

|  |
| --- |
|  |
| Mobile Authentication Corporation |
| MAC OTP System Services API, Version 1.8 |

|  |
| --- |
|  |

# Introduction

This document defines the API for the MAC OTP System (“System”) Web Services. The System is comprised of several Web Services (“Services”) that provide the functionality needed to *Register* End Users, and *Send* and *Verify* One-Time Passwords sent as messages to End Users (“OTP Messages”). OTP Messages are sent to the End Users using SMS (i.e., Short Message Service or texting) or Email networks depending on the Client’s configuration. The API also supports sending regular text messages, containing no OTP, as notification messages.

This document also covers the API for MAC End User Registration (“Registration”). Registering End Users allows the Client the ability to set restrictions on how the End User information can be used.

Note: SMS is the recommended delivery network. Delivering OTP messages via SMS provides the additional security of using a different communication channel.

Contents

[Mobile Authentication Corporation 0](#_Toc409532354)

[Introduction 1](#_Toc409532355)

[Overview 5](#_Toc409532356)

[System diagram 5](#_Toc409532357)

[Diagram Components 5](#_Toc409532358)

[End Users (1) 5](#_Toc409532359)

[Client Managed End Users 5](#_Toc409532360)

[Registered End Users 6](#_Toc409532361)

[End User’s Computing Device (2) 6](#_Toc409532362)

[Client’s Web Server (3) 6](#_Toc409532363)

[MAC OTP Server (4) 6](#_Toc409532364)

[Secure Ads Server (5) 6](#_Toc409532365)

[Aggregator / Gateway Server (6) 6](#_Toc409532366)

[Carriers (7) 6](#_Toc409532367)

[End User’s Mobile Device (8) 6](#_Toc409532368)

[Text Messaging and Short Code Considerations 7](#_Toc409532369)

[Reply messages 7](#_Toc409532370)

[HELP Reply message 7](#_Toc409532371)

[STOP Reply Message 7](#_Toc409532372)

[Service API Requests / Responses 8](#_Toc409532373)

[Request Format Details 8](#_Toc409532374)

[Transaction Type (TrxType) Encoding 8](#_Toc409532375)

[Transaction Details Format and Encoding 9](#_Toc409532376)

[Request Data Encoding Details 9](#_Toc409532377)

[Request Data Encrypted Details 9](#_Toc409532378)

[Response Format Details 10](#_Toc409532379)

[OTP Services 12](#_Toc409532380)

[Registration Services 12](#_Toc409532381)

[Pre End User Verification Registration 12](#_Toc409532382)

[Post End User Verification Registration 12](#_Toc409532383)

[End User File Registration 12](#_Toc409532384)

[End User Management Service 12](#_Toc409532385)

[Client Management Service 12](#_Toc409532386)

[Usage Billing Service 13](#_Toc409532387)

[Service URL 13](#_Toc409532388)

[AdPass 14](#_Toc409532389)

[Overview of how AdPass Works 14](#_Toc409532390)

[AdPass Request Options 14](#_Toc409532391)

[Ad Response Details 14](#_Toc409532392)

[Service Request and Response Details 15](#_Toc409532393)

[Send OTP Service 15](#_Toc409532394)

[Response 15](#_Toc409532395)

[Send OTP Message to a Client Managed End User 15](#_Toc409532396)

[Send OTP Message to a Registered End User 15](#_Toc409532397)

[Cancel an OTP 16](#_Toc409532398)

[Resend OTP Message 16](#_Toc409532399)

[Send Text Message to a Client Managed End User 16](#_Toc409532400)

[Send Text Message to a Registered End User 16](#_Toc409532401)

[Validate OTP Service 17](#_Toc409532402)

[Response 17](#_Toc409532403)

[Verify an OTP 17](#_Toc409532404)

[Registration Services 17](#_Toc409532405)

[Response 17](#_Toc409532406)

[Pre End User Validation Registration Service 17](#_Toc409532407)

[Post End User Verification Registration Service 18](#_Toc409532408)

[End User File Registration 18](#_Toc409532409)

[End User Management Service 19](#_Toc409532410)

[Response 19](#_Toc409532411)

[Check End User Registration 19](#_Toc409532412)

[Deactivate End User 19](#_Toc409532413)

[Activate End User 19](#_Toc409532414)

[Delete End User 19](#_Toc409532415)

[Update End User Information 20](#_Toc409532416)

[Set AdPass Option 20](#_Toc409532417)

[Client Management Service 20](#_Toc409532418)

[Get Client Name 20](#_Toc409532419)

[Get Client Id 20](#_Toc409532420)

[Usage Billing Service 21](#_Toc409532421)

[Request 21](#_Toc409532422)

[Response 21](#_Toc409532423)

[Service Error Replies 22](#_Toc409532424)

