TECHNICAL DOCUMENTATION FOR:

API-MO-MT Interface Specifications

TextPower, Inc.
www.TextPower.com
Twitter.com/TextPower
Support@TextPower.com
888-818-1808

SOAP Basic Interface Version 2.1
This manual describes Version 2.1 of the basic SOAP interface to TextPower, Inc. (TPI) services. There are multiple levels of SOAP interfaces. Only the Basic Interface is described in this manual.

Note that the name of this service has changed from that of previous versions. This service provides versions of each service method that use API Keys in addition to the existing SOAP Header interface. The older service will remain available indefinitely and will continue to work. Since no new basic SMS functionality is part of this API, generally there is no need for existing customers to upgrade. New customers should use this latest API version however.

In this manual, the term MT means Message Terminating. That means a message that you sent to a phone. The terms terminating and originating always are from the perspective of the phone.

Table of Contents

Overview

This manual will not attempt to describe the general SOAP interface and structure. It is expected that the reader already knows this. This manual provides only supplementary information that is not found in the WSDL descriptions. This interface along with the other SOAP interfaces allows the TextPower (TPI) customer to build extremely sophisticated messaging applications. Building a SOAP interface should not be undertaken unless the developer has the proper development tools to auto-generate the necessary interface code and the expertise to develop the SOAP application.

All TPI Basic SOAP functions return an XML fragment or XML node. A SOAP header is required on all calls which do not use the new API Key and the header information is identical on all calls. Structural errors involving invalid credentials, incorrect parameter values, etc. return an XML fragment with errors as the base node. SOAP exceptions are supported. SOAP exceptions always represent internal program faults. Hopefully, you will not receive any but if you do, please forward the information to TPI support. The return of an Errors node represents a detectable error in the information supplied. Please check the information that you are supplying to correct this type of problem.

Changes in Version 3

Documentation has been provided for changes and additions to POST out protocols.

API Keys

An API key is a long 128 bit number that is totally unique. An API Key looks like this: {E103D087-5299-47F5-A769-DBF4BC017B30}. You can get API Keys by going to the TextPower customer web site at https://customer.textpower.com and selecting the **Get API Key** menu item under the **Campaigns** tab. If you have more than one campaign, you must select the campaign you wish to get an API Key for first. To do this, select the Select menu item under the **Campaigns** tab first. On the API Key screen, just select the keyword and the UserID that you want a key for. Click the Add/Change Key button and you will be issued an API Key. Please note that if you return to the Get API Key screen and click this Add/Change API Key button again, you will be issued a new API Key which will invalidate your old key! If you change either your UserID or your password, your API Key will NOT change. If you have more than one UserID, each UserID will have a separate API Key.

WSDL and Header

The WSDL document for Version 2 of the basic SOAP interface is located at http://www.textpower.com/TPIServices/BasicMessageServicesV2.asmx?wsdl

The address for the SOAP calls on the basic interface is:
http://www.textpower.com/TPIServices/BasicMessageServicesV2.asmx

Alternatively, you may access the Basic Message Interface on a secure basis at:
WSDL
https://secure.textpower.com/TPIServices/BasicMessageServicesV2.asmx?wsdl
Address for SOAP Calls
https://secure.textpower.com/TPIServices/BasicMessageServicesV2.asmx

Note 1: This version of the Basic SOAP interface document contains added functions from the previous versions. To access these new functions, you will need to update your web reference!

Note 2: TextPower is adding a high reliability server failover system. In order for this to work, users must use the www in front of textpower.com in URLs! Usage of the secure.textpower.com address already meets the requirements of the high reliability system.

Note that the interface points specified in previous versions of this document remain active.

If you use the SOAP header, it has four parameters on all calls as described in the WSDL. All header parameters are of type String.

