TransNexus

OSP Toolkit

Internal Architecture

Release 2.5.5

09 February 2002

OSP Toolkit

Internal Architecture

Release 2.5.5

09 February 2002

Document 0300-1213-0200

TransNexus
1140 Hammond Drive, Building E
Suite 5250
Atlanta, GA 30328
USA

Phone:	+1 770 671 1888
Fax:	+1 770 671 1188
E-mail:	support@transnexus.com

Contents

Introduction

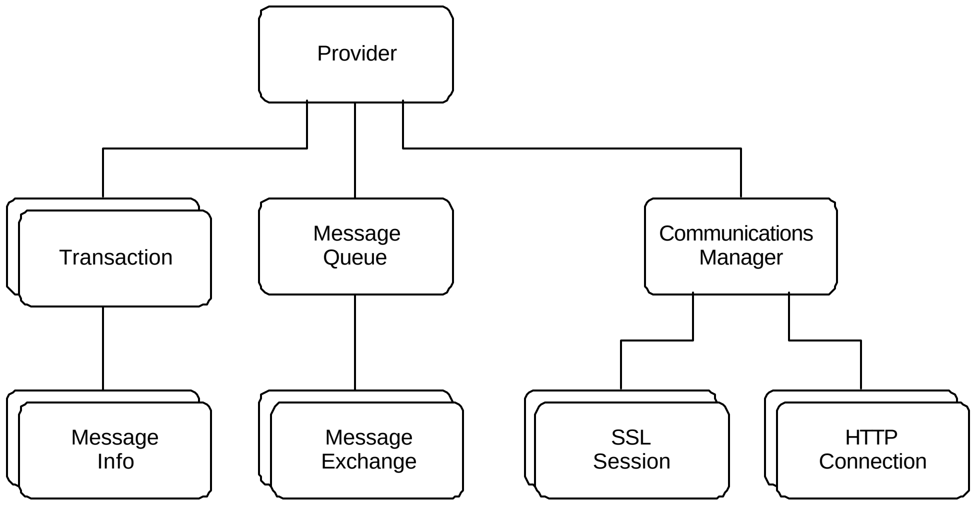
This document describes the internal architecture of release 2.5.5 of the Open Settlement Protocol (OSP) Toolkit. That Toolkit, freely available under license from TransNexus, contains an implementation of the standard settlement protocol endorsed by the European Telecommunications Standards Institute (ETSI) and the International Multimedia Teleconferencing Consortium's Voice over IP (VoIP) Forum. The Toolkit also implements, as an option, extensions to the standard that allow access to enhanced services.

The OSP Toolkit contains eleven separate documents, including this one. The documents are:

- *Introduction*
- *Implementation Guide*
- *How to Build and Test the OSP Toolkit*
- *Errorcode List*
- *Programming Interface*
- *Cisco Interoperability Example*
- *Device Enrollment*
- *Internal Architecture*
- *Porting Guide*
- *Protocol Extensions*
- *ETSI Technical Specification TS 101 321*

The *OSP Toolkit Introduction* includes a "Document Roadmap" section that summarizes the various documents and their application.The sections that follow describe the Toolkit's major components, its formatting and parsing functions, high-level utility services, and platform-specific services. Although implemented entirely in ANSI C, the Toolkit relies an object-oriented architecture. Consequently, the sections of this document are based on the major objects that comprise that architecture.

Note that this document is not intended as exhaustive description of all the objects in the Toolkit library, including details of all their members and member functions. It serves, instead, as an orientation to the design of the library. To avoid any problems that might arise from out-of-date or faulty documentation, library objects are fully documented in the actual source code.



• Figure 1 Major Toolkit Software Objects.

Major Components

At a high level, nine major components perform the Toolkit's essential services. Figure 1 shows those major components, and it illustrates the relationships between them. The following subsections describe each component in greater detail.

> Note:
> Figure 1 does not represent a complete or rigid object hierarchy for the Toolkit library. Rather, it simply shows the major components of the library and their relationship with each other. Other objects, described in later sections of this document, are also part of the library.