[Common Errors 22](#_Toc409532425)

[Service Specific Errors 22](#_Toc409532426)

[Request OTP Service 22](#_Toc409532427)

[Verify OTP Service 23](#_Toc409532428)

[Register End User Services 23](#_Toc409532429)

[End User Management Service 23](#_Toc409532430)

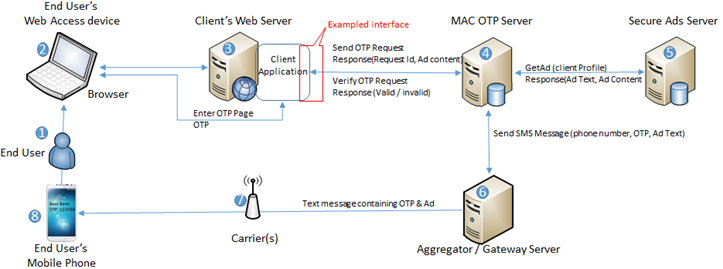
Overview

Contained in this document are brief descriptions of the HTTP/HTTPS methods for sending requests to the System Services, the responses, the message formats, and the Client requirements. Provided in Addendum A are JavaScript/JQuery coding examples. MAC also provides a DLL that can be used in certain IIS deployments; examples of using the MACOTP DLL are covered in Addendum B.

The following diagram is provided to give the reader a general overview of the system flow as a context for the APIs definitions.

Note: The following diagram is just one example of the many ways the OTP System may be deployed and used.

## System diagram



### Diagram Components

The MAC OTP system (4) requires every **Client** (3) to be registered with the system. When registered, the Client will be issued a ***Client ID***. Optionally, the system supports the concept of a **Group.**  A group is a collection of one or more Clients. Every request issued to one of the Services must contain a valid Client Id or Group Id.

**Note**: Examples in this document do not have Group Ids.

## End Users (1)

In the context of the MAC OTP System and this document “End Users (1)” are the customers of the Clients (2). The End Users are the people that receive the OTP messages on their SMS enabled devices (8). The System expects End Users to be controlled by the Client or be registered with the System.

### Client Managed End Users

Using “Client Managed” API calls the Client manages the End User information. The OTP System does not maintain any user information. The required End User information is included in every API call and the OTP system only validates the Client Id and the format of the phone number and email address.

### Registered End Users

In the “Registered” API calls the End User must be registered before the OTP requests are processed by the System (4). A end user record is created in the System database where the first name, last name, phone number and the email address are maintain. The Service API requires that the caller supply the User Id as one of the parameters of the request.

#### Create User Id using User’s last name and Unique Identifier

Normally the User Id is created by doing an MD5 Hash of the End User’s Last Name and a Unique Identifier. Two variations are provided for the Client to use their uniquely generated Ids.

1. Client supplies a Unique Identifier (other than the End User’s email address which is the default) that is hashed with the user’s last name to create the User Id.
2. The Client supplies its own unique User Id.

**Note**: In both cases the registration process may run into conflicts when registering an End User when the User Id is not unique as Group Registered or Open Registered.

Example of generating a User Id using the Client supplied unique identifier:

Var UserId = hex\_md5(LastName.toLowerCase() + UniqueIdentifier.toLowerCase()).toUpperCase();

#### Client created User Id

The Client can generate a User Id and use it in the requests as long as the Client insures the Id is unique to the user. If not, the registration will fail or the message may get sent to the incorrect user.

## End User’s Computing Device (2)

The End User computing device accesses a Client’s application where an OTP is required to complete a process.

## Client’s Web Server (3)

Normally the Client’s application is being hosted on the Client’s or Client Agent’s web server. The Client’s application is using the OTP process for End User Authentication, or Transaction Authorization and Verification.

## MAC OTP Server (4)

The server(s) that host the various Web Services that make up the MAC OTP System.

## Secure Ads Server (5)

The server(s) that manage the Ad content of the Client’s Ad Campaigns for the MAC Clients.

## Aggregator / Gateway Server (6)

The Aggregator / Gateway’ servers that provide a single point of contact to all the Phone Carriers (7).

## Carriers (7)

Verizon, T-Mobile, etc. that provide the telecommunications network that delivers the message to the End User’s mobile device.

## End User’s Mobile Device (8)

The End User’s phone that supports text messaging.

# Text Messaging and Short Code Considerations

The carriers and gateways (aggregators) have several restrictions they place on companies that use the text messaging service to send text message to a end user base. This section covers the ones that are specific to sending One Time Passwords.

## Reply messages

Reply messages are the text messages that are send from the End User. The End user either replies to an text message of send a text message to the short code. There are only 2 reply messages that the OTP system processes the “HELP” message and “STOP” message all other text messages are ignored.

