Open Pluggable Edge Services
Internet-Draft
Expires: May 17, 2004

A. Rousskov
The Measurement Factory

November 17, 2003

OPES Callout Protocol Core
draft-ietf-opes-ocp-core-03

Status of this Memo

Copyright Notice

Abstract

This document specifies Open Pluggable Edge Services (OPES) Callout Protocol (OCP). OCP is an application-agnostic protocol that facilitates exchange of application messages between an OPES processor and a callout server, for the purpose of adaptation of application messages at the callout server.

Table of Contents

1. Introduction

The Open Pluggable Edge Services (OPES) architecture [I-D.ietf-opes-architecture], enables cooperative application services (OPES services) between a data provider, a data consumer, and zero or more OPES processors. The application services under consideration analyze and possibly transform application-level messages exchanged between the data provider and the data consumer.

The OPES processor can delegate the responsibility of service execution by communicating with remote callout servers. As described in [I-D.ietf-opes-protocol-reqs], an OPES processor communicates with and invokes services on a callout server by using a callout protocol. This document specifies the core of such a protocol.

OCP Core specification documents general, application-independent protocol mechanisms. A separate series of documents describe application-specific aspects of OCP. For example, "HTTP adaptation with OPES" [I-D.ietf-opes-http] describes, in part, how HTTP messages and HTTP meta-information can be communicated over OCP.

1.1 Scope

As an application proxy, OPES processor proxies a single application protocol or converts from one application protocol to another. At the same time, OPES processor may be an OCP client, using OCP to facilitate adaptation of proxied messages at callout servers. It is therefore natural to assume that OPES processor takes application messages being proxied, passes them over OCP to callout servers, and then puts the adaptation results back on the wire. However, such an assumption implies that OCP is applied directly to application messages that OPES processor is proxing, which may not be the case.

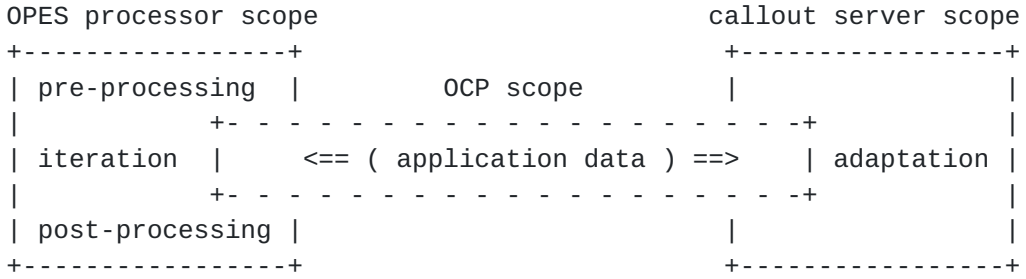

Figure 1

OPES processor may preprocess (or postprocess) proxied application messages before (or after) they are adapted at callout servers. For example, a processor may take an HTTP response being proxied and pass

it as is, along with metadata about the corresponding HTTP connection. Another processor may take an HTTP response, extract its body, and pass that body, along with the content-encoding metadata. Moreover, to perform adaptation, OPES processor may execute several callout services, iterating over several callout servers. Such preprocessing, postprocessing, and iterations make it impossible to rely on any specific relationship between application messages being proxied and application messages being sent to a callout service. Similarly, specific adaptation actions at the callout server are outside of OCP Core scope.

This specification does not define or require any specific relationship among application messages being proxied by the OPES processor and application messages being exchanged with callout servers via OCP. OPES processor usually provides some mapping among these application messages, but processor's specific actions are beyond OCP scope. In other words, this specification is not concerned with the OPES processor role as an application proxy, or as an iterator of callout services. The scope of OCP Core is communication between a single OPES processor and a single callout server.

Furthermore, an OPES processor is at liberty to choose which proxied application messages or information about them to send over OCP. All proxied messages on all proxied connections (if connections are defined for a given application protocol), everything on some connections, selected proxied messages, or nothing might be sent over OCP to callout servers. OPES processor and callout server state related to proxied protocols can be relayed over OCP as application message metadata.

1.2 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. When used with the normative meanings, these keywords will be all uppercase. Occurrences of these words in lowercase comprise normal prose usage, with no normative implications.

OPES processor works with messages from application protocols and may relay information about those application messages to a callout server. OCP is also an application protocol. Thus, protocol elements like "message", "connection", or "transaction" exist in OCP and other application protocols. In this specification, all references to elements from application protocols other than OCP are used with an explicit "application" qualifier. References without the "application" qualifier, refer to OCP elements.

OCP message: OCP message is a basic unit of communication between an OPES processor and a callout server. Message is a sequence of octets formatted according to syntax rules (Section 3.1). Message semantics is defined in Section 11.

application message: An entity defined by OPES processor and callout server negotiation. Usually, the negotiated definition would match the definition from an application protocol (e.g., [RFC2616] definition of an HTTP message, including headers, and body).

application message data: An opaque sequence of octets representing complete or partial application message. OCP Core does not distinguish application message structure (if any). Application message data may be empty.

data: Same as application message data.

original Referring to application message flowing from the OPES processor to a callout server.

adapted Referring to application message flowing from an OPES callout server to the OPES processor.

adaptation: Any kind of access by a callout server, including modification and copying. For example, translating or logging an SMTP message is adaptation of that application message.

agent: Actor for a given communication protocol. OPES processor and callout server are OCP agents. An agent can be referred to as a sender or receiver, depending on its actions in a particular context.

immediate: Performing the specified action before reacting to new incoming messages or sending any new messages unrelated to the specified action.

OCP extension: A specification extending or adjusting this document to cover an application protocol (a.k.a., application binding, e.g., [I-D.ietf-opes-http]), new OCP functionality (e.g., transport encryption and authentication), and/or new OCP Core version.

2. Overall Operation

OPES processor may use OPES callout protocol (OCP) to communicate with callout servers. Adaptation using callout services is sometimes called a "bump in the wire" architecture.

2.1 Initialization

OPES processor establishes transport connections with callout servers for the purpose of exchanging application messages with the callout server(s) using OCP. After a transport-layer connection (usually TCP/IP) is established, communicating OCP agents exchange Connection Start (CS) messages. Next, OCP features can be negotiated between the processor and the callout server (see Section 6). For example, OCP agents have to agree on transport encryption and application message definition. When enough settings are negotiated, OCP agents may start exchanging application messages.

2.2 Original Dataflow

When OPES processor wants to adapt an application message, the OPES processor sends a Transaction Start (TS) message to initiate an OCP transaction dedicated to that application message. The processor then sends an Application Message Start (AMS) message to prepare the callout server for application data that will follow. Once application message scope is established, application data can be sent to the callout server, using Data Use Mine (DUM) and related OCP message(s). All these messages correspond to the original dataflow.

2.3 Adapted Dataflow

The callout server receives data and metadata sent by the OPES processor (original data flow). The callout server analyses metadata and adapts data as it comes in. The server usually builds its version of metadata and responds to OPES processor with an Application Message Start (AMS) message. Adapted application message data can be sent next, using Data Use Mine (DUM) OCP message(s). The application message is then announced to be "completed" or "closed" using an Application Message End (AME) message. The transaction may be closed using a Transaction End (TE) message as well. All these messages correspond to adapted data flow.

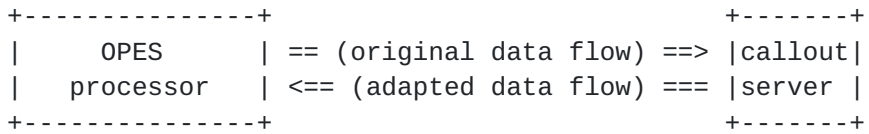

Figure 2

Depending on the negotiated application message definition, it may be possible or even required for callout server to respond with more than one application message within the same transaction. In other words, the callout server may adapt a single original application message into multiple application messages. Each application message sent by the callout server is individually identified by an "am-id" parameter (Section 10.9) and can be sent independently from other application messages within the same transaction (this allows for logical- and transport-level interleaving of OCP messages related to different application messages).

The OPES processor receives the application message sent by the callout server. Other OPES processor actions specific to the application message received are out of this specification scope.

2.4 Termination

Either OCP agent can terminate application message delivery, transaction, or connection by sending an appropriate OCP message. Usually, the callout server terminates application message delivery and the transaction. Abnormal terminations at arbitrary times are supported. The termination message includes a result description.

2.5 Exchange Patterns

In addition to messages carrying application data, OCP agents may also exchange messages related to their configuration, state, transport connections, application connections, etc. A callout server may remove itself from the application message processing loop. A single OPES processor can communicate with many callout servers and vice versa. Though many OCP exchange patterns do not follow a classic client-server model, it is possible to think of an OPES processor as an "OCP client" and of a callout server as an "OCP server". The OPES architecture document [I-D.ietf-opes-architecture] describes configuration possibilities.

The following informal rules illustrate relationships between connections, transactions, OCP messages, and application messages:

- An OCP agent may communicate with multiple OCP agents. Communication with multiple OCP agents is outside of this specification scope.

- An OPES processor may have multiple concurrent OCP connections to a callout server. Communication over multiple OCP connections is outside of this specification scope.

- A connection may carry multiple concurrent transactions. A transaction is always associated with a single connection (i.e., a transaction cannot span multiple concurrent connections).

- A connection may carry at most one message at a time, including control messages and transaction-related messages. A message is always associated with a single connection (i.e., a message cannot span multiple concurrent connections).

- A transaction is a sequence of messages related to application of a given set of callout services to a single application message. A sequence of transaction messages from an OPES processor to a callout server is called original flow. A sequence of transaction messages from a callout server to an OPES processor is called adapted flow. The two flows may overlap in time.

- A transaction is always associated with a single (original) application message. Adapted flow may transfer information about multiple (adapted) application messages.

- An application message (adapted or original) is transferred using a sequence of OCP messages.

2.6 Timeouts

OCP violations, external dependencies, and other factors may lead to states when an OCP agent is not receiving required messages from the other OCP agent. This specification does not document any specific mechanisms to address such situations. In the absence of any extension mechanism, OCP agents MUST implement timeouts for OCP operations. That is, any OCP connection, negotiation, transaction, etc. that is not making progress must be eventually forcefully terminated. This rule covers both dead- and livelock situations.

