GLOBAL MOBILE MESSAGING

Technical Specification

Q-CASTER 3.0

Monday, April 07, 2003

custsupport@quios.net

Table of Contents

1 Introduction

Authorized users of the Q-Caster 3.0 services can programmatically submit XML messages through a simple SOAP interface. These messages are converted to SMS and distributed to mobile Handsets worldwide. Q-Caster is ideal for sending large numbers of messages directly from a database or other content provider application.

Q-Caster 3.0 accepts two types of submissions: single and multiple. Use the single submission mode, `send_to_number`, to submit one message to one Handset. Use the multiple mode, `send_to_numbers`, to submit one message to multiple Handsets.

Preliminary results of the message are available immediately, and indicate the initial validation and error check that Q-Caster performs before sending the message to any downstream providers. Further delivery information is also available, up to and including final receipt by the Handset.

Q-Caster 3.0 represents an improvement in functionality and feature set of the Q-Caster system. This version offers support for 8-bit messages, ringtones, 8-bit text (UCS2), SMTP submissions, and enhanced responses.

2 Prerequisites

To submit messages to Q-Caster for transmission, the following conditions must be met:

- Provider must have a current, valid Contract for Message Distribution Services with Quios.
- Provider must have an existing Quios account with valid authentication information (username and password).
- At least one valid static IP number or range (CIDR) must be associated with Provider's username/password.
- Provider must have a Calling Application capable of transmitting and receiving SOAP requests and responses in accordance with these specifications.
- Provider must have a valid access number for each Handset intended as a Message destination, including necessary country codes and area codes.

This technical specifications document assumes familiarity with the following standards, protocols, specifications, and RFCs:

- RFC 1738 “Uniform Resource Locators (URL)”
- W3C Note “Simple Object Access Protocol (SOAP) 1.1”
- W3C Note "Web Services Description Language (WSDL) 1.1"
- "SMTP Transport Binding for SOAP 1.1"
- ETSI GSM 03.38 Specification
- The Unicode Standard
- Nokia Smart Messaging Spec
- Nokia Smart Messaging FAQ

3 System elements and terminology

In addition to industry-standard terminology, this document defines additional terms as listed in Table 3-1.

Table 3-1 Terminology used in Q-Caster Technical Specifications

Calling Application	The programmatic interface that produces the Provider's message and receives resultant success/error notifications.
Request	The complete SOAP request transmitted to Q-Caster.
Response	The complete SOAP response transmitted from Q-Caster.
Message	The content (text, ringtone, etc.) intended for a Handset. Long Messages are divided into multiple SMSs.
SMS	A single SMS to be delivered to a single Handset. Can be text, ringtone, etc. Can be a portion of a long Message that was too large to deliver as a single SMS.
Handset	Mobile telephone, text pager, or other device capable of receiving SMS messages.
Provider	The organization that provides the Request and sends it to Q-Caster for distribution to the Handsets. Quios also uses the services of "downstream providers".

4 Connecting to the Q-Caster server

In order to transmit the Request to the Q-Caster system for distribution, the Calling Application must establish a network (TCP/IP) connection with the Q-Caster access server.

Connections must always be made from an IP address that is registered with Quios as an authorized address for the Account. Connection attempts from unauthorized locations are rejected.

4.1 Connecting through HTTP

To transmit Requests, the Provider directs the Calling Application to connect to the following URI:

```
http://soap.ewingz.com/SOAP/
```

It is recommended that the connection always be made to the symbolic DNS name for the access server rather than to the IP address. The IP address associated with the server name is subject to change without notice.

4.2 Connecting through SMTP

Alternately, the Calling Application can send Requests to Q-Caster via SMTP by sending to the following address:

```
qc30@soap.ewingz.com
```

It is recommended that the Provider relays e-mail through a local SMTP server, rather than connecting through SMTP directly to Q-Caster. The Q-Caster SMTP server is a minimal implementation, and will not queue or store messages.

5 Sending messages

Requests are transmitted to Q-Caster as a SOAP RPC request. The interface for composing the request is the responsibility of the Calling Application. See the W3C Note "Simple Object Access Protocol (SOAP) 1.1" for more details on using SOAP. The Q-Caster namespace identifier is:

`http://soap.ewingz.com/eWingz/SOAP/QC30`

To use the Q-Caster service, the Calling Application must be able to make a SOAP request and read the responses to it. Most applications will make use of an XML parser and/or SOAP toolkit.

