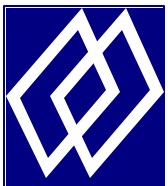


TransNexus

OSP Toolkit

Programming Interface

Release 2.5.5

09 February 2002

OSP Toolkit

Programming Interface

Release 2.5.5

09 February 2002

Document 0300-1212-0200

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 programming interface to 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. This document consists of three sections. The first, "Available Services," provides complete documentation for the library interface. The second section describes typical program flows to show how the various interface functions work together. The document concludes with a detailed example of how the library may be used by Internet telephony devices.

Available Services

The OSP library provides services through two primary objects. Those objects are the provider object and the transaction object. A provider object represents communication to a particular settlement provider; it is typically created during system startup. Transaction objects, each of which must be associated with a specific provider object, represent single transactions with a settlement provider. In the context of Internet telephony, transaction objects are created for each phone call, and are destroyed after the call has been disconnected.

The following subsections describe the interface functions for each of these objects, as well as the initialization function required before any access to the Toolkit functionality.

Software Initialization

Before an application can create and use provider or transaction objects, it must initialize the Toolkit software. That initialization consists of a single function call.

OSPPInit

```
int OSPPInit();
```

The `OSPPInit` function performs internal housekeeping necessary to prepare the Toolkit software for operation. The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

Provider Interface

The programming interface to the provider object includes functions to create and destroy instances of the object, as well as functions to set and return configuration information for these instances. Applications must successfully create a provider object before they can interact with the Toolkit library.

OSPPProviderDelete

```
int
OSPPProviderDelete(
    OSPTPROVHANDLE  ospvProvider,
    int             ospvTimeLimit
);
```

The `OSPPProviderDelete` function tells the Toolkit library to delete a provider object. This function immediately prevents the creation of new transactions for the indicated provider. (Attempts to create new transaction objects will be refused with an appropriate error code.) The function also blocks until all pending transactions for the provider have completed or the time limit has been exceeded.

The `ospvTimeLimit` parameter specifies the maximum number of seconds to wait for pending transactions to complete. A negative value for this parameter instructs the library to wait indefinitely, and a value of zero indicates that the deletion should occur immediately without waiting. If pending transactions are not complete within the time limit, those transactions will be terminated abruptly and information, including information necessary for billing, may be lost.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetAuthorityCertificates

```
int
OSPPProviderGetAuthorityCertificates(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvSizeOfCertificate,
    unsigned       *ospvNumberOfAuthorityCertificates,
    void           *ospvAuthorityCertificates[]
);
```

The `OSPPProviderGetAuthorityCertificates` function returns the certificate authority public keys that are currently trusted by `ospvProvider`. These keys are

returned in the form of X.509 formatted certificates, and they are returned to the `ospvAuthorityCertificates` array. The `ospvSizeOfCertificate` parameter indicates the maximum size of any individual certificate. If any certificate exceeds that value then no certificates are returned and an error is returned. The parameter `ospvNumberOfAuthorityCertificates` points to the maximum number of certificates to return. That variable is updated with the actual number supplied when the function returns. If more certificates are available, then only a partial list is returned.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetHTTPMaxConnections

```
int
OSPPProviderGetHTTPMaxConnections(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvHTTPMaxConnections
);
```

The `OSPPProviderGetHTTPMaxConnections` function returns the maximum number of simultaneous HTTP connections that may be established with `ospvProvider`. That number is returned in the variable pointed to by `ospvHTTPMaxConnections`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetHTTPPersistence

```
int
OSPPProviderGetHTTPPersistence(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvHTTPPersistence
);
```

The `OSPPProviderGetHTTPPersistence` function returns the persistence of HTTP connections established with `ospvProvider`. That value, returned in the location pointed to by `ospvHTTPPersistence`, is measured in seconds.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetHTTPRetryDelay

```
int
OSPPProviderGetHTTPRetryDelay(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvHTTPRetryDelay
);
```

The `OSPPProviderGetHTTPRetryDelay` function returns the delay between retries for HTTP connection attempts with `ospvProvider`. That value, returned in the location pointed to by `ospvHTTPRetryDelay`, is measured in seconds.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetHTTPRetryLimit

```
int
OSPPProviderGetHTTPRetryLimit(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvHTTPRetryLimit
);
```

The `OSPPProviderGetHTTPRetryLimit` function returns the maximum number of retries for HTTP connection attempts with `ospvProvider`. That value is returned in the location pointed to by `ospvHTTPRetryLimit`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetHTTPTimeout

```
int
OSPPProviderGetHTTPTimeout(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvHTTPTimeout
);
```

The `OSPPProviderGetHTTPTimeout` function returns the timeout value that specifies how long to wait for responses from HTTP connections with `ospvProvider`. The value, returned in the location pointed to by `ospvHTTPTimeout`, is measured in milliseconds.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetLocalKeys

```
int
OSPPProviderGetLocalKeys(
    OSPTPROVHANDLE   ospvProvider,
    OSPTPRIVATEKEY  *ospvLocalPrivateKey,
    unsigned         ospvSizeOfCertificate,
    void            *ospvLocalCertificate
);
```

The `OSPPProviderGetLocalKeys` function returns the public and private key information currently in use by `ospvProvider` for signing requests and indications. The RSA private key is returned in the location pointed to by `ospvLocalPrivateKey`, and the X.509 formatted public key certificate is stored in `ospvLocalCertificate`. The `ospvSizeOfCertificate` parameter indicates the maximum size of the `ospvLocalCertificate` array. If the certificate does not fit within that limit, the function returns an appropriate error code.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetLocalValidation

```
int
OSPPProviderGetLocalValidation(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvLocalValidation
);
```

The `OSPPProviderGetLocalValidation` function returns an indication of whether or not `ospvProvider` is currently set to validate authorization tokens locally (i.e. by verifying their digital signature) or via a protocol exchange. The return value is stored in the location pointed to by `ospvLocalValidation`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetNumberOfAuthorityCertificates

```
int
OSPPProviderGetNumberOfAuthorityCertificates(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvNumberOfAuthorityCertificates
);
```

The `OSPPProviderGetNumberOfAuthorityCertificates` function returns the number of certificate authority public keys currently trusted by `ospvProvider`. That value is stored in the location pointed to by `ospvNumberOfAuthorityCertificates`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetNumberOfServicePoints

```
int
OSPPProviderGetNumberOfServicePoints(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvNumberOfServicePoints
);
```

The `OSPPProviderGetNumberOfServicePoints` interface provides the number of service points currently defined for `ospvProvider`. The result is returned in the location pointed to by `ospvNumberOfServicePoints`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetServicePoints

```
int
OSPPProviderGetServicePoints(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvNumberOfServicePoints,
    unsigned        ospvSizeOfServicePoint,
    char           *ospvServicePoints[]
);
```

The `OSPPProviderGetServicePoints` function gives the caller the list of service points currently defined for `ospvProvider`. The `ospvNumberOfServicePoints` parameter indicates the maximum number of service points to include, and the `ospvSizeOfServicePoint` parameter indicates the maximum length of the character string (***including*** the terminating `'\0'`) in which service points are placed. The service points themselves are stored in the character strings indicated by the `ospvServicePoints` array.

If the number of service points is less than `ospvNumberOfServicePoints`, then excess entries in the `ospvServicePoints` array are set to empty strings. If the actual number is more than the parameter, then only the first `ospvNumberOfServicePoints` are supplied. If the string length of any particular service point is greater than `ospvSizeOfServicePoint`, then no service points are supplied (all pointers in the `ospvServicePoints` array are set to empty strings) and an error is returned.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderGetSSLLifetime

```
int
OSPPProviderGetSSLLifetime(
    OSPTPROVHANDLE  ospvProvider,
    unsigned       *ospvSSLLifetime
);
```

The `OSPPProviderGetSSLLifetime` function returns the maximum lifetime of SSL session keys established with `ospvProvider`. That lifetime, expressed in seconds, is returned in the location pointed to by `ospvSSLLifetime`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderNew

```
int
OSPPProviderNew(
    unsigned             ospvNumberOfServicePoints,
    const char          *ospvServicePoints[],
    const char          *ospvAuditURL,
    const OSPTPRIVATEKEY *ospvLocalPrivateKey,
    const void          *ospvLocalCertificate,
    unsigned             ospvNumberOfAuthorityCertificates,
```

```
    const void           *ospvAuthorityCertificates[],
    unsigned              ospvLocalValidation,
    unsigned              ospvSSLLifetime,
    unsigned              ospvHTTPMaxConnections,
    unsigned              ospvHTTPPersistence,
    unsigned              ospvHTTPRetryDelay,
    unsigned              ospvHTTPRetryLimit,
    unsigned              ospvHTTPTimeout,
    const char           *ospvCustomerId,
    const char           *ospvDeviceId,
    OSPTPROVHANDLE       *ospvProvider
);
```

The `OSPPProviderNew` function creates and initializes a provider object. This function must be called and return without errors before any other interaction with the Toolkit library can take place.

The parameters passed to this function provide the initial configuration information for the provider. That information consists of the following items:

`ospvNumberOfServicePoints`: the number of service points included in the list referenced by the `ospvServicePoints` parameter.

`ospvServicePoints`: a list of character strings indicating where the library should send requests and indications. Each service point in the list takes the form of a standard URL, and may consist of up to four components:

- An optional indication of the protocol to be used for communicating with the service point. This release of the Toolkit supports both HTTP and HTTP secured with SSL; they are indicated by `"http://"` and `"https://"` respectively. If the protocol is not explicitly indicated, the Toolkit defaults to HTTP secured with SSL.
- The Internet domain name for the service point. Raw IP addresses may also be used, provided they are enclosed in square brackets such as `"[172.16.1.1]"`.
- An optional TCP port number for communicating with the service point. If the port number is omitted, the Toolkit defaults to port 80 (for HTTP) or port 443 (for HTTP secured with SSL).
- The uniform resource identifier for requests to the service point. This component is not optional and must be included.

The service points are ordered in the list by decreasing preference. The Toolkit library, therefore, attempts to contact the first service point first. Only if that attempt fails will it fall back to the second service point.

Examples of valid service points include

```
"https://service.transnexus.com/scripts/voice/osp.cmd"
"service.uk.transnexus.co.uk/scripts/fax/osp.cmd"
 "http://[172.16.1.2]:443/scripts/video/osp.cmd"
```

`ospvLocalPrivateKey`: the RSA private key to be used for signing messages sent to the settlement service.

`ospvLocalCertificate`: a X.509 formatted certificate containing the RSA public key corresponding to the local private key.

`ospvNumberOfAuthorityCertificates`: the number of certificate authority certificates passed in the next parameter.

`ospvAuthorityCertificates`: an array of X.509 formatted certificates containing certificate authority public keys. These public keys are used to authenticate the settlement provider server during the initial SSL exchange.

`ospvLocalValidation`: a Boolean value to indicate whether or not the Toolkit should validation authorization tokens locally (i.e. by verifying digital signatures) or via a protocol exchange.