Parameter	**Meaning**
UID	Required. The value of your user ID as assigned by TPI.
PWD	Required. The value of your password as assigned by TextPower (this password can be changed at your discretion)
Campaign	The name of the campaign you are using for this call. If you have only one campaign, this parameter may be a zero length string.
Keyword	The keyword in your campaign that you are using for this call. If you have only one keyword in your campaign, this parameter may be zero length string.

The majority of users has only one campaign each and only one keyword and will be able to leave the Campaign and Keyword parameters alone.

Note that this SOAP interface does not support WAP messages in raw form. They are only supported by our classic interface.

Extended Length Messages

The SendToClients and SendToClients_key functions on the Basic interface and the SendToSendList function on the Advanced interface support extended length messages. You cannot use extended length messages on any other send call! Extended length messages can be up to 800 characters long. Since Cell Phones can take only 160

characters in a single message, an extended length message will be sent in more than one packet. The rules for using Extended Length Messages are:

1. An extended length message must be more than 160 characters in length. For messages <= 160 characters, no check for delimiters will be done and the message will be sent exactly as submitted.
2. The user must provide a delimiter between segments of the message. The delimiter is ;:; (semi-colon, colon, semi-colon). The TextPower system will split the message at the given delimiters and transmit it in bursts of 160 characters or less. The delimiters will not be transmitted.
3. Each segment of the message must be <=160 characters in length.
4. When retrieving the status of sends via the GetSendListMembers function on the Advanced interface, only the SendID of the last message segment sent will appear.

Binary Message Sending

TextPower supports binary sending to sending to cell phones on short codes by special provisioning. To send binary you must:

1. Have a campaign specifically set up to send binary. A single campaign can send either normal text or binary but not both.
2. Comply with carrier regulations regarding certification so that specific carriers will allow your binary sends to go through. Please note that for some carriers, particularly T-Mobile and Verizon Wireless, gaining permission to send binary can be both time consuming and expensive.

TextPower supports sending binary content with or without UDH headers. When sending binary, your message must consist of only the hexadecimal character set plus an optional header separator. You may not send binary messages using the Queue option of SendSMS and SendSMSFull or send binary messages on a bulk send. It does not matter whether the hex characters are upper or lower-case.

Sending without a UDH

To send binary without a UDH, simply supply your message as hex text characters. E.g. 546ad3f478

Sending with a custom UDH

To send data with a custom UDH, simply supply your header first, a separation mark consisting of a double pipe symbol (||) followed by your text. E.g. the message `06 05 04` 0b8423f0||567b2a469e means send a UDH of 0b8423f0 and a message content of 567b2a469e.

Function Calls

Type specifications for each call and the parameter data types must be taken from the WSDL. The basic purpose of each function is also described in the WSDL. This section

provides additional information on the meaning of each call, its parameters and return values along with an example of the returned XML node.
Note that in the WSDL, the Integer type is referred to as int, the Date type is referred to as DateTime and the XmlNode type is a complex type. They are the same thing and may be used interchangeably. Sample output XML has been formatted for display on this page using line feeds and indents. These do not appear in actual XML sent from the switch. The Level parameter for output nodes denotes the tag nesting level. The root or base node is 1.

CellNumber Parameter Note: During development of projects, it is sometimes desirable to send messages just for the purpose of testing an interface but which you don't want delivered to a real phone. The TPI SOAP interfaces allow for that case. To send a message without actually having it delivered anywhere, use numbers in the reserved fictitious movie range. These numbers are not cell phones nor are they even real numbers. The TPI SOAP interfaces will accept them and not give you an error. Your logs will be identical to those of real numbers and show these numbers but ***no actual message will be delivered***! Fictitious sends still count against any account limits you may have on your account. Numbers in the reserved fictitious movie range are of the form: NXX55501XX where N is a number from 2-9 and X is a number from 0-9.

Date and Time formats: Note that all dates and times in the basic specification of both inputs and outputs are expressed in Universal Time. You must convert the Universal Time to your local time zone.