### HELP Reply message

When the user replies or sends a HELP message to the short code the gateway will respond with a canned help message. This message must contain a 1-800 number and a link to a web site that provides online help that contains instruction on how to use the OTP system. When using a MAC provided short code MAC provides these services.

### STOP Reply Message

When the end user replies or send a STOP message to the short code the gateway puts a “Block” on the end user’s number. The OTP system will not be able to send a text message to number from the short code until the block is removed. To remove the block the end user must text OPTIN to the short code. The Request OTP service replies with a STOP error when a “SendOTP” or “SendMessage” request is received for a number that is blocked because the gateway received a “STOP” message. The OTP system passes the STOP error to the requester in the response, see Request OTP service.

When the Client receives a STOP error the end user should be told that they have to text “OPTIN” to the short code before the system can deliver an OTP message.

# Service API Requests / Responses

Services support the REST protocol using HTTP/HTTPS Post method where the “data” contains the request details. Depending on the implementation, the ASCII request data is either *encoded* or *encrypted* (recommended when using HTTP) before sending to the System Service to avoid special character conflicts.

* Encoding - The request data is converted into a hexadecimal string.
* Encrypted – The request data using AES encryption algorithm.

### Request Format Details

The parameters for a request are assembled in an ASCII string as key value pairs with each key/value separated by the pipe character “|” and the keys are separated from the values by the colon character “:”.

**Note:** Some values, such as the Transaction details, may contain special characters that would cause problems in the request process. These values must be converted to a hexadecimal string before it is added to the request parameter list.

**Note**: The following example is for a “Client Managed End User” request.

Example before hexadecimal encoding:

**Note**: Key value pairs with keys in red and values in blue (key value separator is in black).

Request:SendOtp|CID:5351674c74846919ec735074|PhoneNumber:4802684076|EmailAddress:tdavis@mobileauthcorp.com|EndUserIpAddress:192.168.168.1|TrxType:2|TrxDetails:4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937|{AdPass Details}|API:???

Where:

1. The request verb (required in every API call): Request:SendOtp
2. Client Id (required in every API call): <CID:5351674c74846919ec735074>
3. End User’s mobile phone number (required Client managed call): PhoneNumber:4802684076
4. End User’s email address (required Client managed call): EmailAddress:tdavis@mobileauthcorp.com
5. End User’s machine IP address (optional): EndUserIpAddress:192.168.168.1
6. Transaction type (optional default is 1 ‘OTP’): TrxType:2
7. Transaction details (optional, default is no details in OTP message): TrxDetails:4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937

**Note 1**: Transaction details (the value) is hexadecimal encoding to avoid issues with special characters.

**Note 2**: See transaction details encoding for formatting details.

1. AdPassDetails (Optional) if included, See AdPass section, Request Details later in this document.
   1. Example of Opt-out for this request: ApOpt:AdDisable.
2. Who is making the request (optional, used for resolving errors): API:???

### Transaction Type (TrxType) Encoding

The transaction type parameter is used by the message assembly method to select the message formatting template.

The templates for SMS, email and voice message formats are defined in the Client’s configuration when the Client is registered with the system.

Transaction types are:

0: (TrxType:0) is for *notification* messages “no OTP will be generated or included in message,”

1: (TrxType:1) is for *authentication* nominally used in the login process “no transaction details,”

2: (TrxType:2) is for *transaction verification* normally includes transaction details that are hexadecimal encoded string. The message assembly function decodes and formats based on the transaction type.

### Transaction Details Format and Encoding

The transaction details could contain new lines and characters that cannot be sent as ASCII characters.

1. The new lines in the transaction details and in the send message text must be replaced by the pipe character “|”. The text message assembly logic replaces the pipe character with the appropriate new line sequence for the message delivery channel.
2. The transaction details and the send message body are hexadecimal encoded.

* Example before encoding:

Hat $17.99|Jacket $150.98|Shirt $33.98|Total $202.95

**Note:** The example will be displayed in the OTP message as 4 lines.

### Request Data Encoding Details

The encoded request is constructed in two parts the header and the request data. The header starts with the data key “Data=99” indicating the request is encoded, followed by the length of the Client Id followed by the Client Id. The encoded request data is appended onto the header to complete the data for the Post. The following is a Java Script example of the request formatting:

var dataToPost = "Data=99" + ClientId.length.toString() + ClientId.toUpperCase() + StringToHex(requestData);

* Example after hexadecimal encoding (complete data packet):

Data=

* Break down of components:

Http post header: Data=

Hexadecimal encoded indicator: 99

Length of Client Id: 24

Client Id (as issued by MAC): 5351674C74846919EC735074

Request data (Hexadecimal encoded): 

### Request Data Encrypted Details

The encrypted call is constructed in two parts- the header and the request data. The header starts with the data key “Data=”.