In their implementation, OCP agents MAY rely on transport-level or other external timeouts if such external timeouts are guaranteed to happen for a given OCP operation. Depending on the OCP operation, an agent MAY benefit from pinging the other side using a Progress Query (PQ) message before terminating an OCP transaction or connection. The latter is especially useful for adaptations that may take a long time at the callout server before producing any adapted data.

2.7 OCP Environment

OCP communication is assumed to usually take place over TCP/IP connections on the Internet (though no default TCP port is assigned to OCP). This does not preclude OCP from being implemented on top of

any other transport protocol, on any other network. High-level transport protocols such as BEEP [RFC3080] may be used. OCP only presumes a reliable connection-oriented transport; any protocol that provides such guarantees can be used; the mapping of OCP message structures onto the transport data units of the protocol in question is outside the scope of this specification.

OCP is application-agnostic. OCP messages can carry application-specific information as payload or as application-specific message parameters.

OCP overhead in terms of extra traffic on the wire is about 100-200 octets per small application message. Pipelining, preview, and data preservation optimizations and as-is encapsulation of application data make fast exchange of application messages possible.

3. Messages

As defined in Section 1.2, an OCP message is a basic unit of communication between an OPES processor and a callout server. A message is a sequence of octets formatted according to syntax rules (Section 3.1). Message semantics is defined in Section 11. Messages are transmitted on top of OCP transport.

OCP messages deal with transport and transaction management as well as application data exchange between a single OPES processor and a single callout server. Some messages can only be emitted by an OPES processor; some only by a callout server; some can be emitted by both OPES processor and callout server. Some messages require responses (one could call such messages "requests"); some can only be used in response to other messages ("responses"); some may be sent without solicitation and/or may not require a response.

3.1 Message Format

An OCP message consists of a message name followed by optional parameters and payload. The exact message syntax is defined by the following Augmented Backus-Naur Form (ABNF) [RFC2234]:

```
message = name [SP anonym-parameters]
          [CRLF named-parameters CRLF]
          [CRLF payload CRLF]
          ";" CRLF

anonym-parameters = value *(SP value)               ; space-separated
named-parameters  = named-value *(CRLF named-value) ; CRLF-separated
list-items        = value *("," value)              ; comma-separated

payload = data

named-value = name ":" SP value

value     = structure / list / atom
structure = "{" [anonym-parameters] [CRLF named-parameters CRLF] "}"
list      = "(" [ list-items ] ")"
atom      = bare-value / quoted-value

name = ALPHA *safe-OCTET
bare-value = 1*safe-OCTET
quoted-value = DQUOTE data DQUOTE
data = size ":" <n>OCTET                  ; n == size

safe-OCTET = ALPHA / DIGIT / "-" / "_"
size = dec-number                         ; 0-2147483647
```

```
dec-number = 1*DIGIT                    ; no leading zeros or signs
```

Figure 3

Several normative rules accompany the above ABNF:

- There is no "implied linear space" (LWS) rule. LWS rules are common to MIME-based grammars, but are not used here. The whitespace syntax is restricted to what is explicitly allowed by the above ABNF.
- All protocol elements are case sensitive unless specified otherwise. In particular, message names and parameter names are case sensitive.
- Sizes are interpreted as decimal values and cannot have leading zeros.
- Sizes do not exceed 2147483647.
- The size attribute in a quoted-value encoding specifies the exact number of OCTETs following the column (':') separator. If size OCTETs are not followed by a quote ('"') character, the encoding is syntactically invalid.
- Empty quoted-values are encoded as a 4-OCTET sequence "0:".
- Any bare-value MAY be encoded as a quoted-value. A quoted-value MUST be interpreted after the encoding is removed. For example, number 1234 can be encoded as four OCTETs 1234 or as eight OCTETs "4:1234", yielding exactly the same meaning.
- By default, agents MUST interpret all values as having UTF-8 encoding. Note that ASCII is a UTF-8 subset, and that the syntax prohibits non-ASCII characters outside of the "data" element.

Messages violating formatting rules are, by definition, invalid. See Section 5 for rules on processing invalid messages.

3.2 Message Rendering

OCP message samples in this specification and its application bindings may not be typeset to depict minor syntactical details of OCP message format. Specifically, SP and CRLF characters are not shown explicitly. No rendering of an OCP message can be used to infer message format. The message format definition above is the only normative source for all implementations.

On occasion, an OCP message line exceeds text width allowed by this specification format. A backslash ("\"), a "soft linebreak" character is used to emphasize a protocol-violating presentation-only linebreak. Bare backslashes are prohibited by OCP syntax. Similarly, a "\r\n" string is sometimes used to emphasize the presence of a CRLF sequence, usually before OCP message payload. Normally, visible end of line corresponds to the CRLF sequence on the wire.

The next section (Section 3.3) contains specific OCP message examples, some of which illustrate the above rendering techniques.

3.3 Message Examples

OCP syntax provides for compact representation of short control messages and required parameters while allowing for parameter extensions. Below are examples of short control messages. The required CRLF sequence at the end of each line is not shown explicitly (see Section 3.2).

```
PQ;
TS 1 2;
DWM 22 1;
DWP 22 1;
x-doit "5:xyzzy";
```

Figure 4

The above examples contain atomic anonymous parameter values such as number and string constants. OCP messages sometimes use more complicated parameters such as item lists or structures with named values. As both messages below illustrate, structures and lists can be nested:

```
NO ({"28:http://iana.org/opes/ocp/TLS"});
NO ({\
"38:http://iana.org/opes/ocp/HTTP/response"
Optional-Parts: (request-header)
},{\
"38:http://iana.org/opes/ocp/HTTP/response"
Optional-Parts: (request-header,request-body)
Transfer-Encodings: (chunked)
});
```

Figure 5

Optional parameters and extensions are possible using named parameters approach as illustrated by the following example. The DWM (Section 11.16) message in the example has three anonymous parameters and two named parameters (the last one being an extension).

```
DWM 1 3 12345
size-request: 16384
x-need-info: "26:twenty six octet extension";
```

Figure 6

Finally, any message may have a payload part. For example, the Data Use Mine (DUM) message below carries 8865 bytes of raw data.

```
DUM 1 3 0 8865
modp: 75
\r\n
8865:... 8865 octets of raw data ...;
```

Figure 7

3.4 Message Names

Most OCP messages defined in this specification have short names, formed by abbreviating or compressing a longer but human-friendlier message title. Short names without a central registration system (like this specification or IANA registry) are likely to cause conflicts. Informal protocol extensions should avoid short names. To emphasize what is already defined by message syntax, implementations cannot assume that all message names are very short.

4. Transactions

OCP transaction is a logical sequence of OCP messages processing a single original application message. The result of the processing may be zero or more application messages, adapted from the original. A typical transaction consists of two message flows: a flow from the OPES processor to the callout server (sending original application message) and a flow from the callout server to the OPES processor (sending adapted application messages). The number of application messages produced by the callout server and whether the callout server actually modifies original application message may depend on the requested callout service and other factors. The OPES processor or the callout server can terminate the transaction by sending a corresponding message to the other side.

A OCP transaction starts with a Transaction Start (TS) message sent by the OPES processor. A transaction ends with the first Transaction End (TE) message sent or received, explicit or implied, which can be sent by either side. Zero or more OCP messages associated with the transaction can be exchanged in between. The figure below illustrates possible message sequence (prefix "P" stands for OCP Client, OPES processor; prefix "S" stands for OCP callout server). Some message details are omitted.

```
P: TS 10;
P: AMS 10 1;
   ... processor sending application data to the callout server
S: AMS 10 2;
   ... callout server sending application data to the processor
   ... processor sending application data to the callout server
P: AME 10 1 result;
S: AME 10 2 result;
P: TE 10 result;
```

Figure 8

5. Invalid input

This specification contains many criteria for valid OCP messages and their parts, including syntax rules, semantics requirements, and relationship to agents state. "Invalid input" in this context means messages or message parts that violate at least one of the normative rules of this specification. A message with an invalid part is, by definition, invalid. If an OCP agent resources are exhausted while parsing or interpreting a message, the agent MUST treat the corresponding OCP message as invalid.

Unless explicitly allowed otherwise, OCP agents MUST terminate the transaction if they receive an invalid message with transaction scope and MUST terminate the connection if they receive an invalid message with a connection scope. Such terminations MUST carry the result status code of 400 and MAY carry termination cause information in result status reason (see Section 10.11). If an OCP agent is unable to determine the scope of an invalid message it received, the agent MUST treat the message as having connection scope.

OCP usually deals with optional but invasive application message manipulations where correctness ought to be valued above robustness. For example, a failure to insert or remove certain optional web page content is usually far less disturbing than corrupting the host page while performing that insertion or removal. Most OPES adaptations are high-level in nature, which makes it impossible to automatically assess correctness of operations, especially if "robustness guesses" are involved.

6. Negotiation

The negotiation mechanism allows OCP client and server to agree on mutually acceptable set of features, including optional and application-specific behavior as well as OCP extensions. For example, transport encryption, data format, and support for a new message can be negotiated. Negotiation implies intent for a behavioral change. For a related mechanism allowing an agent to query capabilities of its counterpart without changing counterpart's behavior, see Ability Query (AQ) and Ability Answer (AA) definitions.

Most negotiations require at least one round trip time delay. In rare cases when other side's response is not required immediately, negotiation delay can be eliminated.

A detected violation of negotiation rules leads to OCP connection termination. This design reduces the number of negotiation scenarios resulting in a deadlock when one of the agents is not compliant.

Two core negotiation primitives are supported: negotiation offer and negotiation response. A Negotiation Offer (NO) message allows an agent to specify a set of features from which the responder has to select exactly one feature it prefers. The selection is sent using a Negotiation Response (NR) message. If the response is positive both sides assume that the selected feature is in effect. If the response is negative, no behavioral changes are assumed. In either case, further offers may follow.

Negotiating OCP agents have to take into account prior negotiated (i.e., already enabled) features. OCP agents MUST NOT make and MUST reject offers that would lead to a conflict with already negotiated features. For example, an agent cannot offer an HTTP application profile for a connection that already has SMTP application profile enabled because there would be no way to resolve the conflict for a given transaction. Similarly, once TLSv1 connection encryption is negotiated, an agent must not offer and must reject offers for SSLv2 connection encryption.

Negotiation Offer (NO) messages may be sent by either agent. OCP extensions documenting negotiation MAY assign initiator role to one of the agents, depending on the feature being negotiated. For example, negotiation of transport security feature should be initiated by OPES processors to avoid situations where both agents wait for each other to make an offer.