Provider

The provider object is the parent object for all other aspects of the Toolkit library. It contains all the information pertaining to a particular settlement service provider, and its public interface is defined by the Toolkit Programming Interface document.

The major child objects of the provider object are transaction objects, a message queue, and a communication manager. Other, less significant, child objects are also part of the provider object; they are described in the "Utility Services" section below.

Transaction

The Toolkit uses a transaction object to represent a single transaction (e.g. a phone call) with a settlement provider. Transaction objects are children of a provider object, and multiple transactions may be active simultaneously with the same parent provider. The public interface of a transaction object is defined in the Toolkit Programming Interface document.

The main functions of a transaction object are creating OSP messages and interpreting responses received from settlement servers. The transaction object uses message info objects to represent these messages.

Processing within transaction object member functions takes place in the thread of the application that calls the Toolkit interface. Figure 2 shows the general flow of execution within the transaction object. All of the steps in that figure execute in the calling program's thread.

The steps in the figure are:

1) An application makes a call to the Toolkit programming interface and invokes a transaction object member function.

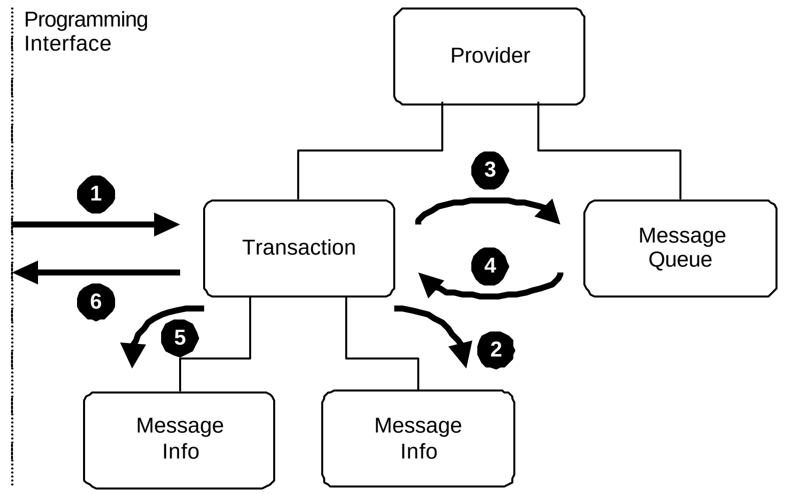
2) If the call requires that the Toolkit send a message to the settlement service provider, the transaction object creates a message info object for that message.

3) The transaction object converts the message info object to a message exchange object and adds the message exchange to the provider's message queue and waits for a reply. The message exchange object, described below, includes a mechanism that allows the transaction object to block while waiting for that reply. The actual transmission of the message and the reception of a reply take place in a separate thread controlled by the provider's communication manager.

4) When the reply is received, the transaction object thread is reactivated and it retrieves the reply from the message exchange object.

5) To interpret the reply, the transaction object parses the reply into a message info object.

6) The transaction object then returns control to the calling application, returning information contained in the reply.



• Figure 2 Flow of Execution within a Transaction Object

Message Info

The message info object represents the information content of an OSP message. This form is not the raw bytes that are transmitted or received, but rather a more abstract representation of the message.

Message Queue

The message queue object stores OSP messages waiting to be transmitted to a service provider, and it serves as the bridge between transaction objects and the provider object's communication manager. As those two objects execute in different threads, the message queue also provides thread synchronization services. Most of that synchronization mechanism is implemented in the message exchange objects, but the message queue does include a condition variable that can wake up the communication manager when the queue transitions from empty to non-empty.

Message Exchange

Message exchange objects represent an OSP message and its associated reply (or error code). They contain raw OSP messages destined for settlement servers and the replies received from those servers. They also include a condition variable to synchronize different threads that may access the exchange information. In the case of message exchange objects on the provider's message queue, the condition variable synchronizes the thread in which the transaction object runs with threads controlled by the provider's communication manager. Thread synchronization takes place as follows:

1) When a transaction object creates an OSP message, it puts that message in a message exchange object and adds the object to the message queue. The transaction thread then blocks on the message's condition variable.