`ospvSSLLifetime`: the lifetime, in seconds, of a single SSL session key. Once this time limit is exceeded, the Toolkit library will negotiate a new session key. Communication exchanges in progress will not be interrupted when this time limit expires.

`ospvHTTPMaxConnections`: the maximum number of simultaneous connections to be used for communication to the settlement provider.

`ospvHTTPPersistence`: the time, in seconds, that an HTTP connection should be maintained after the completion of a communication exchange. The library will maintain the connection for this time period in anticipation of future communication exchanges to the same server.

`ospvHTTPRetryDelay`: the time, in seconds, between retrying connection attempts to the provider. After exhausting all service points for the provider, the library will delay for this amount of time before resuming connection attempts.

`ospvHTTPRetryLimit`: the maximum number of retries for connection attempts to the provider. If no connection is established after this many retry attempts to all service points, then the library will cease connection attempts and return appropriate error codes. This number does ***not*** count the initial connection attempt, so that an `ospvHTTPRetryLimit` of `1` will result in a total of two connection attempts to every service point.

`ospvHTTPTimeout`: the maximum time, in milliseconds, to wait for a response from a server. If no response is received within this time, the current connection is aborted and the library attempts to contact the next service point.

`ospvCustomerId`: an (optional) character string specifying the customer identification for the system with the provider. This value is typically assigned by the settlement provider as a means of uniquely and unambiguously identifying the customer. Some providers may not require this parameter, or they may obtain the necessary information from other means (e.g. from the system's public key certificate). In such cases, this parameter may be the `NULL` pointer or an empty string.

`ospvDeviceId`: an (optional) character string specifying the device identification for the system with the provider. This value is typically assigned by the settlement provider as a means of uniquely and unambiguously identifying the specific device. Some providers may not require this parameter, or they may obtain the necessary information from other means (e.g. from the system's public key certificate). In such cases, this parameter may be the `NULL` pointer or an empty string.

`ospvProvider`: pointer to variable in which to store a handle for the newly created provider object. That handle must be used for all subsequent interactions with the provider.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetAuthorityCertificates

```
int
OSPPProviderSetAuthorityCertificates(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvNumberOfAuthorityCertificates,
    const void     *ospvAuthorityCertificates[]
);
```

The `OSPPProviderSetAuthorityCertificates` function indicates the certificate authority public keys that should be trusted for `ospvProvider`. Those public keys are conveyed in the form of X.509 formatted certificates. The parameter `ospvNumberOfAuthorityCertificates` indicates how many of such certificates are conveyed in the `ospvAuthorityCertificates` array.

Communication exchanges already in progress are not interrupted by this function, but subsequent exchanges will use the new values.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetHTTPMaxConnections

```
int
OSPPProviderSetHTTPMaxConnections(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvHTTPMaxConnections
);
```

The `OSPPProviderSetHTTPMaxConnections` function indicates the maximum number of simultaneous HTTP connections that should be established with `ospvProvider`. The number is passed in the `ospvHTTPMaxConnections` parameter. Changes to this value do not effect active communication exchanges but otherwise take place immediately. In particular, HTTP connections being kept alive strictly because of persistence are terminated immediately if the number of open connections must be reduced.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetHTTPPersistence

```
int
OSPPProviderSetHTTPPersistence(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvHTTPPersistence
);
```

The `OSPPProviderSetHTTPPersistence` function configures the persistence of HTTP connections established with `ospvProvider`. That lifetime, expressed in seconds, is indicated by the `ospvHTTPPersistence` parameter. The Toolkit library keeps a HTTP connection alive for this number of seconds after each communication exchange, anticipating that a subsequent exchange may reuse the existing connection. Changes to this parameter take place immediately.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetHTTPRetryDelay

```
int
OSPPProviderSetHTTPRetryDelay(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvHTTPRetryDelay
);
```

The `OSPPProviderSetHTTPRetryDelay` function configures the delay between retries for connections attempts with `ospvProvider`. That delay, expressed in seconds, is indicated by the `ospvHTTPRetryDelay` parameter. After exhausting all service points for the provider, the library will delay for this amount of time before resuming connection attempts. Changes to this parameter take place immediately.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetHTTPRetryLimit

```
int
OSPPProviderSetHTTPRetryLimit(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvHTTPRetryLimit
);
```

The `OSPPProviderSetHTTPRetryLimit` function configures the maximum number of retries for connections attempts with `ospvProvider`. If no connection is established after this many retry attempts to all service points, then the library will cease connection attempts and return appropriate error codes. This number does ***not*** count the initial connection attempt, so that an `ospvHTTPRetryLimit` of `1` will result in a total of two connection attempts to every service point.

For example, if there are five service points with a Retry Limit of four, then for each service point, the OSP Toolkit will try to connect five times. The OSP Toolkit will try a total of twenty-five times to connect (five service points times five retrys).

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetHTTPTimeout

```
int
OSPPProviderSetHTTPTimeout(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvHTTPTimeout
);
```

The `OSPPProviderSetHTTPTimeout` function configures the maximum amount of time to wait for a reply from `ospvProvider`. That timeout, expressed in milliseconds, is indicated by the `ospvHTTPTimeout` parameter. If no response arrives within this time, the current connection is aborted and the library attempts to connect with another service point. Changes to this parameter do not affect connection attempts already in progress, but take affect for all subsequent attempts.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetLocalKeys

```
int
OSPPProviderSetLocalKeys(
    OSPTPROVHANDLE        ospvProvider,
    const OSPTPRIVATEKEY *ospvLocalPrivateKey,
    const void           *ospvLocalCertificate
);
```

The `OSPPProviderSetLocalKeys` function configures the public and private key pair used by `ospvProvider` to sign its requests and indications. The parameter `ospvLocalPrivateKey` identifies the RSA private key and the parameter `ospvLocalCertificate` points to a X.509 formatted certificate containing the corresponding RSA public key.

Communication exchanges already in progress are not interrupted by this function, but subsequent exchanges will use the new values.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetLocalValidation

```
int
OSPPProviderSetLocalValidation(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvLocalValidation
);
```

The `OSPPProviderSetLocalValidation` function indicates to `ospvProvider` whether authorization tokens should be validated locally (i.e. by verifying their digital signature) or via a protocol exchange. The parameter `ospvLocalValidation` is non-zero for local validation or zero for remote validation.

Communication exchanges already in progress (e.g. AuthorisationIndication and Confirm exchanges) are not interrupted by this function, but subsequent calls to `OSPPTransactionValidateAuthorisation` will use the newly specified technique.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetServicePoints

```
int
OSPPProviderSetServicePoints(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvNumberOfServicePoints,
    const char     *ospvServicePoints[]
);
```

The `OSPPProviderSetServicePoints` function indicates the service points the Toolkit library should use for future communication exchanges with `ospvProvider`. Communication exchanges already in progress are not interrupted.

The format for the `ospvServicePoints` parameter is the same as in the `OSPPProviderNew` function call. In particular, it is a list of service points, each in the form of a standard URL. A service point URL may consist of up to four components:

- An optional indication of the protocol to be used for communicating with the service point. This release of the Toolkit supports both HTTP and HTTP secured with SSL; they are indicated by `"http://"` and `"https://"` respectively. If the protocol is not explicitly indicated, the Toolkit defaults to HTTP secured with SSL.

- The Internet domain name for the service point. Raw IP addresses may also be used, provided they are enclosed in square brackets such as `"[172.16.1.1]"`.

- An optional TCP port number for communicating with the service point. If the port number is omitted, the Toolkit defaults to port 80 (for HTTP) or port 443 (for HTTP secured with SSL).

- The uniform resource identifier for requests to the service point. This component is not optional and must be included.

The service points are ordered in the list by decreasing preference. The Toolkit library, therefore, attempts to contact the first service point first. Only if that attempt fails will it fall back to the second service point.

Examples of valid service points include

```
        "https://service.transnexus.com/scripts/voice/osp.cmd"
        "service.uk.transnexus.co.uk/scripts/fax/osp.cmd"
         "http://[172.16.1.2]:443/scripts/video/osp.cmd"
```

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPProviderSetSSLLifetime

```
int
OSPPProviderSetSSLLifetime(
    OSPTPROVHANDLE  ospvProvider,
    unsigned        ospvSSLLifetime
);
```

The `OSPPProviderSetSSLLifetime` function configures the maximum lifetime of SSL session keys established with `ospvProvider`. That lifetime, expressed in seconds, is indicated by the `ospvSSLLifetime` parameter. The Toolkit library attempts to reuse previously established SSL session keys in order to minimize delay when communicating with a settlement server. This parameter places a maximum lifetime on these keys. Changes to this parameter take place immediately. Note, however, that communication exchanges already in progress are never interrupted when a session key lifetime expires.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

Transaction Interface

Once a provider object has been created, applications may exchange transactions with that settlement service provider. Applications do so by interfacing with a transaction object.

OSPPTransactionAccumulateOneWayDelay

```
int
OSPPTransactionAccumulateOneWayDelay(
    OSPTTRANHANDLE  ospvTransaction,
    unsigned        ospvNumberOfSamples,
    unsigned        ospvMinimum,
    unsigned        ospvMean,
    float           ospvVariance
);
```

The `OSPPTransactionAccumulateOneWayDelay` function accumulates one-way delay statistics for the call. It is used to report one way delay ***from*** the remote peer ***to*** the reporting system. This value may be calculated by comparing the network time protocol (NTP) timestamp included in RTCP messages from the peer with the local NTP time in the reporting system.

Applications may call this function an unlimited number of times during a transaction, but only after the transaction has been authorized and before its usage details are reported (i.e. after calling either the function `OSPPTransactionRequestAuthorisation` or the function `OSPPTransactionValidateAuthorisation` and before calling the function

`OSPPTransactionReportUsage`). Also, each call to this function must report statistics for a separate and distinct set of measurements. In other words, once `OSPPTransactionAccumulateOneWayDelay` is successfully called, the application should discard (at least for subsequent calls to the function) the data and start calculating minimum, mean, variance measures anew.

Applications may use this function to report a single sample, or they may report statistical measurements from a collection of samples. The parameters to the function are:

`ospvTransaction`: handle of the transaction object.

`ospvNumberOfSamples`: the number of samples included in these statistics.

`ospvMinimum`: the minimum delay, in milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should indicate that measurement, and it should be equal to the value of the `ospvMean` parameter.

`ospvMean`: the mean of the delay, in milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should indicate that measurement, and it should be equal to the value of the `ospvMinimum` parameter.

`ospvVariance`: the variance of the delay, in square milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should be zero.

The Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionAccumulateRoundTripDelay

```
int
OSPPTransactionAccumulateRoundTripDelay(
    OSPTTRANHANDLE  ospvTransaction,
    unsigned        ospvNumberOfSamples,
    unsigned        ospvMinimum,
    unsigned        ospvMean,
    float           ospvVariance
);
```

The `OSPPTransactionAccumulateRoundTripDelay` function accumulates round trip delay statistics for the call. These measurements can be made using, for example, H.245 round trip delay requests during the call.