The Q-Caster service is described by a WSDL document. This document is available at:

`http://soap.ewingz.com/wsdl/qc30.wsdl`

5.1 The `send_to_number` RPC and the `send_to_numbers` RPC

To send one or more SMS messages, the Calling Application submits a SOAP request containing a call to `send_to_number` or `send_to_numbers`. The `send_to_number` method sends a single message to a single Handset. The `send_to_numbers` method sends a single message to multiple Handsets. Most parameters of the latter request are similar to the `send_to_number` request, except that `send_to_numbers` groups the `msisdn` and the `uniqueid` into an array. Using this array, the Calling Application can send the message to multiple Handsets.

The parameters for these methods are listed in Table 5-1. Q-Caster does not have default values for these parameters (other than originator; see Table 5-1 for details).

Table 5-1 Parameters to `send_to_number` and `send_to_numbers` requests

Parameter	Type	Constraints	Meaning
`username`	string	must be valid username for the submitting IP address	Provider's username for authentication. This parameter is unchanged from QC2.5.
`password`	string	must be valid password for this username	A valid password for the username; used for authentication. This parameter is unchanged from QC2.5.
`testmode`	boolean	[true | false]	`testmode=true` indicates that the Request is in test mode, which performs all authentication, validation, and parsing steps but does not deliver SMSs to Handsets nor debits Provider accounts. Equivalent to test_mode in QC2.5.
`notification`	string	[none | quios | handset]	Sets the extent of delivery information available in regard to this Message. See Section 7.1.1 for details.

Parameter	Type	Constraints	Meaning
`type`	string	[GSM0338 | UCS2 | Binary | RTTTL]	Indicates the type of information contained in the Message. See Section 6 for more information. In QC2.5, message type was parsed by the Q-Caster engine; this behavior is obsolete.
`class`	integer	[0 | 1 | 2 | 3]	Indicates the SMS class of GSM0338 messages; 0 is a flash message, 1 is delivered to memory. Used only when `type= GSM0338` or `UCS2`. If `type` is `Binary` or `RTTTL`, then this field is ignored. In QC2.5 flash messages were indicated by leading \ character in the text field. This behavior is obsolete.
`udhi`	boolean	[true | false]	`udhi=true` indicates that the `header`+`body`+`footer` of this Message contain a UDHI-compliant message.
`originator`	base64 Binary	see Section 6.9 and 6.10	Sets the SMS originator to the specified string dynamically. Defaults to the `originator` string that is associated with this Provider. If the value of the `originator` string is zero length, then the Message is delivered with the default `originator`. See Section 6.9 and 6.10 for more details. Similar to the QC2.5 `originator` parameter.
`header`	base64 Binary	limited to 80 bytes	Contains the message header. Normally `header`, `body`, and `footer` are concatenated together for display on the Handset. However, if `body` is an array, then `header` is ignored. This Quios `header` parameter is not the same as the UDHI header. In QC2.5, the header was part of the `text` parameter.
`body`	base64 Binary or array of base64 Binary	limited to 4000 bytes; `header`+ `body`+`footer` must be >0.	Contains the binary data or text to be transmitted to the Handset. This Quios `body` parameter is not the same as the UDHI body. Similar to the QC2.5 `body` parameter.
`footer`	base64 Binary	limited to 80 bytes	Contains the message footer. Normally `header`, `body`, and `footer` are concatenated together for display on the Handset. However, if `body` is an array, then `footer` is ignored. In QC2.5, the footer was part of the text parameter.

	Parameter	Type	Constraints	Meaning
In `send_to_numbers`, these are the elements of a struct (`number`), which is returned in an array (`numbers`).	`msisdn`	string	minimum 7 digits, maximum 15 digits. International format. No spaces or alpha characters allowed. The + character is not allowed.	Indicates the MSISDN (phone number) of the Handset. Equivalent to the number parameter in QC2.5.
	`uniqueid`	base64 Binary	unless zero-length, must be unique over the entire lifetime of the account, and limited to 80 bytes	A unique string to identify the Message; multiple SMSs can have the same `uniqueid` if they were split automatically. Used to reference Messages for checking delivery status. Q-Caster assigns a value for `uniqueid` to any Message with a zero-length `uniqueid`.

