        Open Pluggable Edge Services (OPES) Callout Protocol (OCP) Core

Status of This Memo

Copyright Notice

Abstract

   This document specifies the core of the Open Pluggable Edge Services
   (OPES) Callout Protocol (OCP). OCP marshals application messages
   from other communication protocols: An OPES intermediary sends
   original application messages to a callout server; the callout server
   sends adapted application messages back to the processor. OCP is
   designed with typical adaptation tasks in mind (e.g., virus and spam
   management, language and format translation, message anonymization,
   or advertisement manipulation). As defined in this document, the OCP
   Core consists of application-agnostic mechanisms essential for
   efficient support of typical adaptations.

Table of Contents

1. Introduction

   The Open Pluggable Edge Services (OPES) architecture [RFC3835]
   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 callout servers. As described in
   [RFC3836], an OPES processor invokes and communicates with services
   on a callout server by using an OPES callout protocol (OCP). This
   document specifies the core of that protocol ("OCP Core").

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

   Section 1.2 provides a brief overview of the entire OPES document
   collection, including documents describing OPES use cases and
   security threats.

1.1. Scope

   The OCP Core specification documents the behavior of OCP agents and
   the requirements for OCP extensions. OCP Core does not contain
   requirements or mechanisms specific for application protocols being
   adapted.

   As an application proxy, the 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 an OPES processor
   takes application messages being proxied, marshals them over OCP to
   callout servers, and then puts the adaptation results back on the
   wire. However, this assumption implies that OCP is applied directly
   to application messages that OPES processor is proxying, which may
   not be the case.

```
   OPES processor scope                        callout server scope
   +-----------------+                       +-----------------+
   | pre-processing  |        OCP scope      |                 |
   |           +- - - - - - - - - - - - - - - - -+            |
   | iteration  |    <== ( application data ) ==>   | adaptation |
   |           +- - - - - - - - - - - - - - - - -+            |
   | post-processing |                       |                 |
   +-----------------+                       +-----------------+
```
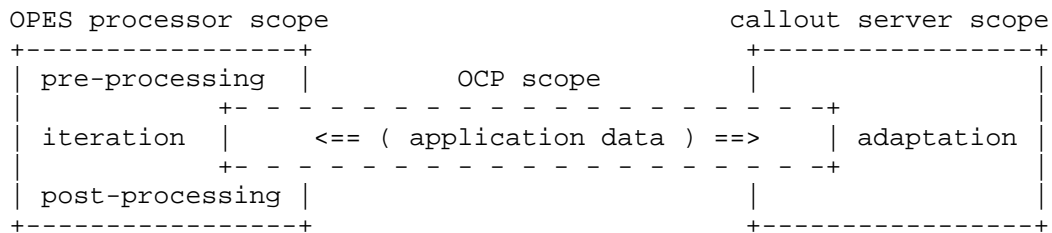
   An OPES processor may preprocess (or postprocess) proxied application
   messages before (or after) they are adapted at callout servers. For
   example, a processor may take an HTTP response being proxied and pass
   it as-is, along with metadata about the corresponding HTTP
   connection. Another processor may take an HTTP response, extract its
   body, and pass that body along with the content-encoding metadata.
   Moreover, to perform adaptation, the 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 OCP Core scope.

   This specification does not define or require any specific
   relationship among application messages being proxied by an OPES
   processor and application messages being exchanged between an OPES
   processor and a callout server via OCP. The OPES processor usually
   provides some mapping among these application messages, but the
   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 may 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. OPES Document Map

   This document belongs to a large set of OPES specifications produced
   by the IETF OPES Working Group. Familiarity with the overall OPES
   approach and typical scenarios is often essential when one tries to
   comprehend isolated OPES documents. This section provides an index
   of OPES documents to assist the reader with finding "missing"
   information.

   o "OPES Use Cases and Deployment Scenarios" [RFC3752] describes a
      set of services and applications that are considered in scope for
      OPES and that have been used as a motivation and guidance in
      designing the OPES architecture.

   o The OPES architecture and common terminology are described in "An
      Architecture for Open Pluggable Edge Services (OPES)" [RFC3835].

   o "Policy, Authorization, and Enforcement Requirements of OPES"
      [RFC3838] outlines requirements and assumptions on the policy
      framework, without specifying concrete authorization and
      enforcement methods.

   o "Security Threats and Risks for OPES" [RFC3837] provides OPES risk
      analysis, without recommending specific solutions.

   o "OPES Treatment of IAB Considerations" [RFC3914] addresses all
      architecture-level considerations expressed by the IETF Internet
      Architecture Board (IAB) when the OPES WG was chartered.

   o At the core of the OPES architecture are the OPES processor and
      the callout server, two network elements that communicate with
      each other via an OPES Callout Protocol (OCP). The requirements
      for this protocol are discussed in "Requirements for OPES Callout
      Protocols" [RFC3836].

   o This document specifies an application agnostic protocol core to
      be used for the communication between an OPES processor and a
      callout server.

   o "OPES Entities and End Points Communications" [RFC3897] specifies
      generic tracing and bypass mechanisms for OPES.

   o The OCP Core and communications documents are independent from the
      application protocol being adapted by OPES entities. Their
      generic mechanisms have to be complemented by application-specific
      profiles. "HTTP Adaptation with OPES" [OPES-HTTP] is such an
      application profile for HTTP. It specifies how
      application-agnostic OPES mechanisms are to be used and augmented
      in order to support adaptation of HTTP messages.

   o Finally, "P: Message Processing Language" [OPES-RULES] defines a
      language for specifying what OPES adaptations (e.g., translation)
      must be applied to what application messages (e.g., e-mail from
      bob@example.com). P language is intended for configuring
      application proxies (OPES processors).

1.3. Terminology

   In this document, 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
   constitute normal prose usage, with no normative implications.