Emails: You can also Opt In and send to Email addresses as well as Cell Numbers.

Note on Parameters: For the API Key version of all calls, the API Key is always the first parameter and is a string. The parameter lists for each function include only the common parameters.

Send SMS

```
Function SendSMS(CellNumber As String, msg As String, Carrier As String, Tariff As Integer, Queued As Boolean) As XmlNode

Function SendSMS_Key(APIKey as String, CellNumber As String, msg As String, Carrier As String, Tariff As Integer, Queued As Boolean) As XmlNode
```

Send out an SMS messaage with a minimum of parameters for common cases.

Input Parameters		
Name	**Data Type**	**Meaning**
CellNumber	String	The number of the cell phone to which the message is to be sent. The cell number must be exactly 10 digits for North American traffic. For international traffic, your number length limits will vary. When you sign up for international traffic, you must request specific length requirements. You may specify multiple numbers by simply separating them with a comma. For an immediate send, you may specify 10 numbers. For a queued or delayed send, you may specify up to 50 numbers. Delayed sends must use the SendSMSFull command. ***NOTE: If a single number in the list is bad, not a cell number or is not opted in for optin sends, NO SENDS will take place! Use with care.***
msg	String	The message to be sent. The message must be at least 5 characters long and not more than 160 characters long unless you are sending binary. For binary data sending, please see the section above. The characters ~[]^|{}` are invalid in an SMS message and if present will cause your message to fail. In addition, all characters with code values above 127 are invalid. Note that some applications like Microsoft Word convert things like quotes into what they call smart quotes. These smart quotes are invalid. Regular quotes are fine but not smart quotes. To be safe, never paste message text from a word processor into an SMS message box.
Carrier	String	The TPI carrier code that handles this number. In

		general, you never have to supply the carrier parameter. If you don't have it, TextPower will look it up automatically. A small charge may apply for lookups but TextPower will cache the carrier permanently for OptIn campaigns and for 3 months for Non-OptIn campaigns. No additional charge applies if the carrier can be recovered from the TextPower internal caches. A table of Carrier Codes can be found in a separate document on the TextPower documentation page. **It is recommended that you use the table of Carrier Codes only for reference in interpreting call logs available on the TextPower Portal. TextPower strongly recommends that carrier codes not be entered into the carrier field on the various API calls that use them. Doing so forces the carrier that you supply to be used which may not be the current correct carrier. The reason is that carrier codes for a number can change. The most common way is that a user simply changes their carrier. Carrier mergers also cause old carrier codes to be discontinued and new carrier codes to be created. TextPower has extensive facilities to track carrier changes from either of these sources. TextPower will update the carrier code to the current valid one automatically in most instances. Supplying the carrier code on a send defeats these automatic mechanisms!**
Tariff	Integer	The tariff should always be zero. Non-zero values were previously used for premium messaging. All carriers have discontinued premium messaging so this value can only be zero now.
Queued	Boolean	If the Queue Parameter is true, the message is put on a queue for sending. This queue is serviced every 0.5 seconds so delivery time will not be noticeably different from a regular send. Use this parameter when you need a faster response for real time reasons. Certain high volume customers may be required to use this parameter. **All customers with the following exceptions are highly encouraged to use Queued=True!** Exceptions are: 1. Sends to the USA Mobility and American

		Messaging carriers 2. International Sends 3. Sends using the unified International/North American feature where international sends can be simply done by prepending a + and a country code to the number.
Output Base Node:		<SMSImmediate>, or <SMSQueued>
Sub Nodes		
Tag	**Level-Type**	**Meaning**
<Customer>	2-String	Customer name
<TimeStamp>	2-String	The time of these messages. TimeStamp is in Universal time.
<MessageStatus>	2-String	The master tag for the following tags describing the message status
<SendResult SeqNum=”n”>	3-String	If there are no errors and sending was attempted, a SendResult tag will be presented for each number in the CellNumber parameter. The SeqNum tag enumerates the numbers. Thre will be one SendResult parameter for each number in the CellNumber parameter.
<MessageID>	4-Integer	A unique number for this message attempt. Immediate, queued and deferred sends all have different number ranges.
<CellNumber>	4-String	The cell number to which the message was sent (10 digits)
<Status>	4-String	*Sent* or *Not Sent* If Not Sent is the status for an immediate send. A colon will follow it with a reason for the not sent status if it was not sent. *Queued* will be returned for all successful delayed or queued messages. Status is not reported if Errors are sent.