Applications may call this function an unlimited number of times during a transaction, but only after the transaction has been authorized and before its usage details are reported (i.e. after calling either the function `OSPPTransactionRequestAuthorisation` or the function `OSPPTransactionValidateAuthorisation` and before calling the function

`OSPPTransactionReportUsage`). Also, each call to this function must report statistics for a separate and distinct set of measurements. In other words, once `OSPPTransactionAccumulateRoundTripDelay` is successfully called, the application should discard (at least for subsequent calls to the function) the data and start calculating minimum, mean, variance measures anew.

Applications may use this function to report a single sample, or they may report statistical measurements from a collection of samples. The parameters to the function are:

`ospvTransaction`: handle of the transaction object.

`ospvNumberOfSamples`: the number of samples included in these statistics.

`ospvMinimum`: the minimum delay, in milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should indicate that measurement, and it should be equal to the value of the `ospvMean` parameter.

`ospvMean`: the mean of the delay, in milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should indicate that measurement, and it should be equal to the value of the `ospvMinimum` parameter.

`ospvVariance`: the variance of the delay, in square milliseconds, measured within the current set of samples. If the function call is used to report a single sample, this parameter should be zero.

The Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionDelete

```
int
OSPPTransactionDelete(
    OSPTTRANHANDLE  ospvTransaction
);
```

The `OSPPTransactionDelete` function destroys the `ospvTransaction` object and releases the resources it consumes. Once this function is called, the application is prohibited from subsequent interaction with the object. (Attempts to do so are refused with an appropriate error code.) The library may continue to use the transaction's resources, however, until it has concluded communication regarding this transaction with the settlement server. An application can ensure the release of all resources only by specifying a time limit in a call to `OSPPProviderDelete`.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionGetFirstDestination

```
int
OSPPTransactionGetFirstDestination(
    OSPTTRANHANDLE  ospvTransaction,
    unsigned        ospvSizeOfTimestamp,
    char           *ospvValidAfter,
    char           *ospvValidUntil,
    unsigned       *ospvTimeLimit,
    unsigned       *ospvSizeOfCallId,
    void           *ospvCallId,
    unsigned        ospvSizeOfCalledNumber,
    char           *ospvCalledNumber
    unsigned        ospvSizeOfDestination,
    char           *ospvDestination,
    unsigned        ospvSizeOfDestinationDevice,
    char           *ospvDestinationDevice,
    unsigned       *ospvSizeOfToken,
    void           *ospvToken
);
```

The `OSPPTransactionGetFirstDestination` function returns the identity of the first authorized destination for a call. The Toolkit library obtains this information during the execution of the `OSPPTransactionRequestAuthorisation` function. The parameters to this function consist of the following:

`ospvTransaction`: handle of the transaction object.

`ospvSizeOfTimestamp`: size of the character strings (***including*** the terminating `'\0'`) in which the function should store validity times for the destination. If this value is zero, then validity times are not returned. If this size is non-zero but not large enough to store either validity time, then an error is indicated and no destination is returned.

`ospvValidAfter`: character string in which to store the earliest time for which the call is authorized to the destination. The format for the string is the same as indicated in the OSP protocol specification. For example, 3:03 P.M. on May 2, 1997, Eastern Daylight Time in the United States is represented as `"1997-05-02T19:03:00Z"`.

`ospvValidUntil`: character string in which to store the latest time for which the call is authorized to the destination. The format for the string is the same as for the `ospvValidAfter` parameter.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is initially authorized. A value of zero indicates that no limit exists. Note that the initial time limit may be extended during the call by either party.

`ospvSizeOfCallId`: pointer to a variable which, on input, contains the size of the memory buffer in which the function should place the H.323 call identifier for the destination. If the value is not large enough to accommodate the call identifier, then an error is indicated and no destination is returned. On output this variable is updated to indicate the actual size of the call identifier.

`ospvCallId`: memory location in which to store the H.323 call identifier for the destination. The call identifier returned here is the same format as the call identifier passed to the `OSPPTransactionRequestAuthorisation` function.

`ospvSizeOfCalledNumber`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the called number. If the value is not large enough to accommodate the called number, then an error is indicated and no destination is returned.

`ospvCalledNumber`: character string in which to store the called number. In general, the called number returned here will be the same as the called number that the application passed to the `OSPPTransactionRequestAuthorisation` function; however, the settlement service provider may perform a number translation on the called number, resulting in a new called number that should be signaled to the peer gateway.

`ospvSizeOfDestination`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the destination information. If the value is not large enough to accommodate the destination, then an error is indicated and no destination is returned.

`ospvDestination`: character string in which to store the identity of the destination. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSizeOfDestinationDevice`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the destination device identity. If the value is not large enough to accommodate the destination device identity, then an error is indicated and no destination is returned.

`ospvDestinationDevice`: character string in which to store the identity of the destination device in a protocol specific manner (e.g. H.323 Identifier); this value is optional and, if it is not present, the returned string is empty.

`ospvSizeOfToken`: pointer to a variable which, on input, contains the size of the memory buffer in which the function should store the authorization token for the destination. If the value is not large enough to accommodate the token, then an error is indicated and no destination is returned. On output this variable is updated to indicate the actual size of the authorization token.

`ospvToken`: memory location in which to store the authorization token for this destination. In general, tokens are opaque, binary objects.

The Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionGetNextDestination

```
int
OSPPTransactionGetNextDestination(
    OSPTTRANHANDLE       ospvTransaction,
    enum OSPEFAILREASON  ospvFailureReason,
    unsigned             ospvSizeOfTimestamp,
    char                *ospvValidAfter,
    char                *ospvValidUntil,
    unsigned            *ospvTimeLimit,
    unsigned            *ospvSizeOfCallId,
    void                *ospvCallId,
    unsigned             ospvSizeOfCalledNumber,
    char                *ospvCalledNumber
    unsigned             ospvSizeOfDestination,
    char                *ospvDestination,
    unsigned             ospvSizeOfDestinationDevice,
    char                *ospvDestinationDevice,
    unsigned            *ospvSizeOfToken,
    void                *ospvToken
);
```

The `OSPPTransactionGetNextDestination` function returns the identity of the next authorized destination for a call. Applications may use this function when attempts to use previously identified authorized destinations (starting with the first destination) fail. The Toolkit library obtains the necessary information for this function during its execution of the `OSPPTransactionRequestAuthorisation`. The parameters to this function consist of the following:

`ospvTransaction`: handle of the transaction object.

`ospvFailureReason`: the reason that attempts to use the previously identified destination failed; values for this parameter are listed in the `ospfail.h` file.

`ospvSizeOfTimestamp`: size of the character strings (***including*** the terminating `'\0'`) in which the function should store validity times for the destination. If this value is zero, then validity times are not returned. If this size is non-zero but not large enough to store either validity time, then an error is indicated and no destination is returned.

`ospvValidAfter`: character string in which to store the earliest time for which the call is authorized to the destination. The format for the string is the same as indicated in the OSP protocol specification. For example, 3:03 P.M. on May 2, 1997, Eastern Daylight Time in the United States is represented as `"1997-05-02T19:03:00Z"`.

`ospvValidUntil`: character string in which to store the latest time for which the call is authorized to the destination. The format for the string is the same as for the `ospvValidAfter` parameter.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is initially authorized. A value of zero indicates that no limit exists. Note that the initial time limit may be extended during the call by either party.

`ospvSizeOfCallId`: pointer to a variable which, on input, contains the size of the memory buffer in which the function should place the H.323 call identifier for the destination. If the value is not large enough to accommodate the call identifier, then an error is indicated and no destination is returned. On output this variable is updated to indicate the actual size of the call identifier.

`ospvCallId`: memory location in which to store the H.323 call identifier for the destination. The call identifier returned here is the same format as the call identifier passed to the `OSPPTransactionRequestAuthorisation` function.

`ospvSizeOfCalledNumber`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the called number. If the value is not large enough to accommodate the called number, then an error is indicated and no destination is returned.

`ospvCalledNumber`: character string in which to store the called number. In general, the called number returned here will be the same as the called number that the application passed to the `OSPPTransactionRequestAuthorisation` function; however, the settlement service provider may perform a number translation on the called number, resulting in a new called number that should be signaled to the peer gateway.

`ospvSizeOfDestination`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the destination information. If the value is not large enough to accommodate the destination, then an error is indicated and no destination is returned.

`ospvDestination`: character string in which to store the identity of the destination. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSizeOfDestinationDevice`: size of the character string (***including*** the terminating `'\0'`) in which the function should store the destination device identity. If the value is not large enough to accommodate the destination device identity, then an error is indicated and no destination is returned.

`ospvDestinationDevice`: character string in which to store the identity of the destination device in a protocol specific manner (e.g. H.323 Identifier); this value is optional and, if it is not present, the returned string is empty.

`ospvSizeOfToken`: pointer to a variable which, on input, contains the size of the memory buffer in which the function should store the authorization token for the destination. If the value is not large enough to accommodate the token, then an error is indicated and no destination is returned. On output this variable is updated to indicate the actual size of the authorization token.

`ospvToken`: memory location in which to store the authorization token for this destination. In general, tokens are opaque, binary objects.

The Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionInitializeAtDevice

```
int
OSPPTransactionInitializeAtDevice(
    OSPTTRANHANDLE  ospvTransaction,
    unsigned        ospvIsSource,
    const char     *ospvSource,
    const char     *ospvDestination,
    const char     *ospvSourceDevice,
    const char     *ospvDestinationDevice,
    const char     *ospvCallingNumber,
    const char     *ospvCalledNumber,
    unsigned        ospvSizeOfCallId,
    const void     *ospvCallId,
    unsigned        ospvSizeOfToken,
    const void     *ospvToken,
    unsigned       *ospvAuthorised,
    unsigned       *ospvTimeLimit,
    unsigned       *ospvSizeOfDetailLog,
    void           *ospvDetailLog
);
```

The `OSPPTransactionInitializeAtDevice` function initializes a (newly created) transaction object. Applications can use this with a distributed architecture in which the systems requesting and validating authorization (e.g. H.323 gatekeepers) are different than the systems that ultimately report usage information (e.g. H.323 gateways). As such, this function is (in a source device) an alternative to the combination of the `OSPPTransactionRequestAuthorisation` function (to initiate a call) and the `OSPPTransactionGetFirstDestination` function (to define the endpoints of the call. In the destination device, this function serves as an alternative to the function `OSPPTransactionValidateAuthorisation`. Parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvIsSource`: indicates whether the system calling this function is acting as the source (if non-zero) or destination (if zero) for the call.

`ospvSource`: character string identifying the source of the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid sources include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvDestination`: character string identifying the destination for the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSourceDevice`: character string identifying the source device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvDestinationDevice`: character string identifying the destination device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvCallingNumber`: character string containing the calling party's number expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvCalledNumber`: character string containing the called number, expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvSizeOfCallId`: size of the memory buffer containing the call identifier.