   The 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 such as "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: A basic unit of communication between an OPES processor
      and a callout server. The 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).

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

   data: Same as application message data.

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

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

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

   agent: The actor for a given communication protocol. The 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
      for adaptation of an application protocol (a.k.a., application
      profile; e.g., [OPES-HTTP]), new OCP functionality (e.g.,
      transport encryption and authentication), and/or new OCP Core
      version.

2. Overall Operation

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

2.1. Initialization

   The OPES processor establishes transport connections with callout
   servers to exchange application messages with the callout server(s)
   by 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 may
   negotiate transport encryption and application message definition.

   When enough settings are negotiated, OCP agents may start exchanging
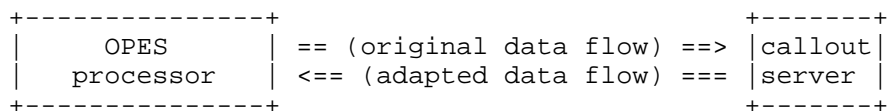   application messages.

   OCP Core provides negotiation and other mechanisms for agents to
   encrypt OCP connections and authenticate each other. OCP Core does
   not require OCP connection encryption or agent authentication.
   Application profiles and other OCP extensions may document and/or
   require these and other security mechanisms. OCP is expected to be
   used, in part, in closed environments where trust and privacy are
   established by means external to OCP. Implementations are expected
   to demand necessary security features via the OCP Core negotiation
   mechanism, depending on agent configuration and environment.

2.2. Original Dataflow

   When the OPES processor wants to adapt an application message, it
   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 the application message
   scope is established, application data can be sent to the callout
   server by using Data Use Mine (DUM) and related OCP message(s). All
   of these messages correspond to the original dataflow.

2.3. Adapted Dataflow

   The callout server receives data and metadata sent by the OPES
   processor (original dataflow). The callout server analyses metadata
   and adapts data as it comes in. The server usually builds its
   version of metadata and responds to the 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"
   by using an Application Message End (AME) message. The transaction
   may be closed by using a Transaction End (TE) message, as well. All
   these messages correspond to adapted data flow.

```
       +---------------+                          +-------+
       |     OPES      | == (original data flow) ==> |callout|
       |   processor   | <== (adapted data flow) === |server |
       +---------------+                          +-------+
```

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

2.4. Multiple Application Messages

   OCP Core specifies a transactions interface dedicated to exchanging a
   single original application message and a single adapted application
   message. Some application protocols may require multiple adapted
   versions for a single original application message or even multiple
   original messages to be exchanged as a part of a single OCP
   transaction. For example, a single original e-mail message may need
   to be transformed into several e-mail messages, with one custom
   message for each recipient.

   OCP extensions MAY document mechanisms for exchanging multiple
   original and/or multiple adapted application messages within a single
   OCP transaction.

2.5. Termination

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

2.6. Message 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 [RFC3835] describes
   configuration possibilities.

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

   o An OCP agent may communicate with multiple OCP agents. This is
      outside the scope of this specification.

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

o 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).

o 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).

o 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.

o In OCP Core, a transaction is associated with a single original
   and a single adapted application message. OCP Core extensions may
   extend transaction scope to more application messages.

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

2.7. Timeouts

   OCP violations, resource limits, external dependencies, and other
   factors may lead to states in which an OCP agent is not receiving
   required messages from the other OCP agent. OCP Core defines no
   messages to address such situations. In the absence of any extension
   mechanism, OCP agents must implement timeouts for OCP operations. An
   OCP agent MUST forcefully terminate any OCP connection, negotiation,
   transaction, etc. that is not making progress. 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 with 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.

2.8. Environment

   OCP communication is assumed usually to take place over TCP/IP
   connections on the Internet (though no default TCP port is assigned
   to OCP in this specification). This does not preclude OCP from being
   implemented on top of other transport protocols, or on other
   networks. High-level transport protocols such as BEEP [RFC3080] may
   be used. OCP Core requires a reliable and message-order-preserving
   transport. Any protocol with these properties 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 Core is application agnostic. OCP messages can carry
   application-specific information as a payload or as
   application-specific message parameters.

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

3. Messages

   As defined in section 1.3, 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, transaction management, and
   application data exchange between a single OPES processor and a
   single callout server. Some messages can be emitted only by an OPES
   processor; some only by a callout server; and some 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 some may not require a response.

3.1. Message Format

   An OCP message consists of a message name followed by optional
   parameters and a 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 ":" *OCTET   ; exactly size octets

   safe-OCTET = ALPHA / DIGIT / "-" / "_"
   size = dec-number   ; 0-2147483647
   dec-number = 1*DIGIT   ; no leading zeros or signs

   Several normative rules accompany the above ABNF:

   o 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.

   o All protocol elements are case sensitive unless it is specified
      otherwise. In particular, message names and parameter names are
      case sensitive.

   o Sizes are interpreted as decimal values and cannot have leading
      zeros.

   o Sizes do not exceed 2147483647.

   o 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.

   o Empty quoted values are encoded as a 4-octet sequence "0:".

   o Any bare value can be encoded as a quoted value. A quoted value
      is 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.

   o Unicode UTF-8 is the default 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 governing processing of invalid messages.

3.2. Message Rendering

   OCP message samples in this specification and its extensions 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 line break"
   character, is used to emphasize a protocol-violating
   presentation-only linebreak. Bare backslashes are prohibited by OCP
   syntax. Similarly, an "\r\n" string is sometimes used to emphasize
   the presence of a CRLF sequence, usually before OCP message payload.
   Normally, the 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;
   DWP 22 16;
   x-doit "5:xyzzy";

   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 ({"32:http://www.iana.org/assignments/opes/ocp/tls"});
   NO ({"54:http://www.iana.org/assignments/opes/ocp/http/response"
   Optional-Parts: (request-header)
   },{"54:http://www.iana.org/assignments/opes/ocp/http/response"
   Optional-Parts: (request-header,request-body)
   Transfer-Encodings: (chunked)
   });

   Optional parameters and extensions are possible with a named
   parameters approach, as illustrated by the following example. The
   DWM (section 11.17) message in the example has two anonymous
   parameters (the last one being an extension) and two named parameters
   (the last one being an extension).