2) The communication manager retrieves the message from the queue and transmits it to the settlement server. When a reply arrives, the communications manager adds the reply to the message exchange object and changes the value of the condition variable. It can also modify the condition variable if an error occurs.

3) The transaction object thread wakes up when the condition variable changes and processes the reply or handles the error, as appropriate.

Communications Manager

The communications manager object coordinates communications to and from the settlement provider server. It does so by managing pools of Secure Sockets Layer (SSL) sessions and HTTP connections. Its typical operation proceeds as follows.

1) The communications manager waits for the provider's message queue to change from empty to non-empty, indicating that a message is ready for transmission.

2) When a message arrives on the queue, the communication manager identifies an HTTP connection on which to send the message. In looking for the connection, the communication manager uses the following priority:

Look for an existing HTTP connection object that is currently idle.

If no idle connections are available, and if the number of current connections is less than the maximum permitted for the provider, create a new HTTP connection object.

If there are no idle connections, and if the number of connections is at the configured limit, wait for a busy HTTP connection object to become idle.

3) If the communication manager had to create a new HTTP connection object for the message, and if SSL security is enabled with the provider, it also assigns an SSL session to that connection. SSL sessions are assigned as follows:

   a) Look for an existing SSL session object that is not currently assigned to an HTTP connection object.

   If no unassigned SSL session objects exist, create a new SSL session object.

Once the message is assigned to an SSL session and HTTP connection, the communications manager has completed its responsibilities with respect to the message.

SSL Session

An SSL session object maintains the information needed to create or resume secure socket layer sessions. That information includes the session identifier and the associated symmetric encryption key.

SSL session objects are created by the communication manager in conjunction with the creation of a new HTTP connection object. SSL session objects, however, are not destroyed with the destruction of the associated HTTP connection. Instead, the communication manager keeps SSL sessions in a pool for re-assignment to a new HTTP connection. The communication manager only destroys SSL objects when the provider itself is destroyed.

SSL session objects do include a lifetime that limits the validity of the data they contain. Logically, this lifetime can expire either while the session is assigned to an HTTP connection, or while the session is inactive and waiting for re-assignment. In the actual implementation, HTTP connection objects are responsible for verifying the session lifetime prior to sending each message. If that lifetime has expired (or if the settlement server requests a new SSL session), the information in the SSL session object is updated to reflect the new SSL session information.

HTTP Connection

HTTP connection objects control a Hypertext Transfer Protocol connection to a settlement server. Their major function is managing the network input and output for that connection, and handling the (minimal) HTTP protocol processing for the connection. Each connection object contains its own message queue that holds messages pending transmission. The communications manager object puts message exchange objects on this queue. As Figure 3 indicates, the typical scenario is for the communication manager to remove messages from the provider’s message queue, assign them to a specific HTTP connection object, and add the message to that connection’s queue.

Each HTTP connection object executes in its own thread, and it synchronized to the connection manager by a condition variable in its message queue. On creation, the connection object immediately waits for this condition variable to indicate that a message is on the queue. When a message appears, the connection object takes the following steps to transmit the message to the settlement server:

1) The connection object attempts to open a TCP connection with the settlement server. It begins with the first server in the list, and, should that attempt fail, continues with subsequent servers on the list until a connection is established or the list is exhausted.