6 Message types and message classes

6.1 Text messages (type GSM0338)

Characters intended for text display in the Message can be encoded by the ETSI GSM 03.38 default alphabet. A chart of the encodings and available characters are shown in Appendix A.

Different character sets are supported by different types of Handset hardware and mobile service carriers. Because Q-Caster cannot determine the specific character display capabilities for any particular Handset, text Messages are not guaranteed to display correctly. However, many Handsets and mobile service carriers, particularly those used outside the United States, are capable of transmitting and displaying ETSI GSM 03.38 characters. In most cases these characters transmit and display correctly.

Note: Many characters in ETSI GSM 03.38 are similar to ASCII and Latin-1, which can give misleading testing results. Calling Applications should undergo thorough testing of unusual characters to ensure that the encoding is correct.

6.1.1 Line breaks in SMS messages

As mentioned previously, the display of characters on Handsets can vary between manufacturers and phone models. However, many phones will end a line and begin the next line in response to this ETSI GSM 03.38 character code: `0x0a` hexadecimal.

6.1.2 Long messages

Text messages of more than 140 bytes (160 characters in GSM0338) are split into multiple SMS transmissions of 140 bytes each. If the message must be split in a particular place (between words, for example), then the Calling Application must submit it as multiple messages, split in the proper place.

6.2 Text messages (type UCS2)

Characters intended for text display in the Message can be encoded by the UCS2 character set. See the information on the Unicode Standard at http://www.unicode.org/ for more information.

Different subsets of UCS2 are supported by different types of Handset hardware and mobile service carriers. Because Q-Caster cannot determine the specific character display capabilities for any particular Handset, text Messages are not guaranteed to display correctly.

6.2.1 Long messages

Text messages of more than 140 bytes (70 UCS2 characters) are split into multiple SMS transmissions of 140 bytes each. If the message must be split in a particular place (between words, for example), then the Calling Application must submit it as multiple messages, split in the proper place.

6.3 Flash messages (class 0)

GSM0338 text messages can be sent as flash SMS messages (class 0 messages). The content of these messages appears on the Handset immediately, and is not stored to memory. Flash capabilities are dependent on the Handset's manufacturer and model; results on hardware that is not flash-capable are undefined.

6.4 Messages delivered to memory (class 1)

Text messages (both GSM0338 and UCS2) can be sent as SMS messages delivered to memory (class 1 messages). These are "ordinary" text messages. Their content doesn't appear on the Handset immediately, but is stored to memory.

6.5 Messages delivered to SIM (class 2)

This feature is rarely used, but is included in the Q-Caster capabilities for completeness.

6.6 Messages delivered to serial port (class 3)

This feature is rarely used, but is included in the Q-Caster capabilities for completeness.

6.7 RTTTL messages (type RTTTL)

Instead of a text message for display on the Handset, the `body` can consist of an RTTTL ringtone for transmission to the Handset as a Smart Message. Q-Caster supports transmission of RTTTL. Lengthy ringtones are automatically split into multiple concatenated SMSs and need not be split before transmission to Q-Caster.

The Calling Application can set the RTTTL attributes for the message. The <name> attribute is required; behavior of RTTTL messages without a <name>

attribute is unspecified. Other attributes are optional. For the attributes listed in Table 6-1, default values will be provided when the Calling Application omits values for these attributes.

Table 6-1: RTTTL attributes

Attribute name	Default value
duration	4
scale	6
beats	63
looping	0

The complete specification for RTTTL is in Appendix B.

RTTTL capabilities are dependent on the Handset's manufacturer and model; results on hardware that is not Smart Message-capable are undefined. Ringtone support is available only on certain hardware (such as Nokia models 3210, 3310, 6110, 6130, 6150, 6210, 6250, 7110, 8110i, 8210, 8810, 8850, 8890, 9110, 9110i, 9210). This information is subject to change.

Note: Certain types of errors in the RTTTL content might cause the message to be identified as a text message, not an RTTTL message. For example, using a hyphen "-" character in a song title is not valid in RTTTL, and will cause Q-Caster to identify and deliver the message as a series of SMS text messages. Inaddition, using an invalid tempo (such as tempo=3) will cause the message to fail identification as an RTTTL submission.