Note: When the data doesn’t start with the “99” indicates that the request data is encrypted using the AES 128 encryption scheme where the upper case Client Id is the key. The header “Data=” is followed by the length of the Client Id, the upper case Client Id and the encrypted data to complete the data for the Post. The following is a Java Script example of the request formatting:

var dataToPost = "Data=" + ClientId.length.toString() + ClientId.toUpperCase() + AES128Encryption(request Data, ClientId.toUpperCase ()));

* Example after AES Encrypted Request (complete data packet):

Data=

* Break down of components:

Http post header: Data=

Length of Client Id: 24

Client Id (as issued by MAC): 5351674C74846919EC735074

Request data (Hexadecimal encoded): 

### Response Format Details

* Example of a successful response with the Replay and Details elements containing ASCII delimited strings.

<?xml version="1.0" encoding="utf-8" ?>

<macResponse>

<calledMethod>WsRequestOtp()</calledMethod>

<Reply>

RequestId:5446d6637484691328eab102 // Request Id

|EnterOTPAd:{Hex Encoded Ad}\* // Optional ad for OTP Page (Hexadecimal encoded)

|ContentAd:{Hex Encoded ad}\* // Optional ad for Content Page (Hexadecimal encoded)

</Reply>

<Details>

Request:SendOtp

|ClientName:Client 1

|TLM:5

|OTPRetriesMax:3

|OTPExpiredTime:10/21/2014 10:05:49 PM

</Details>

</macResponse>

\* See AdPass section for additional information

* Example of error response:

<?xml version="1.0" encoding="utf-8"?>

<macResponse totalProcessTime="1ms">

<calledMethod>FinalizeXmlResponseWithError()</calledMethod>

<Error>Invalid [CID:53ed325e74846912e08d57ad1</Error](CID:53ed325e74846912e08d57ad1%3c/Error)> // the error element contains the error text

</macResponse>

* Example of STOP Error response:

<?xml version="1.0" encoding="utf-8"?>

<macResponse totalProcessTime="1ms">

<calledMethod>FinalizeXmlResponseWithError()</calledMethod>

<Error>Not sent, Blocked user replied '**STOP**' (FromNumber=ShortCode)</Error>

</macResponse>

**Note**: The STOP error indicates that the end user has replied or sent a STOP to the short code the system is using. The End User will not receive the OTP message until they send the opt-in text “OPTIN” to the short code.

**Note**: There are several error responses that could be returned by the MAC OTP Services. There are two categories of errors: (1) Common errors that can be returned by any of the services, and (2) Service specific errors that are only returned by the service called.

## OTP Services

There are two Services that support the One-Time Password and send Message functionality, the Request OTP Service and the Verify OTP Service. Each has their own URL. The base URL (where the MAC OTP System Services are running) combined with the Service URL make up the HTTP address.

Request OTP service URL: Otp/RequestOtp.asmx/WsRequestOtp

Verify OTP service URL: Otp/ValidateOtp.asmx/WsValidateOtp

## Registration Services

The MAC System provides three (3) End User Registration processes: “Pre End User Verification” registration, “Post End User Validation” Verification and End User File Registration. The MAC System does check that the required parameters are provided and that the phone number and email address are in an acceptable format.

### Pre End User Verification Registration

This registration process expects that the End User being registered was verified before the MAC Registration service was called, and that the End User information being used to register the End User is correct. It is assumed that the End User is a “Real” person and that the phone number is for a valid mobile device that can accept text messages. More details are covered in the Registration Section of this document.

Pre End User Validation Registration Service URL: User/STSEndUserRegistration.asmx/WsSTSEndUserRegistration

### Post End User Verification Registration

For an additional charge, this registration process can be configured to use a Verification Service to verify that the End User information being supplied to the registration process is valid. This process uses a two-step registration process by sending an email to the End User to complete the registration process. More details are covered in the Registration Section of this document.

Post End User Verification Registration Service URL: User/EndUserRegistration.asmx/WsEndUserRegistration

### End User File Registration

This registration process is used for bulk registration End User, which is normally used to register a Client’s preexisting user base. The Client creates a text file containing the End User information needed to register their user base. The System does not perform verification on the End User information being supplied in the file, rather it is the Client’s responsibly to insure that the End Users being registered are valid users.

File End User Registration service URL: User/EndUserFileRegistration.asmx/WsEndUserFileRegistration

## End User Management Service

The End User Management service provides service calls that help the Client manage the state of the End User. The service supports calls: (1) to determine if the End User is *registered*, (2) to *deactivate* an End User, (3) to *activate* an End User, (4) to *delete* the End User, or (5) to set the AdPass *Opt-in* or *Opt-out* options.

File End User Management service URL: User/EndUserManagement.asmx/WsEndUserManagement

## Client Management Service

The Client Management Service exposes 2 basic methods, Get Client’s Id using Client’s name and Get Client’s Name using Client’s id.