Since either agent may make an offer, two "concurrent" offers may be made at the same time, by the two communicating agents. Unmanaged concurrent offers may lead to a negotiation deadlock. By giving OPES

processor a priority, offer handling rules (Section 11.17) ensure that only one offer per transport connection is honored at a time, and the other concurrent offers are ignored by both agents.

6.1 Negotiation Phase

A Negotiation Phase is a mechanism to ensure that both agents have a chance to negotiate all features they require before proceeding further. Negotiation Phases have OCP connection scope and do not overlap. For each OCP agent, Negotiation Phase starts with the first Negotiation Offer (NO) message received or the first Negotiation Response (NR) message sent, provided the message is not a part of an already existing Phase. For each OCP agent, Negotiation Phase ends with the first Negotiation Response (NR) message (sent or received) after which the agent expects no more negotiations. Expectation rules are defined later in this section.

During a Negotiation Phase, an OCP agent MUST NOT send messages other than the following "Negotiation Phase messages": Negotiation Offer (NO), Negotiation Response (NR), Ability Query (AQ), Ability Answer (AA), PQ, PA, PR, and Connection End (CE).

Multiple Negotiation Phases may happen during the lifespan of a single OCP connection. An agent may attempt to start a new Negotiation Phase immediately after the old Phase is over, but it is possible that the other agent will send messages other than "Negotiation Phase messages" before receiving the new Negotiation Offer (NO). The agent that starts a Phase has to be prepared to handle those messages while its offer is reaching the recipient.

An OPES processor MUST make a negotiation offer immediately after sending a Connection Start (CS) message. If the OPES processor has nothing to negotiate, the processor MUST send a Negotiation Offer (NO) message with an empty features list. These two rules bootstrap the first Negotiation Phase. Agents are expected to negotiate at least the application binding for OCP Core. Thus, these bootstrapping requirements are unlikely to result in any extra work.

Once a Phase starts, an agent MUST expect further negotiations if and only if the last NO sent or the last NR received contained true "Offer-Pending" parameter value. Informally, an agent can keep the phase open by sending true "Offer-Pending" parameters with negotiation offers or responses. Moreover, if there exist a possibility that the agent may need to continue the Negotiation Phase, the agent must send a true "Offer-Pending" parameter.

6.2 Negotiation Examples

Below is an example of the simplest negotiation possible. The OPES processor is offering nothing and is predictably receiving a rejection. Note that the NR message terminates the Negotiation Phase in this case because neither of the messages contains a true "Offer-Pending" value:

```
P: NO ()
   ;

S: NR
   ;
```

The next example illustrates how a callout server can force negotiation of a feature that an OPES processor ignored. Note that the server sets the "Offer-Pending" parameter to true when responding to the processor Negotiation Offer (NO) message. The processor chooses to accept the feature:

```
P: NO ()
   ;

S: NR
   Offer-Pending: true
   ;

S: NO ({"22:ocp://feature/example/"})
   Offer-Pending: false
   ;

P: NR {"22:ocp://feature/example/"}
   ;
```

If the server changes its mind to continue the above negotiations after sending a true "Offer-Pending" value, the server's only option would be send an empty negotiation offer (see the first example above). If the server does nothing instead, the OPES processor would wait for the server and would eventually timeout the connection.

The following example, shows a dialog with a callout server that insist on encrypting all data communications on OCP connection using some strong encryption mechanism. The OPES processor supports one of the strong encryption mechanisms but prefers not to offer (volunteer support for) strong encryption, perhaps for performance reasons. The server has to send a true "Offer-Pending" with every offer and every response until satisfactory encryption is negotiated or no agreement can be reached. The former is the case in this example. If it were the latter, the server would have to close the OCP connection with a Connection End (CE) message indicating a failure.

```
   P: NO ({"29:ocp://example/encryption/weak"})
      Offer-Pending: false
      ;

   S: NR
      Offer-Pending: true
      ;

   S: NO ({"32:ocp://example/encryption/strongA"},\
      {"32:ocp://example/encryption/strongB"})
      Offer-Pending: true
      ;

   P: NR {"32:ocp://example/encryption/strongB"}
      Offer-Pending: false
      ;

   S: NO ()
      ;

   P: NR
      ;
```

The following example from [I-D.ietf-opes-http] illustrates successful HTTP application profile negotiation:

```
   P: NO ({"38:http://iana.org/opes/ocp/HTTP/response"
      Aux-Parts: (request-header,request-body)
      })
      SG: 5;

   S: NR {"38:http://iana.org/opes/ocp/HTTP/response"
      Aux-Parts: (request-header)
      Pause-At-Body: 30
      Wont-Send-Body: 2147483647
      Content-Encodings: (gzip)
      }
      SG: 5;
```

7. 'Data Preservation' Optimization

Many adaptations do not require any data modifications (e.g., message logging or blocking). Some adaptations modify only a small portion of application message content (e.g., ad filtering or insertion). Yet, in many cases the callout service needs to see complete data. By default, unmodified data would first travel from the OPES processor to the callout server and then back. The "data preservation" optimization in OCP helps to eliminate the return trip if both OCP agents cooperate. This optimization is optional.

To avoid sending unmodified data back, a callout service has to know that the OPES processor has a copy of the data. Since data sizes can be very large and the callout service may not know in advance whether it will be able to utilize the processor copy, it is not possible to require the processor to keep a copy of the entire original data. Instead, it is expected that a processor may keep some portion of the data, depending on processor settings and state.

When processor commits to keeping a data chunk, it announces its decision and the chunk parameters via a Kept parameter of a Data Use Mine (DUM) message. The callout server MAY "use" the chunk by sending a Data Use Yours (DUY) message referring to the preserved chunk. That OCP message does not have payload and, hence, the return trip is eliminated.

Since the mapping between original and adapted data is not known to the processor, the processor MUST keep the chunk until the end of the corresponding transaction, unless the callout server explicitly tells processor that the chunk is not needed. As implied by the above requirement, the processor cannot assume that a data chunk is no longer needed just because the callout server sent a Data Use Yours (DUY) message or adapted data with, say, the same offset as the preserved chunk.

For simplicity, preserved data is always a contiguous chunk of original data, described by an (offset, size) pair using a "Kept" parameter of a Data Use Mine (DUM) message. An OPES processor may volunteer to increase the size of the kept data. An OPES processor may increase the offset if the callout server indicated that the kept data is no longer needed.

Both agents may benefit from data reuse. An OPES processor has to allocate storage to support this optimization while a callout server does not. On the other hand, it is the callout server that is responsible for relieving the processor from data preservation commitments. There is no simple way to resolve this conflict of interest on a protocol level. Some OPES processors may allocate a

relatively small buffer for data preservation purposes and stop preserving data when the buffer gets full. Such technique would benefit callout services that can quickly reuse or discard kept data. Another processor strategy would be to size the buffer based on historical data reuse statistics. To improve chances of beneficial cooperation, callout servers are strongly encouraged to immediately notify OPES processors of unwanted data. The callout server that is not going to send a Data Use Yours (DUY) messages (for a specific data ranges or at all), SHOULD immediately inform the OPES processor of that fact with corresponding Data Preservation Interest (DPI) message(s) or other mechanisms.

8. 'Getting Out Of The Loop' Optimization

Many services are applicable to a small percentage of application messages and yet have to see the beginning of every application message to decide on applicability (e.g., services that adapt based on declared or guessed MIME type). Many services adapt application message "headers" or "prefix" only and are not interested in the remaining parts of an application message (e.g., URL blocking and ad insertion services). 'Getting Out Of The Loop' optimization allows a callout server to get out of application message processing loop when the server is confident that it does not need to see remaining data.

Two conditions are necessary for the callout server to get out of the loop nicely:

No adaptation need: The callout server must finish its primary work. It should sent all adapted data to the processor and should require no more original data from the processor. Since adaptations and adaptation needs might not depend on original data, only the server can evaluate this condition.

No copying need: The OPES processor must receive back all unpreserved data chunks that were sent to the callout server for adaptation. Note that data chunks that are not preserved and are not returned by the callout service would be lost. Since the server may not have seen all the original data sent, only the processor can evaluate this condition.

Since no single agent can determine both conditions, the agents have to cooperate. The callout server has to tell the processor when the first condition is true. This is done via a Want Out of The Data Loop (DWOL) message. The processor has to tell the service that there are no pending unpreserved data chunks. This is done by terminating the application message delivery using an Application Message End (AME) message with a 206 "Get Out" result. Between the above two conditions or messages, the callout server returns all original data unmodified back to the OPES processor, draining pending (uncopied) data queue (if any).

9. Protocol Element Type Declaration Mnemonic (PETDM)

A protocol element type is a named set of syntax and semantics rules. This section defines a simple, formal declaration mnemonic for protocol element types, labeled PETDM. PETDM simplicity is meant to ease type declarations in this specification. PETDM formality is meant to improve interoperability among implementations. Two protocol elements are currently supported by PETDM: parameter values and messages.

All OCP Core parameter and message types are declared using PETDM. OCP extensions MUST use PETDM when declaring new types.

Atom, list, structure, and message constructs are four available base types. Their syntax and semantics rules are defined in Section 3.1. New types can be declared using PETDM to extend base types semantics but not syntax. The following templates are used to extend semantics of types. The new semantics rules are meant to be attached to each declaration using prose text.