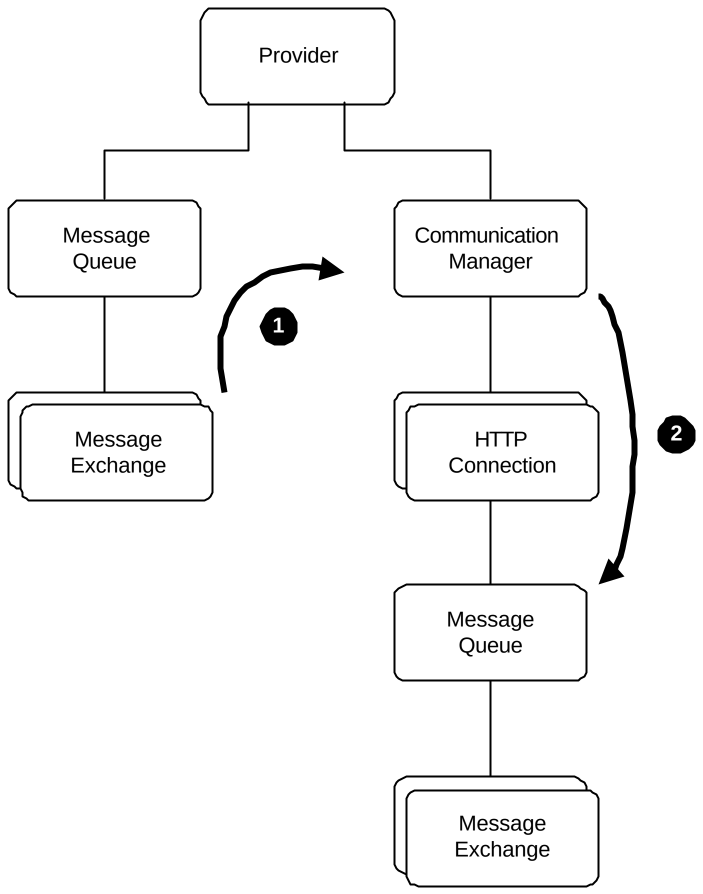
2) If the connection object cannot open a connection, it sleeps for a specified period of time and tries again (returning to step 1). The sleep time and maximum number of retries are controlled by configuration parameters.

3) Once the connection is established, the connection object checks the lifetime of its SSL session. If that lifetime is still valid, the connection object attempts to resume the SSL session. If the session has expired, however, (or if the server rejects the session), the connection object renegotiates the SSL session parameters and updates the SSL session object.

4) The connection object then transmits the message (with appropriate HTTP headers) to the settlement server and blocks, waiting for a reply.

5) If a connection error occurs before the reply arrives, the connection object restarts the process at step 1.

6) When the connection object receives the reply, it ties it to the original message exchange object, remotes that message exchange from its message queue, and signals the original transaction object through the message exchange’s condition variable.



• Figure 3 Message Flow to HTTP Connection Objects.

7) The connection object then checks its message queue for additional messages, If any are present, it processes them beginning with step 3.

8) If the message queue is empty, the HTTP connection object's thread blocks on its message queue condition variable, waiting for the communications manager to add new messages to the queue. The connection object includes a maximum time to be blocked that is equal to the TCP persistence time configured for the provider.

9) If a new message arrives before the persistence time expires, the connection object begins processing at step 3.

10) If no message arrives, the connection object returns its SSL session object to the communication manager's pool and terminates its thread.

Message Formatting and Parsing

In addition to managing the communications between applications and settlement service providers, another significant role of the Toolkit software is message formatting and parsing. As the contents of this section indication, Toolkit includes support for several standards: XML, S/MIME, MIME, Base64, and ASN.1 DER.

The process of creating a formatted message for transmission proceeds in five stages, beginning with a message info object. As Figure 4 illustrates, the message is modified or augmented at each stage.