Client Management Service URL: /AdminServices/ClientServices.asmx/WsClientServices

## Usage Billing Service

The usage and billing service provides a method the Client or group administrator to get the numbers for a billing cycle. A bill is created on the first day of each month for each Client or group depending on the Client’s configuration. The bill will be sent via email to the list of emails configured in the Client’s billing record.

Usage and Billing service URL: Admin/UsageBilling.asmx/WsUsageBilling

## Service URL

The URL provided in this section is just the service specific part of the URL to get a complete URL. The calling application must preceded the service specific part with the base URL. The base URL is the network address when the services are running and will be supplied by MAC operations.

Example of a base URL: <https://corp.mobileauthcory.net/macservices/>

Example of full URL to the Request OTP Service: <https://corp.mobileauthcory.net/macservices/>Otp/RequestOtp.asmx/WsRequestOtp

## AdPass

This section explains how the AdPass feature works, how to use the AdPass Options in the request, and how to process that AdPass data in the response. The Ads returned in the response are formatted as an HTML <div> containing Javascript code snippet designed to be inserted in the “Enter OTP” page, preferably in a predefined <div> on the Enter OTP page.

### Overview of how AdPass Works

The AdPass feature processes three (3) Ads for a Send OTP request. One Ad gets included in the OTP Text Message as a link the second and third Ads get returned in the response, again as hexadecimal encoded HTML <div>. Both Ads are preconfigured in the Ad Server by the Client via the Ad Campaign process.

Note: The Ad Campaign process will be covered in a separate document.

When the Client calls the MAC OTP System to send an OTP to an End User and all of the configuration and Opt-out options for Send Ads to the End User are met, the OTP Server sends a request to the Ad Server requesting an Ad for the Client/User. The response from the Ad Server contains both Ads, and the OTP server includes the “Message Ad” in the OTP Text message that gets sent to the End User’s mobile phone. The OTP server encodes the other Ad and includes the value in the Request OTP response. The Client decodes the Ad and inserts the Ad into the Enter OTP page.

### AdPass Request Options

The AdPass feature supports several “On Request Options” as follows:

1. User or Client Opt-out option – This allows the Client to configure a user opt-out feature. If not present in the request, the ads are enabled:
   1. Key “ApOpt” value “AdDisable” – The MAC OTP system will not send an Ad for this request regardless of Client’s configuration.
   2. TBD

### Ad Response Details

The *Enter OTP* and *Content* Ads are delivered from the ad server as text. The format is ‘HTML div’ that can be inserted directly into the page being delivered to the Client. The MAC OTP system encodes the text as a hexadecimal string to avoid conflicts created by special characters used in the ads.

Example of an Ad:

<div data-ad-id='EnterOTP\_G2'><a id='adURL' target='\_blank' href='http://localhost:8010/demos/Redir.aspx?i=G2' ><img src='http://localhost:8010/demos/ads/golf/golf-ad2.jpg' style='max-width: 300px !important;width: 100% !important;' border='0'></a></div>

Note: The content of the ad ‘div” may change based on how the Client sets up the Ad Campaign.

# Service Request and Response Details

## Send OTP Service

### Response

The response will be formatted based on the Response Format parameter if supplied in the request.

Note: The STOP error indicates that the end user has replied or sent a STOP to the short code the system is using. The End User will not receive the OTP message until they send the opt-in text “OTTIN” to the short code.

The STOP error is formatted as follows:

The response will contain:

1. Reply // Response verb
   1. Sent
2. RequestId // Request Id
   1. 54a962777484690a50ea1538
3. EnterOTPAd // The ad to be displayed with the Request OTP page
   1. 3C64697620646174…
4. Details
   1. Request // Request verb
      1. SendOtp
   2. ClientName // Id of Client requesting the OTP
      1. Client 1
   3. DeliveryMethod // Method the OTP message was delivered
      1. Sms
   4. TLM // Time in minutes the OTP has to live
      1. 10
   5. OTPRetriesMax // Maximum number of retries, as configured in Client
      1. 3
   6. OTPExpiredTime // UTC time when OTP will Expire
      1. 1/4/2015 4:05:38 PM

### Send OTP Message to a Client Managed End User

* Request: **SendOtp** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* PhoneNumber: 4802684076 // End User’s phone number (required, format is validated)
* EmailAddress: [jdoe@company.com](mailto:jdoe@company.com) // End User’s email address(required, format is validated)
* EndUserIpAddress: 72.43.21.1 // (Optional) End User’s machine’s IP address
* TrxType: 2 // (Optional) OTP Message type (optional, default is 0)
* TrxDetails: 4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937

// (Optional) Hexadecimal encoded Transaction Details (included in OTP message)