   DWM 1 3
   Size-Request: 16384
   X-Need-Info: "26:twenty six octet extension";

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

   DUM 1 13
   Modp: 75
   \r\n
   8865:... 8865 octets of raw data ...;

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
   (such as this specification or the 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

   An 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 the 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 the 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.

   An 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. A TE message
   can be sent by either side. Zero or more OCP messages associated
   with the transaction can be exchanged in between. The figure below
   illustrates a possible message sequence (prefix "P" stands for the
   OPES processor; prefix "S" stands for the 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;

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. In this context, "Invalid input" means
   messages or message parts that violate at least one of the normative
   rules. A message with an invalid part is, by definition, invalid.
   If OCP agent resources are exhausted while parsing or interpreting a
   message, the agent MUST treat the corresponding OCP message as
   invalid.

   Unless explicitly allowed to do otherwise, an OCP agent MUST
   terminate the transaction if it receives an invalid message with
   transaction scope and MUST terminate the connection if it receives an
   invalid message with a connection scope. A terminating agent MUST
   use the result status code of 400 and MAY specify termination cause
   information in the result status reason parameter (see section
   10.10). 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 for which 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 (making unusable) the host page while performing that
   insertion or removal. Most OPES adaptations are high level in
   nature, which makes it impossible to assess correctness of the
   adaptations automatically, especially if "robustness guesses" are
   involved.

6. Negotiation

   The negotiation mechanism allows OCP agents to agree on the mutually
   acceptable set of features, including optional and
   application-specific behavior and 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 the counterpart's behavior, see the
   Ability Query (AQ) and Ability Answer (AA) message definitions.

   Most negotiations require at least one round trip time delay. In
   rare cases when the other side's response is not required
   immediately, negotiation delay can be eliminated, with an inherent
   risk of an overly optimistic assumption about the negotiation
   response.

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 at most one feature that it prefers. The selection is sent by
using a Negotiation Response (NR) message. If the response is
positive, both sides assume that the selected feature is in effect
immediately (see section 11.19 for details). 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 an SMTP application profile
enabled, as 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 (unless a negotiated feature explicitly allows
for changing an encryption scheme on the fly).

Negotiation Offer (NO) messages may be sent by either agent. OCP
extensions documenting negotiation MAY assign the 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 the other to make an offer.

As 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 11.18) ensure
that only one offer per OCP connection is honored at a time, and that
the other concurrent offers are ignored by both agents.

6.1. Negotiation Phase

A Negotiation Phase is a mechanism ensuring 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, the 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 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. Agent
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), Progress Query (PQ), Progress Answer (PA), Progress Report
(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 profile for OCP Core. Thus, these
bootstrapping requirements are unlikely to result in any extra work.

Once a Negotiation Phase starts, an agent MUST expect further
negotiations if and only if the last NO sent or the last NR received
contained a 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
is 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 has not negotiated.
   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 seeks to stop the above negotiations after sending a
true "Offer-Pending" value, its 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 time out the connection.

The following example shows a dialog with a callout server that
insists on enabling two imaginary features: strong transport
encryption and volatile storage for responses. The server is
designed not to exchange sensitive messages until both features are
enabled. Naturally, the volatile storage feature has to be
negotiated securely. The OPES processor supports one of the strong
encryption mechanisms but prefers not to offer (to volunteer support
for) strong encryption, perhaps for performance reasons. The server
has to send a true "Offer-Pending" parameter to get a chance to offer
strong encryption (which is successfully negotiated in this case).
Any messages sent by either agent after the (only) successful NR
response are encrypted with "strongB" encryption scheme. The OPES
processor does not understand the volatile storage feature, and the
last negotiation fails (over a strongly encrypted transport
connection).

```
P: NO ({"29:ocp://example/encryption/weak"})
   ;
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"}
   ;
... all traffic below is encrypted using strongB ...
```

```
   S: NO ({"31:ocp://example/storage/volatile"})
      Offer-Pending: false
      ;
   P: NR
      Unknowns: ({"31:ocp://example/storage/volatile"})
      ;
   S: CSE { 400 "33:lack of VolStore protocol support" }
      ;
```