`ospvCallId`: memory location containing the H.323 call identifier for the call.

`ospvSizeOfToken`: size of the memory buffer containing an authorization token for the call.

`ospvToken`: memory location containing an authorization token.

`ospvAuthorised`: pointer to a variable in which the function will indicate whether or not the call is authorized. On return, a non-zero value indicates that the call is authorized by the provider, while a zero value indicates an authorization failure.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is initially authorized. A value of zero indicates that no limit exists. Note that the initial time limit may be extended during the call by using the function `OSPPTransactionRequestReAuthorisation`.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization validation.

`ospvDetailLog`: pointer to a location in which to store a detail log for the validation. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the authorization confirmation obtained from the settlement provider, including the settlement provider's digital signature.

If the provider has been configured to perform local validation, the Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution. If local validation is not used, this function blocks until authorization has been validated, refused, or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionNew

```
int
OSPPTransactionNew(
    OSPTPROVHANDLE   ospvProvider,
    OSPTTRANHANDLE  *ospvTransaction
);
```

The `OSPPTransactionNew` function creates a new transaction object for `ospvProvider`. A handle to that object is returned to the location pointed to by `ospvTransaction`.

After calling this function to allocate storage for a transaction object, applications should call one of the following three functions to initialize the object:

`OSPPTransactionRequestAuthorisation`: used by the source of a call.

`OSPPTransactionValidateAuthorisation`: used by the destination for a call.

`OSPPTransactionInitialize`: used primarily in architectures that separate the call authorization functions from call setup and usage reporting functions.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionRecordFailure

```
int
OSPPTransactionRecordFailure(
    OSPTTRANHANDLE       ospvTransaction,
    enum OSPEFAILREASON  ospvFailureReason,
);
```

The `OSPPTransactionRecordFailure` function allows an application to record the failure of a call attempt. Applications can use this function when they wish to abandon a call attempt without exhausting the list of possible destinations, and in a distributed architecture in which the system retrieving successive destinations (e.g. an H.323 gatekeeper) is different than the system that ultimately reports usage information (e.g. an H.323 gateway).

The parameters to this function consist of the following:

`ospvTransaction`: handle of the transaction object.

`ospvFailureReason`: the reason that attempts to use the previously identified destination failed; values for this parameter are listed in the `ospfail.h` file.

The Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionReinitializeAtDevice

```
int
OSPPTransactionReinitializeAtDevice(
    OSPTTRANHANDLE       ospvTransaction,
    enum OSPEFAILREASON  ospvFailureReason,
    unsigned             ospvIsSource,
    const char          *ospvSource,
    const char          *ospvDestination,
    const char          *ospvSourceDevice,
    const char          *ospvDestinationDevice,
    const char          *ospvCallingNumber,
    const char          *ospvCalledNumber,
    unsigned             ospvSizeOfCallId,
    const void          *ospvCallId,
    unsigned             ospvSizeOfToken,
    const void          *ospvToken,
    unsigned            *ospvAuthorised,
    unsigned            *ospvTimeLimit,
    unsigned            *ospvSizeOfDetailLog,
    void                *ospvDetailLog
);
```

The `OSPPTransactionReinitializeAtDevice` function re-initializes a (previously initialized) transaction object. Applications can use this with a distributed architecture in which the systems requesting and validating authorization (e.g. H.323 gatekeepers) are different than the systems that ultimately report usage information (e.g. H.323 gateways). The reporting device can call this function after failing to reach a previous destination. As such, this function is an alternative to the `OSPPTransactionGetNextDestination` function. Parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvFailureReason`: the reason that attempts to use the previously identified destination failed; values for this parameter are listed in the `ospfail.h` file.

`ospvIsSource`: indicates whether the system calling this function is acting as the source (if non-zero) or destination (if zero) for the call.

`ospvSource`: character string identifying the source of the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid sources include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvDestination`: character string identifying the destination for the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSourceDevice`: character string identifying the source device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvDestinationDevice`: character string identifying the destination device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvCallingNumber`: character string containing the calling party's number expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvCalledNumber`: character string containing the called number, expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvSizeOfCallId`: size of the memory buffer containing the call identifier.

`ospvCallId`: memory location containing the H.323 call identifier for the call.

`ospvSizeOfToken`: size of the memory buffer containing an authorization token for the call.

`ospvToken`: memory location containing an authorization token.

`ospvAuthorised`: pointer to a variable in which the function will indicate whether or not the call is authorized. On return, a non-zero value indicates that the call is authorized by the provider, while a zero value indicates an authorization failure.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is initially authorized. A value of zero indicates that no limit exists. Note that the initial time limit may be extended during the call by using the function `OSPPTransactionRequestReAuthorisation`.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization validation.

`ospvDetailLog`: pointer to a location in which to store a detail log for the validation. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the authorization confirmation obtained from the settlement provider, including the settlement provider's digital signature.

If the provider has been configured to perform local validation, the Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution. If local validation is not used, this function blocks until authorization has been validated, refused, or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionReportUsage

```
int
OSPPTransactionReportUsage(
    OSPTTRANHANDLE   ospvTransaction,
    unsigned         ospvDuration,
    unsigned         ospvLossPacketsSent,
    signed           ospvLossFractionSent,
    unsigned         ospvLossPacketsReceived,
    signed           ospvLossFractionReceived,
    unsigned        *ospvSizeOfDetailLog,
    void            *ospvDetailLog
);
```

The `OSPPTransactionReportUsage` function reports usage information for a call. Once this function returns successfully, it may not be called again for the life of the transaction object. Parameters to the function are:

`ospvTransaction`: handle of the transaction object.

`ospvDuration`: the duration of the call, in seconds.

`ospvLossPacketsSent`: a count of the total number of packets sent by the reporting system that were not received by its peer, as reported in the peer's RTCP sender and receiver reports. If the `ospvLossFractionSent` parameter has a negative value, this parameter is ignored, and the reporting system is assumed to have neither `ospvLossPacketsSent` or `ospvLossFractionSent` statistics available.

`ospvLossFractionSent`: the fraction of packets sent by the reporting system that were not received by its peer, as reported in the peer's RTCP sender and receiver reports. The fraction is expressed as an integer number from 0 (no loss) to 255 (total loss). If the value of this parameter is negative, the reporting system is assumed to have neither `ospvLossPacketsSent` or `ospvLossFractionSent` statistics available.

`ospvLossPacketsReceived`: a count of the total number of packets that the reporting system expected to receive but did not, as reported in the system's RTCP sender and receiver reports. If the `ospvLossFractionReceived` parameter has a negative value, this parameter is ignored, and the reporting system is assumed to have neither `ospvLossPacketsReceived` or `ospvLossFractionReceived` statistics available.

`ospvLossFractionReceived`: the fraction of packets that the reporting system expected to receive but did not, as reported in it's RTCP sender and receiver reports. The fraction is expressed as an integer number from 0 (no loss) to 255 (total loss). If the value of this parameter is negative, the reporting system is assumed to have neither `ospvLossPacketsReceived` or `ospvLossFractionReceived` statistics available.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the usage report.

`ospvDetailLog`: pointer to a location in which to store a detail log for the usage report. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the usage confirmation obtained from the settlement provider, including the settlement provider's digital signature.

As delivered in the Toolkit library, this function blocks until usage has been reported or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionRequestAuthorisation

```
int
OSPPTransactionRequestAuthorisation(
    OSPTTRANHANDLE   ospvTransaction,
    const char       *ospvSource,
    const char       *ospvSourceDevice,
    const char       *ospvCallingNumber,
    const char       *ospvCalledNumber,
    const char       *ospvUser,
    unsigned          ospvNumberOfCallIds,
    OSPTCALLID       *ospvCallIds[],
    const char       *ospvPreferredDestinations[],
    unsigned         *ospvNumberOfDestinations,
    unsigned         *ospvSizeOfDetailLog,
    void             *ospvDetailLog
);
```

The `OSPPTransactionRequestAuthorisation` function allows an application to request authorization and, optionally, routing information for a transaction. The parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvSource`: character string identifying the source of the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid sources include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSourceDevice`: character string identifying the source device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvCallingNumber`: character string containing the calling party's number expressed as a full international number conforming to the ITU E.164 standard (with no punctuation); if the actual calling party number is unavailable (e.g. because the end user has blocked caller ID services), then the application should supply a local phone number for the device originating the call.

`ospvCalledNumber`: character string containing the called number, expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvUser`: character string identifying the end user (e.g. calling card and PIN number assigned to roaming users); this string may be empty.

`ospvNumberOfCallIds`: the number of call identifiers in the `ospvCallIds` list.

`ospvCallIds`: an array of H.323 call identifiers for the call. The `OSPTCALLID` type consists of a length indicator and a pointer to the binary data. Applications may provide a list of call identifiers in anticipation of the authorization request returning multiple potential destinations. In that case each potential destination is assigned a separate call identifier. An application may also provide only a single call identifier yet still receive multiple potential destinations. In that case the same call identifier value must be used for each destination. If the `ospvCallIds` list contains more than one entry, the number of entries in that list must be the same as the input value of the `ospvNumberOfDestinations` parameter. (Otherwise, an error is returned.)

The value of a call identifier is opaque to the Toolkit and is treated as an arbitrary array of bytes.

`ospvPreferredDestinations`: a list of character strings containing preferred destinations for the call, expressed as either DNS names or IP addresses enclosed in square brackets, followed by an optional colon and TCP port number. The list is terminated by an empty string or a `NULL` pointer, and, if the application has no preferred destinations, the list may be empty. If multiple preferred destinations are included, they are listed in order of decreasing preference.

Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvNumberOfDestinations`: pointer to a variable which, on input, contains the maximum number of destinations the application wishes to consider; on output the variable will be updated with the actual number of destinations authorized.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization request.

`ospvDetailLog`: pointer to a location in which to store a detail log for the authorization request. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the authorization response obtained from the settlement provider, including the settlement provider's digital signature.

As delivered in the Toolkit library, this function blocks until authorization has been received or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionRequestReAuthorisation

```
int
OSPPTransactionRequestReAuthorisation(
    OSPTTRANHANDLE   ospvTransaction,
    unsigned         ospvDuration,
    unsigned        *ospvSizeOfToken,
    void            *ospvToken,
    unsigned        *ospvAuthorised,
    unsigned        *ospvTimeLimit,
    unsigned        *ospvSizeOfDetailLog,
    void            *ospvDetailLog
);
```

The `OSPPTransactionRequestReAuthorisation` function asks the Toolkit library to refresh a previously granted authorization, perhaps, for example, because the time limit for that authorization is nearing its expiration. Parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvDuration`: the duration of the call so far, in seconds.

`ospvSizeOfToken`: pointer to a variable which, on input, contains the size of the memory buffer in which the function should store the authorization token for the destination. If the value is not large enough to accommodate the token, then an error is indicated and no destination is returned. On output this variable is updated to indicate the actual size of the authorization token.