* AdPass Options: ??? // Ad Pass Option, see the Ad Pass section for details
* API:??? // Caller’s tag name (optional, used for debugging)

### Send OTP Message to a Registered End User

* Request: **SendOtp** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (used if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* EndUserIpAddress: 72.43.21.1 // End User’s machine’s IP address (optional)
* TrxType: 2 // OTP Message type (optional, default is 0)
* TrxDetails: 4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937

// (Optional) Hexadecimal encoded Transaction Details (included in OTP message)

* AdPass Options: ??? // Ad Pass Option, (optional, see the Ad Pass section)
* API: ??? // Caller’s tag name (optional, used for debugging)

### Cancel an OTP

* Request: **CancelOtp** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* RequestId: 5351674c74846919ec735074 // OTP Id (required, returned by the Send OTP request)
* API: ??? // Caller’s tag name (optional, used for debugging)

### Resend OTP Message

* Request: **ResendOtp** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* RequestId: 5351674c74846919ec735074 // OTP Id (required, returned by the Send OTP request)
* API: ??? // Caller’s tag name (optional, used for debugging)

### Send Text Message to a Client Managed End User

* Request: **SendMessage** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* PhoneNumber: 4802684076 // End User’s phone number (required, format is validated)
* EmailAddress: [jdoe@company.com](mailto:jdoe@company.com) // End User’s email address(required, format is validated)
* EndUserIpAddress: 72.43.21.1 // (Optional) End User’s machine’s IP address
* Message, : 4861742031372… // (Optional) Hexadecimal encoded Message Details (Pipe characters are converted to new lines in message)
* AdPass Options: ??? // Ad Pass Option, see the Ad Pass section for details
* API:??? // Caller’s tag name (optional, used for debugging)

### Send Text Message to a Registered End User

* Request: **SendMessage** // Request Verb(required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (used if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* EndUserIpAddress: 72.43.21.1 // (Optional) End User’s machine’s IP address
* Message, : 4861742031372… // (Optional) Hexadecimal encoded Message Details (Pipe characters are converted to new lines in message)
* AdPass Options: ??? // Ad Pass Option, see the Ad Pass section for details
* API:??? // Caller’s tag name (optional, used for debugging)

## Validate OTP Service

### Response

Response will be formatted based on the Response Format parameter if supplied in the request.

The response will contain:

1. Reply // Response verb
   1. Valid
2. Details
   1. Request // The request
      1. VerifyOtp
   2. ClientName // Name of Client that requested the OTP
      1. The Client
   3. DeliveryMethod // Method the message was delivered
      1. SMS
   4. TLM // Time to live for OTP, in minutes, As configured in Client
      1. 10
   5. OTPRetriesMax // Maximum number of retries, as configured in Client
      1. 3
   6. OTPRetriesCurrent // Retry count, on 0 (zero) the OTP will be disabled
      1. 2
   7. OTPExpiredTime // UTC time when OTP will Expire
      1. 11/25/2014 4:50:12 PM

### Verify an OTP

* Request: **VerifyOtp** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* RequestId: 5351674c74846919ec735074 // OTP Id (required, returned by the Send OTP request)
* Otp: ???? // OTP (required, entered by the End User)
* API:??? // Caller’s tag name (optional, used for debugging)

## Registration Services

### Response

TBD

### Pre End User Validation Registration Service

* Request: **EndUserRegister** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (used if user IS BEING group registered)
* RegType: ClientRegister // Registration Type:

Where:

ClientRegister: User is restricted to Client,

GroupRegister: User is restricted to the group (Client must belong to the group)

OpenRegistered: User can be used by any Client that is Open Enabled (Reference Client configuration)

* FirstName: John // End User’s first name (required)
* LastName: Doe // End User’s last name (required, used to create User Id)
* PhoneNumber: 4895551212 // End User’s mobile device phone number(required)
* EmailAddress: [jdoe@company.com](mailto:jdoe@company.com) // End User’s email address(required)
* UID: 351674c74846919ec333333 // (Optional) End User’s Unique Identifier

// Normally generated by the Client or Client’s service

### Post End User Verification Registration Service

* Request: **EndUserRegister** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (used if user IS BEING group registered)
* RegType: ClientRegister // Registration Type:

Where:

ClientRegister: User is restricted to Client,

GroupRegister: User is restricted to the group (Client must belong to the group)

OpenRegistered: User can be used by any Client that is Open Enabled (Reference Client configuration)

* FirstName: John // End User’s first name (required)
* LastName: Doe // End User’s last name (required, used to create User Id)
* PhoneNumber: 4895551212 // End User’s mobile device phone number(required)
* EmailAddress: [jdoe@company.com](mailto:jdoe@company.com) // End User’s email address(required)
* UID: 351674c74846919ec333333 // (Optional) End User’s Unique Identifier

// Normally generated by the Client

Note: The following End User Information may be required depending on what End User Verification Service is being used.