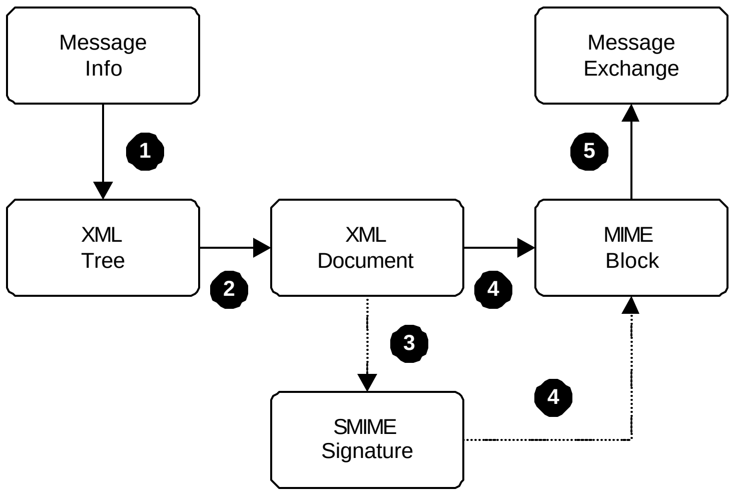
The complementary process, parsing a received message for interpretation, uses the same objects, though in reverse order. In addition, the parsing process may require an additional steps to interpret authorization tokens. Figure 5 (on the following page) illustrates the entire process. The additional step required for tokens is base64 decoding.

The following subsections describe the objects of the formatting and parsing processes. The starting and ending points—message info and message exchange objects—were discussed in the preceding section.

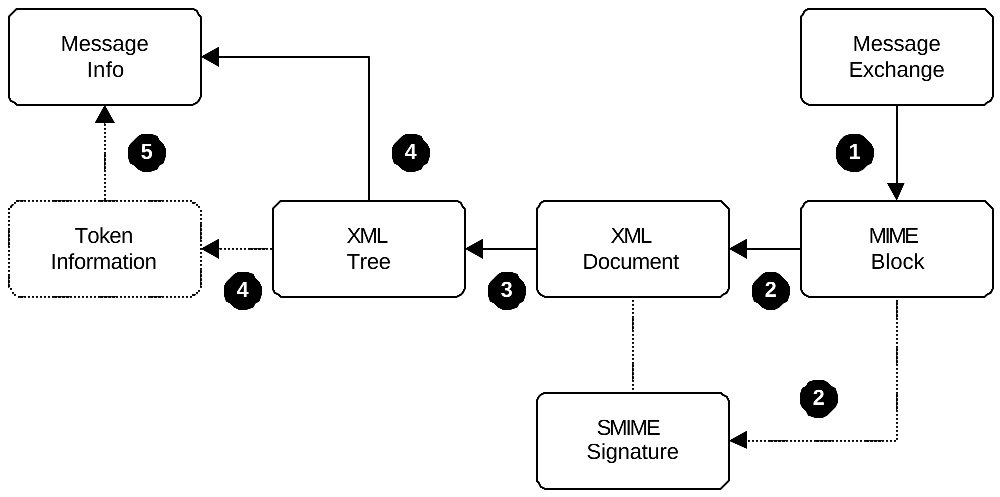
XML Tree

The XML tree object contains a logical representation of the contents of an XML document. It mirrors the nested logical structure of XML content through tree-like data structures. Each node in the tree represents an XML element; children of a node are sub-elements of the parent. The node also includes the attributes associated with the element.

XML trees may be created in two ways. The formatting process creates an XML tree from a message object. Since message info objects are specific to each OSP message type, this process is itself specific to individual message types. In the reverse direction, XML trees are created from XML documents. This process is a straightforward example of standard XML parsing, though some significant simplifications from true XML parsing are possible. In particular, OSP messages do not use external entity references or processing instructions, so handling of external entities or processing instructions is not required. In addition, the Toolkit library does not validate XML documents, nor does it exhaustively check for well-formedness. (The Toolkit's XML parser is an adherent to the Internet protocol philosophy: be conservative in what you send and liberal in what you accept.)



• Figure 4 Formatting an OSP Message.



• Figure 5 Parsing an OSP Message.

XML Document

The XML document object contains the linear, character string for an XML document. It represents the bit-by-bit content of OSP messages in a format ready for transmission, or as received. For outgoing messages, XML documents are built from XML trees. And for incoming messages, XML documents are extracted from MIME blocks.

SMIME Signature

SMIME signature objects contain a digital signature conforming to the Secure Multipurpose Internet Mail Extension standards. During formatting, SMIME signature objects are generated directly by the cryptographic services. For received messages, SMIME signatures are extracted from MIME blocks.

