data item specified by the key. A disabled Field implies that the outgoing record will omit the specified data item. The enabling/disabling mechanism further reduces bandwidth requirement; it could also reduce processing in Service Elements, as only needed data items are produced.

In an IPDR/SP Session, all the Collectors and the Exporter MUST use the same Template Set Configuration. The Templates' configuration and connectivity to an end application MUST be the same in all Collectors. The Exporter MUST publish the relevant Templates to all Collectors in a Session, before it starts to send data according to the Templates.

The complete set of Templates per each Exporter Session MUST bear a Configuration ID that identifies the Template Set Configuration. Each Data Record is delivered with the Template ID and the Configuration ID, so that the correct Template can be referenced. A Collector, when receiving a record with an older Configuration ID, may handle the record gracefully by keeping some Template history. The transport layer SHOULD ensure that a Collector would not get Messages with future Configuration IDs. The Configuration ID is not guaranteed to maintain the same value between different Sessions unless externally declared otherwise by the Exporter. However, for specific session, the Configuration ID is guaranteed to maintain the same value between different connections unless externally declared otherwise by the Exporter

2.6.1 Template Representation

Templates reference IPDR Service Definition as a formal, externally defined specification of the Template structure. Section 4.4 below describes the IDL fragment that defines the formal specification of the "Template Block" and "Field Descriptor" used to describe the structure of a single Template.

2.6.2 Template Transmission and Negotiation

In the simplest form, the Exporter declares the Templates it employs and communicates this information to one or more Collectors. Optionally, Templates may be negotiated. The negotiation is an advanced and advantageous feature compared to most of the alternative protocols for usage exchange. This allows the Collector and Exporter to negotiate a set of Fields within Templates to be sent. Rather than requiring the Exporter to export all Fields, whether the Collector is interested in receiving them or not, this allows the Collector(s) to indicate to the Exporter which Fields they are interested in and this allows the Exporter to determine whether or not to support the requirements of the Collector.

Ultimately, the Exporter determines which Fields to send. A simple Exporter MAY indicate that Template negotiation is not a supported capability. Another Exporter MAY support negotiation, but may based on local configuration chose to send Fields in the initial proposed Template even if the Collector did not indicate an interest.

IPDR/SP has a feature of capability negotiation. The capability to negotiate Templates is a capability that may exist or be absent from a compliant implementation. A Collector and an Exporter MUST be able to communicate with a party that is different than its own capabilities. For example, a Collector that supports Template negotiation should be able to interoperate with both an Exporter that does and doesn't support Template negotiation. It can determine which type of Exporter it is connected to through the capability negotiation that precedes the potential Template negotiation.

Templates are negotiable between an Exporter and a Collector. A Collector MAY propose changes to the Templates received from an Exporter (e.g., enabling some keys and disabling others), or it can acknowledge the Templates as is. In the case that a Template or a Field is not recognized by the Collector (e.g., they might be added to the Exporter after the Collector configuration has completed), the Collector MAY choose to disable each unknown Field or unknown Templates in order to avoid unnecessary traffic. A Template is disabled when all the keys are disabled. It is the Exporter's responsibility to decide what would be the final set of Templates used by a Session.

The Collector MUST not disable Fields in the Template if they are mandatory based on the IPDR Service Definition unless the Collector is disabling all the Fields in a Template to effectively disable transmission of that Data Record type.

Templates are negotiated only with the highest priority Collector within a Session.

2.6.3 Changing Templates

Changing Templates includes changing the Field enable/disable setting of existing Template/s and/or introduction of (a) new Template/s.

Over time the set of information available from an Exporter may change. In this case it may be desirable for the Exporter to modify any existing Sessions in order to begin transmitting Data Records which incorporate these changes. IPDR/SP offers a mechanism to accomplish this change.

End the current Session by sending a SESSION STOP Message. The new set of Templates may then be announced and a negotiation on this set of Templates may occur. Subsequently a new SESSION START Message is issued indicating the resuming of the Session. The Exporter (in case the set of Templates did change) **MUST** change the Configuration ID and indicates a boundary in the Data Record stream by specifying a new Document ID in the SESSION START Message and reset the sequence number to 0. In the case of multiple Collectors available for the same Session, the Exporter **SHOULD** inform the Collectors with lower Priority of the change.

This specification does not define the means by which an Exporter is configured to utilize a new set of Templates.

Over time a Collector may require different set of Fields e.g., it requires reception of more/less data for certain purposes, or it can handle data that previously was meaningless. In this case the Collector request a Template negotiation. The Exporter **MAY** either accept or reject this request.

2.7 Connection Establishment

The IPDR/SP state machine allows for connections to be established by either the Exporter or the Collector.

An implementation of an Exporter may choose to only enable connection establishment in one direction. The Exporter may only initiate connections and not accept inbound connections, or it may only accept inbound connections and not attempt to create connections itself.

To ensure interoperability a Collector **SHOULD** support both directions of connection establishment.

It may prove useful in some deployments to choose a model for connection establishment that is effectively unidirectional; e.g., all Collectors establish connections or accept connections. For instance security policies or device constraints may dictate a unidirectional approach.

In general, support for both directions of connection may reduce the delay in reestablishing communication when either a Collector or an Exporter is restarting. Initiating a connection upon startup provides the quickest means to identify that an Exporter or Collector is available. Alternatively polling strategies may be employed, where connection reestablishment attempts are made during some configurable interval. Means of configuring such intervals are outside the scope of this document.

2.8 Exporter Query Messages

A Collector MAY query an Exporter's status by sending query Messages after it has established a connection with the Exporter. The Exporter MUST respond with response Messages. Only the Collector can initiate the query Messages.

Two query Messages are defined:

- GET SESSIONS – returns information about all the Sessions that are available from this Exporter.
- GET TEMPLATES – returns information about the Templates used on a particular Session.

2.9 Flow Control

Flow control mechanisms are employed to ensure that data is sent from an Exporter to a Collector only if it is ready to receive data. Flow control mechanisms are likewise used to indicate to the Collector whether an Exporter is sending to it data as the primary Collector or it is a redundant/backup Collector for some other Collector that is currently primary. The Flow control also provides information on the DSNs and Document ID so that the Collectors can collectively guarantee that no Data Records are lost.

Four Messages (FLOW START, FLOW STOP, SESSION START and SESSION STOP) are employed to support flow control.

FLOW START and FLOW STOP are sent by the Collector to indicate whether it is ready or not ready to receive data, in the case of FLOW STOP it provides information to the Exporter about why the Collector is no longer willing to receive Data Records (see the [FLOW START](#) section and the [FLOW STOP](#) section)

SESSION START and SESSION STOP Messages are sent by the Exporter to designate the associated Collector the active/inactive data transfer and to provide information static within the Session. (See the [SESSION START](#) section and the [SESSION STOP](#) section)

In addition to explicit flow control Messages mentioned above, KEEP ALIVE Messages can be periodically sent between either communicating parties to the other to ensure the connection is still available (see the [Reliability](#) section).

2.10 Data Transfer

After Templates have been negotiated or set, if negotiation is supported, and both Session and Flow have been started, DATA Messages are sent from the Exporter to the active Collector. This is usually the highest priority Collector that is connected and operational. Each DATA Message contains a Data Sequence Number (DSN). The primary Collector MUST accept the data as long as it is in-sequence. Out-of-sequence DATA Messages SHOULD be discarded.

Upon reception of the Message with initial DSN (designated in the SESSION START Message), the Collector MUST accept all in-sequence DATA Messages. The DSN MUST be incremented by 1 for each new DATA Message originated from the Exporter.

A Collector MUST acknowledge the reception and correct processing of DATA Messages by intermittently sending DATA ACKNOWLEDGE Messages when the window of outstanding Data Records is closing or acknowledgement timers fire. The DATA ACKNOWLEDGE MUST contain the DSN of the last processed in-sequence DATA Message. See section [Reliability](#) for more information

2.11 Sessions

It is sometimes desirable to support multiple different sets of Templates for different applications. For instance, one set of Templates may relate to Accounting/Billing data as well as associated Audit information while another set of Templates may be used for Fraud application or Traffic Engineering.

Sessions define the set of Templates supported by an Exporter. The basic Exporter supports at least one Session that defines the set of Templates it exports.

Each Session has its set of Templates (these may be the same Templates, but the Fields could be enabled or disabled differently). The available Sessions are configured in the Exporter, each with a different Session name with associated Session IDs. The means of this configuration is outside the scope of this specification.

Once a Collector's relationship with an Exporter is in the connected state, the Collector indicates its willingness to participate in Sessions by issuing a FLOW START Message for each target Session.

An Exporter indicates the status of a given Session with a Collector via the SESSION START and SESSION STOP Messages.

A Collector MAY take part in different Sessions. When configuring a Collector, it needs to know the Sessions in which it participates. The Exporter can issue a GetSession Message to receive a list of available Sessions provided by an Exporter. The configuration of Sessions that a Collector MAY participate is outside the scope of this specification

Whether within a single Session or within multiple Sessions, information related to multiple Templates is also multiplexed over the same connection

2.11.1 Multiplexed Streams within the same single Session

In order to associate each DATA Message to the appropriate Template, each DATA Message indicates in its header the Template ID to which it relates (see the [DATA](#) section).

2.11.2 Simultaneous multiple Sessions within a single connection

Although it is desirable to support multiple sets of Templates in some instances, it is not generally so. IPDR/SP supports the ability to have multiple Sessions for communication of different types of accounting records to different Collectors. For example, performance related information to one Collector and billing-related information to another. At times, the same Collector MAY serve as the Collector for multiple types of information. Therefore, the ability to have multiple Sessions on the same link is desirable.

The capability to support multiple Sessions is a capability that may be optionally supported and is negotiated as part of the capabilities negotiation stage of the protocol.

The ability to support multiple Sessions is indicated by setting the MULTISESSION bit in the capabilities bit-mask set within the CONNECT Message. If both communicating parties support MULTISESSION then multiple Sessions MAY exist. An Exporter MAY indicate support for MULTISESSION but still export only one Session.

An Exporter MAY deliver Data Records to different Collection Systems by establishing different Sessions. Each Session MAY consist of several Collectors in a redundant configuration. The Session ID embedded in all the IPDR/SP Messages determines which Session a given IPDR/SP Message is associated with.