* Dob: 10/10/1988 // End User’s date of birth
* SSN4: 5678 // last 4 digits of End User’s SSN
* Street: 1234 E Main // End User’s street address
* Unit: #123 // End User’s apartment, suite
* City: New Town // End User’s city or town
* State: AZ // End User’s state
* ZipCode: 85123 // End User’s zip code
* DriverLic: 1212121212 // End User’s Driver’s License Id
* DriverLicSt: AZ // End User’s Driver’s License state
* API:??? // Caller’s tag name (optional, used for debugging)

### End User File Registration

* Request: **EndUserRegister** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (used if user IS BEING group registered)
* **FileType**: xml, cvs/txt or json // File Type:

// where:

// - xml is an text file that is in XML document

// format

// - cvs or txt is a file where each line contains

// a user record that is formatted using the

// pipe “|” and Column “:” characters as separators.

// - json is a file that contains a serialized

// dictionary

* **FileName:** MyRegistrationFile.txt // File name including the file extent
* API:??? // Caller’s tag name (optional, used for debugging)

## End User Management Service

### Response

<Reply>Success</Reply>

### Check End User Registration

This service call checks the registration status of the End User.

* Request: **EndUserRegister** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* API:??? // Caller’s tag name (optional, used for debugging)

### Deactivate End User

This service call sets the disable state in the End User record. Any subsequent service calls targeting the End User will be rejected.

* Request: **DeactivateEndUser** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* API:??? // Caller’s tag name (optional, used for debugging)

### Activate End User

This service call resets the disable state in the End User record allowing service calls that target this End User.

* Request: **ActivateEndUser** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* API:??? // Caller’s tag name (optional, used for debugging)

### Delete End User

* Request: **DeleteEndUser** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* API:??? // Caller’s tag name (optional, used for debugging)

### Update End User Information

* Request: **UpdateEndUser** // Request Verb (required)
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* PhoneNumber: 4895551212 // End User’s mobile device phone number
* Dob: 10/10/1988 // End User’s date of birth
* SSN4: 5678 // last 4 digits of End User’s SSN
* Street: 1234 E Main // End User’s street address
* Unit: #123 // End User’s apartment, suite
* City: New Town // End User’s city or town
* State: AZ // End User’s state
* ZipCode: 85123 // End User’s zip code
* DriverLic: 1212121212 // End User’s Driver’s License Id
* DriverLicSt: AZ // End User’s Driver’s License state
* API:??? // Caller’s tag name (optional, used for debugging)

**Note**: If the User’s name, User’s email, or User Id changes the User will have to be deleted and reregistered.

### Set AdPass Option

* Request: **SetAdPassOption** // Request Verb
* CID: 5351674c74846919ec735074 // Client Id (required)
* GroupId: 5351674c74846919ec333333 // Group Id (required if user was group registered)
* UserId: (computed) // User Id (required)

**Note**: Computed by Client normally by hashing the End User’s LastName and email/UID

* AdPassOption: AdPassEnable, AdPassDisable // Set Opt-in / Opt-Out state (required)

**Note**: AdPassDisable (setting the Opt-out state) overrides Client options.

## Client Management Service

### Get Client Name

* Request: GetClientName // Request Verb
* [CID: 53ed325e74846912e08d57ad](CID:53ed325e74846912e08d57ad) // Client Id
* API: ???

Response

<**Reply**>5351674c74846919ec735074<**Reply**>

### Get Client Id

* Request: GetClientId // Request Verb
* [ClientName: Client](CID:53ed325e74846912e08d57ad) A // Client’ Name
* API: ???

Response

<**Reply**>5351674c74846919ec735074</**Reply**>

## Usage Billing Service

### Request

* Request: GetUsageBillingForMonth // Request Verb
* [CID: 53ed325e74846912e08d57ad](CID:53ed325e74846912e08d57ad) // Client Id (required if requesting billing numbers for a Client bill)
* GroupId: 5351674c74846919ec333333 // Group Id (required if requesting numbers for a group bill)
* BillDate: 12/2014 // Month and year of bill
* API: ???

### Response

This service only returns XML responses.

Example of response where bill was found:

<Reply>

<Client Name="Kohl's" CID="53ed325e74846912e08d57ad" >  
 <OTPSent>  
 <Sms count="11" amount="11.01" />  
 <Email count="22" amount="22.02" />  
 <Voice count="33" amount="33.03" />

</OTPSent>  
 <Ads>  
 <MessageAds count="31" amount="31.01" clickedcount="0" clickedamount="0" />  
 <EnterOTPAds count="32" amount="32.02" clickedcount="0" clickedamount="0" />  
 <ContentAds count="33" amount="33.03" clickedcount="0" clickedamount="0" />  
 </Ads>  
 <EndUser>  
 <Registrations count="44" amount="44.04" />  
 <Verifications count="55" amount="55.05" />  
 </EndUser>  
 </Client>  
</Reply>

Example of error response:

<Error>Could not retrieve bill for Client 1 on 12/2014 from bill archives!</Error>

# Service Error Replies

Errors returned by the System services are defined in two categories: Common Errors and Service Specific Errors.