Output Example 1(Immediate Send)

```
<SMSImmediate xmlns="">
<Customer>CUSTOMER A</Customer>
<MessageStatus>
        <TimeStamp>2/6/2008 4:05:29 PM</TimeStamp>
        <SendResult SeqNum="0">
                <MessageID>136986</MessageID>
                <CellNumber>312555788</CellNumber>
                <Status>Sent</Status>
        </SendResult>
</MessageStatus>
</SMSImmediate>
```

Output Example 2-Queued Send)

```
<SMSQueued xmlns="">
<Customer>CUSTOMER A</Customer>
<MessageStatus>
        <TimeStamp>2/8/2008 3:16:44 PM</TimeStamp>
        <SendResult SeqNum="0">
                <QueueID>350</QueueID>
                <CellNumber>312555788</CellNumber>
                <Status>Queued</Status>
        </SendResult>
</MessageStatus>
</SMSQueued>
```

Output Example 2- Double Number Send, one failing do to No OptIn

```
<SMSImmediate xmlns="">
<Customer>CUSTOMER A</Customer>
<MessageStatus>
        <TimeStamp>2/10/2008 8:49:27 PM</TimeStamp>
        <SendResult SeqNum="0">
                <MessageID>137017</MessageID>
                <CellNumber>312555788</CellNumber>
                <Status>Sent</Status>
        </SendResult>
        <SendResult SeqNum="1">
                <MessageID>0</MessageID>
                <CellNumber>6305558866</CellNumber>
                <Status>Not Sent:No OptIn</Status>
        </SendResult>
</MessageStatus>
</SMSImmediate>
```

SendSMSFull

Send an SMS Message using the full range of capabilities that TPI offers on this interface. The SendSMSFull function produces the same output as SendSMS but has more parameters to cover less widely used functions. SendSMSFull can also send a message at any later time. Consequently, it has one more tag for the deferred or delayed send response. The table below only supplies the data on the additional parameters. See the documentation for SendSMS for the remainder.

```
Function SendSMSFull(CellNumber As String, msg As String, Carrier As
String, Tariff As Integer, CarrierLookup As Boolean, ForceOptIn As
Boolean, DelaySend As Boolean, SendTime As Date, Queued As Boolean) As
XmlNode

Function SendSMSFull_Key(APIKey as String, CellNumber As String, msg As
String, Carrier As String, Tariff As Integer, CarrierLookup As Boolean,
ForceOptIn As Boolean, DelaySend As Boolean, SendTime As Date, Queued
As Boolean) As XmlNode
```

Input Parameters		
Name	**Data Type**	**Meaning**
CarrierLookup	Boolean	If your account has Carrier Lookup permission, you may set this parameter to True. If this parameter is true, you do not need to provide the carrier. TPI will look up the carrier for you. Extra charges may apply.
ForceOptIn	Boolean	If your account has permission, you may force an OptIn for this account by setting this parameter to true. If a subscriber has done an OptOut however from an earlier OptIn, the ForceOptIn parameter has no effect.
DelaySend	Boolean	If the DelaySend Parameter is True, the message will be sent at a later time determined by the SendTime parameter.
SendTime	Date	The SendTime parameter must consist of a date and time string in US time format (m/d/y). Valid examples: 2/15/2007 12:10 PM 2/15/2007 14:10 2/15/2007 8:05 AM 2/15/2007 8:05 The time is interpreted as the time zone that is set for this user in their User information. **Warning: If no time is supplied after the date, the message will be sent at midnight!**