The protocol message header indicates which Session that Message relates to, to support the multiplexing of multiple Sessions over the same connection (See the [Common Header](#) section)

2.12 Reliability

Two primary means are employed to reduce the likelihood of data loss:

- Data Records are acknowledged by Collectors when received.
- Redundant Collectors can be connected to an Exporter.

2.12.1 Data acknowledgment

The Exporter is responsible for delivering all the records. A Collector MUST acknowledge the reception and correct processing of DATA Messages by intermittently sending DATA ACKNOWLEDGE Messages when the window of outstanding Data Records is closing or acknowledgement timers fire. The DATA ACKNOWLEDGE MUST contain the DSN of the last processed in-sequence DATA Message. The Collector SHOULD make every attempt to make the data recoverable upon Collector failure before acknowledging Data Records. As an example, a Collector MAY write the records received to a redundant persistent storage array, flush and sync the disk to assure survivability in case of Collector failure. This provides a rather high degree of confidence that the data can be recovered. The indication of acknowledgement from a Collector can be assumed by the Exporter to mean that it no longer has responsibility for these Data Records and MAY remove them from its transient buffer.

2.12.2 Collector high availability

Collector redundancy: For purposes of improved reliability and robustness, redundant Collectors configuration may be employed. Deployment of redundant Collectors is a deployment choice by the operator which will be based on the business impact of lost Data Records.

Additional features such as load balancing may be implemented in a multi-Collector environment. The process of configuring Exporter is carried out using the NE's configuration system and is outside the scope of this document.

KEEP ALIVE Messages are sent periodically between either communicating parties to the other to ensure the connection is still available. These KEEP ALIVE Messages are sent on both primary connections and standby/backup connections to ensure that backup

links are also operational in case of a failover due to failure of link or active/primary Collector.

Transport layer detection: The transport layer (together with some other means) is responsible for monitoring Collector's responsiveness and notifying protocol for any failures. The Exporter MAY choose to transition to an alternate Collector.

Unacknowledged data: When the amount of unacknowledged Data Records reaches a threshold or the time since the last DATA ACKNOWLEDGE Message exceeds that set during the Session initiation, all unacknowledged Data Records SHOULD be transmitted to an alternate Collector. The duplication flag of this/these DATA Message/s SHOULD be set to indicate possible duplication. If alternate Collectors are not available; the Exporter MAY in response to system limitations choose to drop some accounting records. This loss would be indicated by a gap in sequence numbers

Failover: An Exporter SHOULD deliver Data Records to its perceived operating Collector with the highest priority; if this Collector is deemed unreachable, the Exporter MUST deliver the Data Records to the next highest priority Collector that is perceived to be operating.

In this scenario one Collector does not receive all the records but another redundant Collector for the same Collection System receives the rest of the records. For example, Collector #1 could receive records 3042-3095 and then 3123-..., with Collector #2 receiving records 3096-3122. It is the Exporter's responsibility to deliver all the records, in-sequence, but not necessarily to the same Collector

The protocol does not specify how an Exporter SHOULD redirect Data Records to other Collectors, which is considered an implementation issue. But all the supporting mechanisms are provided by the protocol to work in a multiple-Collector environment (e.g., the Template negotiation process, and configuration procedures, etc.).

Data Record delivery SHOULD revert to the higher priority Collector when it is perceived to be operating again.

Recovery: When a Session is initially established from an Exporter to a Collector, a unique Document Id is assigned by the Exporter in the SESSION START Message. See the [Documents](#) section for more information.

If a Session is subsequently reestablished with a Collector as part of recovery, and it represents the continuation of some prior stream of data, the Exporter SHOULD indicate this by using the same Document Id as was used in the previous Active Session. In this recovery situation the DSN SHOULD be in sequence with the previous stream. This MAY include sequence numbers already used, if some records which were not acknowledged on the old stream are being retransmitted.

In summary, the Collection System eventually receives all the Data Records, possibly through more than one Collector. The Exporter MUST convey all the records it received to the Collection System. This may result in duplicate records in the Collection System. In this case, the DSN MUST be used to remove duplicates. To aid the process of

duplicate removal, whenever a Data Record is re-sent (sent for any time past the first time an attempt to send it has been attempted) to another Collector, an appropriate duplication flag of the DATA Message is set to indicate that this Data Record might be a duplicate.

2.13 Protocol Summary

The following diagrams (figures 2 and 3) illustrate the basic state machine employed by an IPDR Exporter and Collector pair. . .