`ospvToken`: memory location in which to store the authorization token for this destination. In general, tokens are opaque, binary objects.

`ospvAuthorised`: pointer to a variable in which the function will indicate whether or not the call is reauthorized. On return, a non-zero value indicates that the call is reauthorized by the provider, while a zero value indicates an authorization failure.

`ospvTimeLimit`: pointer to a variable in which to place the total number of seconds for which the call is now authorized. A value of zero indicates that no limit exists.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization reauthorization.

`ospvDetailLog`: pointer to a location in which to store a detail log for the reauthorization. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the reauthorization response obtained from the settlement provider, including the settlement provider's digital signature.

This function blocks on network input/output until authorization has been refreshed, refused, or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionValidateAuthorisation

```
int
OSPPTransactionValidateAuthorisation(
    OSPTTRANHANDLE  ospvTransaction,
    const char     *ospvSource,
    const char     *ospvDestination,
    const char     *ospvSourceDevice,
    const char     *ospvDestinationDevice,
    const char     *ospvCallingNumber,
    const char     *ospvCalledNumber,
    unsigned        ospvSizeOfCallId,
    const void     *ospvCallId,
    unsigned        ospvSizeOfToken,
    const void     *ospvToken,
    unsigned       *ospvAuthorised,
    unsigned       *ospvTimeLimit,
    unsigned       *ospvSizeOfDetailLog,
    void           *ospvDetailLog
);
```

The `OSPPTransactionValidateAuthorisation` function asks the Toolkit library to validate a requested incoming call, based on the call's parameters and authorization tokens included in the call. This function may be called multiple times for a single call, so that, for example, call requests with multiple authorization tokens may result in multiple calls to this function, one for each authorization token in the request. Parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvSource`: character string identifying the source of the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid sources include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvDestination`: character string identifying the destination for the call. The value is expressed as either a DNS name or an IP address enclosed in square brackets, followed by an optional colon and TCP port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

`ospvSourceDevice`: character string identifying the source device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvDestinationDevice`: character string identifying the destination device in a protocol specific manner (e.g. H.323 Identifier); this string is optional and may be empty.

`ospvCallingNumber`: character string containing the calling party's number expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvCalledNumber`: character string containing the called number, expressed as a full international number conforming to the ITU E.164 standard (with no punctuation).

`ospvSizeOfCallId`: size of the memory buffer containing the call identifier.

`ospvCallId`: memory location containing the H.323 call identifier for the call.

`ospvSizeOfToken`: size of the memory buffer containing an authorization token for the call.

`ospvToken`: memory location containing an authorization token.

`ospvAuthorised`: pointer to a variable in which the function will indicate whether or not the call is authorized. On return, a non-zero value indicates that the call is authorized by the provider, while a zero value indicates an authorization failure.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is initially authorized. A value of zero indicates that no limit exists. Note that the initial time limit may be extended during the call by using the function `OSPPTransactionRequestReAuthorisation`.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization validation.

`ospvDetailLog`: pointer to a location in which to store a detail log for the validation. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the authorization confirmation obtained from the settlement provider, including the settlement provider's digital signature.

If the provider has been configured to perform local validation, the Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution. If local validation is not used, this function blocks until authorization has been validated, refused, or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

OSPPTransactionValidateReAuthorisation

```
int
OSPPTransactionValidateReAuthorisation(
    OSPTTRANHANDLE  ospvTransaction,
    unsigned        ospvSizeOfToken,
    const void     *ospvToken,
    unsigned       *ospvAuthorised,
    unsigned       *ospvTimeLimit,
    unsigned       *ospvSizeOfDetailLog,
```

```
    void           *ospvDetailLog
);
```

The `OSPPTransactionValidateReAuthorisation` function asks the Toolkit library to re-validate an existing call, perhaps in order to increase the time limit for the call. Applications may call this function when they receive an updated authorization token for the call. Parameters to the function are

`ospvTransaction`: handle of the (previously created) transaction object.

`ospvToken`: memory location containing an authorization token.

`ospvAuthorised`: pointer to a variable in which the function will indicate whether or not the call is still authorized. On return, a non-zero value indicates that the call is authorized by the provider, while a zero value indicates an authorization failure.

`ospvTimeLimit`: pointer to a variable in which to place the number of seconds for which the call is now authorized. A value of zero indicates that no limit exists. Note that the resulting time limit is the cumulative time limit for the call, and it includes the duration of the call up to the reauthorization.

`ospvSizeOfDetailLog`: pointer to a variable which, on input, contains the maximum size of the detail log; on output, the variable will be updated with the actual size of the detail log. By setting this value to zero, applications indicate that they do not wish a detail log for the authorization validation.

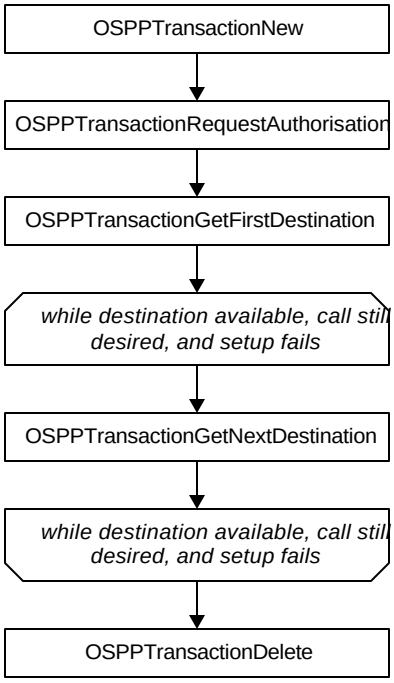
`ospvDetailLog`: pointer to a location in which to store a detail log for the validation. If this pointer is not `NULL`, and if the `ospvSizeOfDetailLog` parameter is non-zero, then the library will store a copy of the reauthorization confirmation obtained from the settlement provider, including the settlement provider's digital signature.

If the provider has been configured to perform local validation, the Toolkit library is able to perform this function without network interaction, and, therefore, does not block for network input or output during its execution. If local validation is not used, this function blocks until authorization has been validated, refused, or an error has been detected. The *Open Settlement Protocol Toolkit Porting Guide* includes information on modifying that behavior to prevent blocking.

The function returns an error code or zero (if the operation was successful). Specific error codes and their meanings can be found in the *OSP Toolkit Errorcode List* document.

Application Program Flow

This section describes how an application creates and manages transaction objects. It does this by documenting the possible program flows for both source and destination systems, each under three different scenarios. The scenarios correspond to systems perform authorization functions only, usage reporting only, and both authorization and reporting. Consult the Toolkit Implementation Guide for more information on the applicability of these scenarios to particular network environments and architectures.



- Figure 1 Transaction Object Program Flow for Authorizing Source Systems.

Source System

Three different processing flows are valid for source systems, depending on the role played by that system in the call. The possible roles include authorization, usage reporting, or both authorization and reporting.

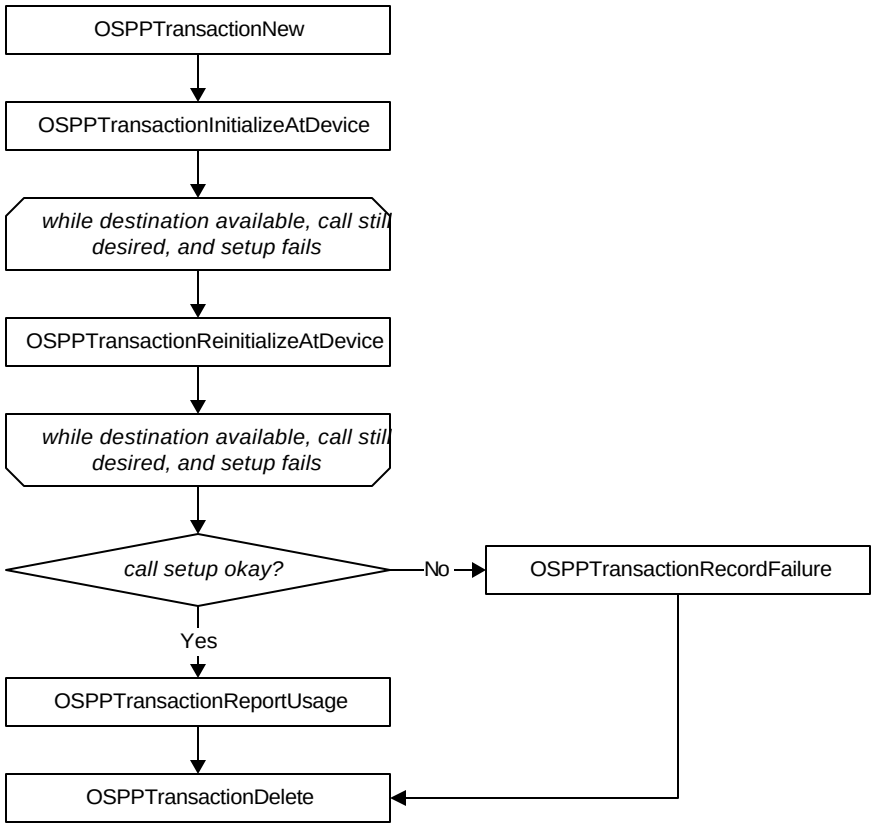
Authorization Only

If a source system performs authorization services only, it interacts with the Toolkit according to the program flow of Figure 1. The system creates a new transaction object, requests authorization for the phone call, and then enters a loop retrieving candidate destinations.

The loop begins with a call to `OSPPTransactionGetFirstDestination`. If that destination is not available, or is otherwise unacceptable to the system, it can call the function `OSPPTransactionGetNextDestination`. The application loops on repeated calls to this function until one of three things happen: either (a) the call setup succeeds, (b) the application exhausts all possible destinations, or (c) the application abandons the call attempt (without trying all candidate destination). At that point, the application calls the function `OSPPTransactionDelete`.

Usage Reporting Only

A source system may be part of a network architecture in which it simply reports usage information; it relies on a different system for authorization. In that environment, the source system uses a transaction object as Figure 2 shows. As the figure indicates, the source systems begins by creating a transaction object and then entering a loop.



• Figure 2 Transaction Object Program Flow for Reporting Source Systems.

The loop itself beings with a call to `OSPPTransactionInitializeAtDevice` and, if necessary, includes multiple calls to `OSPPTransactionReinitializeAtDevice`. The process continues until either the call setup succeeds or the system abandons the call attempt. If the call is successful, then the system simply reports the usage information and deletes the transaction object. If the call attempt ultimately fails, however, the system must call `OSPPTransactionRecordFailure` to indicate the reason for the failure of the final destination. It can then report usage (presumably a value of zero) and delete the object.

Authorization and Reporting

Although source devices may act solely as authorizing or reporting systems, a single system will typically perform both functions. In that environment, the same source system requests authorization for a call, and it reports usage information for the call. Figure 3 shows the program flow that the system follows in interacting with a transaction object.