		If the SendTime is in the past, the message will be sent within the next minute. SendTime is accurate only to the nearest minute. Adding seconds to the time string is not an error but is ignored. If DelaySend is false, the SendTime parameter must still be set to a valid date but will be ignored.
Output Base Node:		<SMSImmediate>, <SMSQueued> or <SMSDeferred>
Sub Nodes		See SendSMS function
Output Example- Delayed Send		**Output Example 3- Delayed Send**

```
<SMSDeferred xmlns="">
<Customer>CUSTOMER A</Customer>
<MessageStatus>
        <TimeStamp>2/8/2008 3:19:45 PM</TimeStamp>
        <SendResult SeqNum="0">
                <DeferID>75</DeferID>
                <CellNumber>312555788</CellNumber>
                <Status>Deferred</Status>
        </SendResult>
</MessageStatus>
</SMSDeferred>
```

SendToClients

```
Function SendtoClients(ByVal msg As String, ByVal DelaySend As Boolean,
ByVal SendTime As Date) As XmlNode
```

The SendToClients function sends an SMS to all SMS Opted In Clients as well as as Email to all Email Opted In Clients. The clients are the complete set of all Cell Numbers and Emails that are Opted In under the campaign and keyword referenced in the header.

Input Parameters		
Name	**Data Type**	**Meaning**
Msg	String	The message to be sent. This command supports extended length messages. See the Extended Length message section above.
DelaySend	Boolean	See SMSFull
SendTime	Date	See SMSFull
Output Base Node:		
Sub Nodes		
Tag	**Level-Type**	**Meaning**
Type	2-String	The literal “Clients”
TimeFrame	2-String	The literals “Now” or “Deferred” depending upon the type of send requested
Count	2-Integer	The count of clients (SMS+Email) that the message will be sent to.
Output Example		
<GroupSend xmlns=""> <Type>Client</Type> <TimeFrame>Now</TimeFrame> <Count>9</Count> </GroupSend>		

SendToClientsByTag

```
Function SendtoClientsByTag(ByVal msg As String, ByVal TagList As
String, ByVal DelaySend As Boolean, ByVal SendTime As Date) As XmlNode

Function SendtoClientsByTag_Key(ByVal APIKey as String, ByVal msg As
String, ByVal TagList As String, ByVal DelaySend As Boolean, ByVal
SendTime As Date) As XmlNode
```

The SendToClientsByTag function allows a client send to be tailored to only a defined sub-set of the total client list. This is achieved by using Send Tags. Please see our separate documentation on Send Tags at http://www.TextPower.com/Util/Docs/SendTags.html. The SendToClientsByTag operates exactly as does the SendToClients function but provides the one additional input parameter of TagList. A Tag List consists of a comma separated list of Send Tags that have been installed on your client list. Send Tags can be installed by using the ManageSendTag function which is found on the Advanced SOAP interface and by other means which are detailed in the documentation file listed above. Send Tags are not case sensitive and are arbitrary alphanumeric strings with some restrictions. The **TagList** parameter is limited to 600 characters. See the above documentation for all the details on Tag Lists.

Refer to the documentation on the SendToClients call for all other particulars of the SendToClientsByTag function. If Email Clients are defined, emails will also be sent to the Email Clients.

Examples:

TagList=”VIP” sends to only those clients who have the Send Tag, VIP set on them

TagList=”204,VIP” sends to only those clients who have both the VIP and the 204 Send Tag set on them.

TagList=”” sends to all clients and is exactly equivalent to a SendToClients call.