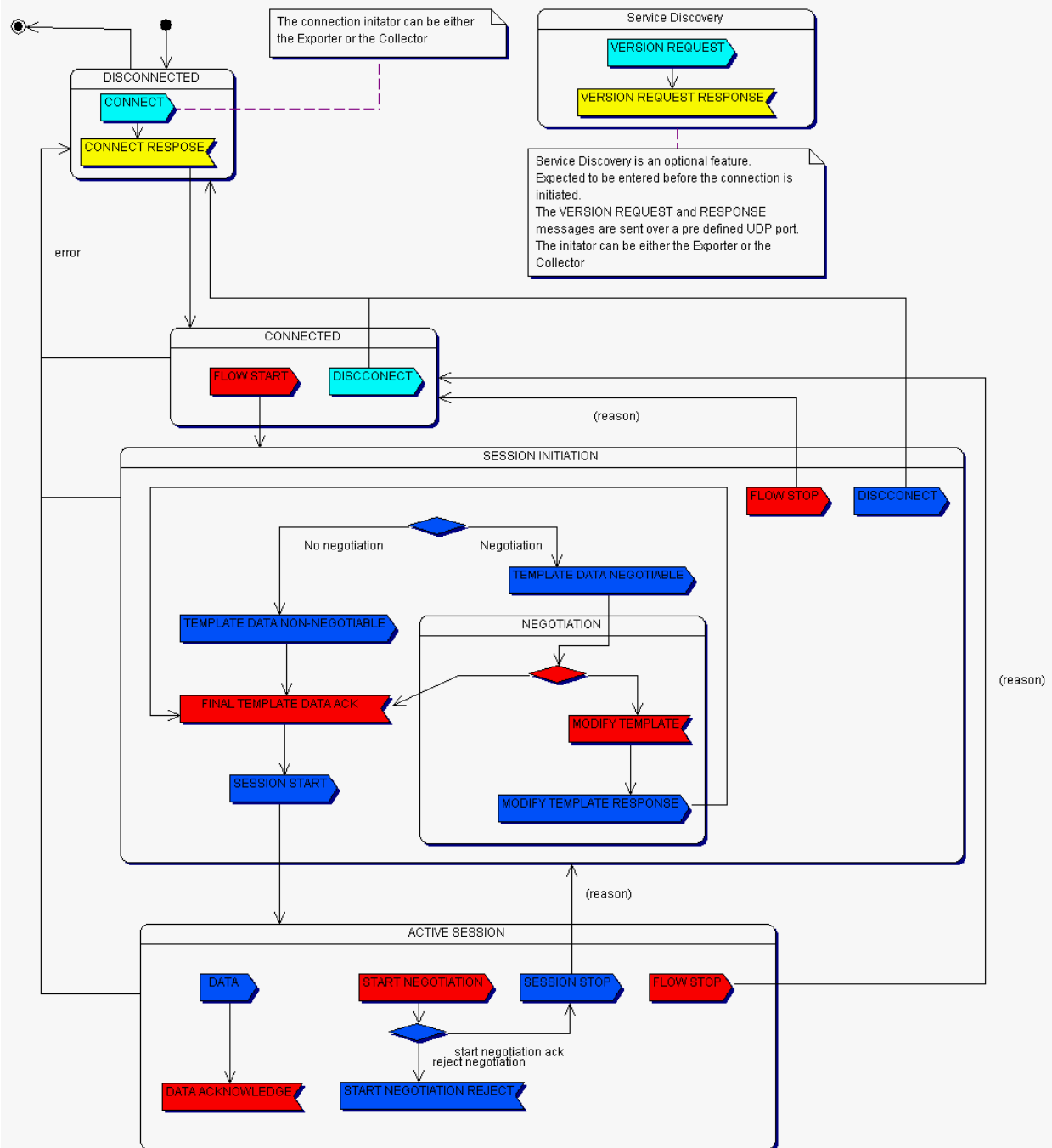


Figure 2: IPDR/SP State Diagram 1

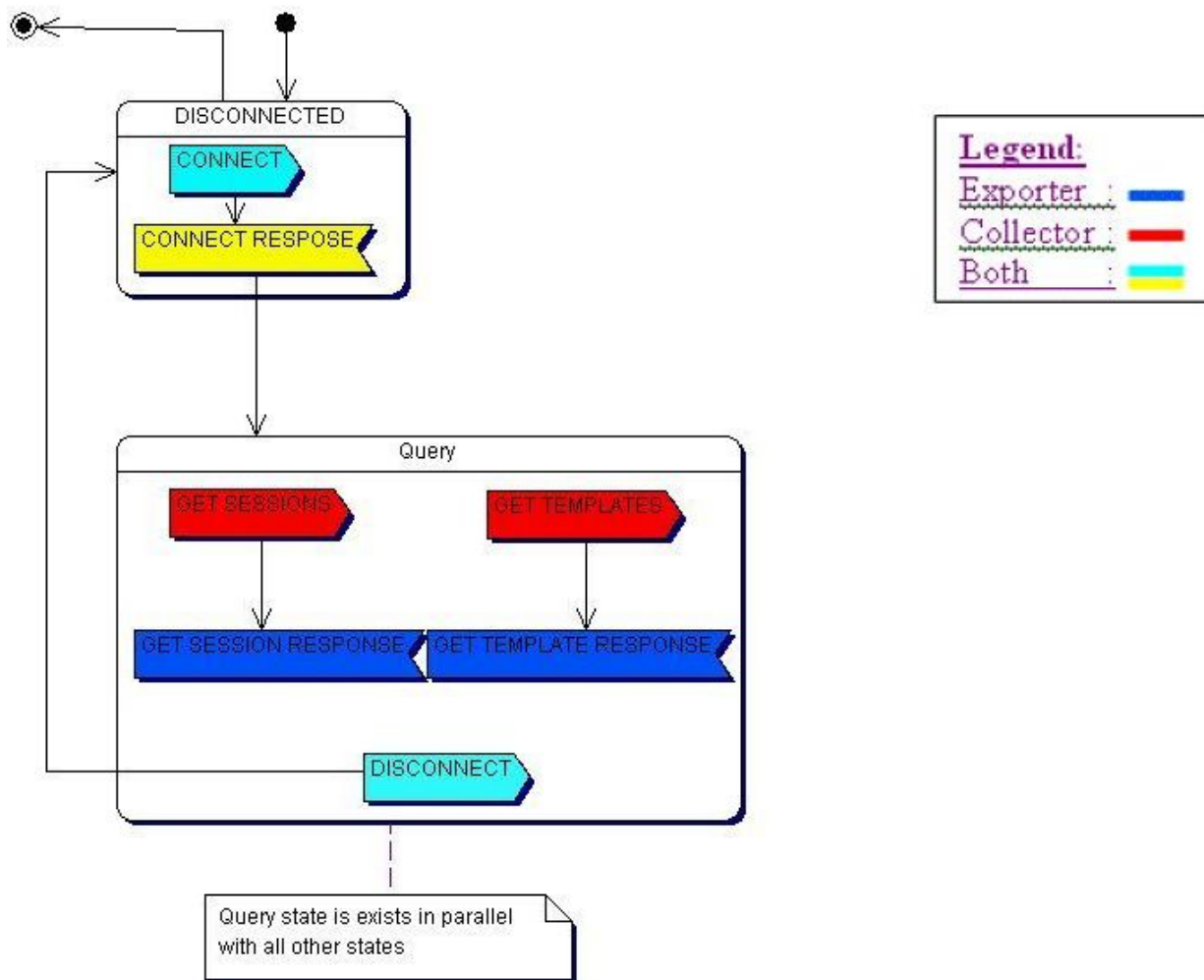


Figure 3: IPDR/SP State Diagram 2

Figure 4 illustrates the IPDR/SP protocol sequence diagram.

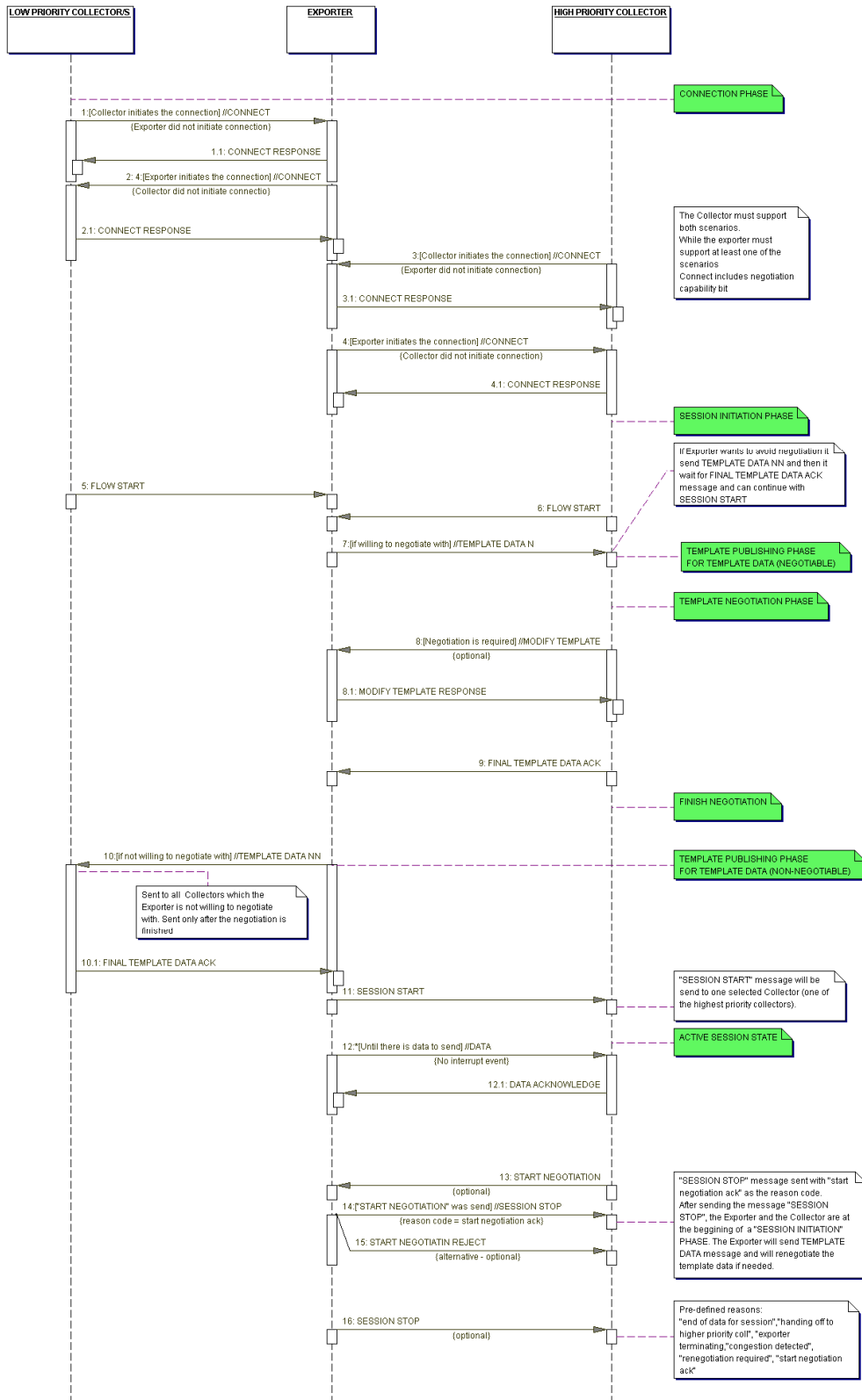


Figure 4: IPDR/SP protocol sequence diagram

2.13.1 IPDR/SP version discovery

Either party (either Exporter or Collector) MAY inquire about the IPDR/SP version and transport layer support by sending a UDP packet on an agreed UDP port. If the receiving party implements version discovery, it MUST respond to this request with a UDP packet carrying the protocol version, the transport type and the port number used for the specific transport.

The inquiring party sends a VERSION REQUEST Message to query the other party's protocol support. The receiving party will respond with a VERSION REQUEST RESPONSE Message that contains details about the protocols that it supports

2.13.2 Protocol Sequence

1. Connection phase

- a. A Collector or an Exporter initiates a connection by sending a CONNECT Message.
- b. The corresponding Exporter or Collector respectively responds to the CONNECT Message with a CONNECT RESPONSE Message

During this phase the Collector and the Exporter agree on:

- The KEEP ALIVE interval (of the remote side). In case the KEEP ALIVE Message is not received on time (during a low traffic period – no other Message is received) the connection is terminated. It is measured in second units.
- The support of (based on the capabilities flags):
 - Template negotiation
 - Simultaneous multi-Sessions
 - Structure Data Types

2. Session initiation phase

- a. A Collector sends the FLOW START Message as soon as it is ready to start the Session.
- b. The Exporter sends the TEMPLATE DATA Message as soon as it receives the FLOW START Message from the Collector. The TEMPLATE DATA Message it sends is either:
 - i. Negotiable (sent only to the high priority Collector, only if both the Exporter and the Collector support Template negotiation and only if the Exporter policy determines that Template negotiation is needed – see 2.c) or
 - ii. Non-negotiable (see 2.d)
- c. Negotiation phase (this phase is reached in case the Exporter sends Negotiable Template Data, see 2.b.i)
 - i. Optionally the Collector can respond to the negotiable TEMPLATE DATA Message with a MODIFY TEMPLATE Message (in case Template negotiation is requested) in order to change the accounting information for a particular Session. For example it may be required to receive a subset of the Fields available from the Exporter in order to reduce throughput requirements. At this stage the Exporter determines the Template information (the result of the Template negotiation) for the Session and will respond with MODIFY TEMPLATE RESPONSE

-
- ii. Next (if the collector does not want to negotiate the Template Data or after a Collector that requested Template negotiation receives a MODIFY TEMPLATE RESPONSE Message), the Collector MUST send a FINAL TEMPLATE DATA ACK Message.
 - iii. When the Exporter receives a FINAL TEMPLATE DATA ACK Message from the negotiator Collector, it MUST first send non-negotiable TEMPLATE DATA to all other Collectors of the specific Session (case 2.d). Then it will send a START SESSION Message to the high priority Collector and the Session will begin.
 - d. In case the Exporter sends non negotiable TEMPLATE DATA (see 2.b.ii)
 - i. The Collector MUST respond with FINAL TEMPLATE DATA ACK
 - ii. Following that the Exporter MUST respond with SESSION START Message and the Session will begin.
 - 3. Active Session phase
 - a. At this stage the Exporter sends DATA Messages according to the agreed set of Templates.
 - b. The Collector MUST respond with DATA ACKNOWLEDGE Message
 - c. In some cases a Collector MAY need to renegotiate the Template data (assuming Template negotiation is supported).
 - i. The Collector can request a Template data negotiation by sending a START NEGOTIATION Message to the Exporter.
 - ii. The Exporter MUST respond to the START NEGOTIATION Message with either:
 - 1. A START NEGOTIATION REJECT Message. In this case the Session will continue with no effect.
 - 2. A SESSION STOP Message with the reason code flag set to "start negotiation ack". Then negotiation begins as described in 2b-2d.
 - d. In some cases an Exporter MAY need to renegotiate the Template data (assuming Template negotiation is supported). In order to do it, it MUST send SESSION STOP Message with the reason code flag set to "renegotiation is required". At this stage the negotiation begins as described in 2b-2d.
 - 4. During a connection the Collector can send to the Exporter:
 - a. a GET SESSION Message. The Exporter MUST respond to this Message with the GET SESSION RESPONSE that contains all the existing Sessions.
 - b. a GET TEMPLATES Messages. The Exporter MUST respond to this Message with the GET TEMPLATES RESPONSE Message that contains the Template set for the specific Session.

2.14 Backwards Compatibility

IPDR/SP builds upon two existing specifications, namely IPDR's XDR file format and RFC3423, Common Reliable Accounting for Network Elements (CRANE). As part of this evolution, some level of compatibility with previous versions is a goal.

IPDR/SP uses the same basic version negotiation mechanism as the original CRANE and borrows much of its message set from CRANE. IPDR/SP Messages are distinguished from CRANE Messages by advancing the version Field in the Message headers to 2.

Beginning with IPDR version 3.5 IPDR/XDR and IPDR/XML formats are able to record the contents of any IPDR/SP stream as documents.

In the case of IPDR/XML, this is supported by further expansion of the XML-Schema subset which may be used in creating IPDR Service Definitions, and the generation of XML document instances which are valid according to the Schema.