MIME Block

MIME block objects contain a complete OSP protocol message (which is, by definition, a MIME block transferred by HTTP). When messages are prepared for transmission, MIME blocks are formed by combining an XML document with an SMIME signature. In the reverse direction, MIME blocks are simply extracted from message exchange objects.

Token Information

OSP authorization response messages may include tokens to validate the authorization. These tokens, if present, are base64 encoded in the XML content. Before passing the token to an application, it must be decoded. The also Toolkit includes functionality to interpret the contents of an authorization token. (That functionality is needed to verify tokens, for example.)

Utility Services

The Toolkit includes several utility objects in that support the Open Settlement Protocol implementation. The major utility objects, each described in this section, include a delay probe, a statistics accumulator, and a transaction ID tracking object. Figure 6 shows the relationship of these objects with the Toolkit's primary objects.

Statistics Accumulator

The statistics accumulator object is used to track network performance statistics for a call. Specifically, it can keep track of either one-way or round trip delay measurements as the application reports them. Each transaction object includes two statistics accumulators, one for each measure quantity.

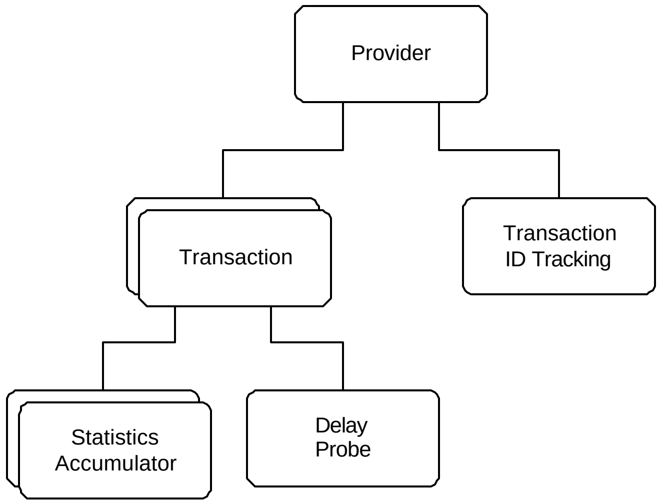
Each statistics accumulator includes the following parameters:

m	minimum measured value
n	number of measuremen ts
$\bar{x}$	sample mean
q	sum of square of samples
S^2	sample variance

As the application collects sample measurements, it reports those measurements to the Toolkit using the following values:

$\hat{m}$	minimum measured value within sample set
$\hat{n}$	number of measuremen ts within sample set
$\hat{\bar{x}}$	mean of sample set
$\hat{S}^2$	sample variance of sample set

The new sample set is used to update the old parameters (time i) and calculate new parameters (time $i+1$) according to the following equations:



• Figure 6 Utility Service Objects.

$$m_{i+1} = \min(m_i, \hat{m})$$
$$n_{i+1} = n_i + \hat{n}$$
$$\overline{x}_{i+1} = \frac{n_i \cdot \overline{x}_i + \hat{n} \cdot \hat{\overline{x}}}{n_{i+1}}$$
$$q_{i+1} = q_i + (\hat{n} - 1) \cdot \hat{S}^2 + \hat{n} \cdot \hat{\overline{x}}^2$$
$$S_{i+1}^2 = \frac{q_{i+1} - n_{i+1} \cdot \overline{x}_{i+1}^2}{n_{i+1} - 1}$$

Delay Probe

The delay probe object is responsible for performing a round-trip latency test to one or more destinations. Such tests may be used to support quality of service extensions to the basic OSP standard. The delay probe performs its function by opening sockets and sending small UDP datagrams to the echo service of each destination. It then measures the time until a response is received. The delay probe incorporates a maximum time to wait for responses.

Transaction ID Tracking

The transaction ID tracking object ensures the uniqueness of transaction IDs included in authorization tokens. It is designed to help prevent the inappropriate re-use of those tokens, and it makes sure that the same transaction ID is not re-used within the lifetime of the authorization token that contains it.