Things in angle brackets are template placeholders, to be substituted with actual type names or parameter name tokens. Square brackets surround optional elements such as structure members or message payload.

o Declaring a new atomic type:
<new-type-name>: extends atom;

o Declaring a new list with old-type-name items:
<new-type-name>: extends list of <old-type-name>;
Unless explicitly noted otherwise, empty lists are valid and have semantics of a absent parameter value.

o Declaring a new structure with members:

```
<new-type-name>: extends structure with {
     <old-type-nameA> <old-type-nameB> [<old-type-nameC>] ...;
     <member-name1>: <old-type-name1>;
     <member-name2>: <old-type-name2>;
     [<member-name3>: <old-type-name3>];
     ...
};
```

The new structure may have anonymous members and named members. Neither group have to exist. Note that it is always possible for extensions to add more members to old structures without affecting type semantics because unrecognized members are ignored by compliant agents.

o Declaring a new message with parameters:

```
<new-type-name>: extends message with {
     <old-type-nameA> <old-type-nameB> [<old-type-nameC>] ...;
     <parameter-name1>: <old-type-name1>;
     <parameter-name2>: <old-type-name2>;
     [<parameter-name3>: <old-type-name3>];
     ...
};
```

The new type name becomes the message name. Just like when extending a structure, the new message may have anonymous parameters and named parameters. Neither group have to exist. Note that it is always possible for extensions to add more parameters to old messages without affecting type semantics because unrecognized parameters are ignored by compliant agents.

o Extending a type with more semantics details:

```
<new-type-name>: extends <old-type-name>;
```

o Extending a structure- or message-base type:

```
<new-type-name>: extends <old-type-name> with {
     <old-type-nameA> <old-type-nameB> [<old-type-nameC>] ...;
     <member-name1>: <old-type-name1>;
     <member-name2>: <old-type-name2>;
     [<member-name3>: <old-type-name3>];
     ...
};
```

New anonymous members are appended to the anonymous members of the old type, and new named members are merged with named members of the old type.

o Extending a message-base type with payload semantics:

```
<new-type-name>: extends <old-type-name> with {
     ...
} and payload;
```

Any any OCP message can have payload, but only some message types have known payload semantics. Like any parameter, payload may be required or optional.

Except for message-based types, all extended types may be used as a replacement of the types they extend. For example, a Negotiation Offer (NO) message uses a parameter of type Features. Features (Section 10.13) is a list of feature (Section 10.12) items. A Feature is a structure-based type. An OCP extension (e.g., an HTTP application binding) may extend the feature type and use a value of that extended type in a negotiation offer. Recipients that are aware of the extension will recognize added members in feature items and negotiate accordingly. Other recipients will ignore them. Thus, declaring a protocol element type freezes that element base syntax, but allows for fine-tuning its semantics in extensions.

9.1 Optional Parameters

Anonymous parameters are positional: parameter's position (i.e., the number of anonymous parameters to the left) is its "name". Thus, when a structure or message have multiple optional anonymous parameters, parameters to the right can be used only if all parameters to the left are present. The following notation:
[name1] [name2] [name3] ... [nameN]
 is interpreted as:
[name1 [name2 [name3 ... [nameN] ...]]]

When adding an anonymous parameter to a structure or message that have optional anonymous parameters, the new parameter has to be optional, and the new parameter can only be used if all old optional parameters are in use. Named parameters do not have such limitations because they are not positional but associative; they are identified by their unique names.

10. Message Parameter Types

This section defines parameter value types that are used for message definitions (Section 11). Before using a parameter value, an OCP agent MUST check whether the value has the expected type (i.e., whether it complies with all rules from the type definition). A single rule violation means that the parameter is invalid. See Section 5 for rules on processing invalid input.

OCP extensions MAY define their own types. If they do, OCP extensions MUST define types with exactly one base format and MUST specify type of every new parameter they introduce.

10.1 uri

uri: extends atom;

Uri (universal resource identifier) is an atom formatted according to URI rules in [RFC2396].

Often, a uri parameter is used as a unique (within a given scope) identifier. Many uri parameters are URLs. Unless noted otherwise, URL identifiers do not imply existence of a serviceable resource at the location they specify. For example, an HTTP request for an HTTP-based URI identifier may result in a 404 (Not Found) response.

10.2 uni

uni: extends atom;

Uni (unique numeric identifier) is an atom formatted as dec-number and with a value in the [0, 2147483647] inclusive range.

A uni parameter is used as a unique (within a given scope) identifier. Uni semantics is incomplete without the scope specification. Some OCP messages create identifiers (i.e., bring them into scope). Some OCP messages destroy them (i.e, remove them from scope). An OCP agent MUST NOT create the same uni value more than once. When creating a new identifier of the same type and within the same scope as some old identifier, an OCP agent MUST use a higher numerical value for the new identifier. The first rule makes uni identifiers suitable for cross-referencing logs and other artifacts. The second rule makes efficient checks of the first rule possible.

For example, a previously used application message identifier "am-id" must not be used for a new Application Message Start (AMS) message within the same OCP transaction, even if a prior Application Message End (AME) message was sent for the same transaction.

An OCP agent MUST terminate the associated scope if all unique values

have been used up.

10.3 size

size: extends atom;

Size is an atom formatted as dec-number and with a value in the [0, 2147483647] inclusive range.

Size value is the number of octets in the associated data chunk.

OCP Core cannot handle application messages that exceed 2147483647 octets in size, require larger sizes as a part of OCP marshaling process, or use sizes with granularity other than 8 bits. This limitation can be addressed by OCP extensions as hinted in Section 15.1.

10.4 offset

offset: extends atom;

Offset is an atom formatted as dec-number and with a value in the [0, 2147483647] inclusive range.

Offset is an octet position expressed in the number of octets relative to the first octet of the associated dataflow. The offset of the first data octet has a value of zero.

10.5 percent

percent: extends atom;

Percent is an atom formatted as dec-number and with a value in the [0, 100] inclusive range.

Percent semantics is incomplete without associating its value with a boolean statement or assertion. The value of 0 indicates absolute impossibility. The value of 100 indicates an absolute certainty. In either case, the associated statement can be relied upon as if it was expressed in boolean rather than probabilistic terms. Values in the [1,99] inclusive range indicate corresponding levels of certainty that the associated statement is true.

10.6 boolean

boolean: extends atom;

Boolean type is an atom formatted as dec-number and with a value in the [0, 1] inclusive range. A boolean parameter expresses truthfulness of the corresponding statement. Two atomic values are valid: true and false.

10.7 xid

xid: extends uni;

"Xid", an OCP transaction identifier, uniquely identifies an OCP transaction within an OCP connection.

10.8 sg-id

sg-id: extends uni;

"Sg-id", a service group identifier, uniquely identifies a group of services on an OCP connection.

10.9 am-id

am-id: extends uni;

"Am-id", an application message identifier, uniquely identifies an application message within an OCP transaction scope, including both original and adapted dataflows.

10.10 modp

modp: extends percent;

Modp extends the percent type to express senders confidence that application data will be modified. The boolean statement associated with the percentage value is "data will be modified". Modification is defined as adaptation that changes numerical value of at least one data octet.

10.11 result

```
result: extends structure with {
     atom [atom];
};
```

OCP processing result is expressed as a structure with two documented members: a required Uni status code and an optional reason. The reason member contains informative textual information not intended for automated processing. For example,

```
{ 200 OK }
{ 200 "6:got it" }
{ 200 "27:27 octets in UTF-8 encoding" }
```

This specification defines the following status codes:

Result Status Codes

code	text	semantics
200	OK	Overall success. This specification does not contain any general actions for 200 status code recipients.
206	partial data	Partial success. When sent by an OPES processor, the remaining original data is not going to be sent due to the callout server disinterest expressed via a Ignoring Your Data (DIY) or a Want Out of The Data Loop (DWOL) message. When sent by a callout server, the remaining adapted application data is identical to the remaining original data flow and should come directly from the OPES processor that already committed to "Out Of The Loop" optimization by sending a corresponding AME message after receiving a Want Out of The Data Loop (DWOL) message.
400	failure	An error, exception, or trouble. A recipient of a 400 (failure) result of a AME, TE, or CE message MUST destroy any state or data associated with the corresponding data flow, transaction, or connection. For example, adapted version of the application message data must be purged from the processor cache if the OPES processor receives an Application Message End (AME) message with result code of 400.

Specific OCP messages may require code-specific actions.

Extending result semantics is possible by adding new "result" structure members or negotiating additional result codes (e.g., as a part of a negotiated profile). A recipient of an unknown (in then-current context) result code MUST act as if code 400 (failure) was received.

The recipient of a message without the actual result parameter, but with an optional formal result parameter MUST act as if code 200 (OK)

was received.

Textual information (the second anonymous parameter of the result structure) is often referred to as "reason" or "reason phrase". To assist troubleshooting efforts, OCP agents are encouraged to include descriptive reasons with all results indicating a failure.

An OCP message with result status code of 400 (failure) is called "a message indicating a failure" in this specification.

10.12 feature

```
feature: extends structure with {
     uri;
};
```

Feature extends structure to relay an OCP feature identifier and to leave "space" for optional feature-specific parameters (sometimes called feature attributes). Feature values are used to declare support for and negotiate use of OCP features.

This specification does not define any features.

10.13 features

```
feature: extends list of feature;
```

10.14 service

```
service: extends structure with {
     uri;
};
```

"Service" structure has one anonymous member, an OPES service identifier of type "uri". Services may have service-dependent parameters. An OCP extension defining a service for use with OCP MUST define service identifier and service-dependent parameters as additional "service" structure members, if any. For example, a service value may look like this:

```
{"37:http://iana.org/assigned/opes/ocp/tls"
"8:blowfish"}
```

10.15 services

```
services: extends list of service;
```

"Services" is a list of "service" values. Unless noted otherwise, the list can be empty and the order of the values is the requested or actual service application order.

10.16 Dataflow Specializations

Several parameter types such as "offset" apply to both original and adapted dataflow. It is relatively easy to misidentify type's dataflow affiliation, especially when parameters with different affiliation are mixed together in one message declaration. The following statements declare new dataflow-specific types using their dataflow-agnostic versions (denoted by a <type> placeholder).

The following new types refer to original data only:
org-<type>: extends <type>;

The following new types refer to adapted data only:
adp-<type>: extends <type>;

The following new types refer to sender's dataflow only:
my-<type>: extends <type>;

The following new types refer to recipient's dataflow only:
your-<type>: extends <type>;

OCP Core use the above type naming scheme to implement dataflow specialization for the following types: offset, size, am-id, sg-id. OCP extensions SHOULD use the same scheme.

11. Message Definitions

This section describes specific OCP messages. Each message is given a unique name and usually has a set of anonymous and/or named parameters. The order of anonymous parameters is specified in the message definitions below. No particular order for named parameters is implied by this specification. No more than one named-parameter with a given name can appear in the message; messages with multiple equally-named parameters are semantically invalid.

A recipient MUST be able to parse any message in valid format (see Section 3.1), subject to recipient resources limitations.

Unknown or unexpected message names, parameters, and payloads may be valid extensions. For example, an "extra" named parameter may be used for a given message, in addition to what is documented in the message definition below. A recipient MUST ignore any valid but unknown or unexpected name, parameter, member, or payload.