In the case of IPDR/XDR, there are complementary extensions which align it with the encoding model of IPDR/SP messages. Advancing the version number in the IPDR/XDR header from 3 to 4 indicates IPDR/XDR documents capable of preserving any IPDR/SP flow of accounting records.

The reader may notice some discrepancies between IPDR/XDR's Template and data encodings and those of IPDR/SP. These are considered artifacts of the streaming versus documented oriented design center. IPDR/XDR and IPDR/SP are aligned in terms of their information content.

3 Message Format

IPDR/SP describes its message format using an augmented form of RFC1832, External Data Representation (XDR).

Two augmentations of XDR used by IPDR are as follows:

1. Support for indefinite length specification. This allows for stream based encoding of information without knowing or calculating the entire length of a Message or document in advance. The value of -1 in a length Field indicates that, based on Template information, a decoder SHOULD be able to determine where a Message completes.
2. No 32-bit alignment padding. Beginning in IPDR 3.5, both IPDR/XDR and IPDR/SP remove the padding constraint specified by XDR. This allows for specification to the byte level of structures. This augmentation is described in section 3.20 of RFC1832, "Areas for Future Enhancement".

By using these two extensions the protocol specification can move away from the use of "ASCII art" common in most IETF protocol specifications and use a more concise and formal C style syntax for describing protocol message formats.

Note, it would be possible to create automated tools to generate ASCII art from the XDR IDL specification language, the converse would be significantly more difficult.

3.1 XDR equivalence to "ASCII art"

The use of XDR specification of protocol message structure is a departure from the majority of IETF protocols operating in the accounting domain. Rather the typical mode of specification is based on the use of "ASCII art" diagrams which illustrate a series of bytes and the position various Fields occupy in that byte table.

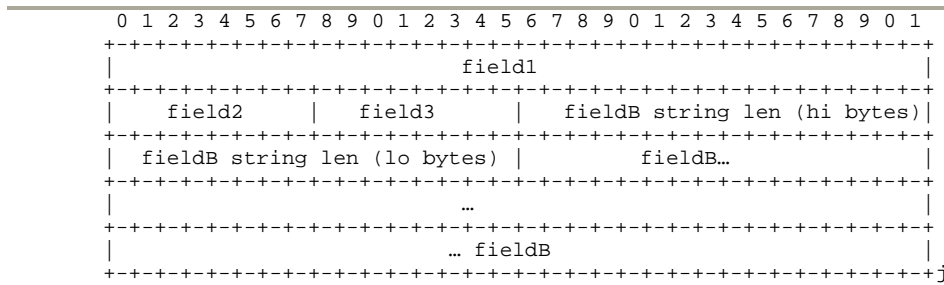
XDR Augmentation number 2 cited above, allows the same information to be conveyed in the more concise and machine consumable format.

Note that one might also consider a "pure" XML mechanism for this based on additional Schema constraints and mapping policies. However, existing tools and specifications supporting XDR's byte level encoding make the use of XDR IDL convenient for our current problem.

By way of example, the basic mapping from an XDR style specification to its equivalent ASCII art model is illustrated below:

```
struct ExampleA {
    int field1;
    char field2;
    char field3;
}

struct ExampleB {
    struct ExampleA fieldA;
    UTF8String fieldB;
}
Example B layout:
0                               1                               2                               3
```



The Example B structure has at its beginning an embedded Example A structure. The ASCII art shows the Example. A structure elements in their appropriate sizes and position.

UTF8String's have a 32-bit length Field according to XDR encoding rules. Since we are augmenting the XDR encoding by doing byte level packing, the string length is not aligned to 32-bit word boundaries. The fieldB string is variable length, so is shown spanning multiple bytes of indeterminate length. Note that fieldB's contents also begin at the packed byte boundary, no 32-bit boundary alignment is performed.

If 32-bit alignment is desired, it can be achieved by populating structures with a reserved char, or several to pad out the location of a desired aligned Field.

3.2 Common Header

All IPDR/SP Messages begin with the following 8 byte header:

```

struct IPDRStreamingHeader {
    char version;          /* version of protocol, for this version, set to 2          */
    char messageId;        /* the ID of this Message (see MessageIds)                          */
    char sessionId;        /* the ID of this Session or 0 if not Session-specific              */
    char messageFlags;     /* reserved/unused and set to 0                                     */
    int messageLen;        /* length in bytes of Message including header                      */
};

```

It is followed by a variable length message payload. Some of the Messages do not have any additional payload part. All IPDR/SP Messages are XDR encoded in Network Byte order.

Details of each Field are described below:

version

Indicates the supported IPDR/SP implementation. This Field **MUST** be set to 2.

messageId

Identifies the type of the Message. The Message IDs defined by IPDR/SP 2.1 are defined as follows. See Table 1 in [section 4](#) for additional details of each Message.

sessionId

Identifies the Session with which the Message is associated. The Session ID is ignored in the case of basic connection related Messages. Connection related Messages are: CONNECT,

CONNECT RESPONSE, DISCONNECT, KEEP ALIVE, FINISH NEGOTIATION, GET SESSIONS, GET SESSIONS RESPONSE and ERROR. For implementations which do not support multiple Sessions, the sessionId 0 SHOULD always be used.

messageFlags

used to identify options associated with the Message. Currently no flags are defined.

messageLen

the total length of the IPDR/SP Message in octets including the header.

The Message set supported by a basic valid implementation is dependent on the capabilities identified in the connection request.

4 Message Details

Table 1 summarizes the message set defined for IPDR/SP v2.1. The following sections will provide details on the structure of each Message, using the augmented XDR encoding described in the [Message Format](#) section.

Table 1

Category	ID	Message	Direction	Comments
Connection	0x05	CONNECT	Either	
	0x06	CONNECT RESPONSE	Either	Opposite of CONNECT
	0x07	DISCONNECT	Either	
Errors	0x23	ERROR	Either	In lieu of response, underlying connection cleared
Flow Control	0x01	FLOW START	C E	Framed in Sessions
	0x08	SESSION START	E C	
	0x03	FLOW STOP	C E	
	0x09	SESSION STOP	E C	
Template	0x10	TEMPLATE DATA	E C	
	0x1a	MODIFY TEMPLATE	C E	Optional
	0x1b	MODIFY TEMPLATE RESPONSE	E C	Optional
	0x13	FINAL TEMPLATE DATA ACK	C E	
	0x1d	START NEGOTIATION	C E	Optional
	0x1e	START NEGOTIATION REJECT	E C	Optional
Data	0x20	DATA	E C	
	0x21	DATA ACKNOWLEDGE	C E	
State Independent	0x14	GET SESSIONS	C E	
	0x15	GET SESSIONS RESPONSE	E C	
	0x16	GET TEMPLATES	C E	
	0x17	GET TEMPLATES RESPONSE	E C	
	0x40	KEEP ALIVE	Either	May affect fail-over

4.1 Connection

Connection related Messages deal with moving the state machine for an IPDR/SP Exporter and Collector from the Disconnected State to the Connected state. For more details on the state machine, see the figures and discussion in the [Protocol Summary](#) section.

The CONNECT and CONNECT RESPONSE Messages provide for both the acknowledgement of both peers that they are capable of participating in the exchange of records via IPDR/SP as well as a means of both peers indicating any optional protocol capabilities they support. To use any optional capability, both Exporter and Collector MUST indicate their support of that capability.

4.1.1 CONNECT

The creator of the underlying reliable transport connection **MUST** send the connection message. Either the Exporter or the Collector **MAY** initiate the underlying transport connection.

Following is the IDL fragment that defines the formal specification of the CONNECT Message:

```
struct Connect {
    struct IPDRStreamingHeader header;
    int initiatorAddress; /* IPv4 IP address of the initiator (Collector/Exporter */
    short initiatorPort; /* The transport protocol port number of the initiator */
    int capabilities; /* an array of capability bits for capability negotiation */
    int keepAliveInterval /* the maximum delay between some indication from remote, */
    /* expressed in seconds */
    /*

    UTF8String vendorId; /* The vendor ID of the initiator (Exporter/Collector) */
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

initiatorAddress

IPv4 IP address of the initiator (Collector or Exporter).

initiatorPort

The transport protocol port number of the initiator

capabilities

An array of capability bits for capability negotiation.

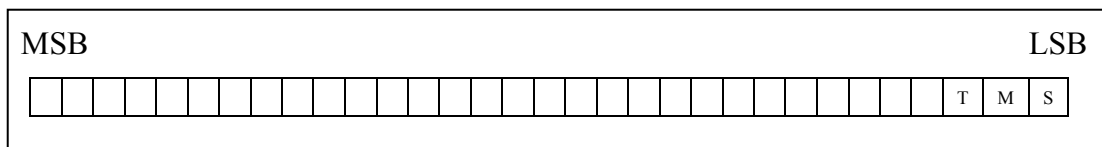


Figure 5

The capabilities flags indicate supported options for this connection. Specifically the following capabilities are optional:

- **STRUCTURES** (see 'S' in figure 5 above) – if set, this flag indicates that the accounting records being sent may be structured as supported in IPDR 3.5, vs. being in first normal form, as specified in IPDR 3.1.1. Structured data includes repeating primitive Type Fields or substructures within a single record. Common scenarios where this might arise are in conference calls or other multiparty accounting that may group all parties in a single record. Note that typical Relational Databases require data in first normal form, so there are benefits to observing this model. However, the need to represent existing industry standard structures and still carry them in IPDR has driven the support for STRUCTURES

- MULTISESSION (see 'M' in figure 5 above) – if set indicates that the party allows support for multiple Sessions of accounting records. Simpler equipment and implementations may chose to only support a single stream
- TEMPLATE NEGOTIATION (see 'T' in figure 5 above) – if set allows the Collector to request a different set of accounting records or Fields than that advertised initially by the Exporter. Simpler Exporters MAY choose not to support negotiation. In this case this capability SHOULD be off, and completing the connection dialog puts the two parties in the Connected state. Negotiating Exporters and Collectors enter a Template negotiation state after exchanging the connection dialog when this flag is set

keepAliveInterval

The KEEP ALIVE interval indicates the maximum amount of idle time on a connection before a KEEP ALIVE Message MUST be sent to assure the underlying transport connection is still available. It is the maximum allowed delay between some indications from a remote are received in the connection level. This value is expressed in seconds.

vendorId

A Vendor Identifier is a string that identifies the vendor that created the initiator (Exporter/Collector).

The CONNECT Message has the messageId set to 0x05 in the header.

4.1.2 CONNECT RESPONSE

The recipient of a connect Message MUST send a CONNECT RESPONSE. The response indicates the desired Keep Alive interval of the responder as well as its capabilities.

Following is the IDL fragment that defines the formal specification of the CONNECT RESPONSE Message:

```
struct ConnectResponse {
    struct IPDRStreamingHeader header;
    int capabilities; /* an array of capability bits for capability negotiation */
    int keepAliveInterval; /* the maximum delay between some indication from remote */
    /* It is expressed in seconds. */
    UTF8String vendorId; /* the vendor ID of the responder (Exporter/Collector) */
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

capabilities

The meaning of capabilities is the same as defined in CONNECT.