As the figure shows, the system creates a new transaction object and requests authorization. It then enters a loop retrieving candidate destinations, beginning with a call to `OSPPTransactionGetFirstDestination` and continuing (if necessary) with subsequent calls to `OSPPTransactionGetNextDestination`.

The system breaks out of the loop when one of three things occurs:

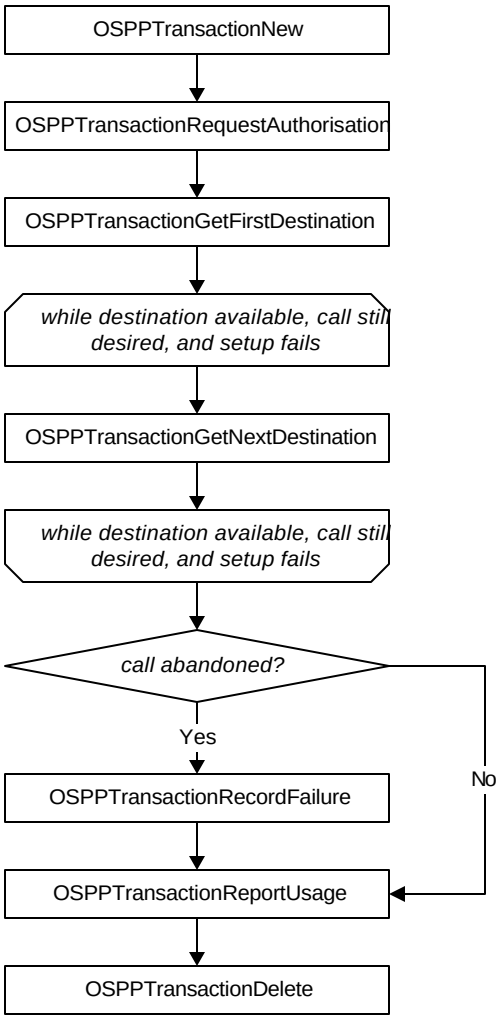
The call succeeds: In this case the system reports usage and deletes the transaction object.

The list of candidate destinations is exhausted: In this case the system reports usage (with a value of zero) and deletes the transaction object

The system abandons the call attempt without exhausting all candidate destinations: In this case the system must call `OSPPTransactionRecordFailure` before reporting usage information and deleting the transaction. Without this call, the Toolkit will assume that the last destination it provided was successfully reached, even though the duration may have value of zero.

Destination System

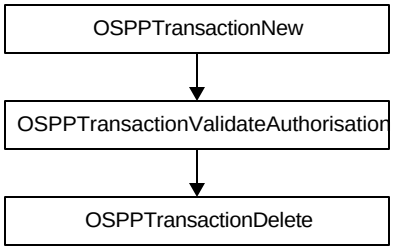
Program flow in destination systems is simpler than that in source systems, primarily because there is no issue of multiple, candidate peers (i.e. there is always only a single source system). Nonetheless, three different architectures are still valid: an authorizing only system, a reporting only system, and an authorizing and reporting system.



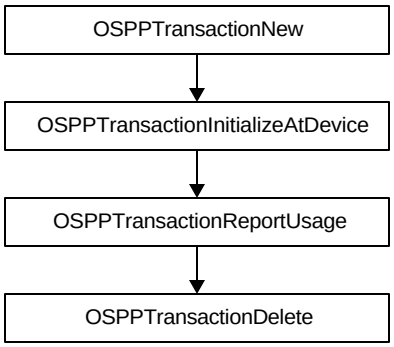
• Figure 3 Transaction Object Program Flow for Authorizing and Reporting Source Systems.

Authorization Only

The simplest case of all is shown in Figure 4 below, in which the destination system simply authorizes calls. To do that, it creates a new transaction object, calls `OSPPTransactionValidateAuthorisation`, and then deletes the object.



• Figure 4 Transaction Object Program Flow in Authorizing Destination Systems.



• Figure 5 Transaction Object Program Flow in Reporting Destination Systems.

Usage Reporting Only

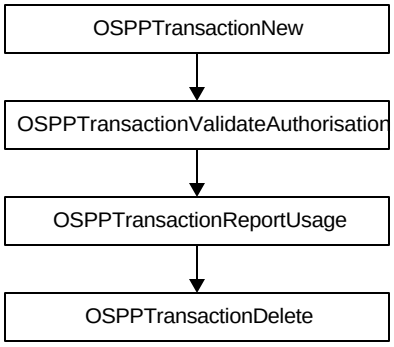
When the destination simply reports usage information, it can follow the program flow of Figure 5. The application creates a new transaction object, initializes it, reports the usage information, and deletes the object. Note that, in contrast with the source system, the destination system never reinitializes a transaction object.

Authorization and Reporting

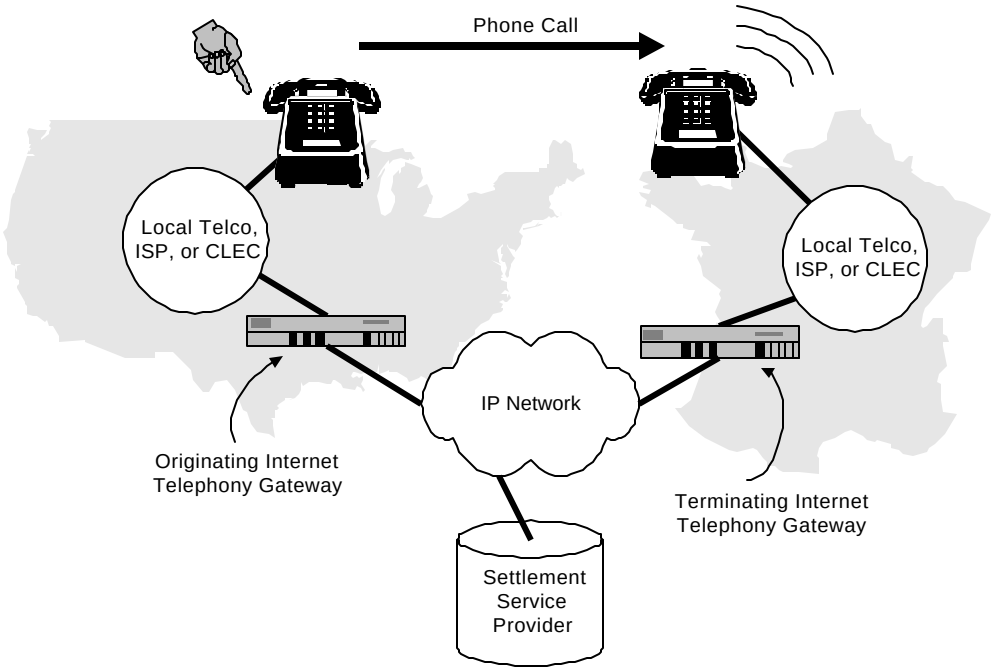
The final case is a destination system that both authorizes calls and reports their usage information. Such systems interact with transaction objects based on the program flow of Figure 6. As that figure shows, the system creates a new object and then validates the authorization for the transaction. After the phone call has finished, it reports usage information and deletes the transaction object.

Example Usage

This section presents an example usage scenario for the Toolkit library. It shows how to use the various Toolkit functions during a normal point-to-point telephone call. (Note: the code fragments in this section are designed strictly to illustrate the use of Toolkit library functions; they do not necessarily represent recommended coding practices.) The scenario is divided into four difference phases, beginning with system startup and finishing with system shutdown. In between, the text considers the actions of an originating gateway or gatekeeper and a terminating gateway or gatekeeper. Figure 7 provides the



• Figure 6 Transaction Object Program Flow for Authorizing and Reportting Destination Systems.



• Figure 7 Systems Involved in Example Transaction.

context for the example. In the example of the figure, the Toolkit library executes on both the originating and terminating gateways.

Note: Additional usage scenarios may be found in the Toolkit document *Implementation Guidelines*. Although that document does not provide the same level of detail as this section, it does offer guidance for the use of the Toolkit library in scenarios other than a normal, point-to-point telephone call.

System Startup

```
 1  #include "osp.h"           /* include Open Settlement Protocol */
 2
 3  ...
 4
 5  int errCode;                    /* place to store error code */
 6
 7  ...
 8
 9  errCode = OSPPInit();         /* initialize OSP Toolkit functions */
10
11  if (errCode != OSPC_ERR_NO_ERROR) {
12       /* respond to error condition, and break, return, or exit */
13  }
```

line 1

The `osp.h` header file includes all necessary function prototypes, as well as type and constant definitions.

line 9

The call to `OSPPInit` initializes the open settlement protocol Toolkit functions.

lines 11-13

Be sure to check for possible error returns from the initialization routine.

Provider Initiation

As part of their system startup, each of the two gateways must initialize provider objects for the settlement service providers that they intend to use. Initialization requires a call to `OSPPProviderNew` with appropriate configuration information, which is typically kept in some form of non-volatile storage. How the gateways store and access that information it outside the scope of the OSP software development kit. A call to `OSPPProviderNew` might be executed as shown in the following code fragment. Parameters in the code fragment are shown as constants for clarity; actual implementations would typically use variables.

```
14  OSPTPROVHANDLE hProv;         /* handle to settlement provider */
15
16  ...
17
18  errCode = OSPPProviderNew (
19              4,                    /* number of service points */
20              {                   /* service points for provider */
21               "service.transnexus.com/scripts/voice/osp.cmd",
22               "service.transnexus.ga.us/scripts/voice/osp.cmd",
23               "service.transnexus.co.uk/scripts/voice/osp.cmd",
24               "service.transnexus.co.jp/scripts/voice/osp.cmd"
25              },
26              "",                          /* audit URL (not used) */
27              &privKey,          /* private key used for signing */
28              &localCert,       /* local public key certificate */
29              10,                 /* number of CA keys to trust */
30              CACerts,              /* array of CA certificates */
31              1,                          /* use local validation */
32              3600,              /* SSL session key life of 1 hr */
33              32,                /* max simultaneous connections */
34              60,                   /* HTTP persistence of 1 min */
35              600,                /* HTTP retry delay of 10 min */
36              3,                  /* HTTP retry limit of 3 times */
37              3500,                   /* HTTP timeout of 3.5 sec */
38              "",                      /* Customer ID (not used) */
39              "",                         /* Device ID (not used) */
40              &hProv                           /* returned handle */
41          );
```

line 14

Although other parameters may, in theory, be passed as constants, a memory location for the provider handle is essential and must be declared. Applications must treat `OSPTPROVHANDLE` as an opaque data structure.

line 19

This provider supports four service points.

lines 20-25

The four service points are listed in order of preference. In the example above, `service.transnexus.com` is the DNS name of the preferred service point. Should that service point be unavailable, the Toolkit library would then try to use the service point `service.transnexus.ga.us`. The function call also specifies additional fallbacks of `service.transnexus.co.uk` and finally to `service.transnexus.co.jp`.