Some message parameter values use uni identifiers to refer to various OCP states (see Section 10.2 and Appendix B). These identifiers are created, used, and destroyed by OCP agents via corresponding messages. Except when creating a new identifier, an OCP agent MUST NOT send a uni identifier that does not correspond to an active state (i.e., that was either never created or was already destroyed). Such identifiers invalidate the host message (see Section 5). For example, the recipient must terminate the transaction when the xid parameter in a Data Use Mine (DUM) message refers to an unknown or already terminated OCP transaction.

11.1 Connection Start (CS)

CS: extends message;

A Connection Start (CS) message indicates the start of an OCP connection. An OCP agent MUST send this message before any other message on the connection. If the first message an OCP agent receives is not Connection Start (CS), the agent MUST terminate the connection with a Connection End (CE) message having 400 (failure) result status code. An OCP agent MUST send Connection Start (CS) message exactly once. An OCP agent MUST ignore repeated Connection Start (CS) messages.

At any time, a callout server MAY refuse further processing on an OCP connection by sending a Connection End (CE) message with status code 400 (failure). Note that the above requirement to send a CS message first still applies.

With TCP/IP as transport, raw TCP connections (local and remote peer IP addresses with port numbers) identify an OCP connection. Other transports may provide OCP connection identifiers to distinguish logical connections that share the same transport. For example, a single BEEP [RFC3080] channel may be designated as a single OCP connection.

11.2 Connection End (CE)

```
CE: extends message with {
     [result]
};
```

Connection End (CE) Indicates an end of an OCP connection. The agent initiating closing or termination of a connection MUST send this message immediately prior to closing or termination. The recipient MUST free associated state, including transport state.

Connection termination without a Connection End (CE) message indicates that the connection was prematurely closed, possibly without the closing-side agent prior knowledge or intent. When an OCP agent detects a prematurely closed connection, the agent MUST act as if a Connection End (CE) message indicating a failure was received.

A Connection End (CE) message implies the end of all transactions, negotiations, and service groups opened or active on the connection being ended.

11.3 Create Service Group (SGC)

```
SGC: extends message with {
     my-sg-id services;
};
```

An Create Service Group (SGC) message instructs the recipient to associate a list of services with a given service group identifier ("my-sg-id"). The group can then be referred by the sender using the identifier. The recipient MUST maintain the association until a matching Connection End (CE) message is received or the corresponding OCP connection is closed.

Service groups have a connection scope. Transaction management messages do not affect existing service groups.

Maintaining service group associations requires resources (e.g., storage to keep the group identifier and a list of service IDs). Thus, there is a finite number of associations an implementation can maintain. Callout servers MUST be able to maintain at least one

association for each OCP connection they accept. If a recipient of a Create Service Group (SGC) message does not create the requested association, it MUST immediately terminate the connection with a Connection End (CE) message indicating a failure.

11.4 Destroy Service Group (SGD)

```
SGD: extends message with {
     my-sg-id;
};
```

A Destroy Service Group (SGD) message instructs the recipient to forget about the service group associated with the specified identifier. The recipient MUST destroy the identified service group association.

11.5 Transaction Start (TS)

```
TS: extends message with {
     xid my-sg-id;
};
```

Sent by an OPES processor, a TS message indicates the start of an OCP transaction. Upon receiving of this message, the callout server MAY refuse further transaction processing by responding with a corresponding Transaction End (TE) message. A callout server MUST maintain the state until it receives a message indicating the end of the transaction or until it terminates the transaction itself.

The required "my-sg-id" identifier refers to a service group created with a Create Service Group (SGC) message. The callout server MUST apply the list of services associated with "my-sg-id", in the specified order.

This message introduces transaction identifier (xid). An OPES processor MUST use a new transaction identifier for each new transaction on a given connection.

11.6 Transaction End (TE)

```
TE: extends message with {
     xid [result];
};
```

Indicates the end of the identified OCP transaction. The recipient MUST free associated state.

This message terminates the life of the transaction identifier (xid).

An OCP agent MUST send a Transaction End (TE) message immediately after it makes a decision to send no more messages related to the corresponding transaction. Violating this requirement may cause, for example, unnecessary delays, rejection of new transactions, and even timeouts for agents that rely on this end-of-file condition to proceed.

11.7 Application Message Start (AMS)

```
AMS: extends message with {
     xid my-am-id;
};
```

Indicates the start of processing of an application message. The recipient MAY refuse processing with an Application Message End (AME) message indicating a failure.

When an AMS message is sent to the callout server, the callout server usually sends an AMS message back, announcing the creation of an adapted version of the original application message. Such response may be delayed. For example, the callout server may wait for more information to come from the OPES processor Depending on the application protocol, there may be multiple such responses. An OCP application binding specification MUST document whether multiple adapted versions of an original message are allowed.

When an AMS message is sent to the OPES processor, an OPTIONAL "services" parameter describes callout services that the server MAY apply to the application message. Usually, the "services" value matches what was asked by the OPES processor.

This message introduces application message identifier (am-id).

11.8 Application Message End (AME)

```
AMS: extends message with {
     xid my-am-id [result];
};
```

Informs the recipient that there will be no more data for the corresponding application message and indicates the end of application message processing. The recipient MUST free associated application message state.

An Application Message End (AME) message ends any data preservation commitments associated with the corresponding application message.

This message terminates the life of the application message

identifier (am-id).

An OCP agent MUST send an Application Message End (AME) message immediately after it makes a decision to send no more data for the corresponding application message. Violating this requirement may cause, for example, unnecessary delays, rejection of new transactions, and even timeouts for agents that rely on this end-of-file condition to proceed.

11.9 Data Use Mine (DUM)

```
DUM: extends message with {
     xid my-am-id my-offset;
     [As-is: org-am-id org-offset];
     [Kept: org-offset org-size ];
     [Modp: modp];
} and payload;
```

This is the only OCP Core message that carries application data as payload. The sender MUST NOT make any gaps in data supplied by Data Use Mine (DUM) and Data Use Yours (DUY) messages (i.e., the my-offset of the next data message must be equal to my-offset plus the payload size of the previous data message). Messages with gaps are invalid. The sender MUST send payload and MAY use empty payload (i.e., payload with zero size). Empty payloads are useful for communicating meta-information about the data (e.g., modification predictions or preservation commitments) without sending data.

An OPES processor MAY send a "Kept" parameter to indicate its new data preservation commitment (Section 7) for original data. When an OPES processor sends a "Kept" parameter, the processor MUST keep a copy of the specified data (the preservation commitment starts). The Kept offset parameter specifies the offset of the first octet of the preserved data. The Kept size parameter is the size of preserved data. Note that data preservation rules allow (i.e., do not prohibit) OPES processor to decrease offset and to specify a data range not yet fully delivered to the callout server.

If the "Kept" parameter value violates data preservation rules, but the recipient has not sent any Data Use Yours (DUY) messages for the given OCP transaction yet, then the recipient MUST NOT use any preserved data for the given transaction (i.e., must not sent any Data Use Yours (DUY) messages). If the "Kept" parameter value violates data preservation rules, and the recipient has already sent Data Use Yours (DUY) messages, the DUM message is invalid and the rules of Section 5 apply. These requirements help preserve data integrity when "Kept" optimization is used by the OPES processor.

A callout server MUST send a "Modp" parameter if the server can provide a reliable value and has not already sent the same parameter value for the corresponding application message. The definition of "reliable" is entirely up to the callout server.

A callout server SHOULD send an "As-is" parameter if the attached data is identical to a fragment in the original data flow. An "As-is" parameter specifying a data fragment that have not been sent to the callout server is invalid. The recipient MUST ignore invalid "As-is" parameters. Identical means that all adapted octets have the same numeric value as the corresponding original octets. The "am-id" field MUST correspond to the original application message identifier for the same transaction. This parameter is meant to allow for partial data preservation optimizations without a preservation commitment. The preserved data still crosses the link to the callout server twice, but OPES processor may be able to optimize its handling.

The OPES processor MUST NOT terminate its data preservation commitment (Section 7) in reaction to receiving a Data Use Mine (DUM) message.

11.10 Data Use Yours (DUY)

```
DUY: extends message with {
     xid adp-am-id org-offset org-size;
};
```

The callout server tells the OPES processor to use the "size" bytes of preserved data starting at the specified offset, as if that data chunk came from the callout server in a Data Use Mine (DUM) message with adp-am-id parameter.

The OPES processor MUST NOT terminate its data preservation commitment (Section 7) in reaction to receiving a Data Use Yours (DUY) message.

11.11 Data Preservation Interest (DPI)

```
DPI: extends message with {
     xid org-am-id org-offset org-size;
};
```

The Data Preservation Interest (DPI) message describes an original data chunk using the first octet offset and size as parameters. The chunk is the only area of original data that callout server may be interested in referring to in future Data Use Yours (DUY) messages. This data chunk is referred to as "reusable data". The rest of the

original data is referred to as "disposable data". Thus, disposable data consists of octets below the specified offset and above the (offset+size) offset.

After sending this message, the callout server MUST NOT send Data Use Yours (DUY) messages referring to disposable data. chunk(s). If an OPES processor is not preserving some reusable data, it MAY start preserving that data. If the OPES processor preserves some disposable data, it MAY stop preserving that data. If an OPES processor does not preserves some disposable data, it MAY NOT start preserving that data.

A callout server MUST NOT indicate reusable data areas that overlap with disposable data areas indicated in previous Data Preservation Interest (DPI) messages. In other words, reusable data must not grow and disposable data must not shrink. If a callout server violates this rule, the Data Preservation Interest (DPI) message is invalid (see Section 5).

The Data Preservation Interest (DPI) message cannot force the OPES processor to preserve data. The term reusable in this context stands for callout server interest in reusing the data in the future, given OPES processor cooperation.

For example, an offset value of zero and the size value of 2147483647 indicates that the server may want to reuse all the original data. The size value of zero indicates that the server is not going to send any more Data Use Yours (DUY) messages.

11.12 Ignoring Your Data (DIY)

```
DIY: extends message with {
     xid org-am-id org-offset;
};
```

The Ignoring Your Data (DIY) message informs OPES processor that the callout server is going to ignore and discard original data starting with the specified offset. After sending this message, the callout server MUST NOT send Data Use Yours (DUY) messages referring to data at or above the given offset. An OPES processor that preserves any data at or above the offset MAY stop preserving that data. An OPES processor that does not preserve data at or above the offset MAY NOT start preserving that data.