Note that the responder **SHOULD** only indicate those capabilities which were previously sent by the connecting party and which it is willing to support. Identifying capabilities which were not proposed by the connector has no effect, those capabilities are not available for the current connection.

keepAliveInterval

The meaning of keepAliveInterval is the same as defined in Connect.

vendorId

A Vendor Identifier is a string that identifies the vendor that created the responder (Exporter/Collector).

The CONNECT RESPONSE Message has the messageId set to 0x06 in the header

4.1.3 DISCONNECT

Either party in the course of graceful termination of a connection **MAY** send the DISCONNECT Message. It is not acknowledged and **SHOULD** be followed by sending party disconnecting the underlying transport, and the receiving party, issuing the corresponding close of connection on their side.

Following is the IDL fragment that defines the formal specification of the DISCONNECT Message:

```
struct Disconnect {  
    struct IPDRStreamingHeader header;  
};
```

Field description:**header**

The common header (see the [Common Header](#) section)

The Disconnect has a messageId of 0x07 in the header.

4.2 Errors

Either the Exporter or the Collector **MAY** issue an ERROR Message in the event where either a communication error has been detected or other failures impacting the connection. An error Message, that is not session oriented, **MUST** be followed by the sender initiating disconnection of the underlying transport. The recipient of an ERROR will also issue the corresponding close of connection on their side.

4.2.1 ERROR

Following is the IDL fragment that defines the formal specification of the ERROR Message:

```
struct Error {  
    struct IPDRStreamingHeader header;  
    int timeStamp;           /* time of error (in seconds from epoch time) */  
    short errorCode;         /* The error code field consists of two parts: */  
};
```

```

/* Session oriented flag: it is a one bit flag. */
/* It is the MSB of the errorCode field. It      */
/* indicates whether the error is specific for    */
/* the session (=1) or it is a general error and */
/* thus it is not specific to the session (=0). */
/* The code ID: The rest 15 LSBs of the          */
/* errorCode field, specifies the error code ID */
/* (0 - 32767). Values of 0-255 are reserved for */
/* standard error codes.                        */
/* 0 = keepalive expired                       */
/* 1 = Message invalid for capabilities         */
/* 2 = Message invalid for state                */
/* 3 = Message decode error                     */
/* 4 = process terminating                      */
/* Values > 255 may be used for vendor specific */
/* error codes                                 */

    UTF8String description;
};

```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

timestamp

The time of the error occurrence (in seconds from epoch time)

errorCode

The error code field consists of two parts:

1. Session oriented flag: It is a one bit flag. It is the MSB of the errorCode field. It indicates whether the error is specific for the session (=1) or it is a general error and thus it is not specific to the session (=0). In the last case the ERROR Message MUST be followed by the sender initiating disconnection of the underlying transport.
2. The code ID: The rest 15 LSBs of the errorCode field, specifies the error code ID (0 - 32767) as follows:
 - Values of 0-255 are reserved for standard reason codes.
 - 0 = keep alive expired
 - 1 = message invalid for capabilities
 - 2 = message invalid for state
 - 3 = message decode error
 - 4 = process terminating
 - Values > 255 may be used for vendor specific codes.

description

Description of the error

The ERROR Message has the messageId set to 0x23 in the header

4.3 Flow Control

Appropriate flow control is necessary for managing the reliable delivery of accounting records and ensuring the Collector has recorded them.

Flow control deals with the acknowledging and throttling of exported streams of records, as well as enabling redundant Collectors to perform rapid failover with duplicate detection.

Bear in mind that IPDR/SP already assumes an underlying reliable transport. However, the protocols at this level leave ambiguous details about the disposition of data in transit at the time of a failure. A simple application level acknowledgement scheme, based on windows of configurable sizes, allows for this necessary exchange between the Exporter and Collector, while imposing minimal overhead.

4.3.1 FLOW START

FLOW START Messages may only be sent by the Collector to indicate its willingness to participate in a Session for a particular stream of accounting records.

Following is the IDL fragment that defines the formal specification of the FLOW START Message:

```
struct FlowStart {  
    struct IPDRStreamingHeader header;  
};
```

Field description:

header

The common header (see the [Common Header](#) section)

The FLOW START Message has the messageId set to 0x01 in the header

4.3.2 SESSION START

SESSION START Messages are only sent by the Exporter to indicate that an accounting stream is now actively flowing to a Collector. This distinguishes between the situations where no accounting records are available and when accounting records are being sent to an alternate Collector.

Following is the IDL fragment that defines the formal specification of the SESSION START Message:

```
struct SessionStart {  
    struct IPDRStreamingHeader header;  
    int exporterBootTime; /* boot time of Exporter(in seconds from epoch time)*/  
    long firstRecordSequenceNumber; /* first sequence number to be expected */  
    long droppedRecordCount /* number of records dropped in gap situations */  
    boolean primary; /* indication that the Collector is the primary */  
    int ackTimeInterval; /* the maximum time between acks from Collector */  
    /* (in second units) */  
    int ackSequenceInterval; /* the maximum number of unacknowledged records */  
    char documentId[16]; /* the UUID associated with the info being sent */  
    /* in this Session */  
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

exporterBootTime

Boot time of Exporter (in seconds from epoch time).

firstRecordSequenceNumber

First sequence number, of the Record Data, to be expected

droppedRecordCount

Number of records dropped in gap situations.

primary

Indication that the Collector is the primary

ackTimeInterval

The maximum time between acknowledges from Collector (in second units)

ackSequenceInterval

The maximum number of unacknowledged records

documentId

The UUID associated with the info being sent in this session

Notice: Data Records have 64-bit counters. Together with a UUID that represents the document, it is possible to uniquely distinguish records from one another.

The SESSION START Message has the messageId set to 0x08 in the header

4.3.3 FLOW STOP

FLOW STOP Messages may only be sent by the Collector to indicate that it is no longer able to participate in a particular Session. The reasonInfo Field contains a string which the Collector MAY use to provide additional details. The Exporter MAY choose to log this information for operational support purposes.

Following is the IDL fragment that defines the formal specification of the FLOW STOP Message:

```
struct FlowStop {
    struct IPDRStreamingHeader header;
    short reasonCode;           /* values of 0-255 are reserved for standard */
                                /* reason codes. Values > 255 may be used for */
                                /* vendor specific codes. */
                                /* 0 = normal process termination */
                                /* 1 = termination due to process error */
    UTF8String reasonInfo;
};
```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

reasonCode

The reasons codes:

- values of 0-255 are reserved for standard reason codes.
 - 0 = normal process termination
 - 1 = termination due to process error
- Values > 255 may be used for vendor specific codes.

reasonInfo

Description of the reason to stop the Flow.

The FLOW STOP Message has the messageId set to 0x03 in the header

4.3.4 SESSION STOP

SESSION STOP Messages are only sent by the Exporter to indicate that a Collector is no longer the active recipient of a stream of accounting records.

Following is the IDL fragment that defines the formal specification of the SESSION STOP Message:

```
struct SessionStop {
    struct IPDRStreamingHeader header;
    short reasonCode;
    UTF8String reasonInfo;
};
```

/* values of 0-255 are reserved for standard */
 /* reason codes. Values of > 255 may be used for */
 /* vendor-specific codes. */
 /* 0 = end of data for session */
 /* 1 = handing off to higher priority Collector */
 /* 2 = Exporter terminating */
 /* 3 = congestion detected */
 /* 4 = renegotiation is required */
 /* 5 = start negotiation acknowledge */
 /* 6 = end of IPDRDoc */
 /* 7 = Template data was updated */

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

reasonCode

The reasons codes are enumerated in the IDL fragment above.

reasonInfo

Description of the reason to stop the Session.

The SESSION STOP Message has the messageId set to 0x09 in the header

4.4 Template

Templates form the foundation for efficient exchange of large volumes of similar accounting records. By summarizing the different accounting record Fields, their Types and order in a Template, the individual encoded accounting records can be efficiently packed according to augmented XDR.

IPDR/SP borrows the same self describing data model as used by IPDR/XDR and IPDR/File formats. Specifically Templates point to XML-Schemas which are compliant with the IPDR Service Definition Guidelines. The Templates also identify the Field's name and Type code. Type code (typeId) MUST indicate derived Types when applicable. By putting the name and Type code in the Template, along with the schema, a consumer need not have access to the schema directly in order to decode DATA Messages, because the Types all have well defined encoding policies.

Following is the IDL fragment that defines the formal specification of the FieldDescriptor and TemplateBlock used to describe the structure of a single Template:

```

struct FieldDescriptor{
    int         typeId;      /* ID of Field Type                */
    int         fieldId;     /* unqualified Field code that can be used */
                                /* as alternative accessor handles to Fields */
    UTF8String  fieldName;  /* Note that Field names are to be qualified */
                                /* with the Namespace name, as an example: */
                                /* http://www.ipdr.org/namespace:movieId */
                                /* The namespace MUST match one of those */
                                /* targeted by the schema or schema imports */
    boolean     isEnabled   /* true=enabled Field false=disabled Field */
};

```

Fields descriptions:

typeId

ID of Field Type. Defines the Data Type of the Field.

fieldId

Unqualified Field code that can be used as alternative accessor handles to Fields. The fieldId can be optionally set by the Exporter to uniquely allow access to (set/get of Fields) without using the textual fieldName. The fieldId is not guaranteed to maintain the same value between different Sessions and different connections unless externally declared otherwise by the Exporter.

fieldName

identifies a Field within a Template. A fieldName MUST be qualified within the namespace name, as an example: <http://www.ipdr.org/namespace:movieId> The namespace MUST match one of those targeted by the schema or schema imports

isEnabled

an enable/disable flag – true = an enabled Field false = a disabled Field