If the response contains a <Error> element, this element will contain the details.

## Common Errors

There are four sub-categories for “Common Errors” returned by the System Services.

1. Request Format errors
   1. Corrupt request data. The service could not decode request.
   2. Request type required. The service could not find the request in the data after decoding.
2. API errors
   1. Invalid request. The service does not support the request, normally caused by the request being sent to the wrong service
   2. Invalid Client Id. The service could not find the Client using the Id supplied in the request data
   3. Invalid Client name. The service could not find the Client using the Client’s name supplied in the request data. Normally caused by a difference in spelling or case.
   4. Invalid Group Id. The service could not find the group using the Id supplied in the request data.
   5. Invalid Group name. The service could not find the group using the group’s name supplied in the request data. Normally caused by a difference in spelling or case.
   6. Group disabled. Group has the disabled status set.
   7. Invalid End User. Service could not find the end user using the parameters supplied in the request data. Normally caused by the user not being registered or registered with usage restrictions that do not match the request.
3. System Errors
   1. Exceptions (General). The service failed when processing the request. Unplanned service issue, report to MAC for a resolution.

## Service Specific Errors

### Request OTP Service

1. Invalid End User. Service could not find End User in the database. Normally returned because the End User is not registered or the Client does not have access to the End User’s registration record.
2. Message delivery service error. The system could not deliver the message. Normally occur when the service could not contact the message delivery service or the backup. The message delivery service and the retry list are setup in the Client configuration via the System Admin interface.
3. Unregistered End User. Service could not find the End User. Normally caused the End User is not registered before call or the End User was registered with restrictions.
4. End User (Inactive). End User state is inactive. Normally caused by Client deactivating the End User.
5. Client Managed, Requires user's email and phone number. Either the End User’s email address or phone number is not supplied in request.
6. Client Managed, invalid email. Normally caused by an error in the calling system that allows an invalid email format.
7. Client Managed, invalid phone number. Normally caused by an error in the system that allows an invalid phone number format.
8. Disabled can not resend. Resend OTP request for an OTP that was previously used or disabled.
9. Timeout can not resend. Resend OTP request for an OTP that has timed out.
10. Too Many Retries can not resend. Resend OTP request for an OTP that has been previously retried too many times.
11. Not sent, Blocked user replied '**STOP’** (FromNumber=9995551212). Normally caused by the End User replying “STOP” to an OTP message or sending a “STOP” message to the number the system is using. The End User must send an “OPTIN” message to the same system number to resume the OTP service.
12. Unable to parse response [434445….]. System error please report error to the system administrator.
13. All providers failed to send message. System could not contact a provider to send the message. Normally caused by a network error where the system was unable to contact any of the message delivery providers. Contact the primary system administrator as soon as possible.
14. No document template for “message type". Client configuration error. Contact the primary system administrator as soon as possible.

### Verify OTP Service

1. Invalid. The OTP supplied in the request did not match the OTP sent to the End User.
2. Request Id (Invalid). Service could not find the OTP record in the database using the supplied request Id. Normally caused by the requester using an invalid request Id.
3. Disabled. The request Id supplied in the request data is for an OTP record that was already used. Normally caused by the requester using an invalid request Id.
4. Invalid OTP. Normally caused by the End User entering an incorrect OTP.
5. Timeout. The End User waited too long to enter the OTP. Note: The time allotted to enter the OTP is in the details of the reply of the Request OTP response.
6. Invalid Client. The Client Id in the request did not match the Client Id used when the OTP was sent.
7. OTP missing in request data. Invalid request, service could not find the OTP parameter.
8. OTP can't be of zero length. OTP invalid, cannot be 0 (zero) length.
9. Client Id (Invalid for this OTP). The Client Id supplied in the Verify OTP request is different that the Client Id in the Request OTP.

### Register End User Services

1. End User Exists. The User Id was not unique and the End User is already in the database. Normally caused when the End User is already registered, the Client supplied Unique Identifier for the End User is not unique, or the Client supplied User Id is not unique.

### End User Management Service

1. Invalid End User. Service could not find End User in the database. Normally returned because the End User is not registered or the Client does not have access to the End User’s registration record.
2. No End User to delete. Normally caused by a difference in the spelling of the unique End User information (such as email), or incorrect User Id supplied in the request.