GetCounts

Return the count of all current OptIn Clients and of all subscription packages for this account

```
Function GetCounts() As XmlNode
```

Input Parameters		(None)
Output Base Node:		AccountStats
Sub Nodes		
Tag	**Level-Type**	**Meaning**
Clients	2-Integer	The count of all currently Opted In clients is in the Count attribute.
Output Example		
<AccountStats xmlns=""> <Clients Count="9" /> </AccountStats>		

GetAccountData

Get basic data on your TPI account.

```
Function GetAccountData() As XmlNode
```

Input Parameters		(None)
Output Base Node:		AccountData
Sub Nodes		
Tag	**Level-Type**	**Meaning**
Account	2-Integer	Your internal account number
SendsLeft-3	2-Integer	If your account is on a pay as you go basis, this parameter gives the number of sends left in your account. If your account is not on a pay as you go basis, this number is -1.
ShortCode	2-String	The short code that this account is on.
OptInLevel	2-Integer	The OptIn level of your account. 0=No OptIn 1=Single OptIn 2=Double OptIn
TimeStamp	2-Date	The current server timestamp in Universal time
TimeAdjust	2-Integer	The number of hours to subtract from Universal time to get your local time. This assumes that you have set the Time Zone and Daylight Savings Time parameters for your account using the TextPower customer web site. If you have not set them, TimeAdjust will show the difference between Universal time and Pacific time.
LastReceiveID	2-Integer	The ID of the last received message. If no messages have been received, LastRecieveID=0
LastSendID	2-Integer	The ID of the last sent message. If no messages have been sent, LastSendID=0

Output Example

```
<AccountData xmlns="">
<Account>57</Account>
<SendsLeft>-1</SendsLeft>
<ShortCode>85700</ShortCode>
<OptInLevel>1</OptInLevel>
<TimeStamp>2/10/2009 5:00:30 PM</TimeStamp>
<TimeAdjust>5</ TimeAdjust >
<LastReceiveID>255837</LastReceiveID>
<LastSendID>2378160</LastSendID>
</AccountData>
```

LookupCarrier

Lookup carrier can be used to look up a cell phone carrier. You must have additional permission to execute this function. A charge applies for the use of this function.

```
Function LookupCarrier(PhoneNumber As String) As XmlNode

Function LookupCarrier_Key(APIKey as String, PhoneNumber As String) As
XmlNode
```

Input Parameters		
Name	**Data Type**	**Meaning**
PhoneNumber	String	The phone number that you wish to look up
Output Base Node:		CarrierData
Sub Nodes		
Tag	**Level-Type**	**Meaning**
PhoneNumber	2-String	It may have punctuation but must evaluate to a 10 digit US phone number
Result	2-String	The Result of the lookup. OK means the number was found as a cell phone and all the other parameters are valid.
CarrierCode	2-String	The TPI carrier code string. If the carrier code is INACP then that means that this number "Is Not A Cell Phone". Do not try to send to this number! Only failures will result! Lookups of fictitious movie numbers will receive this code and will be charged.
CarrierName	2-String	The common name for the carrier. Note that this carrier name comes from a national data base and the carrier name may not match the exact form that carriers require for marks purposes. If the carrier is not found, this parameter will be UNK.

Output Example 1: Phone is a cell phone

```
<CarrierData xmlns="">
<PhoneNumber>312555788</PhoneNumber>
<Result>OK</Result>
<CarrierCode>31003</CarrierCode>
<CarrierName>Verizon</CarrierName>
</CarrierData>
```

Output Example 2: Phone is not a cell phone

```
<CarrierData xmlns="">
<PhoneNumber>6304690000</PhoneNumber>
<Result>Not a Cell Phone</Result>
<CarrierCode>INACP</CarrierCode>
<CarrierName>UNK</CarrierName>
```

```
</CarrierData>
```