   The following example from [OPES-HTTP] illustrates successful HTTP
   application profile negotiation:

```
   P: NO ({"54:http://www.iana.org/assignments/opes/ocp/http/response"
      Aux-Parts: (request-header,request-body)
      })
      SG: 5;
   S: NR {"54:http://www.iana.org/assignments/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., HTTP cookies filtering or ad
   insertion). Yet, in many cases, the callout service has 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 eliminate the return trip if
   both OCP agents cooperate. Such cooperation is optional: OCP agents
   MAY support data preservation optimization.

   To avoid sending back unmodified data, a callout service has to know
   that the OPES processor has a copy of the data. As data sizes can be
   very large and the callout service may not know in advance whether it
   will be able to use 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 an OPES 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, therefore, the
return trip is eliminated.

As the mapping between original and adapted data is not known to the
processor, the processor MUST keep the announced-as-preserved chunk
until the end of the corresponding transaction, unless the callout
server explicitly tells the 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, for instance,
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, but 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 becomes full. This 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
made a decision not to send Data Use Yours (DUY) messages (for a
specific data ranges or at all) SHOULD immediately inform the OPES
processor of that decision with the corresponding Data Preservation
Interest (DPI) message(s) or other mechanisms.

8. 'Premature Dataflow Termination' Optimizations

Many callout services adapt small portions of large messages and
would preferably not to be in the loop when that adaptation is over.
Some callout services may not seek data modification and would
preferably not send data back to the OPES processor, even if the OPES
processor is not supporting the data preservation optimization
(Section 7). By OCP design, unilateral premature dataflow
termination by a callout server would lead to termination of an OCP
transaction with an error. Thus, the two agents must cooperate to
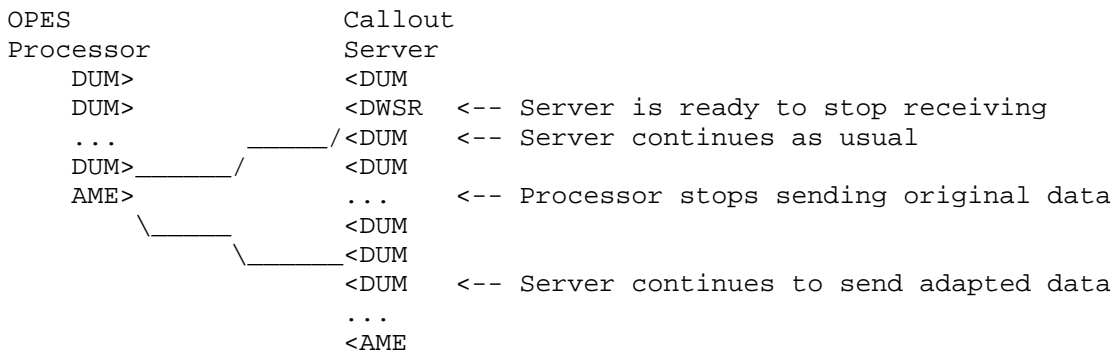allow for error-free premature termination.

This section documents two mechanisms for premature termination of
original or adapted dataflow. In combination, the mechanisms allow
the callout server to get out of the processing loop altogether.

8.1. Original Dataflow

There are scenarios where a callout server is not interested in the
remaining original dataflow. For example, a simple access blocking
or "this site is temporary down" callout service has to send an
adapted (generated) application message but would preferably not
receive original data from the OPES processor.

OCP Core supports premature original dataflow termination via the
Want Stop Receiving Data (DWSR) message. A callout server that does
not seek to receive additional original data (beyond a certain size)
sends a DWSR message. The OPES processor receiving a DWSR message
terminates original dataflow by sending an Application Message End
(AME) message with a 206 (partial) status code.

The following figure illustrates a typical sequence of events. The
downward lines connecting the two dataflows illustrate the
transmission delay that allows for more OCP messages to be sent while
an agent waits for the opposing agent reaction.

```
OPES                  Callout
Processor             Server
    DUM>                  <DUM
    DUM>                  <DWSR  <-- Server is ready to stop receiving
    ...          _____/<DUM    <-- Server continues as usual
    DUM>_____/           <DUM
    AME>                  ...    <-- Processor stops sending original data
       _____            <DUM
              _____     <DUM
                          <DUM   <-- Server continues to send adapted data
                          ...
                          <AME
```

The mechanism described in this section has no effect on the adapted
dataflow. Receiving an Application Message End (AME) message with
206 (partial) result status code from the OPES processor does not
introduce any special requirements for the adapted dataflow
termination. However, it is not possible to terminate adapted
dataflow prematurely after the original dataflow has been prematurely
terminated (see section 8.3).

8.2. Adapted Dataflow

   There are scenarios where a callout service may want to stop sending
   adapted data before a complete application message has been sent.
   For example, a logging-only callout service has to receive all
   application messages but would preferably not send copies back to the
   OPES processor.

   OCP Core supports premature adapted dataflow termination via a
   combination of Want Stop Sending Data (DWSS) and Stop Sending Data
   (DSS) messages. A callout service that seeks to stop sending data
   sends a DWSS message, soliciting an OPES processor permission to
   stop. While waiting for the permission, the server continues with
   its usual routine.

   An OPES processor receiving a Want Stop Sending Data message responds
   with a Stop Sending Data (DSS) message. The processor may then pause
   to wait for the callout server to terminate the adapted dataflow or
   may continue sending original data while making a copy of it. Once
   the server terminates the adapted dataflow, the processor is
   responsible for using original data (sent or paused after sending
   DSS) instead of the adapted data.

   The callout server receiving a DSS message terminates the adapted
   dataflow (see the Stop Sending Data (DSS) message definition for the
   exact requirements and corner cases).

   The following figure illustrates a typical sequence of events,
   including a possible pause in original dataflow when the OPES
   processor is waiting for the adapted dataflow to end. The downward
   lines connecting the two dataflows illustrate the transmission delay
   that allows for more OCP messages to be sent while an agent waits for
   the opposing agent reaction.