Line 26

Separate URL that accepts audit data. In this example auditing is not used.

lines 27-28

Pointers to the local system's private key and corresponding public key certificate. The public key certificate, which must conform to X.509 distinguished encoding rules, includes identification of the digest and signing algorithms.

lines 29-30

The number of certificate authority public keys and the corresponding public key certificates. These are the public keys trusted by the local system for authenticating the identity of the settlement service provider during the SSL exchange.

line 31

This provider validates authorization tokens locally.

lines 32-37

Communication parameters for this provider. The example specifies a lifetime of 1 hour (3600 seconds) for SSL session keys, and HTTP persistence of 1 minute (60 seconds), and it limits the library to no more than 32 simultaneous active connections with the provider. On failure, connection attempts are retried every 10 minutes (600 seconds) up to a limit of 3 retries. The library is also instructed to wait no more than 3.5 seconds (3500 milliseconds) for a response.

lines 38-39

The customer and device IDs for the system with the indicated provider. In this example, these parameters are not used (because, for example, the particular provider does not need the information or it can obtain the information from other forms, such as the public key certificate). In other cases, though, the settlement provider will assign values for these parameters.

line 40

The address of a variable in which to return a handle to the new provider object.

Originating Gateway

This example scenario breaks up the actions of the originating gateway (or gatekeeper) into several different phases. First, the gateway requests authorization from a settlement server. Once authorization is obtained, it then retrieves the authorized destinations and sets up a call with the terminating gateway. During the life of that call, the gateway reports network performance measurements that it makes, and, once the call is finished, the gateway reports the final usage information.

Requesting Authorization

A transaction begins when the originating gateway needs to complete a phone call. To obtain routing and authorization information, the gateway must create a transaction object and request authorization. The following code fragment shows this operation. Parameters in the code fragment are shown as constants for clarity; actual implementations would typically use variables.

```
42  OSPTTRANHANDLE hTrans;              /* handle to a transaction */
43  unsigned numDest;                    /* number of destinations */
44  unsigned logSize = 0;                     /* size of detail log */
45
46  ...
47
48  errCode = OSPPTransactionNew(hProv, &hTrans);
49
50  if (errCode != OSPC_ERR_NO_ERROR) {
51       /* respond to error condition, and break, return, or exit */
52  }
53
54  numDest = 5;                    /* accept up to 5 destinations */
55
56  errCode = OSPPTransactionRequestAuthorisation (
57                hTrans,                     /* transaction handle */
58                "[172.16.1.2]",                /* source for call */
59                "",                           /* source device ID */
60                "14048724887",             /* calling party number */
61                "33169186100",              /* called party number */
62                NULL,                         /* not a roaming user */
63                1,                           /* one call ID provided */
64                &CallId,                            /* H.323 Call ID */
65                NULL,                        /* no preferred routes */
66                &numDest,                       /* max destinations */
67                &logSize,                     /* size of detail log */
68                NULL                                /* no detail log */
69           );
```

line 42

A variable in which to store the handle to a transaction, once it's created.

line 43

A variable used for input and output to `OSPPTransactionRequestAuthorisation` (and thus cannot be a constant). On input, it indicates the maximum number of different destinations to return; on output, it contains the actual number of destinations available.

line 44

Another variable used for both input and output. In this case it's the size of the detail log. This example does not use a detail log, and so this variable remains set to zero.

line 48

Before using a transaction object, the application must create one. Note that the call to `OSPPTransactionNew` must indicate a (previously created) provider object to which the transaction applies.

line 50-52

The call to `OSPPTransactionNew` must return successfully (with no error) before the application can proceed to use the transaction object. The body of this `if` clause must not fall through to the following code.

line 54

Set up the maximum number of destinations the application is prepared to accept.

line 56

Actually request authorization. Note that this function will block until a response is received or an error is detected.

line 57

The handle to the transaction object created in the call to `OSPPTransactionNew` (line 39).

line 58

The IP address (or DNS name) of the device initiating the call. As the example shows, IP addresses are expressed using the DNS notation, which encloses the dotted decimal address in square brackets.

line 59

The protocol-specific device identifier for the source. In this example this value is not used.

line 60

The calling party's number in full E.164 notation. In this case the calling party is from the United States (country code 1) and is at number (404) 872-4887. When the application

does not know the calling party number (because, for example, that party has blocked caller ID), it may supply a default local number of the initiating device.

line 61

The called number in full E.164 notation. In this case the called party is in France (county code 33) and is at the number (0)1-69-18-6100.

line 62

This example does not represent a roaming user, and thus this parameter is `NULL`. If the user was roaming, this parameter would be a character string containing the user's calling card and/or personal identification numbers.

line 63-64

The number of, and values for, H.323 (version 2) call identifiers which uniquely identify the call. This example supplies a single call ID to be used for all destinations; the actual value (and its size, in bytes) is contained in the CallId array.

line 65

In this example the application does not indicate any preferred routes, and thus this parameter is `NULL`. If the application did wish to suggest preferred destinations, it would do so by passing a list of character strings in this parameter.

line 66

On input, this variable indicates the maximum number of destinations the application is prepared to accept. On output, the variable will hold the actual number of destinations available.

lines 67-68

In this example, the application does not need a detailed log of the transaction. If such a log were required, the application would indicate the maximum size for the log and a pointer to the memory location for the log. On successful return, that memory location would contain the complete binary message received from the settlement server, including any digital signatures. The `logSize` variable would also be updated with the correct size of that message.

Retrieving the First Destination

Once the authorization request has succeeded, the originating gateway may retrieve the destinations that have been authorized. It begins this process by asking the Toolkit library for the first authorized destination.

```
70  char dest[262];                  /* place to store destination */
71  char calledNum[32];            /* place to store called number */
72  unsigned char callId[40];    /* place to store call identifier */
73  unsigned callIdSize = sizeof(callId);         /* and it's size */
74  unsigned char token[2048];             /* place to store token */
```

```
75  unsigned tokenSize = sizeof(token);        /* and token's size */
76  unsigned timeLimit;             /* initial time limit for call */
77
78  errCode = OSPPTransactionGetFirstDestination (
79                hTrans,                     /* transaction handle */
80                0,                       /* no timestamps needed */
81                NULL,
82                NULL,
83                &timeLimit,  /* for how long is call authorised? */
84                &callIdSize,                    /* H.323 call ID */
85                callId,
86                sizeof(calledNum),    /* (translated) called num */
87                calledNum,
88                sizeof(dest),            /* destination for call */
89                dest,
90                0,                         /* device ID not used */
91                NULL,
92                &tokenSize,               /* authorisation token */
93                token
94            );
```

lines 70-75

These variables store the output from `OSPPTransactionGetFirstDestination`. That output includes a destination, the (possible translated) called number, an H.323 call identifier, an authorization token, and a time limit for the call. The destination may include a complete DNS name (up to 255 characters), a colon, and a 16-bit (or 5 digit) port number. Including the terminating `'\0'` gives the final size of 262 characters. Call identifier sizes depend on the application protocol, and token sizes depend on the settlement service provider.

lines 80-82

The application may use these parameters to request the validity times for the authorization. In this example, no such request is made so that the parameters are `0`, `NULL`, and `NULL`, respectively.

line 83

A variable in which to store the initial time limit authorized for the call. A value of zero indicates no time limit. Note that the peer gateway may refresh the time limit during a call.

lines 84-85

The maximum size and location for storing the H.323 (version 2) call identifier for this destination. The value is stored as a binary array, and its actual length is placed in `callIdSize`.

lines 86-87

The maximum size and location for storing the called number. In general, the number returned here is the same as passed to `OSPPTransactionRequestAuthorisation` (line 61), but that is not guaranteed to be the case. The server may have performed a

number translation that results in a new called number, in which case the new number is returned here. The application should always use this number in its setup message to the peer gateway.

lines 88-89

The maximum size and location for storing the destination's addressing information. That information is stored as a character string containing either the DNS name or the IP address (enclosed in square brackets), optionally followed by a port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

lines 90-91

The maximum size and location for storing the protocol-specific device identifier. Not used in this example.

lines 92-93

The maximum size and location for storing the authorization token for this destination. On output, the token will be in the `token` array and `tokenSize` will indicate its size in bytes.

Retrieving Subsequent Destinations

If the first destination is unavailable (or otherwise unacceptable to the application), it may request additional destinations. The Toolkit library will provide up to as many destinations as were returned in the authorization request. Each destination is retrieved from the library with calls to `OSPPTransactionGetNextDestination`.

```
 95  tokenSize = sizeof(token);           /* reset token's max size */
 96  callIdSize = sizeof(callId);         /* and reset call ID size */
 97
 98  errCode = OSPPTransactionGetNextDestination (
 99                hTrans,                     /* transaction handle */
100                OSPC_FAIL_REMOTE_TCPFIN,   /* why prev. failure? */
101                0,                        /* no timestamps needed */
102                NULL,
103                NULL,
104                &timeLimit,               /* time limit for call */
105                &callIdSize,                    /* H.323 call ID */
106                callId,
107                sizeof(calledNum),    /* (translated) called num */
108                calledNum,
109                sizeof(dest),            /* destination for call */
110                dest,
111                0,                        /* device ID not used */
112                NULL,
113                &tokenSize,               /* authorisation token */
114                token
115             );
```

lines 95-96

Reset the variables that hold the size of the call ID and token. These value would have been set to the actual sizes on return from `OSPPTransactionGetFirstDestination`. They need to be reset to properly indicate maximum sizes on the forthcoming call to `OSPPTransactionGetNextDestination`.

line 100

A constant (defined in ospfail.h) indicating the reason for the failure of the previous destination. In this example, the application indicates that it's TCP connection to that destination request was refused.

lines 101-103

The application may use these parameters to request the validity times for the authorization. In this example, no such request is made so that the parameters are `0`, `NULL`, and `NULL`, respectively.

line 104

A variable in which to store the initial time limit authorized for the call. A value of zero indicates no time limit. Note that the peer gateway may refresh the time limit during a call.

lines 105-106

The maximum size and location for storing the H.323 (version 2) call identifier for this destination. The value is stored as a binary array, and its actual length is placed in `callIdSize`.

lines 107-108

The maximum size and location for storing the called number. In general, the number returned here is the same as passed to `OSPPTransactionRequestAuthorisation` (line 61), but that is not guaranteed to be the case. The server may have performed a number translation that results in a new called number, in which case the new number is returned here. The application should always use this number in its setup message to the peer gateway.

lines 108-110

The maximum size and location for storing the destination's addressing information. That information is stored as a character string containing either the DNS name or the IP address (enclosed in square brackets), optionally followed by a port number. Examples of valid destinations include `"gateway1.carrier.com"` and `"[172.16.1.2]:112"`.

lines 111-112

The maximum size and location for storing the protocol-specific device identifier. Not used in this example.

lines 113-114

The maximum size and location for storing the authorization token for this destination. On output, the token will be in the `token` array and `tokenSize` will indicate its size in bytes.