GetSMSStatus

GetSMSStatus returns the delivery status of a previously sent message. There are 3 ways of referring to the message that you want to query for. If you have a MessageID, use that. You got a MessageID when you did an immediate send earlier. If you have previously queried for status on this message using the Queued or Deferred IDs and the message had been sent, you received a MessageID back. Use that MessageID for all subsequent queries on that call.
If you have only a QueueID, use that parameter.
If you have only a DeferID, use that parameter.
Only one parameter may be non-zero. Both the other parameters must be zero! If you use QueueID or a DeferID and the message has not been sent, you will receive an Errors return.
GetSMSStatus can only be guaranteed to return status for messages sent in the previous 2 days. Fictitious movie number sends will always show no status since no send took place.

```
Function GetSMSStatus(MessageID As Integer, QueueID As Integer, DeferID
As Integer) As XmlNode

Function GetSMSStatus_Key(APIKey as String, MessageID As Integer,
QueueID As Integer, DeferID As Integer) As XmlNode
```

Input Parameters		
Name	**Data Type**	**Meaning**
MessageID	Integer	The MessageID if you sent the message originally without using either the Delay or Queued parameters.
QueueID	Integer	The QueueID if you sent the message originally using the Queue parameter
DeferID	Integer	The DeferID if you sent the message originally using the Delay parameters
Output Base Node:		MTStatus
Sub Nodes		
Tag	**Level-Type**	**Meaning**
MessageID	2-Integer	The MessageID which references this message. You may not have received a MessageID on your original send but all QueueIDs and DeferIDs will be converted to a MessaageID after the message has been sent. For QueueIDs, this usually occurs within a second or two For a DeferID, it will not occur until the message has been sent. Depending upon the parameters used to send the message, this could be months, days, hours, etc. in the future.
FirstStatus	2-String	The status returned from the carrier when the message was first sent. If this is not OK, then there

		will be no valid LastStatus.
LastStatus	2-String	The Last Status on this message. Messages may go through several status points before reaching a final status. Last Status is a word description of the status. NA means **Not Available**.
StatusDate	2-Date	The date and time that the after status was received.
Reason	2-Integer	A numeric code for the last status. 0 indicates no status.

Output Example 1- Successful retrieval

```
<MTStatus xmlns="">
<MessageID>137001</MessageID>
<FirstStatus>OK</FirstStatus>
<LastStatus>Message successfully delivered.</LastStatus>
<StatusDate>2/8/2008 1:05:17 PM</StatusDate>
<Reason>4</Reason>
</MTStatus>
```

Output Example 2 –Query on a deferred message that has not yet been sent

```
<Errors Count="1" xmlns="">
<Error Number="23" Type="MessageID" Description="Message ID 0 not found for this campaign." />
</Errors>
```

Errors Return

Input Parameters		None
Output Base Node:		Errors. The Count attribute has the number of errors contained in the block.
Sub Nodes		
Tag	**Level-Type**	**Meaning**
Error	2-Complex	Each error is described by a attributes in the Error tag. A Number, Type and Description attribute are supplied for each error.
Output Example 1-Invalid Subscription ID		
<Errors Count="1" xmlns=""> <Error Number="30" Type="SubscriptionData" Description="Subscription ID 1 was not found as belonging to this account." /> </Errors>		
Output Example 2-Invalid UserID		
<Errors Count="1" xmlns=""> <Error Number="1" Type="User" Description="Invalid UserID" /> </Errors>		

Soap Exception

Applications must handle the SOAP exception. Please report SOAP exceptions along with the complete text of the exception to support@textpower.com.

MO (Mobile Originating) service

MO service uses a combination of a short code number and keywords to route your message. Keywords are the first words at the beginning of the message. TextPower does a Post or Get to your application with the data for the call. POST/GETs can be performed for received calls, keyword help requests or OptOut requests. The basic interface is the same for each type of call. It is expected that different URLs will posted to for each type of function. There is no parameter supplied to distinguish between different call types. While technically, this is not a SOAP/Web Service interface, it is also published here to provide complete specifications for both the MO interface and the MT SOAP Basic interface in the same document.

.