```
    OPES                   Callout
    Processor              Server
        DUM>                   <DUM
        DUM>                   <DWSS     <-- Server is ready to stop sending
        ...           _____/<DUM       <-- Server continues as usual,
        DUM>_____/           <DUM          waiting for DSS
        DSS>                   ...
              _____          <DUM
      possible    _____<DUM
      org-dataflow          <AME 206 <-- Server terminates adapted dataflow
      pause          _____/              upon receiving the DSS message
         _____/
        DUM>                             <-- Processor resumes original dataflow
        DUM>                                 to the server and starts using
        ...                                  original data without adapting it
        AME>
```

   Premature adapted dataflow preservation is not trivial, as the OPES
   processor relies on the callout server to provide adapted data
   (modified or not) to construct the adapted application message. If
   the callout server seeks to quit its job, special care must be taken
   to ensure that the OPES processor can construct the complete
   application message. On a logical level, this mechanism is
   equivalent to switching from one callout server to another
   (non-modifying) callout server in the middle of an OCP transaction.

   Other than a possible pause in the original dataflow, the mechanism
   described in this section has no effect on the original dataflow.
   Receiving an Application Message End (AME) message with 206 (partial)
   result status code from the callout server does not introduce any
   special requirements for the original dataflow termination.

8.3. Getting Out of the Loop

   Some adaptation services work on application message prefixes and do
   not seek to be in the adaptation loop once their work is done. For
   example, an ad insertion service that did its job by modifying the
   first fragment of a web "page" would not seek to receive more
   original data or to perform further adaptations. The 'Getting Out of
   the Loop' optimization allows a callout server to get completely out
   of the application message processing loop.

   The "Getting Out of the Loop" optimization is made possible by
   terminating the adapted dataflow (section 8.2) and then by
   terminating the original dataflow (section 8.1). The order of
   termination is very important.

If the original dataflow is terminated first, the OPES processor
would not allow the adapted dataflow to be terminated prematurely, as
the processor would not be able to reconstruct the remaining portion
of the adapted application message. The processor would not know
which suffix of the remaining original data has to follow the adapted
parts. The mapping between original and adapted octets is known only
to the callout service.

An OPES processor that received a DWSS message followed by a DWSR
message MUST NOT send an AME message with a 206 (partial) status code
before sending a DSS message. Informally, this rule means that a
callout server that wants to get out of the loop fast should send a
DWSS message immediately followed by a DWSR message; the server does
not have to wait for the OPES processor's permission to terminate
adapted dataflow before requesting that the OPES processor terminates
original dataflow.

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 supported by PETDM: message parameter values
and messages.

All OCP Core parameter and message types are declared by using PETDM.
OCP extensions SHOULD 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 in PETDM to extend base types semantics by
using the following declaration templates. The new semantics rules
are meant to be attached to each declaration using prose text.

Text 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 it is explicitly noted otherwise, empty lists are valid and
have the semantics of an 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 has 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 as when a structure
is extended, the new message may have anonymous parameters and named
parameters. Neither group has 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.

o Extending type semantics without renaming the type:
<old-type-name>: extends <namespace>::<old-type-name>;
The above template can be used by OCP Core extensions that seek to
change the semantics of OCP Core types without renaming them. This
technique is essential for extending OCP messages because the message
name is the same as the message type name. For example, an SMTP
profile for OCP might use the following declaration to extend an
Application Message Start (AMS) message with Am-Id, a parameter
defined in that profile:

AMS: extends Core::AMS with {
        Am-Id: am-id;
};

All extended types may be used as replacements of the types they
extend. For example, a Negotiation Offer (NO) message uses a
parameter of type Features. Features (section 10.12) is a list of
feature (section 10.11) items. A Feature is a structure-based type.
An OCP extension (e.g., an HTTP application profile) 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.

The OCP Core namespace tag is "Core". OCP extensions that declare
types MUST define their namespace tags (so that other extensions and
documentation can use them in their PETDM declarations).

9.1. Optional Parameters

Anonymous parameters are positional: The parameter's position (i.e.,
the number of anonymous parameters to the left) is its "name". Thus,
when a structure or message has 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 an anonymous parameter is added to a structure or message that
    has optional anonymous parameters, the new parameter has to be
    optional and can only be used if all old optional parameters are in
    use. Named parameters do not have these limitations, as they are not
    positional, but associative; they are identified by their explicit
    and 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 the type of every new protocol element 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. Uni semantics is incomplete without the scope
    specification. Many uri parameters are URLs. Unless it is noted
    otherwise, URL identifiers do not imply the 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] range, inclusive.

    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 within the same scope. 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 transaction identifier "xid" must not
be used for a new Transaction Start (TS) message within the same OCP
transaction, even if a prior Transaction End (TE) message was sent
for the same transaction.

An OCP agent MUST terminate the state associated with uni uniqueness
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] range, inclusive.

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

OCP Core cannot handle application messages that exceed 2147483647
octets in size, that require larger sizes as a part of OCP marshaling
process, or that 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] range, inclusive.

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] range, inclusive.

Percent semantics is incomplete unless its value is associated 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
were 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 with two valid values: true and false. A
boolean parameter expresses the truthfulness of the associated
statement.

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. modp

modp: extends percent;

Modp extends the percent type to express the sender's 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 the numerical value of at least
one data octet.

10.10. result

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

The 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 a 200|
|        |              | status code recipient.                   |
|   206  |   partial    | Partial success.  This status code is    |
|        |              | documented for Application Message End   |
|        |              | (AME) messages only.  The code indicates |
|        |              | that the agent terminated the            |
|        |              | corresponding dataflow prematurely (i.e.,|
|        |              | more data would be needed to reconstruct |
|        |              | a complete application message).         |
|        |              | Premature termination of one dataflow    |
|        |              | does not introduce any special           |
|        |              | requirements for the other dataflow      |
|        |              | termination.  See dataflow termination   |
|        |              | optimizations (section 8) for use cases. |
|   400  |   failure    | An error, exception, or trouble.  A      |
|        |              | recipient of a 400 (failure) result of an|
|        |              | AME, TE, or CE message MUST destroy any  |
|        |              | state or data associated with the        |
|        |              | corresponding dataflow, transaction, or  |
|        |              | connection.  For example, an 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 made possible by adding new "result"
structure members or by 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)
   were 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) were received.

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

   In this specification, an OCP message with result status code of 400
   (failure) is called "a message indicating a failure".

10.11. feature

   feature: extends structure with {
           uri;
   };

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

   This specification does not define any features.

10.12. features

   features: extends list of feature;

   Features is a list of feature values. Unless it is noted otherwise,
   the list can be empty, and features are listed in decreasing
   preference order.

10.13. 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, if
   there are any, as additional "service" structure members. For
   example, a service value may look like this:

```
    {"41:http://www.iana.org/assignments/opes/ocp/tls" "8:blowfish"}
```

10.14. services

    services: extends list of service;

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

10.15. Dataflow Specializations

    Several parameter types, such as offset apply to both original and
    adapted dataflow. It is relatively easy to misidentify a type's
    dataflow affiliation, especially when parameters with different
    affiliations are mixed together in one message declaration. The
    following statements declare new dataflow-specific types by 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 the sender's dataflow only:

    my-<type>: extends <type>;

    The following new types refer to the recipient's dataflow only:

    your-<type>: extends <type>;

    OCP Core uses the above type-naming scheme to implement dataflow
    specialization for the following types: offset, size, and 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. OCP extensions MUST NOT introduce
    order-dependent named parameters. 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 the limitations of the recipient's
resources.

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 corresponds to an inactive state
(i.e., that was either never created or already destroyed). Such
identifiers invalidate the host OCP 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 it sends 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 the 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];
};
```

A Connection End (CE) Indicates the 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's 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. Service Group Created (SGC)

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

A Service Group Created (SGC) message informs the recipient that a
list of adaptation services has been associated with the given
service group identifier ("my-sg-id"). Following this message, the
sender can refer to the group by using the identifier. The recipient
MUST maintain the association until a matching Service Group
Destroyed (SGD) 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
Service Group Created (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. Service Group Destroyed (SGD)

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

   A Service Group Destroyed (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 Transaction Start (TS) message indicates
   the start of an OCP transaction. Upon receiving 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 an a Service Group Created (SGC) message. The callout server
   MUST apply the list of services associated with "my-sg-id", in the
   specified order.

   This message introduces the transaction identifier (xid).

11.6. Transaction End (TE)

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

   A Transaction End (TE) indicates the end of the identified OCP
   transaction.

   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.

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

11.7. Application Message Start (AMS)

```
AMS: extends message with {
        xid;
        [Services: services];
};
```

An Application Message Start (AMS) message indicates the start of the
original or adapted application message processing and dataflow. The
recipient MAY refuse further processing by sending an Application
Message End (AME) message indicating a failure.

When an AMS message is sent by the OPES processor, the callout server
usually sends an AMS message back, announcing the creation of an
adapted version of the original application message. This
announcement may be delayed. For example, the callout server may
wait for more information from the OPES processor.

When an AMS message is sent by the callout server, an optional
"Services" parameter describes OPES services that the server MAY
apply to the original application message. Usually, the "services"
value matches what was asked by the OPES processor. The callout
server SHOULD send a "Services" parameter if its value would differ
from the list of services requested by the OPES processor. As the
same service may be known under many names, the mismatch does not
necessarily imply an error.

11.8. Application Message End (AME)

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

An Application Message End (AME) message indicates the end of the
original or adapted application message processing and dataflow. The
recipient should expect no more data for the corresponding
application message.

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

An OCP agent MUST send an Application Message End (AME) message
immediately after it makes a decision to stop processing of its
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-offset;
        [As-is: org-offset];
        [Kept: org-offset org-size ];
        [Modp: modp];
} and payload;
```

A Data Use Mine (DUM) message carries application data. It is the
only OCP Core message with a documented 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 the 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). A
DUM message without payload is invalid. 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 current
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 or
continues). 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) an OPES processor to decrease offset and to
specify a data range not yet fully delivered to the callout server.
OCP Core does not require any relationship between DUM payload and
the "Kept" parameter.

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. The data
   modification prediction includes DUM payload. That is, if the
   attached payload has been modified, the modp value cannot be 0%.

   A callout server SHOULD send an "As-is" parameter if the attached
   data is identical to a fragment at the specified offset in the
   original dataflow. An "As-is" parameter specifying a data fragment
   that has 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. This parameter is meant to allow for
   partial data preservation optimizations without a preservation
   commitment. The preserved data still crosses the connection with the
   callout server twice, but the OPES processor may be able to optimize
   its handling of the data.

   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 org-offset org-size;
   };

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

   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-offset org-size;
   };

   The Data Preservation Interest (DPI) message describes an original
   data chunk by using the first octet offset and size as parameters.
   The chunk is the only area of original data that the 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 at
or 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 an OPES processor preserves some disposable
data, it MAY stop preserving that data. If an OPES processor does
not preserve 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. In this context, the term reusable
stands for callout server interest in reusing the data in the future,
given the OPES processor cooperation.

For example, an offset value of zero and the size value of 2147483647
indicate 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. Want Stop Receiving Data (DWSR)

```
DWSR: extends message with {
        xid org-size;
};
```

The Want Stop Receiving Data (DWSR) message informs OPES processor
that the callout server wants to stop receiving original data any
time after receiving at least an org-size amount of an application
message prefix. That is, the server is asking the processor to
terminate original dataflow prematurely (see section 8.1) after
sending at least org-size octets.

An OPES processor receiving a Want Stop Receiving Data (DWSR) message
SHOULD terminate original dataflow by sending an Application Message
End (AME) message with a 206 (partial) status code.

An OPES processor MUST NOT terminate its data preservation commitment
(section 7) in reaction to receiving a Want Stop Receiving Data
(DWSR) message. Just like with any other message, an OPES processor
may use information supplied by Want Stop Receiving Data (DWSR) to
decide on future preservation commitments.

11.13. Want Stop Sending Data (DWSS)

```
DWSS: extends message with {
        xid;
};
```

The Want Stop Sending Data (DWSS) message informs the OPES processor
that the callout server wants to stop sending adapted data as soon as
possible; the server is asking the processor for permission to
terminate adapted dataflow prematurely (see section 8.2). The OPES
processor can grant this permission by using a Stop Sending Data
(DSS) message.

Once the DWSS message is sent, the callout server MUST NOT
prematurely terminate adapted dataflow until the server receives a
DSS message from the OPES processor. If the server violates this
rule, the OPES processor MUST act as if no DWSS message were
received. The latter implies that the OCP transaction is terminated
by the processor, with an error.

An OPES processor receiving a DWSS message SHOULD respond with a Stop
Sending Data (DSS) message, provided the processor would not violate
DSS message requirements by doing so. The processor SHOULD respond
immediately once DSS message requirements can be satisfied.

11.14. Stop Sending Data (DSS)