```

struct TemplateBlock{
    short       templateId; /* ID of Template - context within configId */
                                /* Provides numeric Identifier to */
                                /* IPDR service specification for context of */
                                /* Session/config */
    UTF8String  schemaName; /* Reference to IPDR service specification */
    UTF8String  typeName;  /* Reference to IPDR service specification */
    FieldDescriptor fields<>; /* Fields in this Template */
};

```

Fields descriptions:

templateId

ID of a Template. A numeric Identifier assigned by the Exporter to the Template. It provides numeric Identifier to IPDR service specification for the context of Session/Template-Set-

Configuration. The templateId is not guaranteed to maintain the same value between different Sessions and different connections unless externally declared otherwise by the Exporter

schemaName*

Reference to IPDR service specification.

typeName*

Reference to IPDR service specification

fields

Arrays of Fields in this Template. See the above definition of the FieldDescriptor structure.

* **Notice:** Field schemaName and typeName form the external reference to the IPDR Service Definition whereas the Template would later be referenced only with templateId.

4.4.1 TEMPLATE DATA

The Exporter MUST send a TEMPLATE DATA Message after it gets a FLOW START Message from the Collector (before the establishment of a Session). The LSB of the TEMPLATE DATA flags Field, indicates whether the TEMPLATE DATA is negotiable (=1) or not (=0). The Template block identifies all the Templates that will be used over this Session. Sending DATA Messages that identify Template Ids not carried in the TEMPLATE DATA Messages is invalid, and SHOULD result in an ERROR Message being sent with decode error as the cause.

Following is the IDL fragment that defines the formal specification of the TEMPLATE DATA Message:

```
struct TemplateData {
    struct IPDRStreamingHeader header;
    short configId;                /* Identifies context of Template definitions */
                                   /* Changes in Template MUST use a different */
                                   /* configId (0 if unused) */
    char flags;                   /* LSB 0=NN 1=N; rest of bits Unused (reserved) */
    TemplateBlock templates<>;    /* Definitions of Templates supported */
};
```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

configId

The configId is the Identifier of a specific Template Set Configuration. It is changed whenever the Template Set Configuration is changed. It is set to 0 if unused.

flags

LSB 0 = Non Negotiable 1 = Negotiable; rest of bits Unused(reserved)

templates

templates is an array of templateBlock - Definitions of the supported Templates Set Configuration. If it is negotiable TEMPLATE DATA Message, this is the basis for the negotiation.

The TEMPLATE DATA Message has the messageId set to 0x10 in the header

4.4.2 MODIFY TEMPLATE

If the Collector receives a negotiable TEMPLATE DATA Message, the Collector MAY issue a MODIFY TEMPLATE request in order to alter the set of accounting records and their Fields which will be transferred.

The Exporter is not obligated to recognize any of the proposed changes, but indicates on a MODIFY TEMPLATE RESPONSE the set of Templates for that Session after applying any approved changes in the MODIFY TEMPLATE Message.

Following is the IDL fragment that defines the formal specification of the MODIFY TEMPLATE Message:

```
struct ModifyTemplate {  
    struct IPDRStreamingHeader header;  
    short configId;                /* Identifies context of Template definitions.*/  
                                   /* Changes in Template MUST use a different */  
                                   /* configId (0 if unused) */  
    char flags;                   /* Unused and reserved */  
    TemplateBlock changeTemplates<>; /* Definitions of Templates */  
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

configId

The configId is the Identifier of a specific Template Set Configuration. The Collector has to set it to be the same as received from the Exporter.

flags

Unused and reserved

changeTemplates

changeTemplates is an array of templateBlock – The suggested Template Set Configuration.

The MODIFY TEMPLATE Message has the messageId set to 0x1a in the header

4.4.3 MODIFY TEMPLATE RESPONSE

Upon receiving a MODIFY TEMPLATE Message, the Exporter is not obligated to recognize any of the proposed changes. It indicates on a MODIFY TEMPLATE

RESPONSE the set of Templates for that Session after applying any approved changes in the MODIFY TEMPLATE Message.

Following is the IDL fragment that defines the formal specification of the MODIFY TEMPLATE RESPONSE Message:

```
struct ModifyTemplateResponse {
    struct IPDRStreamingHeader header;
    short configId;                /* Identifies context of Template definitions.*/
                                   /* Changes in Template SHOULD use a different */
                                   /* configId (0 if unused) */
    char flags;                    /* Unused and reserved */
    TemplateBlock resultTemplates<>; /* Definitions of Templates - final results */
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

configId

The configId is the Identifier of a specific Template Set Configuration. It is changed whenever the Template Set Configuration is changed. It is set to 0 if unused.

flags

Unused and reserved.

resultTemplates

The resultTemplates is an array of templateBlocks – The determined Template Set Configuration for the following Active Session.

The MODIFY TEMPLATE RESPONSE Message has the messageId set to 0x1b in the header

4.4.4 FINAL TEMPLATE DATA ACK

The Collector sends a FINAL TEMPLATE DATA ACK to indicate that it is satisfied with the received TEMPLATE DATA and does not require a negotiation or to confirm that it got the MODIFY TEMPLATE DATA RESPONSE Message and it is ready to begin a Session-based delivery of accounting records.

Following is the IDL fragment that defines the formal specification of the FINAL TEMPLATE DATA ACK Message:

```
struct FinalTemplateDataAck {
    struct IPDRStreamingHeader header;
};
```

Field description:

header

The common header (see the [Common Header](#) section)

The FINAL TEMPLATE DATA ACK Message has the messageId set to 0x13 in the header

4.4.5 START NEGOTIATION

The Collector sends a START NEGOTIATION in order to request re-negotiation. The Exporter can either:

- accept the request and initiate a negotiation by stopping the Session or
- reject the request by sending the START NEGOTIATION REJECT Message.

Following is the IDL fragment that defines the formal specification of the START NEGOTIATION Message:

```
struct StartNegotiation {  
    struct IPDRStreamingHeader header;  
};
```

Field description:

header

The common header (see the [Common Header](#) section)

The START NEGOTIATION Message has the messageId set to 0x1d in the header

4.4.6 START NEGOTIATION REJECT

The Exporter sends a START NEGOTIATION REJECT in order reject the Collector request for renegotiating the Template Data

Following is the IDL fragment that defines the formal specification of the START NEGOTIATION REJECT Message:

```
struct StartNegotiationReject {  
    struct IPDRStreamingHeader header;  
};
```

Field description:

header

The common header (see the [Common Header](#) section)

The START NEGOTIATION REJECT Message has the messageId set to 0x1e in the header

4.5 Data

The DATA Message is the vehicle for all accounting records moved from the Exporter towards a Collector. The DATA Message uses the templating model described earlier to reduce the amount of redundant information passed in each Message. Specifically, by specifying in advance the Fields and their Types and their order, the encoded Data Record can pack all the values together according to the augmented XDR encoding.

A sequence number does data acknowledgement on a configurable window of outstanding DATA Messages determined by the Exporter. Because the underlying transport is reliable, this window may be set very large (e.g., hundreds or thousands of records) and does not grow and shrink over the Session.

The Exporter also specifies a minimal ack interval, to ensure timely acknowledgements when accounting record traffic volumes are low.

4.5.1 DATA

The Exporter sends DATA Messages to a Collector in the context of a Session. The Session ID is carried in the message header.

Following is the IDL fragment that defines the formal specification of the DATA Message:

```
struct Data {
    struct IPDRStreamingHeader header;
    short templateId;      /* a Template ID relative to the Session defined      */
                          /* in the header. The Fields in this Template were      */
                          /* reported at the beginning of the Session via          */
                          /* TEMPLATE DATA Messages                               */
    short configId;        /* (see above)                                           */
    char  flags;           /* currently just duplicate flag                         */
    long  sequenceNum;     /* Session relative sequence number of this record      */
    opaque dataRecord<>;   /* XDR encoded Fields based defined by templateId       */
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

templateId

The Template ID in the DATA Message refers to a Template Identifier previously sent by the Exporter on a TEMPLATE DATA Message. This describes the Fields, their order and Type and is used to define the augmented XDR encoding applied to construct the binary dataRecord.

configId

The configId is the Identifier of a specific Template Set Configuration. It is changed whenever the Template Set Configuration is changed. It is set to 0 if unused.

flags

The flags Field contains one defined flag, duplicate, represented by the flags LSB, which if set indicates that this record was previously sent to another Collector which failed to acknowledge its receipt.

sequenceNum

The sequence number is a relative to a Session. It increases by one for each accounting record sent. The number begins at 0 when a new Session Document is initiated. See the [SESSION START](#) section.

dataRecord

The dataRecord is an array of opaque - XDR encoded Fields based defined by templateId and configId.

The DATA Message has the messageId set to 0x20 in the header

4.5.2 DATA ACKNOWLEDGE

The DATA ACKNOWLEDGE Message is the indication of acknowledgement from a Collector that Data received and handled. It can be assumed by the Exporter to mean that it no longer has responsibility for these Data Records and may remove them from its transient buffer.

Following is the IDL fragment that defines the formal specification of the DATA ACKNOWLEDGE Message:

```
struct DataAcknowledge {
    struct IPDRStreamingHeader header;
    short configId;           /* (see above) */
    long sequenceNum;         /* Session relative sequence number of last record */
                             /* received in a series. */
};
```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

configId

The configId is the Identifier of a specific Template Set Configuration. It is changed whenever the Template Set Configuration is changed. It is set to 0 if unused.

sequenceNum

Session relative sequence number of the last record received and handled in a series.

The DATA ACKNOWLEDGE Message has the messageId set to 0x21 in the header

4.6 State Independent

The following Messages are available at any point in the state machine for IPDR/SP.

The two query Messages, GET SESSIONS and GET TEMPLATES, are always initiated by a Collector to identify the available streams of information emanating from an Exporter.

The KEEP ALIVE Message is sent in low traffic periods to ensure that both parties are still in communication. Note that some reliable transports such as TCP may have large periods of time before signaling connection loss, hence an application KEEP ALIVE allows for independent and low overhead monitoring of the connection between and Exporter and a Collector. SCTP's more

configurable timeout and failover mechanism may lessen the need for KEEP ALIVE, but this conclusion has not yet been reached.

4.6.1 GET SESSIONS

The GET SESSIONS Message is sent by a Collector to identify the streams of accounting records available on each stream provided by this Exporter.

Note that based on authorization mechanisms not defined in this specification, an Exporter MAY report different available Sessions to different Collectors.