Accumulating Statistics

Once the call is established, the gateway may report delay statistics for the call as they are gathered. The following code fragment demonstrates how to report one-way and round trip delay statistics. As before, note that the sample code uses constants as the parameters for clarity. Actual implementations would certainly use variables to store the data.

```
116 errCode = OSPPTransactionAccumulateOneWayDelay (
117               hTrans,                     /* transaction handle */
118               100,                         /* number of samples */
119               109,                  /* minimum delay of 109 ms */
120               193,                      /* mean delay of 193 ms */
121               96.322                /* variance of 96.3 (ms)^2 */
122           );
123
124 if (errCode != OSPC_ERR_NO_ERROR) {
125     /* respond to error condition appropriately */
126 }
127
128 errCode = OSPPTransactionAccumulateRoundTripDelay (
129               hTrans,                     /* transaction handle */
130               1,                       /* just a single sample */
131               340,                  /* minimum delay of 340 ms */
132               340,                     /* mean delay of 340 ms */
133               0.0                      /* variance of 0 (ms)^2 */
134           );
```

line 116

This call reports one-way delay statistics.

lines 118-121

The application reports a summary of 100 one-way delay measurements in which the minimum delay was 0.109 s, the mean of all 100 was 0.193 s, and the variance was 0.000096322 s^2.

lines 124-126

The application should be prepared to handle an error return from the function call. The actual action to be taken depending on the specific error encountered.

lines 128-134

The application also reports statistics for round trip delay. In this case, the report is for a single measurement of 0.34 s. Note that for a single sample, the minimum and mean values are the same, and the variance is zero.

Reporting Usage

Once the call is finished, the originating gateway calls `OSPPTransactionReportUsage` to report the usage details to the settlement provider. The following code fragment illustrates such a call, as well as the call to `OSPPTransactionDelete` that destroys the transaction object.

```
135  errCode = OSPPTransactionReportUsage (
136                hTrans,                      /* transaction handle */
137                300,                               /* 5 min call */
138                401,                 /* 401 pkts xmit'd but lost */
139                5,                  /* 5/255 pkts xmit'd but lost */
140                252,                       /* 252 pkts not rcv'd */
141                3,                       /* 3/255 pkts not rcv'd */
142                &logSize,              /* max size of detail log */
143                NULL                     /* no detail log desired */
144           );
145
146  if (errCode != OSPC_ERR_NO_ERROR) {
147                    /* respond to error condition appropriately */
148  }
149
150  errCode = OSPPTransactionDelete(hTrans);
```

line 135

Report the final usage information for the call. Note that this function will block until the usage is successfully reported or an error is detected.

line 136

Handle to the transaction object originally created for this call.

line 137

The duration of the call in seconds, in this case the call was 5 minutes (300 s) long.

line 138

Of the packets transmitted by this system, 401 were not received by the peer system.

line 139

The 401 lost packets were 2% (5/255) of the total packets sent by this system.

line 140

This system did not receive 252 packets that were sent by its peer.

line 141

The 252 missing packets were 1% (3/255) of the total that were sent by the peer.

lines 142-143

The application does not wish a detail log for the transaction. The `logSize` variable is zero (line 44) and the pointer to the memory area for logging is `NULL`. Either condition by itself will prevent logging, but both are set here to be safe.

line 150

At the conclusion of the transaction, delete the transaction object and free its resources.

Terminating Gateway

The peer, or terminating gateway, takes a complementary set of actions in this example scenario. It first validates the authorization token contained in the call setup message. After accepting the call, the gateway then reports network performance measurements made during the call and, once the call is finished, final usage information.

Validating Authorization

A transaction begins when the terminating gateway receives a setup message that it cannot independently authorize. To see if it should accept the call, the gateway creates a new transaction object and asks the Toolkit library to validate any tokens contained in the setup message.

```
151  OSPTTRANHANDLE hTrans;              /* handle to a transaction */
152  unsigned authorized = 0;                /* is call authorized? */
153
154  ...
155
156  errCode = OSPPTransactionNew(hProv, &hTrans);
157
158  if (errCode != OSPC_ERR_NO_ERROR) {
159       /* respond to error condition, and break, return, or exit */
160  }
161
162  errCode = OSPPTransactionValidateAuthorisation (
163                hTrans,                     /* transaction handle */
164                "[172.16.1.2]",                /* source for call */
165                "[10.1.2.3]",             /* destination for call */
166                "",                 /* source device ID (not used) */
167                "",                  /* dest. device ID (not used) */
168                "14048724887",             /* calling party number */
169                "33169186100",               /* called party number */
170                38,                       /* size of H.323 Call ID */
171                callId,                  /* array containing call ID */
172                1155,                /* token is 1155 bytes in size */
173                token,                     /* array containing token */
174                &authorized,                /* is call authorized? */
175                &timeLimit,                 /* time limit for call */
176                &logSize,                       /* detail log size */
177                NULL                       /* no detail log needed */
178           );
179
180  if (errCode != OSPC_ERR_NO_ERROR) {
```

```
181     /* respond to error condition */
182 }
183
184 if (!authorized) {
185           /* if the authorisation was not valid, deny the call */
186 }
```

line 151

A variable in which to store the handle to a transaction, once it's created.

line 152

A variable that `OSPPTransactionValidateAuthorisation` will use to indicate whether or not the call is authorized..

line 156

Before using a transaction object, the application must create one. Note that the call to `OSPPTransactionNew` indicates a (previously created) provider object to which the transaction applies.

line 158-160

The call to `OSPPTransactionNew` must return successfully (with no error) before the application can proceed to use the transaction object. The body of this `if` clause must not fall through to the following code.

line 162

Actually request validation. In general, his function does may or may not block for network input/output depending on whether the provider supports local validation.

line 163

The handle to the transaction object created in the call to `OSPPTransactionNew` (line 155).

line 164

The IP address (or DNS name) of the device initiating the call. As the example shows, IP addresses are expressed using the DNS notation, which encloses the dotted decimal address in square brackets.

line 165

The IP address (or DNS name) of the device deciding whether or not to accept the call request. As the example shows, IP addresses are expressed using the DNS notation, which encloses the dotted decimal address in square brackets.

lines 166-167

The protocol-specific device identifiers for the source and destination devices. Not used in this example.

line 168

The calling party's number in full E.164 notation. In this case the calling party is from the United States (country code 1) and is at number (404) 872-4887.

line 169

The called number in full E.164 notation. In this case the called party is in France (county code 33) and is at the number (0)1-69-18-6100.

lines 170-171

The size and value for the H.323 (version 2) call identifier, which uniquely identifies the call. The value itself is an arbitrary array of bytes.

lines 172-173

The size and value of the authorization token being used to validate the call request. Note that if the setup message contains multiple tokens, the application may use this function multiple times, one for each token, until a valid token is found or all tokens are exhausted.

line 174

A variable in which `OSPPTransactionValidateAuthorisation` can store an indication of whether or not the call is authorized.

line 175

A variable in which the function can store the initial time limit for the call. A value of zero indicates no limit. Note that the gateway may request extension of this time limit during the call.

lines 176-177

The application does not wish a detail log for the transaction. The `logSize` variable is zero (line 44) and the pointer to the memory area for logging is `NULL`. Either condition by itself will prevent logging, but both are set here to be safe.

lines 179-181

Once more, check for error return before proceeding.

lines 184-186

If the authorization was valid, accept the call; otherwise, the call should be refused.

Accumulating Statistics

Once the call is established, the gateway may report delay statistics for the call as they are gathered. The following code fragment demonstrates how to report one-way and round trip delay statistics.

```
187 errCode = OSPPTransactionAccumulateOneWayDelay (
188               hTrans,                     /* transaction handle */
189               100,                         /* number of samples */
190               109,                  /* minimum delay of 109 ms */
191               193,                      /* mean delay of 193 ms */
192               96.322                /* variance of 96.3 (ms)^2 */
193           );
194
195 if (errCode != OSPC_ERR_NO_ERROR) {
196      /* respond to error condition appropriately */
197 }
198
199 errCode = OSPPTransactionAccumulateRoundTripDelay (
200               hTrans,                     /* transaction handle */
201               1,                       /* just a single sample */
202               340,                  /* minimum delay of 340 ms */
203               340,                     /* mean delay of 340 ms */
204               0.0                      /* variance of 0 (ms)^2 */
205           );
```

line 187

This call reports one-way delay statistics.

lines 188-193

The application reports a summary of 100 one-way delay measurements in which the minimum delay was 0.109 s, the mean of all 100 was 0.193 s, and the variance was 0.000096322 s^2.

lines 194-197

The application should be prepared to handle an error return from the function call. The actual action to be taken depending on the specific error encountered.

lines 199-205

The application also reports statistics for round trip delay. In this case, the report is for a single measurement of 0.34 s. Note that for a single sample, the minimum and mean values are the same, and the variance is zero.

Reporting Usage

Once the phone call is finished, the terminating gateway reports usage details to the settlement provider. It does so via a call to `OSPPTransactionReportUsage`. The following code fragment illustrates this and a call to `OSPPTransactionDelete` that destroys the transaction object.

```
206  errCode = OSPPTransactionReportUsage (
207                hTrans,                      /* transaction handle */
208                308,                               /* 308 s call */
209                252,                 /* 252 pkts xmit'd but lost */
210                3,                  /* 3/255 pkts xmit'd but lost */
211                401,                       /* 401 pkts not rcv'd */
212                5,                       /* 5/255 pkts not rcv'd */
213                &logSize,              /* max size of detail log */
214                NULL                    /* no detail log desired */
215             );
216
217  if (errCode != OSPC_ERR_NO_ERROR) {
218                    /* respond to error condition appropriately */
219  }
220
221  errCode = OSPPTransactionDelete(hTrans);
```

line 206

Report the final usage information for the call. Note that this function will block until the usage is successfully reported or an error is detected.

line 207

Handle to the transaction object originally created for this call.

line 208

The duration of the call in seconds, in this case the call was approximately 5 minutes (308 seconds) long.

line 209

Of the packets transmitted by this system, 252 were not received by the peer system.

line 210

The 252 lost packets were 1% (3/255) of the total packets sent by this system.

line 211

This system did not receive 401 packets that were sent by its peer.

line 212

The 401 missing packets were 2% (5/255) of the total that were sent by the peer.

lines 213-214

The application does not wish a detail log for the transaction. The `logSize` variable is zero (line 44) and the pointer to the memory area for logging is `NULL`. Either condition by itself will prevent logging, but both are set here to be safe.

line 224

At the conclusion of the transaction, delete the transaction object and free its resources.

System Shutdown

When either gateway wishes to shutdown, or otherwise terminate its association with a settlement provider, it should delete the provider object as indicated in the following code fragment.

```
222  errCode = OSPPProviderDelete (
223               hProv,                        /* provider to delete */
224               -1                             /* wait indefinitely */
225             );
```

line 222

The function that deletes a provider object and frees its resources. Note that this function may block until all pending transactions with the provider are complete. The time limit parameter (line 224) can be used to place a limit on the blocking time.

line 223

The handle to the provider to delete.

line 224

The time limit to wait for successful deletion. A negative value indicates that the function should wait as long as necessary for pending transactions to complete.