6.8 8-bit binary messages (type Binary)

Q-Caster accepts messages in 8-bit binary format. It performs no validation or encoding; the Calling Application is responsible for proper validation and encoding.

The user data header and binary body are limited to 280 characters, with the space character bringing the total to 281. The SMS header is not included in this length limitation.

Each binary SMS is limited to a maximum of 140 encoded bytes and 281 total characters. However, the Calling Application can encode the `body` as an array of base64 items instead of a single base64 item. This single Message submission is delivered as multiple SMS messages. The SMSs are reassembled upon reaching the Handset.

Binary capabilities are dependent on the Handset's manufacturer and model.

Note: Although Quios cannot provide detailed support regarding binary content, online resources are available to assist in assembling binary content, particularly for Nokia Smart Messaging. Refer to the Nokia Smart Messaging Spec or the Nokia Smart Messaging FAQ for assistance with Smart Messaging formats.

Both of these documents are available, after registering, from the Nokia website http://www.forum.nokia.com/. For the Smart Messaging Spec, choose

Technologies, then Messaging, then Documents. For the Smart Messaging FAQ, choose Technologies, then Messaging, then Smart Messaging.

Note: The Calling Application is responsible for setting the MCC/MCN octets of operator logos.

6.9 Dynamic originator

Q-Caster's `originator` field allows the Provider to supply a customized originator for each SOAP Request. If a non-zero-length `originator` is supplied, then each SMS resulting from the `send_to_number` or `send_to_numbers` call will reflect that originator. In this way the originator can by dynamic; it can be set individually for each message.

Or, the SOAP Request can use a zero length `originator`, which causes Q-Caster to default to the `originator` text that was supplied with the Provider's account setup.

6.10 The originator field

The requirements for the originator field depend on the Provider's contractual agreement with Quios. Quios allows different types of originators. These specifications for originators apply to both the default originator field and to dynamic originator settings.

A standard originator must contain at least one non-numeric character, must be at least 2 and no more than 11 alphanumeric characters, and must match the following regular expression:

```
/^[-.+0-9A-Za-z]{2,11}$/
```

With prior contractual agreement, a Provider can use an International originator. With this type of originator, the Handset can call directly back to the message sender as represented in the originator. This type of originator must use a valid international format MSISDN in the originator field, optionally preceded by the plus "+" character. International originators must be at least 5 and no more than 15 digits.

With prior contractual agreement, a Provider can use a National originator (also called a Short Code). This type of originator must at least one and no more than 5 digits, and may not include a plus "+" character.

7 Responses to messages

Each successful SOAP request receives a SOAP response; in Q-Caster, these responses are used to communicate to the Calling Application information about the message's progress through its delivery lifecycle. The initial SOAP response to `send_to_number` or `send_to_numbers` reports the initial status of the Request. Major error conditions result in SOAP faults, as described in Section 7.2.

The initial response indicates the result of initial parsing of the Request, including preliminary validity checks on the MSISDN and other contents of the Request. A `send_to_number` response is a `send_to_number_result` struct, which is described in Section 7.1. A `send_to_numbers` response is an array. Each element of the array is a `send_to_number_result` struct, which is described in Section 7.1.

The original Request assigns one `uniqueid` to each Message, or allows Q-Caster to assign this `uniqueid`. However, if any Message is too large for a single SMS, then Q-Caster divides it into multiple SMSs ("segments") before delivery. The number of segments of a message is reported in the `send_to_numbers` result.

Note: A successful `send_to_number` response does **not** indicate successful delivery to the Handset. This preliminary status indicates only that Q-Caster has determined that the request is valid, and will send the request to its downstream providers. For additional information about the message's status, the Calling Application must call the `status` or `history` method, as described in Section 8.

Q-Caster responses always include numeric codes and associated text. The numeric codes, such as `disposition_code`, are the definitive result and should be used for the Calling Application's logic. The associated text, such as `disposition_text`, is provided solely for the convenience of the developer, and should not be referred to directly in the Calling Application.

7.1 The `send_to_number` result

The `send_to_number` result is used to convey information about the current or historical status of each SMS. See Table 7-1 for a list of the information returned in this struct. Following sections describe each of these keys.