```
DSS: extends message with {
        xid;
};
```

The Stop Sending Data (DSS) message instructs the callout server to
terminate adapted dataflow prematurely by sending an Application
Message End (AME) message with a 206 (partial) status code. A
callout server is expected to solicit the Stop Sending Data (DSS)
message by sending a Want Stop Sending Data (DWSS) message (see
section 8.2).

A callout server receiving a solicited Stop Sending Data (DSS)
message for a yet-unterminated adapted dataflow MUST immediately
terminate dataflow by sending an Application Message End (AME)
message with a 206 (partial) status code. If the callout server

already terminated adapted dataflow, the callout server MUST ignore
the Stop Sending Data (DSS) message. A callout server receiving an
unsolicited DSS message for a yet-unterminated adapted dataflow MUST
either treat that message as invalid or as solicited (i.e., the
server cannot simply ignore unsolicited DSS messages).

The OPES processor sending a Stop Sending Data (DSS) message MUST be
able to reconstruct the adapted application message correctly after
the callout server terminates dataflow. This requirement implies
that the processor must have access to any original data sent to the
callout after the Stop Sending Data (DSS) message, if there is any.
Consequently, the OPES processor either has to send no data at all or
has to keep a copy of it.

If a callout server receives a DSS message and, in violation of the
above rules, waits for more original data before sending an
Application Message End (AME) response, a deadlock may occur: The
OPES processor may wait for the Application Message End (AME) message
to send more original data.

11.15. Want Data Paused (DWP)

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

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

The "your-offset" parameter refers to dataflow originating at the OCP
agent receiving the parameter.

If, at the time the Want Data Paused (DWP) message is received, the
recipient has already sent data at the specified offset, the message
recipient MUST stop sending data immediately. Otherwise, the
recipient MUST stop sending data immediately after it sends the
specified offset. 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 are 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.

11.16. Paused My Data (DPM)