Once the processor sends all original data below the given offset (or if the processor has already sent all original data below the given offset), the processor SHOULD immediately terminate the original application message delivery using a Application Message End (AME)

message with a 206 result status code.

A callout server MAY send DIY offsets higher than or equal to the previously sent DIY offsets. However, as the above rules imply, such offsets would have no effect if processor already acted on a previous DIY message with an equal or lower offset. Informally, a processor may maintain a single "not useful for sending" offset per transaction and that offset would never increase.

For example, an offset value of zero indicates that the server is going to ignore all original data and generate an adapted application message from scratch. The OPES processor should most likely not send any data to the callout server in this case.

See the Want Out of The Data Loop (DWOL) message description for requirements related to using both DWOL and DIY messages within one transaction.

11.13 Want Out of The Data Loop (DWOL)

```
DWOL: extends message with {
     xid org-am-id org-offset;
};
```

The Want Out of The Data Loop (DWOL) message informs OPES processor that the callout server wants to get out of the processing loop once the original data at or above the specified offset is received (see Section 8). After sending all requested data, the OPES processor MAY respond with an Application Message End (AME) message carrying a 206 result status code. If the callout server receives that 206 status code response, the server MAY terminate adaptation by sending an Application Message End (AME) message with 206 result status code and getting out of the loop.

While waiting for the 206 status code response, the callout server MAY perform original data modifications, including modifications of data above the specified offset. The Want Out of The Data Loop (DWOL) message indicates desire to get out of the loop, not a commitment to stop data modifications and not a decision to stop forwarding adapted data.

A callout server MUST NOT use both Want Out of The Data Loop (DWOL) and Ignoring Your Data (DIY) messages during the same transaction. Doing so may make OPES processor 206 (partial content) status code ambiguous. If an OPES processor receives both DWOL and DIY messages during the same transaction, the processor MUST terminate the transaction with a Transaction End (TE) message indicating a failure.

An OPES processor MUST NOT terminate its data preservation commitment (Section 7) in reaction to receiving a Want Out of The Data Loop (DWOL) message. Just like with any other message, an OPES processor may use information supplied by Want Out of The Data Loop (DWOL) to decide on future preservation commitments.

11.14 Want Data Paused (DWP)

```
DWP: extends message with {
     xid your-am-id your-offset;
};
```

The Want Data Paused (DWP) message indicates sender's temporary lack of interest in receiving data starting with the specified offset. This disinterest in receiving data is temporary in nature and implies nothing about sender's intent to send data.

Message parameters always refer to dataflow originating at the other OCP agent. When sent by an OPES processor, am-id is adp-am-id and offset is adp-offset. When sent by a callout server, am-id is org-am-id and offset is org-offset.

If the specified offset has already been reached at the time the Paused My Data (DPM) message was received, the recipient MUST stop sending data immediately. Otherwise, the recipient MUST stop sending data immediately when the specified offset is reached. Once the recipient stops sending more data, it MUST immediately send a Paused My Data (DPM) message and MUST NOT send more data until it receives a Want More Data (DWM) message.

As most OCP Core mechanisms, data pausing is asynchronous. The sender of the Want Data Paused (DWP) message MUST NOT rely on the data being paused exactly at the specified offset or at all.

11.15 Paused My Data (DPM)

```
DPM: extends message with {
     xid my-am-id;
};
```

The Paused My Data (DPM) message indicates sender's commitment to send no more data until the sender receives a Want More Data (DWM) message.

The recipient of the Paused My Data (DPM) message MAY expect the data delivery being paused. If the recipient receives data despite this expectation, it MAY abort the corresponding transaction with a Transaction End (TE) message indicating a failure.

11.16 Want More Data (DWM)

```
DWM: extends message with {
     xid your-am-id;
     [Size-request: your-size];
};
```

The Want More Data (DWM) message indicates sender's need for more data.

Message parameters always refer to dataflow originating at the other OCP agent. When sent by an OPES processor, am-id is adp-am-id and size is adp-size. When sent by a callout server, am-id is org-am-id and size is org-size.

If a "Size-request" parameter is present, its value is the suggested minimum data size. It is meant to allow the recipient to deliver data in fewer chunks. The recipient MAY ignore the "Size-request" parameter. An absent "Size-request" parameter implies "any size".

The message also cancels the Paused My Data (DPM) message effect. If the recipient was not sending any data because of its DPM message, the recipient MAY resume sending data. Note, however, that the Want More Data (DWM) message can be sent regardless of whether the dataflow in question has been paused. The "Size-request" parameter makes this message a useful stand-alone optimization.

11.17 Negotiation Offer (NO)

```
NO: extends message with {
     features;
     [SG: my-sg-id];
     [Offer-Pending: boolean];
};
```

A Negotiation Offer (NO) message solicits a selection of a single "best" feature out of a supplied list, using a Negotiation Response (NR) message. The sender is expected to list preferred features first when possible. The recipient MAY ignore sender preferences. If the list of features is empty, the negotiation is bound to fail but remains valid.

Both OPES processor and callout server are allowed to send Negotiation Offer (NO) messages. The rules in this section ensure that only one offer is honored if two offers are submitted concurrently. An agent MUST NOT send a Negotiation Offer (NO) message if it still expects a response to its previous offer on the same connection.

If an OPES processor receives a Negotiation Offer (NO) message while its own offer is pending, the processor MUST disregard the server offer. Otherwise, it MUST respond immediately.

If a callout server receives a Negotiation Offer (NO) message when its own offer is pending, the server MUST disregard its own offer. In either case, the server MUST respond immediately.

If an agent receives a message sequence that violates any of the above rules in this section, the agent MUST terminate the connection with a Connection End (CE) message indicating a failure.

An optional "Offer-Pending" parameter is used for Negotiation Phase maintenance (Section 6.1). The option's value defaults to "false".

An optional "SG" parameter is used to narrow the scope of negotiations to the specified service group. If SG is present, the negotiated features are negotiated and enabled only for transactions that use the specified service group ID. Connection-scoped features are negotiated and enabled for all service groups. The presence of scope does not imply automatic conflict resolution common to programming languages; no conflicts are allowed. When negotiating connection-scoped features, an agent MUST check for conflicts within each existing service group. When negotiating group-scoped features, an agent MUST check for conflicts with connection-scoped features already negotiated. For example, it must not be possible to negotiate a connection-scoped HTTP application profile if one service group already has an SMTP application profile and vice versa.

OCP agents SHOULD NOT send offers with service groups used by pending transactions. Unless explicitly noted otherwise in a feature documentation, OCP agents MUST NOT apply any negotiations to pending transactions. In other words, negotiated features take effect with the new OCP transaction.

11.18 Negotiation Response (NR)

```
NR: extends message with {
     [feature];
     [SG: my-sg-id];
     [Rejects: features];
     [Unknowns: features];
     [Offer-Pending: boolean];
};
```

A Negotiation Response (NR) message conveys recipient's reaction to a Negotiation Offer (NO) request. An accepted offer is indicated by the presence of an anonymous "feature" parameter, containing the

selected feature. If the selected feature does not match any of the offered features, the offering agent MUST consider negotiation failed and MAY terminate the connection with a Connection End (CE) message indicating a failure.

A rejected offer is indicated by omitting the anonymous "feature" parameter.

If negotiation offer contains an SG parameter, the responder MUST include that parameter in the Negotiation Response (NR) message. The recipient of a NR message without the expected SG parameter MUST treat negotiation response as invalid.

If negotiation offer lacks an SG parameter, the responder MUST NOT include that parameter in the Negotiation Response (NR) message. The recipient of a NR message with an unexpected SG parameter MUST treat negotiation response as invalid.

An optional "Offer-Pending" parameter is used for Negotiation Phase maintenance (Section 6.1). The option's value defaults to "false".

When accepting or rejecting an offer, the sender of the Negotiation Response (NR) message MAY supply additional details via Rejects and Unknowns parameters. The Rejects parameter can be used to list features that were known to the Negotiation Offer (NO) recipient but could not be supported given negotiated state that existed when NO message was received. The Unknowns parameter can be used to list features that were unknown to the NO recipient.

11.19 Ability Query (AQ)

```
AQ: extends message with {
     feature;
};
```

A Ability Query (AQ) message solicits an immediate Ability Answer (AA) response. The recipient MUST respond immediately with a AA message. This is a read-only, non-modifying interface. The recipient MUST NOT enable or disable any features due to an AQ request.

OCP extensions documenting a feature MAY extend AQ messages to supply additional information about the feature or the query itself.

The primary intended purpose of the ability inquiry interface is debugging and troubleshooting rather than automated fine-tuning of agent behavior and configuration. The latter may be better achieved by the OCP negotiation mechanism (Section 6).

11.20 Ability Answer (AA)

```
AA: extends message with {
     boolean;
};
```

A Ability Answer (AA) message expresses senders support for a feature requested via a Ability Query (AQ) message. The sender MUST set the value of the anonymous boolean parameter to the truthfulness of the following statement: "at the time of this answer generation, the sender supports the feature in question". The meaning of "support" and additional details are feature-specific. OCP extensions documenting a feature MUST document the definition of "support" in the scope of the above statement and MAY extend AA messages to supply additional information about the feature or the answer itself.

11.21 Progress Query (PQ)

```
PQ: extends message with {
     [xid] [your-am-id];
};
```

A Progress Query (PQ) message solicits an immediate Progress Answer (PA) response. The recipient MUST immediately respond to a PQ request, even if transaction or application message identifiers are invalid from the recipient point of view.

11.22 Progress Answer (PA)

```
PA: extends message with {
     [xid] [my-am-id];
     [Org-Data: org-size];
};
```

A PA message carries a senders state. If an agent is sending an application message ID (my-am-id) parameter, the agent MUST send the "Org-Data" parameter. Otherwise, the agent MUST NOT send an "Org-Data" parameter. The "Org-Data" size is the total original data size received or sent by the agent so far for the identified application message (an agent can be either sending or receiving original data so there is no ambiguity). When referring to received data, progress information does not imply that the data has been otherwise processed in some way.

The progress inquiry interface is useful for several purposes, including keeping idle OCP connections "alive", gauging the agent processing speed, verifying agent's progress, and debugging OCP communications. Verifying progress, for example, may be essential to

implement timeouts for callout servers that do not send any adapted data until the entire original application message is received and processed.