Following is the IDL fragment that defines the formal specification of the GET SESSIONS Message:

```
struct GetSessions {  
    struct IPDRStreamingHeader header;  
    short requestId; /* numeric ID of request for Sessions */  
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

Note that the Session ID setting is ignored by the Exporter for this Message.

requestId

The requests Identifier - allows match up of response to request (it starts at zero and is incremented. It is unique per Session)

The GET SESSIONS Message has the messageId set to 0x14 in the header

4.6.2 GET SESSIONS RESPONSE

An Exporter MUST respond to a GET SESSION Message with GET SESSIONS RESPONSE Message that includes the list of Sessions available to this Collector. The Session lists, may then in turn, be used by a Collector to issue GET TEMPLATES Messages for specific Sessions of interest.

Following is the IDL fragment that defines the formal specification of the GET SESSIONS RESPONSE Message:

```
struct GetSessionsResponse {  
    struct IPDRStreamingHeader header;  
    short requestId; /* allows match up of response to request */  
    struct SessionBlock sessionBlocks <>; /* description of supported Sessions */  
};
```

Fields descriptions:

header

The common header (see the [Common Header](#) section)

Note that the SessionId setting is ignored by the Collector for this Message.

requestId

The requests Identifier - allows match up of response to request

sessionBlocks

An array of SessionBlocks - description of supported Sessions

Following is the IDL fragment that defines the formal specification of the SessionBlock structure:

```
struct SessionBlock {
    char sessionId;
    char reserved;
    UTF8String sessionName; /* Optional names and description for the Session */
    UTF8String sessionDescription;
    int ackTimeInterval; /* the maximum time between acks for this Session in sec. */
    int ackSequenceInterval; /* the maximum number of unacknowledged data items */
};
```

Fields descriptions:**sessionId**

sessionId is the Session Identifier.

reserved

reserved

sessionName

The name of the Session

sessionDescription

The description of the Session

ackTimeInterval

The maximum time between acknowledges from Collector (in second units)

ackSequenceInterval

The maximum number of unacknowledged records

The GET SESSIONS RESPONSE Message has the messageId set to 0x15 in the header

4.6.3 GET TEMPLATES

A Collector MAY identify the available Templates for a given Session context by issuing the GET TEMPLATES Message. Responses are linked to requests by setting the requestId.

Following is the IDL fragment that defines the formal specification of the GET TEMPLATES Message:

```
struct GetTemplates {
    struct IPDRStreamingHeader header;
    short requestId; /* numeric ID of request for Templates */
};
```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

Note that the Session ID setting is ignored by the Exporter for this Message.

requestId

The requests Identifier - allows match up of response to request (unique per Session)

The GET TEMPLATES Message has the messageId set to 0x16 in the header

4.6.4 GET TEMPLATES RESPONSE

An Exporter MUST respond with a GET TEMPLATES RESPONSE for any valid Session which the Collector is allowed to consume. If a sessionId is not available, an ERROR Message SHOULD be sent and the connection terminated.

Following is the IDL fragment that defines the formal specification of the GET TEMPLATES RESPONSE Message:

```
struct GetTemplatesResponse {  
    struct IPDRStreamingHeader header;  
    short requestId;  
    short configId;  
    TemplateBlock currentTemplates <>;  
};
```

Fields descriptions:**header**

The common header (see the [Common Header](#) section)

Note that the Session ID setting is ignored by the Exporter for this Message.

requestId

The requests Identifier - allows match up of response to request (unique per Session)

configId

The configId is the Identifier of a specific Template Set Configuration. It is changed whenever the Template Set Configuration is changed. It is set to 0 if unused.

currentTemplates

An array of templateBlock (including the enable/disable setting for each field) for the specific Session. (see the [Template](#) section).

The GET TEMPLATES RESPONSE Message has the messageId set to 0x17 in the header

4.6.5 KEEP ALIVE

In situations where there is low traffic, it is useful to have application controlled knowledge about the availability of the connection peer.

The KEEP ALIVE MUST be sent by either peer when no traffic has been sent by that party within the time requested on the CONNECT or CONNECT RESPONSE Message.

An implementation MAY send KEEP ALIVEs at any time.

Following is the IDL fragment that defines the formal specification of the KEEP ALIVE Message:

```
struct KeepAlive {  
    struct IPDRStreamingHeader header;  
};
```

Field description:

header

The common header (see the [Common Header](#) section)

Note that the Session ID setting is ignored by the Exporter for this Message.

The KEEP ALIVE Message has the messageId set to 0x40 in the header

5 Underlying Transport

TCP may be used as a transport layer for IPDR/SP. If TCP is used an implementation MUST follow these rules:

- Either the Exporter or Collector MAY initiate TCP over a specific TCP port.
- The initiator of a connection is responsible for reestablishing a connection in case of a failure.
- Messages are written as a stream of bytes into a TCP connection, the size of a Message is

6 Service Discovery

6.1 UDP Protocol

Since the Streaming Protocol may evolve in the future and it MAY run over different transport layers, a transport neutral version negotiation mechanism running over UDP is defined. An Exporter or a Collector SHOULD implement version discovery and negotiation as defined herein. Either party MAY inquire about the Streaming Protocol version and transport layer support by sending a UDP packet on an agreed UDP port. If the receiving party implements version discovery and negotiation, it MUST respond to this request with a UDP packet carrying the protocol version, the transport type and the port number used for the specific transport.

The inquiring party (either Collector or Exporter) sends the following Message to query the other party's protocol support. The following IDL fragment defines the formal specification of this Message

```
struct VersionRequest {
    int requesterAddress; /* IPv4 address of the version request initiator */
    int requesterBootTime; /* boot time of the version request initiator (in seconds*/
                          /* from epoch time) */
    char [4] msg; /* MUST be 'CRAN' for version 1 and 'IPDR' for version 2 */
}
```

Fields descriptions:

requesterAddress

IPv4 address of the version request initiator

requesterBootTime

Boot time of the version request initiator (in seconds from epoch time).

msg

MUST be 'CRAN' for version 1 and 'IPDR' for version 2

The receiving party SHOULD respond with the following Message that describes the protocols that it supports.

Following is the IDL fragment that defines the formal specification of this Message.

```
struct VersionResponse {
    ProtocolInfo defaultProtocol;
    ProtocolInfo [] additionalProtocols;
}

struct ProtocolInfo {
    int transportType;
    int protocolVersion;
    short portNumber;
    short reserved;
}
```

Fields descriptions:

defaultProtocol

The default protocol

additionalProtocols

Additional supported protocols

transportType

TransportTypeId of the transport protocol

protocolVersion

IPDR Streaming Protocol version supported over transport

portNumber

Transport protocol port used

reserved

Reserved/unused

6.2 Capability Files

In addition to the UDP based mechanism described above, IPDR's existing transport mechanisms utilize an XML based capability file to define the supported protocols of an IPDR Transmitter (an Exporter in the case of IPDR/SP). The basic mechanism is described in the associated IPDR document [1].

This section defines the extensions to the IPDR Capability file structure to describe an IPDR/SP capable IPDR Transmitter.

The capability mechanism allows IPDR consumers to determine the formats and protocols supported by an IPDR Transmitter. IPDR Capability files are intended to be extended to address other IPDR transfer mechanisms in a uniform, extensible manner.

The "supportedProtocolItem" XML element is the basic unit for defining a means of communicating with an IPDR Transmitter (an Exporter). IPDR/SP represents one means of transmission and as such the information for IPDR/SP is contained in one of these elements.

The following example shows the structure for XML capability files describing the IPDR/SP. It mirrors the information content of the UDP based "versioning", described previously. The items shown in **bold** indicate specific details related to IPDR/SP. The non-bold items are directly taken from the existing Capabilities specification.

```
<CapabilityRsp>
  <supportedProtocolList>
    <supportedProtocolItem version="2"
      protocolMapping="Streaming"
      encoding="XDR">
      <primitiveList>
        <primitiveItem>Push</primitiveItem>
      </primitiveList>
    </supportedProtocolItem>
  </supportedProtocolList>
</CapabilityRsp>
```

```
<extension>
  <exporterAddress>192.168.1.22</exporterAddress>
  <transportType>TCP</transportType>
  <portNumber>666</portNumber>
</extension>
</supportedProtocolItem>
</supportedProtocolList>
</CapabilityRsp>
```

The information content in the UDP model “versioning” is:

- a version number – the version shall be 2 for IPDR/SP.
- an Exporter address – a fully qualified domain name or IP address.
- a transport type – for now this is the string “TCP”. SCTP or BEEP or other protocols may be mapped later.
- a port number – the integer port number where the Exporter may be located.

This is sufficient for a Collector to determine its ability to communicate with an Exporter. Additional information can be determined using the Messages defined in IPDR/SP itself.

An Exporter MAY thus advertise a URI which locates this capabilities file and from the information contained within a Collector can determine its options for communicating with that system.

7 Security

The IPDR/SP can be viewed as an application running over a reliable transport layer, such as TCP or SCTP. The protocol is end-to-end in the sense that the Messages are communicated between Exporters and Collectors identified by the host address and the transport protocol port number. Before any Sessions can be initiated, a set of Collectors' addresses **SHOULD** be provisioned on the Exporter. Similarly, a Collector maintains a list of Exporters' addresses with which it communicates. The provisioning is typically carried out securely using a network management system; in this way, the end-points can be authenticated and authorized. As this scheme is static, without additional security protections the protocol is vulnerable to attacks such as address spoofing.

IPDR/SP does not offer strong security facilities; therefore, it cannot ensure confidentiality and integrity or non-repudiation of its Messages. It is strongly recommended that administrators evaluate their deployment configurations and implement appropriate security policies. For example, if the IPDR/SP is deployed over a local area network or a dedicated connection that ensures security, no additional security services or procedures may be required; however, if Exporters and Collectors are connected through the Internet, lower layer security services **SHOULD** be used.

To achieve strong security, lower layer security services are strongly recommended. The lower layer security services are transparent to IPDR/SP. Security mechanisms may be provided at the IP layer using IPSec [12], or it may be implemented for transport layer using TLS [13]. The provisioning of the lower layer security services is out of the scope of this document.

8 Complete IPDR/SP IDL Definition

The following listing shows the **normative** IDL specification for constructing IPDR/SP Messages.

```

/*****
ipdr_streaming.idl - IPDR/SP XDR IDL definition

This XDR IDL is used to describe all Messages and structures used by
IPDR/SP.