Table 7-1 Values of `send_to_number` result

Key	Value Type	Meaning
`uniqueid`	base64 Binary	The `uniqueid` for this Message as provided by the Calling Application, or as generated by Q-Caster.
`segments`	integer	If the Message is too large for a single SMS, then Q-Caster divides it into multiple SMSs. This number indicates the total number of SMSs for this Message.
`response_code`	integer	A 3-digit code indicating the most recent action performed on the SMS. See Table 7-4 for all legal values.
`response_text`	string	A human-readable string that corresponds to the `response_code`. This string is subject to change at any time.
`disposition_code`	integer	A 1-digit code indicating the most recent status of the SMS. See Table 7-3 for all legal values.
`disposition_text`	string	A human-readable string that corresponds to the `disposition_code`. This string is subject to change at any time.

7.1.1 Notification levels

The extent of information available for a particular message depends on its progress through its delivery lifecycle, and on its notification level. The `notification` parameter is set in the `send_to_number(s)` request when the message is initially submitted to Q-Caster, **not** when the message status is requested. The notification level changes the information available to the Calling Application regarding the message's progress through its delivery lifecycle. For example, redelivery is attempted for 24 hours for every message that encounters an unavailable Handset, but final delivery confirmation is available only for those sent whose `notification` is `handset`. Without this notification level, information about the redelivery results is unavailable to the Calling Application (and to Quios).

According to the notification level, different types of information are available regarding the message's status. See Table 7-2 for an explanation of the notification levels. Subsequent tables will also refer to notification levels.

Table 7-2 Notification levels

Pricing	Level	Meaning
standard	`none`	Status of message lifecycle up to the point where it exits the Quios system for a downstream provider. Includes information on validity checks (for number length, country code, etc.) , blocked numbers, and other preliminary results. Does not report final delivery to Handset.
premium	`quios`	Status of entire message lifecycle for most messages. Includes final success/failure after first delivery attempt and immediate retries. Does not include status of delayed retries (as used when Handset is off or unreachable).
premium	`handset`	Status of entire message lifecycle, including final delivery success/failure after immediate and delayed retries.

7.1.2 Message disposition: `disposition_code` and `disposition_text`

The message disposition indicates the progress of the message during the process of acceptance, testing, and transmission. A higher-numbered disposition state indicates that the message has progressed further in the transmission process. The message moves through the disposition states in order, but does not necessarily experience each state. See Table 7-3 for a listing of the values for `disposition_code`.

The last three dispositions, `disposition_code` 4, 5, and 6 (success, failure, and final), indicate that this message has reached the end of its transmission lifecycle, and no further action will be performed on this message.

Table 7-3 Corresponding `disposition_code` and `disposition_text` values

Code	Text	Meaning
1	tested	Message was sent in testmode; tests were successful; if testmode were off then this message would be passed downstream.
2	processing	Message is active in Quios system.
3	waiting	Quios is waiting for status information from a downstream provider.
4	success	Message was delivered to Handset.
5	failure	Message will not be delivered to Handset.
6	final	Message is finished in Quios system, but final status cannot be determined.

7.1.3 Message response: `response_code` and `response_text`

The message response indicates the results of actions performed on the message. See Table 7-4 for a listing of the legal values of `response_code`.

In `send_to_number` response or `send_to_numbers` response, the `response_code` will be limited to responses to the preliminary validity testing that the message undergoes. Messages in testmode will also be limited to these responses. These responses are in the range of 100 to 199, and are indicated in Table 7-4 by shading.

Note: The Calling Application must use only `response_code` in its logic; `response_text` values are for convenience only and are subject to change.