POST and GET Interfaces

The Post and GET interfaces are identical in parameters. The only difference is in how the parameters are sent.

Parameters:

Parameters	**Meaning**
UID	The user ID that you provided us
PWD	The Password that you provided us
CellNumber	The originating number of the phone that sent the message.
Carrier	The carrier code of the phone that sent the message.
Message	The actual message. Not applicable on. The Message parameter contains the raw message received. The keyword may be on the front or not. It will not be there if this was a sticky reply or if a Block Keyword was found. The message will be text in ASCII for most customers. For certain kinds of accounts receiving binary, the information will be in ASCII encoded hexadecimal. If the original MO was a STOP message, the Message parameter will be "QuitQuit"
KW	The keyword that was used to route the message. Note that this keyword may not actually appear in the message. It may have been inferred due to a sticky reply or may be a Block Keyword.
ShortCode	The short code that applies to the call type.
StickyID	If this MO was routed to you via a normal Short Code/Keyword combination, StickyID is 0. If there was no keyword on the MO and an MT was sent to this phone in the last 30 minutes and you have sticky processing on your

	account, StickyID is the MT that this MO is responding to. The StickyID will be the same as that of a MessageID parameter for an SMSImmediate XML return.
ReceiveID	A unique integer reference number for this MO
OrigMessage	This parameter is only supplied when a STOP message is received. The Message parameter will be "QuitQuit" on all STOP calls. In case you need the original raw text which came in, it will be supplied on the OrigMessage parameter. Remember that this parameter is only supplied for STOP messages. Be prepared in your code for this parameter to not be present.
RouteHint	The RouteHint is non-zero only if StickyID is non-zero. If the RouteHint is non-zero, it a number unique to the user that sent the sticky message.
Tags	If SendTags are present on the CellNumber, the optional Tags parameter will appear. It will contain a comma separated list of the SendTags for this number. E.G. "tag1,tag2,tag3". Remember that this parameter is only supplied for numbers that have SendTags on them. Be prepared in your code for this parameter to not be present.

Each POST must have a response sent by the user. Please do not send a whole HTML page. A simple OK is sufficient. Note that the carrier is always sent to you on a POST or GET. If you will be responding to TextPower with a message send over Sender.aspx, please include the carrier you were sent in the initiating POST or GET. It will have a slight effect in lowering your message charges. It is recommended that when returning messages to TextPower after a POST, you use the queued option on the send. This will help in mitigating timeout errors. Even better is to implement queuing on your server also. If you separate the receiving process from the sending process, timeouts will be very rare.

Opt Out Interface

You may choose to have Opt Out notifications sent to your application. If you choose this option, your application is responsible for sending the required carrier compliant OptOut message to the user. The TextPower system has already performed its own OptOut action before you receive this notification. The interface is identical to the POST/GET interface. When an OptOut has occurred, the Message parameter always contains *QuitQuit*. This is the same regardless of the actual quit word entered by the user. No extra information from the MO is provided.

OptIn/OptOut Notifications

You may choose to have Optin and/or Optout events POSTed or emailed to your application. The data supplied is as follows:

Parameters	**Meaning**
ShortCode	The Short Code on which the optin or optout occurred
Keyword	The Keyword on which the optin or optout occurred
CellNumber	The CellNumber on which the optin or optout occurred
Action	The word OptIn or OptOut depending upon which action occurred.
Time	The Time at which the action occurred. The time is in the user's local time.
UID	The UID parameter which you have selected to be sent to you. If you have not selected this option, UID will be empty.
PWD	The PWD parameter which you have selected to be sent to you. If you have not selected this option, PWD will be empty.

Help Interface

You may choose to have Help requests sent to your application. If you choose this option, your application is responsible for sending the required Help message to the user. The interface is identical to the POST/GET interface.

Opt Out and Help messages may be sent to the same URL. The Opt Out/Help URL must be different from the MO delivery URL.