The IDL is organized in the following logical sub-sections:

- Header structure definition - common to all transport protocol messages.
- Enumeration of Message IDs.
- Structure of messages used in transport protocol.
- Data structures and enumerations used within Messages.
- Messages and structures used in UDP version discovery and negotiation.

Note:
Comments within the text are not normative but rather only used to provide
minimal documentation. The complete description is available within the
IPDR/SP specification main body.
*****/

/*****
Header structure definition:
*****/

struct IPDRStreamingHeader {
    char version;          /* version of protocol, for this version, set to 2          */
    char messageId;        /* the ID of this Message (see MessageIds)                      */
    char sessionId;        /* the ID of this Session or 0 if not Session-specific          */
    char messageFlags;     /* reserved/unused and set to 0                                 */
    int messageLen;        /* length in bytes of Message including header                  */
};

/*****
Enumeration of Message IDs:
*****/

enum MessageIds {
    CONNECT = 0x05,
    CONNECT_RESPONSE = 0x06,
    DISCONNECT = 0x07,
    FLOW_START = 0x01,
    FLOW_STOP = 0x03,
    SESSION_START = 0x08,
    SESSION_STOP = 0x09,
    KEEP_ALIVE = 0x40,
    TEMPLATE_DATA = 0x10,
    MODIFY_TEMPLATE = 0x1a,
    MODIFY_TEMPLATE_RESPONSE = 0x1b,
    FINAL_TEMPLATE_DATA_ACK = 0x13,
    START_NEGOTIATION = 0x1d,
    START_NEGOTIATION_REJECT = 0x1e,
    GET_SESSIONS = 0x14,
    GET_SESSIONS_RESPONSE = 0x15,
    GET_TEMPLATES = 0x16,
    GET_TEMPLATES_RESPONSE = 0x17,
    DATA = 0x20,
    DATA_ACK = 0x21,
    ERROR = 0x23
};

```

```

/*****
Structure of messages used in transport protocol:
*****/

struct Connect {
    struct IPDRStreamingHeader header;
    int initiatorAddress; /* IPv4 IP address of the initiator (Collector/Exporter */
    short initiatorPort; /* The transport protocol port number of the initiator */
    int capabilities; /* an array of capability bits for capability negotiation */
    int keepAliveInterval /* the maximum delay between some indication from remote, */
                        /* expressed in seconds */
    UTF8String vendorId; /* The vendor ID of the initiator (Exporter/Collector) */
};

struct ConnectResponse {
    struct IPDRStreamingHeader header;
    int capabilities; /* an array of capability bits for capability negotiation */
    int keepAliveInterval; /* the maximum delay between some indication from remote */
                        /* It is expressed in seconds. */
    UTF8String vendorId; /* the vendor ID of the responder (Exporter/Collector) */
};

struct Disconnect {
    struct IPDRStreamingHeader header;
};

struct Error {
    struct IPDRStreamingHeader header;
    int timeStamp; /* time of error (in seconds from epoch time) */
    short errorCode; /* The error code field consists of two parts: */
                    /* Session oriented flag: it is a one bit flag. */
                    /* It is the MSB of the errorCode field. It */
                    /* indicates whether the error is specific for */
                    /* the session (=1) or it is a general error and */
                    /* thus it is not specific to the session (=0). */
                    /* The code ID: The rest 15 LSBs of the */
                    /* errorCode field, specifies the error code ID */
                    /* (0 - 32767). Values of 0-255 are reserved for */
                    /* standard error codes. */
                    /* 0 = keepalive expired */
                    /* 1 = Message invalid for capabilities */
                    /* 2 = Message invalid for state */
                    /* 3 = Message decode error */
                    /* 4 = process terminating */
                    /* Values > 255 may be used for vendor specific */
                    /* error codes */
    UTF8String description;
};

struct FlowStart {
    struct IPDRStreamingHeader header;
};

struct SessionStart {
    struct IPDRStreamingHeader header;
    int exporterBootTime; /* boot time of Exporter(in seconds from epoch time)*/
    long firstRecordSequenceNumber; /* first sequence number to be expected */
    long droppedRecordCount /* number of records dropped in gap situations */
    boolean primary; /* indication that the Collector is the primary */
    int ackTimeInterval; /* the maximum time between acks from Collector */
                        /* (in second units) */
    int ackSequenceInterval; /* the maximum number of unacknowledged records */
    char documentId[16]; /* the UUID associated with the info being sent */
                        /* in this Session */
};

struct FlowStop {
    struct IPDRStreamingHeader header;
};

```

```

    short reasonCode;          /* values of 0-255 are reserved for standard */
                                /* reason codes. Values > 255 may be used for */
                                /* vendor specific codes. */
                                /* 0 = normal process termination */
                                /* 1 = termination due to process error */
    UTF8String reasonInfo;
};

struct SessionStop {
    struct IPDRStreamingHeader header;
    short reasonCode;          /* values of 0-255 are reserved for standard */
                                /* reason codes. Values of > 255 may be used for */
                                /* vendor-specific codes. */
                                /* 0 = end of data for session */
                                /* 1 = handing off to higher priority Collector */
                                /* 2 = Exporter terminating */
                                /* 3 = congestion detected */
                                /* 4 = renegotiation is required */
                                /* 5 = start negotiation acknowledge */
                                /* 6 = end of IPDRDoc */
                                /* 7 = Template data was updated */
    UTF8String reasonInfo;
};

struct TemplateData {
    struct IPDRStreamingHeader header;
    short configId;            /* Identifies context of Template definitions */
                                /* Changes in Template MUST use a different */
                                /* configId (0 if unused) */
    char flags;                /* LSB 0=NN 1=N; rest of bits Unused (reserved) */
    TemplateBlock templates<>; /* Definitions of Templates supported */
};

struct ModifyTemplate {
    struct IPDRStreamingHeader header;
    short configId;            /* Identifies context of Template definitions. */
                                /* Changes in Template MUST use a different */
                                /* configId (0 if unused) */
    char flags;                /* Unused and reserved */
    TemplateBlock changeTemplates<>; /* Definitions of Templates */
};

struct ModifyTemplateResponse {
    struct IPDRStreamingHeader header;
    short configId;            /* Identifies context of Template definitions. */
                                /* Changes in Template SHOULD use a different */
                                /* configId (0 if unused) */
    char flags;                /* Unused and reserved */
    TemplateBlock resultTemplates<>; /* Definitions of Templates - final results */
};

struct FinalTemplateDataAck {
    struct IPDRStreamingHeader header;
};

struct StartNegotiation {
    struct IPDRStreamingHeader header;
};

struct StartNegotiationReject {
    struct IPDRStreamingHeader header;
};

struct Data {
    struct IPDRStreamingHeader header;
    short templateId;          /* a Template ID relative to the Session defined */
                                /* in the header. The Fields in this Template were */
                                /* reported at the beginning of the Session via */
                                /* TEMPLATE DATA Messages */
    short configId;            /* (see above) */
    char flags;                /* currently just duplicate flag */
    long sequenceNum;          /* Session relative sequence number of this record */
    opaque dataRecord<>;       /* XDR encoded Fields based defined by templateId */
};

```

```

struct DataAcknowledge {
    struct IPDRStreamingHeader header;
    short  configId;          /* (see above) */
    long   sequenceNum;       /* Session relative sequence number of last record */
                                /* received in a series. */
};

struct GetSessions {
    struct IPDRStreamingHeader header;
    short requestId;          /* numeric ID of request for Sessions */
};

struct GetSessionsResponse {
    struct IPDRStreamingHeader header;
    short requestId;          /* allows match up of response to request */
    struct SessionBlock sessionBlocks <>; /* description of supported Sessions */
};

struct GetTemplates {
    struct IPDRStreamingHeader header;
    short requestId;          /* numeric ID of request for Templates */
};

struct GetTemplatesResponse {
    struct IPDRStreamingHeader header;
    short requestId;          /* allows match up of response to request */
    short configId;           /* (see above) */
    TemplateBlock currentTemplates <>; /* supported active Templates */
};

struct KeepAlive {
    struct IPDRStreamingHeader header;
};

/*****
Data structures and enumerations used within Messages:
*****/

struct FieldDescriptor{
    int      typeId;          /* ID of Field Type */
    int      fieldId;         /* unqualified Field code that can be used */
                                /* as alternative accessor handles to Fields */
    UTF8String fieldName;     /* Note that Field names are to be qualified */
                                /* with the Namespace name, as an example: */
                                /* http://www.ipdr.org/namespace:movieId */
                                /* The namespace MUST match one of those */
                                /* targeted by the schema or schema imports */
    boolean  isEnabled       /* true=enabled Field false=disabled Field */
};

struct TemplateBlock{
    short     templateId;     /* ID of Template - context within configId */
                                /* Provides numeric Identifier to */
                                /* IPDR service specification for context of */
                                /* Session/config */
    UTF8String schemaName;    /* Reference to IPDR service specification */
    UTF8String typeName;      /* Reference to IPDR service specification */
    FieldDescriptor fields<>; /* Fields in this Template */
};

struct SessionBlock {
    char sessionId;
    char reserved;
    UTF8String sessionName;   /* Optional names and description for the Session */
    UTF8String sessionDescription;
    int  ackTimeInterval;     /* the maximum time between acks for this Session in sec. */
    int  ackSequenceInterval; /* the maximum number of unacknowledged data items */
};

```

```

    } ;

    enum Capabilities {
        STRUCTURES = 0x01,
        MULTISESSION = 0x02,
        TEMPLATE_NEGOTIATION = 0x04
    }

    /*****
    Messages and structures used in UDP version discovery and negotiation:
    *****/

    struct VersionRequest {
        int requesterAddress; /* IPv4 address of the version request initiator */
        int requesterBootTime; /* boot time of the version request initiator (in seconds*/
                                /* from epoch time) */
        char [4] msg; /* MUST be 'CRAN' for version 1 and 'IPDR' for version 2 */
    }

    struct VersionResponse {
        ProtocolInfo defaultProtocol;
        ProtocolInfo [] additionalProtocols;
    }

    struct ProtocolInfo {
        int transportType; /* TransportTypeId of the transport protocol */
        int protocolVersion; /* IPDR Streaming Protocol version supported over transport */
        short portNumber; /* Transport protocol port used */
        short reserved; /* Reserved/unused */
    }

    enum TransportTypeIds {
        TCP = 1,
        SCTP = 2
    }

```