Table 7-4 Corresponding `response_code` and `response_text` values

Code	Text	Meaning
110	message accepted	Q-Caster has accepted the message for validation and delivery.
111	message accepted for testing	Q-Caster has accepted the message in testmode. No delivery attempts will be made.
115	internal error, fatal	Q-Caster has encountered an unrecoverable error.
130	number blocked, account	This number is not reachable from this account.
131	number blocked, eWingz	This number is not reachable from the Quios system.
134	no route to subscribed carrier	Message is not deliverable because there is no route to this Handset.
136	invalid number	The MSISDN is not valid.
137	invalid number, 7-15 digits only	The MSISDN is not valid because it is the wrong length.
138	invalid number, bad country code	The MSISDN is not valid because no such number exists in the provided country code.
139	invalid number, not a handset	The MSISDN is not a valid number.
140	illegal originator	The originator is not allowed. Note: numeric originators are allowed only by prior contractual agreement.
141	invalid content	The content of the message is invalid; usually this indicates a blank text message or an invalid RTTTL message.
144	daily limit exceeded	The Provider's account has exceeded the contractual daily limit for messages.
145	monthly limit exceeded	The Provider's account has exceeded the contractual monthly limit for messages.
301	message defunneled	SMS has been removed from input funnel and is added to the Q-Caster database.
302	message selected for routing	The message has been retrieved from the Q-Caster database.
303	message routed	A downstream provider has been chosen based on the SMS's requirements.
304	message selected for delivery	The SMS has been retrieved from the database for sending to a downstream provider.
305	message added to process queue	The message has been queued for delivery to a downstream provider.
306	too many attempts	The message has exceeded the allowed number of delivery attempts.
307	message routed waiting for parent	This is part of a multipart message, and is waiting for the earlier message parts to be delivered.
312	status query sent	Quios has requested delivery status from downstream provider.
314	internal error, retryable	The message has encountered a non-fatal internal error; delivery will be re-attempted.
315	internal error, fatal	The message has encountered a fatal internal error.
318	external error, fatal	The message has encountered a fatal external error.
410	message accepted	Message has been accepted by a downstream provider for delivery attempt.
413	status query failed	Downstream provided could not provide delivery status.
414	internal error, retryable	The message has encountered a non-fatal internal error; delivery will be re-attempted by eWingz.
415	internal error, fatal	The message has encountered a fatal internal error.
420	no further status	No further status information is available from the downstream provider.
505	message added to process queue	The downstream provider has queued the message to the final carrier for delivery.

Code	Text	Meaning
510	message accepted	The final carrier has accepted the message for delivery.
515	internal error, fatal	The message has encountered a fatal internal error.
516	external error, retryable	The message has encountered a non-fatal external error; delivery will be re-attempted by eWingz.
517	external error, waiting	The message has encountered a non-fatal external error; delivery will be re-attempted at a later time by the downstream provider.
518	external error, fatal	The message has encountered a fatal external error.
519	handset is off	The destination Handset is powered off; message delivery will be re-attempted at a later time by the downstream provider.
520	no further status	No further status is available for this message.
534	no route to subscribed carrier	The carrier that the Handset is subscribed to is not reachable.
535	carrier has no SMS service	The carrier that the Handset is subscribed to is not reachable.
536	invalid number	The downstream provider has determined that this MSISDN is not valid.
540	illegal originator	The downstream provider has disallowed this originator.
541	invalid content	The downstream provider has disallowed this content.
542	inexpensive SMS only	The downstream provider's charges exceed those allowed for this message.
570	SMS not provisioned	Regardless of the Handset's physical capabilities, the service provider does not allow this Handset to receive SMSs.
571	validity period expired	The validity time period for this message has been exceeded.
572	unknown subscriber	The downstream provider has no record of this subscriber.
573	SMS temporarily barred	This subscriber cannot receive SMS.
574	no route to visited carrier	This subscriber is roaming, and there is no route to the Handset.
576	temp. delivery failure, fatal, SIM full?	The downstream provider has encountered a temporary failure; delivery will be re-attempted at a later time.
610	message accepted	Message was accepted by the destination Handset.

7.2 SOAP faults

Major errors in the message request result in SOAP faults. These faults indicate unrecoverable errors, and the entire SOAP request is rejected. These faults are generally not considered to be part of normal Calling Application operations, and should be encountered rarely (if ever) in the Calling Application's production environment. In particular, Client.Authentication and Client.Input faults should never be encountered outside the new account setup and development activities.

A representative listing of SOAP faults is in Table 7-5; this listing is subject to change without notice. These faults can contain variable text, indicated by italics in the table.