A recipient of a PA message MUST NOT assume that the sender is not working on any transaction or application message not identified in the PA message. A PA message cannot carry information about multiple transactions or application messages.

If an agent is working on the transaction identified in the Progress Query (PQ) request, the agent MUST send the corresponding transaction ID (xid) when answering PQ with a PA message. Otherwise, the agent MUST NOT send the transaction ID. If an agent is working on the application message identified in the PQ request (your-am-id), the agent MUST send that application message ID (my-am-id). Otherwise, the agent MUST NOT send the application message ID.

Informally, the answer needs to match query identifiers, provided those identifiers are valid at the time the response is generated. Absent identifiers in the answer indicate invalid identifiers in the query (from the query recipient point of view).

11.23 Progress Report (PR)

```
PR: extends message with {
     [xid] [my-am-id];
     [Org-Data: org-size];
};
```

A PR message carries senders state. The message semantics and associated requirements are identical to that of a Progress Answer (PA) message except that the PR message is sent unsolicited. The sender MAY report progress at any time. The sender MAY report progress unrelated to any transaction or application message or related to any valid (current) transaction or application message.

Unsolicited progress reports are especially useful for OCP extensions dealing with "slow" callout services that introduce significant delays for the final application message recipient. The report may contain progress information that will make that final recipient more delay-tolerant.

12. IAB Considerations

OPES treatment of IETF Internet Architecture Board (IAB) considerations [RFC3238] are documented in [I-D.ietf-opes-iab].

13. Security Considerations

This section examines security considerations for OCP. OPES threats are documented in [I-D.ietf-opes-threats].

OCP relays application messages that may contain sensitive information. Appropriate transport encryption can be negotiated to prevent information leakage or modification (see Section 6), but OCP agents may support unencrypted transport by default. Such default OCP agent configurations will expose application messages to third party recording and modification, even if OPES proxies themselves are secure.

OCP implementation bugs may lead to security vulnerabilities in OCP agents, even if OCP traffic itself remains secure. For example, a buffer overflow in a callout server caused by a malicious OPES processor may grant that processor access to information from other (100% secure) OCP connections, including connections with other OPES processors.

Careless OCP implementations may rely on various OCP identifiers to be unique across all OCP agents. A malicious agent can inject an OCP message that matches identifiers used by other agents, in an attempt to get access to sensitive data. OCP implementations must always check an identifier for being "local" to the corresponding connection before using that identifier.

OCP is a stateful protocol. Several OCP commands increase the amount of state that the recipient has to maintain. For example, a Create Service Group (SGC) message instructs the recipient to maintain an association between a service group identifier and a list of services.

Implementations that cannot handle resource exhaustion correctly increase security risks. The following are known OCP-related resources that may be exhausted during a compliant OCP message exchange:

OCP message structures: OCP message syntax does not limit the nesting depth of OCP message structures and does not place an upper limit on the length (number of OCTETs) of most syntax elements.

concurrent connections: OCP does not place an upper limit on the number of concurrent connections that a callout server may be instructed to create via Connection Start (CS) messages.

service groups: OCP does not place an upper limit on the number of service group associations that a callout server may be instructed to create via Create Service Group (SGC) messages.

concurrent transactions: OCP does not place an upper limit on the number of concurrent transactions that a callout server may be instructed to maintain via Transaction Start (TS) messages.

concurrent flows: OCP Core does not place an upper limit on the number of concurrent adapted data flows that an OPES processor may be instructed to maintain via Application Message Start (AMS) messages.

14. IANA Considerations

This specification contains no resources suitable for Internet Assigned Numbers Authority (IANA) maintenance.

15. Compliance

This specification defines compliance for the following compliance subjects: OPES processors (OCP client implementations), callout servers (OCP server implementations), OCP extensions. An OCP agent (a processor or callout server) may also be referred to as "sender" or "recipient" of an OCP message.

A compliance subject is compliant if it satisfies all applicable "MUST" and "SHOULD" level requirements. By definition, to satisfy a "MUST" level requirement means to act as prescribed by the requirement; to satisfy a "SHOULD" level requirement means to either act as prescribed by the requirement or have a reason to act differently. A requirement is applicable to the subject if it instructs (addresses) the subject.

Informally, OCP compliance means that there are no known "MUST" violations, and all "SHOULD" violations are conscious. In other words, a "SHOULD" means "MUST satisfy or MUST have a reason to violate". It is expected that compliance claims are accompanied by a list of unsupported SHOULDs (if any), in an appropriate format, explaining why preferred behavior was not chosen.

Only normative parts of this specification affect compliance. Normative parts are: parts explicitly marked using the word "normative", definitions, and phrases containing unquoted capitalized keywords from [RFC2119]. Consequently, examples and illustrations are not normative.

15.1 Adapting OCP Core

OCP extensions MAY change any normative requirement documented in this specification, including OCP message format, except for the following rule: OCP extensions MUST require that changes to normative parts of OCP Core are negotiated prior to taking effect. For example, if an RTSP binding for OCP requires support for sizes exceeding 2147483647 octets, the binding specification can document appropriate OCP message format and semantics changes while requiring that RTPS adaptation agents negotiate "large size" support before using large sizes. Such negotiation can be bundled with negotiating another feature (e.g., negotiating an RTSP profile may imply support for large sizes).

Appendix A. Message Summary

This appendix is not normative. The table below summarizes key OCP message properties. For each message, the table provides the following information:

name: message name as seen on the wire;

title: human-friendly message title;

P: whether this specification documents message semantics as sent by an OPES processor;

S: whether this specification documents message semantics as sent by a callout server;

tie: related messages such as associated request or response message or associated state message.

name	title	P	S	tie
CS	Connection Start	X	X	CE
CE	Connection End	X	X	CS
SGC	Create Service Group	X	X	SGD TS
SGD	Destroy Service Group	X	X	SGC
TS	Transaction Start	X		TE SGC
TE	Transaction End	X	X	TS
AMS	Application Message Start	X	X	AME
AME	Application Message End	X	X	AMS DWOL
DUM	Data Use Mine	X	X	DUY DWP
DUY	Data Use Yours		X	DUM DPI
DPI	Data Preservation Interest		X	DUY
DIY	Ignoring Your Data		X	DWOL AME
DWP	Want Data Paused	X	X	DPM

DPM	Paused My Data	X	X	DWP DWM	
DWM	Want More Data	X	X	DPM	
DWOL	Want Out of The Data Loop		X	AME DIY	
NO	Negotiation Offer	X	X	NR SGC	
NR	Negotiation Response	X	X	NO	
PQ	Progress Query	X	X	PA	
PA	Progress Answer	X	X	PQ PR	
PR	Progress Report	X	X	PA	
AQ	Ability Query	X	X	AA	
AA	Ability Answer	X	X	AQ	

Appendix B. State Summary

This appendix is not normative. The table below summarizes long-lived OCP states that survive past a single request/response dialog. For each state, the table provides the following information:

state: short state label

ID: associated identifier, if any

birth: messages creating this state

death: messages destroying this state

state	ID	birth	death
connection		CS	CE
service group	sg-id	SGC	SGD
transaction	xid	TS	TE
application message	am-id	AMS	AME
preservation commitment		DUM	DPI DIY AME
paused dataflow		DPM	DWM

Appendix C. Acknowledgements

The author gratefully acknowledges the contributions of: Abbie Barbir (Nortel Networks), Oskar Batuner (Independent Consultant), Karel Mittig (France Telecom R&D), Markus Hofmann (Bell Labs), Hilarie Orman (The Purple Streak), Reinaldo Penno (Nortel Networks), Martin Stecher (Webwasher) as well as an anonymous OPES working group participant.

Special thanks to Marshall Rose for his xml2rfc tool.

Appendix D. Change Log

Internal WG revision control ID: $Id: ocp-core.xml,v 1.70 2003/11/17 23:50:09 rousskov Exp $

2003/11/01

* Simplified/steamlined ping-pong interface: Moved "unsolicited pong" semantics to a Progress Report (PR) message. Moved "solicited pong" semantics to a Progress Answer (PA) message. Renamed Progress Request (ping) to Progress Query (PQ). Renamed "Progress" parameter to "Org-Data".

* Added informative summaries of OCP messages and states as appendices.

* Added a requirement for uni values to increase so that agents can easily enforce uni uniqueness.

* Added Dataflow-specific types for size, offset, am-id, and sg-id. Resolved several ambiguities in message declarations: "which am-id should this message use, original or adapted?".

* Renamed Data Interested in Using Yours (DIUY) message to Data Preservation Interest (DPI).

* Renamed Data Won't Look At Yours (DWLY) message to Ignoring Your Data (DIY).

* Renamed Data Pause (data-pause) message to Want Data Paused (DWP).

* Renamed Data Paused (data-paused) message to Paused My Data (DPM).

* Renamed Data Need (data-need) message to Wont More Data (DWM).

* Renamed Data Want Out (DWO) message to Want Out Of The Data Loop (DWOL).

2003/10/31

* Changed Kept parameter syntax and clarified/simplified/improved its semantics. Renamed DWSY message to DIUY and clarified/simplified/improved its semantics. All data preservation interface is now built around a single continues data chunk that Kept parameter and DIUY message refer to when they need to specify what is preserved or needs to be.

* Added Negotiation Phase and an optional "Offer-Pending" parameter to NO and NR messages to ensure that an OCP agent can negotiate vital features before application data is seen on the wire.

* Polished dataflow pausing interface and made its support mandatory. Gave an OPES processor the same abilities to pause/resume dataflow that a callout server has.

* Added a Timeouts section, requiring all OCP agents to support timeouts of some sort.

* Removed data loss to-do item. Extensions would have to take care of that complication.

2003/10/30

* Merged Capability and State Inquiry mechanisms into a simpler Ability Query/Answer (AQ/AA) interface. Added a new MUST: OCP extensions must document what it means to "support" a given feature they document. The definition is needed for generation of AA messages.

* Removed DoS attacks against callout service as a security consideration because its place is in OPES architecture or OPES security drafts.

* Merged DACK mechanism into a polished ping-pong mechanism.

* Added a new requirement: An OCP application binding specification MUST document whether multiple adapted versions of an original message are allowed.

* Declared all OCP messages using PETDM.

* Deleted "Application Protocol Requirements" Section as essentially unused.