```
DPM: extends message with {
       xid;
};
```

The Paused My Data (DPM) message indicates the 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.

11.17. Want More Data (DWM)

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

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

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

The "Size-request" parameter refers to dataflow originating at the
OCP agent receiving the parameter. 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.

11.18. 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 it is possible. The recipient MAY ignore sender
preferences. If the list of features is empty, the negotiation is
bound to fail but remains valid.

Both the OPES processor and the 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 it is 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.

As with other protocol elements, OCP Core extensions may document
additional negotiation restrictions. For example, specification of a
transport security feature may prohibit the use of the SG parameter
in negotiation offers, to avoid situations where encryption is
enabled for only a portion of overlapping transactions on the same
transport connection.

11.19. 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 (a.k.a., positive
response) 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 (negative response) is indicated by omitting the
anonymous "feature" parameter.

The successfully negotiated feature becomes effective immediately.
The sender of a positive response MUST consider the corresponding
feature enabled immediately after the response is sent; the recipient
of a positive response MUST consider the corresponding feature
enabled immediately after the response is received. Note that the
scope of the negotiated feature application may be limited to a
specified service group. The negotiation phase state does not affect
enabling of the feature.

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

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

11.20. Ability Query (AQ)

   AQ: extends message with {
         feature;
   };

   An Ability Query (AQ) message solicits an immediate Ability Answer
   (AA) response. The recipient MUST respond immediately with an 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 and not automated fine-tuning of agent
   behavior and configuration. The latter may be better achieved by the
   OCP negotiation mechanism (section 6).

11.21. Ability Answer (AA)

   AA: extends message with {
         boolean;
   };

An Ability Answer (AA) message expresses the sender's support for a
feature requested via an 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.

11.22. Progress Query (PQ)

    PQ: extends message with {
          [xid];
    };

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

11.23. Progress Answer (PA)

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

    A PA message carries the sender's state. 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 otherwise been 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 the 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 does not 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 the PQ with a PA message. Otherwise, the
agent MUST NOT send the transaction ID. If an agent is working on
the original application message for the specified transaction, the
agent MUST send the Org-Data parameter. If the agent has already
sent or received the Application Message End (AME) message for the
original dataflow, the agent MUST NOT send the Org-data parameter.

Informally, the PA message relays the sender's progress with the
transaction and original dataflow identified by the Progress Query
(PQ) message, provided the transaction identifier is still valid at
the time of the answer. Absent information in the answer indicates
invalid, unknown, or closed transaction and/or original dataflow from
the query recipient's point of view.

11.24. Progress Report (PR)

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

A Progress Report (PR) message carries the sender's state. The
message semantics and associated requirements are identical to those
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 original
application message or related to any valid (current) transaction or
original dataflow.

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 [RFC3914].

13. Security Considerations

This section examines security considerations for OCP. OPES threats
are documented in [RFC3837]

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. These
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 gain 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 Service
Group Created (SGC) message instructs the recipient to maintain an
association between a service group identifier and a list of
services.

Implementations that cannot correctly handle resource exhaustion
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 Service Group Created (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 flows that an OPES processor may be
    instructed to maintain via Application Message Start (AMS)
    messages.

14. IANA Considerations

   The IANA maintains a list of OCP features, including application
   profiles (section 10.11). For each feature, its uri parameter value
   is registered along with the extension parameters (if there are any).
   Registered feature syntax and semantics are documented with PETDM
   notation (section 9).

   The IESG is responsible for assigning a designated expert to review
   each standards-track registration prior to IANA assignment. The OPES
   working group mailing list may be used to solicit commentary for both
   standards-track and non-standards-track features.

   Standards-track OCP Core extensions SHOULD use
   http://www.iana.org/assignments/opes/ocp/ prefix for feature uri
   parameters. It is suggested that the IANA populate resources
   identified by such "uri" parameters with corresponding feature
   registrations. It is also suggested that the IANA maintain an index
   of all registered OCP features at the
   http://www.iana.org/assignments/opes/ocp/ URL or on a page linked
   from that URL.

   This specification defines no OCP features for IANA registration.

15. Compliance

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

   A compliance subject is compliant if it satisfies all applicable
   "MUST" and "SHOULD" requirements. By definition, to satisfy a "MUST"
   requirement means to act as prescribed by the requirement; to satisfy
   a "SHOULD" requirement means either to act as prescribed by the
   requirement or to 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 that all "SHOULD" violations are deliberate. In
   other words, "SHOULD" means "MUST satisfy or MUST have a reason to
   violate". It is expected that compliance claims be accompanied by a

list of unsupported SHOULDs (if any), in an appropriate format,
explaining why the preferred behavior was not chosen.

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

15.1. Extending OCP Core

OCP extensions MUST NOT change the OCP Core message format, as
defined by ABNF and accompanying normative rules in Section 3.1.
This requirement is intended to allow OCP message viewers,
validators, and "intermediary" software to at least isolate and
decompose any OCP message, even a message with semantics unknown to
them (i.e., extended).

OCP extensions are allowed to change normative OCP Core requirements
for OPES processors and callout servers. However, OCP extensions
SHOULD NOT make these changes and MUST require on a "MUST"-level that
these changes are negotiated prior to taking effect. Informally,
this specification defines compliant OCP agent behavior until changes
to this specification (if any) are successfully negotiated.

For example, if an RTSP profile for OCP requires support for offsets
exceeding 2147483647 octets, the profile specification can document
appropriate OCP changes while requiring that RTSP adaptation agents
negotiate "large offsets" support before using large offsets. This
negotiation can be bundled with negotiating another feature (e.g.,
negotiating an RTSP profile may imply support for "large offsets").

As implied by the above rules, OCP extensions may dynamically alter
the negotiation mechanism itself, but such an alternation would have
to be negotiated first, using the negotiation mechanism defined by
this specification. For example, successfully negotiating a feature
might change the default "Offer-Pending" value from false to true.

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, response message,
      or associated state message.

name	title	P	S	tie
CS	Connection Start	X	X	CE
CE	Connection End	X	X	CS
SGC	Service Group Created	X	X	SGD TS
SGD	Service Group Destroyed	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 DSS
DUM	Data Use Mine	X	X	DUY DWP
DUY	Data Use Yours		X	DUM DPI
DPI	Data Preservation Interest		X	DUY
DWSS	Want Stop Sending Data		X	DWSR DSS
DWSR	Want Stop Receiving Data		X	DWSS
DSS	Stop Sending Data	X		DWSS
DWP	Want Data Paused	X	X	DPM
DPM	Paused My Data	X	X	DWP DWM
DWM	Want More Data	X	X	DPM
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 OCP
   states. Some states are maintained across multiple transactions and
   application messages. Some correspond to a single request/response
   dialog; the asynchronous nature of most OCP message exchanges
   requires OCP agents to process other messages while waiting for a
   response to a request and, hence, while maintaining the state of the
   dialog.

   For each state, the table provides the following information:

   state: Short state label.

   birth: Messages creating this state.

   death: Messages destroying this state.

   ID: Associated identifier, if any.

state	birth	death	ID
connection	CS	CE	
service group	SGC	SGD	sg-id
transaction	TS	TE	xid
application message and dataflow	AMS	AME	
premature org-dataflow termination	DWSR	AME	
premature adp-dataflow termination	DWSS	DSS AME	
paused dataflow	DPM	DWM	
preservation commitment	DUM	DPI AME	
negotiation	NO	NR	
progress inquiry	PQ	PA	
ability inquiry	PQ	PA	

Appendix C. Acknowledgements

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

   Special thanks to Marshall Rose for his xml2rfc tool.

16. References

16.1. 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.

   [RFC3835]   Barbir, A., Penno, R., Chen, R., Hofmann, M., and H.
                 Orman, "An Architecture for Open Pluggable Edge Services
                 (OPES)", RFC 3835, August 2004.

16.2. Informative References

   [RFC3836]   Beck, A., Hofmann, M., Orman, H., Penno, R., and A.
                 Terzis, "Requirements for Open Pluggable Edge Services
                 (OPES) Callout Protocols", RFC 3836, August 2004.

   [RFC3837]   Barbir, A., Batuner, O., Srinivas, B., Hofmann, M., and
                 H. Orman, "Security Threats and Risks for Open Pluggable
                 Edge Services (OPES)", RFC 3837, August 2004.

   [RFC3752]   Barbir, A., Burger, E., Chen, R., McHenry, S., Orman,
                 H., and R. Penno, "Open Pluggable Edge Services (OPES)
                 Use Cases and Deployment Scenarios", RFC 3752, April
                 2004.

   [RFC3838]   Barbir, A., Batuner, O., Beck, A., Chan, T., and H.
                 Orman, "Policy, Authorization, and Enforcement
                 Requirements of the Open Pluggable Edge Services
                 (OPES)", RFC 3838, August 2004.

   [RFC3897]   Barbir, A., "Open Pluggable Edge Services (OPES)
                 Entities and End Points Communication", RFC 3897,
                 September 2004.

   [OPES-RULES] Beck, A. and A. Rousskov, "P: Message Processing
                 Language", Work in Progress, October 2003.

   [RFC3914]   Barbir, A. and A. Rousskov, "Open Pluggable Edge
                 Services (OPES) Treatment of IAB Considerations", RFC
                 3914, October 2004.

   [OPES-HTTP]   Rousskov, A. and M. Stecher, "HTTP adaptation with
                 OPES", Work in Progress, January 2004.

   [RFC2616]   Fielding, R., Gettys, J., Mogul, J., Frystyk, 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/

Full Copyright Statement

Intellectual Property

Acknowledgement