2003/10/28

* Simplified and polished CS message rules. Callout servers MUST send CS now so that processors can be sure the other end is talking OCP.

* Made "Type Declaration Mnemonic (TDM)" a top-level section titled "Protocol Element Type Declaration Mnemonic (PETDM)" and documented OCP message declaration mnemonic.

* Merged parameter type declarations with parameter declarations.

* Polished parameter type declarations.

2003/10/26

* Started using TDM for Core value types.

* Added Data Want Out (DWO) message.

* Added Data Wont Look at Yours (DWLY) message.

* Renamed Wont-Use to more specific Wont-Send. Made Wont-Send parameter into a Data Wont Send Yours (DWSY) message because it controls original data flow and is not specific to a particular adapted AM (there can be many). This change means that Data Use Yours (DUY) messages are no longer terminating preservation commitment by default. Thus, we lost a little in terms of performance (unless processors look ahead for DWSYs) but gained a lot of simplicity in terms of support for multiple adapted application messages (SMTP case).

* Added 206 (partial data) status code definition.

* 206 status code should be used with AME, not TE.

* Replaced "global scope" with "connection scope" in negotiation rules.

2003/10/25

* Clarified negotiation mechanism when it comes to negotiating multiple [possibly conflicting] features.

* Clarified service group-scoped negotiations. Agents must watch out for global conflicts when doing group-scoped negotiations and vice-versa.

2003/10/24

* Added 'Out Of The Loop' Optimization section.

* Added 'Data Recycling' Optimization section.

* Added "Type Declaration Mnemonic" (TDM) to facilitate type declarations here and in OCP extensions.

2003/10/19

* Removed optional "sizep" parameter. HTTP needs content-dependent parameter (AM-EL), and we do not know any generic application for sizep that would be worth supporting in Core.

2003/10/08

* Documented backslash (\) and CRLF (\r\n) OCP message rendering tricks.

2003/10/07

* Added named structure members to message BNF. Used MIME-like syntax already used for named parameters. Named members are needed to represent optional structure members.

head-sid15

* Removed leftovers of data-have message name. Use Data Use Mine instead (Karel Mittig).

* Anonymized named parameters and removed currently unused "rid" parameter in ping and pong messages (Karel Mittig).

* Renamed DUM.please-ack to "DUM.ack" (Karel Mittig). More work is needed to polish and simplify acknowledgment mechanism.

head-sid14

* Documented known resource-exhaustion security risks.

* Polished compliance definition. Avoid two levels of compliance.

head-sid13

* Added SG parameter to Negotiation Offer (NO) and Negotiation Response (NR) messages to limit the result of negotiations to the specified service group. Still need to document SG-related logic in the Negotiation section.

* Removed "services" parameter from Transaction Start (TS) message because we have to rely on service groups exclusively, because only service groups can have negotiated application profiles associated with them.

* Replaced data-id parameter with "Kept: kept-offset" and

"Wont-Use: used-size" parameter. We probably need octet-based granularity, and old data-id only offered fragment-based granularity.

* Made AME and TE messages required.

* Documented result parameter syntax and two result codes: 200 (success) and 400 (failure).

* Added optional "result" parameter to CE.

head-sid12

* Fixed BNF to remove extra SP and "," in front of structure and list values.

* Fixed the type of "id" field in a "service" structure.

* Documented "sg-id" parameter.

* Renamed "copied" to "data-id" so that it can be used by both agents. An OPES processor uses named "Copied: data-id" parameter and a callout server uses anonymous "data-id" parameter (instead of previously documented "copy-am-offset").

* Removed "rid" parameter from Negotiation Offer (NO) message as unused.

* Removed "size" parameter from messages with payload since payload syntax includes an explicit size value.

* Renamed Data Have (DH) message to Data Use Mine (DUM) message to preserve the symmetry with Data Use Yours (DUY) message and to prepare for possible addition of Data Check Mine (DCM) message.

* Finished phasing out the "modified" message parameter.

* Added an "As-is" named-parameter to mark adapted pieces of data identical to the original.

* Replaced a huge "message nesting" figure with a set of short specific rules illustrating the same concept. Added a new "Exchange Patterns" subsection to accommodate the rules and related matters. The figure was not clear enough. Hopefully, the rules are.

head-sid10

* Removed the concept of OCP connection as a group of messages sharing the same group of callout services. Now there is no difference between OCP connection and transport connection.

* Added a concept of a Service Group, which is a list of services with an identifier, for now. A given Service Group is referenced by the creating/destroying side only, to prevent destruction synchronization.

* Removed Connection Services (CSvc) message.

* Removed connection priority until proven generally useful. Can be implemented as an extension.

head-sid9

* Added Negotiation and Capability Inquiry sections.

* Deleted data-end message because AME (Application Message End) already does the same thing and because there is no data-start message.

* Deleted meta-* messages. Data-* messages are now used for both metadata and data since OCP does not know the difference, but must provide the same exchange mechanism for both.

* Use a single message name (short or long, depending on the message) instead of using full and abbreviated versions and trying to enforce abbreviations on the wire. Be more consistent in creating short message names.

* Resurrected OCP scope figure based on popular demand.

* Applied Martin Stecher comments dated 2003/05/30.

head-sid8

* Added structure and list values to ABNF syntax.

* Messages with multiple equally-named parameters are semantically invalid.

* Added types for message parameters.

* Started replacing complicated, error-prone, and probably mostly useless "modified" parameter with a clear and simple "as-is"

parameter.

* Converted parameter descriptions from list items to subsections.

* OCP syntax requires one or two character lookups to determine the next message part. Fixed a comment for implementors saying that one lookup is always sufficient.

head-sid7

* Mentioned TCP/IP/Internet as assumed transport/network, with any other reliable connection-oriented transport/network usable as well. We do not document how OCP messages are mapped to TCP but it should be obvious. See Overall Operation section.

* Applied Martin Stecher's corrections to OCP message syntax and definitions of messages.

* Restricted full message name use to documentation, debuggers, and such. The differences in abbreviated and full name usage still need more consideration and polishing.

* IAB Considerations section now refers to the future opes-iab draft.

head-sid6

* Added OCP message syntax. Reformatted message descriptions to match new syntax concepts.

* Started adding meta-have message to exchange metadata details. Removed negotiation messages for now (posted new messages to the list for a discussion).

* Added Security Considerations section (based on Abbie Barbir's original text).

head-sid4

* Changed document labels to reflect future "WG draft" status.

* Added Acknowledgments section.

* Added "Core" to the title since we expect application specific drafts to follow and because this document, even when complete, cannot specify a "working" protocol without application-specific parts. This change is still debatable.

* Added reference to required future application-specific specs in the Introduction.

* Moved all rant about irrelevance of application protocols proxied by an OPES processor to the "Application proxies and OCP scope" section. Removed "processor input" and "processor output" terms. No reason to define a new term when its only purpose is to document irrelevance?

* Moved "OCP message" definition to the terminology section.

* Clarified "application message" definition based on recent WG discussions and suggestions. There seems to be consensus that "application message" is whatever OPES processor and callout server define or agree on, but OCP needs some minimal structure (content + metadata)

* Synced data and metadata definitions with the new "application message" definition.

* Simplified "Overall Operation" section since it no longer need to talk about irrelevance of application protocols proxied by an OPES processor.

* Illustrated nesting/relationship of key OCP concepts (application message, OCP message, transaction, connection, transport connection, etc.). The figure needs more work.

* Listed all from-processor and from-server OCP messages in one place, with references to message definitions.

* Added "services" message parameter, assuming that more than one service may be requested/executed with one transaction.

* Gave callout server ability to report what services were actually applied (see "services" parameter definition).

head-sid3

* clarified application message definition and OCP boundaries by introducing three kinds of "applications": processor input, processor output, and OCP application

* made "Overall Operation" a top-level section since it got long and has its own subsections now; lots of editorial changes in this sections, new figures

* added illustrations of OCP messages, transactions, and

connections

head-sid2

* introduced a notion of meta-data to both simplify OCP and make OCP agnostic to application meta-data; previous approach essentially assumed existence of a few common properties like protocol name or application message source/destination while not allowing any other properties to be exchanged between OCP agents); specific meta-data format/contents is not important to OCP but OCP will help agents to negotiate that format/contents

* removed wording implying that OCP adapts application messages; OCP only used to exchange data and meta-data (which facilitates adaptation)

* changed most of the definitions; added definitions for meta-data, original/adapted flows, and others

* split 'data-pause' message into 'data-pause' request by the callout server and 'data-paused' notification by the OPES processor; fixed "paused" state management

* added motivation for data acking mechanism

* replaced "am-proto", "am-kind", "am-source", and "am-destination" parameters with "meta-data"

* replaced SERVER and CLIENT placeholders with "callout server" and "OPES processor"

* added editing marks

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[I-D.ietf-opes-architecture] Barbir, A., "An Architecture for Open Pluggable Edge Services (OPES)", draft-ietf-opes-architecture-04 (work in progress), December 2002.

Informative References

[I-D.ietf-opes-protocol-reqs]
Beck, A., "Requirements for OPES Callout Protocols", draft-ietf-opes-protocol-reqs-03 (work in progress), December 2002.

[I-D.ietf-opes-threats]
Barbir, A., "Security Threats and Risks for Open", draft-ietf-opes-threats-02 (work in progress), September 2003.

[I-D.ietf-opes-scenarios]
Barbir, A., "OPES Use Cases and Deployment Scenarios", draft-ietf-opes-scenarios-01 (work in progress), August 2002.

[I-D.ietf-opes-iab]
Barbir, A. and A. Rousskov, "OPES Treatment of IAB Considerations", draft-ietf-opes-iab-03 (work in progress), October 2003.

[I-D.ietf-opes-http]
Rousskov, A. and M. Stecher, "HTTP adaptation with OPES", draft-ietf-opes-http-01 (work in progress), October 2003.

[I-D.ietf-fax-esmtp-conneg]
Toyoda, K. and D. Crocker, "SMTP Service Extension for Fax Content Negotiation", draft-ietf-fax-esmtp-conneg-08 (work in progress), June 2003.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[RFC3238] Floyd, S. and L. Daigle, "IAB Architectural and Policy Considerations for Open Pluggable Edge Services", RFC 3238, January 2002.

Author's Address

Alex Rousskov
The Measurement Factory

EMail: rousskov@measurement-factory.com
URI: http://www.measurement-factory.com/

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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