Table 7-5: SOAP faults

Type	Text	Meaning
Server	General Failure - Untrapped Error	Error in Quios server; please send diagnostic information to custsupport@quios.net
Client .Authentication	*username*/*password*/*IP* combination does not authenticate	The authentication information is not valid; check that the username, password, and IP address are authorized and correct.
Client.Input	`history` received unknown `uniqueid`	No Message was ever sent with this `uniqueid`; validate the `uniqueid` against the local database before submitting request.
Client.Input	`history` received wrong number of parameters: *count*, must be 3	The `history` request was malformed.
Client.Input	received empty message content, `header` + `body` + `footer` must be > 0	Message content was zero length; validate for empty content before submitting request.
Client.Input	received invalid `class`: *value*, must be 0, 1, 2 or 3	Value for `class` was not valid; validate for parameter values before submitting request.
Client.Input	received invalid notification: *value*, must be none, quios, or handset	Value for `notification` was not valid; validate for parameter values before submitting request.
Client.Input	received invalid `originator`: *value*	Value for `originator` was not valid; validate for parameter values before submitting request.
Client.Input	received invalid type: *value*, must be GSM0338, Binary, RTTTL, UCS2	Value for `type` was not valid; validate for parameter values before submitting request.
Client.Input	`send_to_number` received invalid msisdn, *msisdn*	Value for `misisdn` was not valid; validate for parameter values before submitting request.
Client.Input	`send_to_number` received too long `uniqueid`, *length* bytes, maximum 80	Value for `uniqueid` was not valid; validate for parameter values before submitting request.
Client.Input	`send_to_number` received wrong number of parameters: *count,* must be 13	The `send_to_number` request was malformed.
Client.Input	`send_to_numbers` received invalid msisdn, *msisdn*	Value for `misisdn` was not valid; validate for parameter values before submitting request.
Client.Input	`send_to_numbers` received non-array numbers parameter, numbers must be an array	The `send_to_numbers` request was malformed.
Client.Input	`send_to_numbers` received non-struct numbers element	The `send_to_numbers` request was malformed.
Client.Input	`send_to_numbers` received too long `uniqueid`, *length* bytes, maximum 80	Value for `uniqueid` was not valid; validate for parameter values before submitting request.
Client.Input	`send_to_numbers` received wrong number of parameters: *count,* must be 12	The `send_to_numbers` request was malformed.

Type	Text	Meaning
Client.Input	`status` received unknown `uniqueid`	No Message was ever sent with this `uniqueid`; validate the `uniqueid` against the local database before submitting request.
Client.Input	`status` received wrong number of parameters: *count*, must be 3	The status request was malformed.

8 Checking message status

Q-Caster offers additional SOAP RPCs for retrieving information about how a Message moves through its delivery lifecycle. The extent of information available is determined by the Message's `notification` level, as explained in Section 7.1.1.

8.1 The `status` request and `history` request

To receive information on the current delivery status of a Message, the Calling Application submits a `status` request referring to the Message's `uniqueid`. This request will retrieve information about where the Message is in its lifecycle.

To receive information on the current and historical delivery status of a Message, the Calling Application submits a `history` request referring to the Message's `uniqueid`. This request will retrieve information about the Message's lifecycle, from the initial response to the current information.

For the types of data available via `status` request and `history` request, see Section 8.4.

Table 8-1 lists the parameters required by the `status` and `history` calls.

Table 8-1 Parameters to `status` request and `history` request

Parameter	Type	Constraints	Meaning
`username`	string	must be valid username for the submitting IP address	Provider's username for authentication.
`password`	string	must be valid password for this username	A valid password for the username; used for authentication.
`uniqueid`	base64 Binary	must be valid `uniqueid` for this username	A unique string to identify the Message; multiple SMSs can have the same `uniqueid` if they were split automatically. Used to reference Messages for checking delivery status.

8.2 The `status` response

Each `status` request is answered with a `status` response, which is an array called `status_result`. This response indicates the current status of the Message in its delivery lifecycle. Although the `status` response reports the status of a single Message, it is nonetheless an array of `sms_status_entry` structs, because the `status` request for information on a single `uniqueid` can refer to a number of SMSs if the original Message was split due to length. See Section 8.4 for information on `sms_status_entry`.

8.3 The `history` response

Each `history` request is answered with a `history` response. This response contains all the current and historical information regarding the progress of the Message in its delivery lifecycle. The `history` response consists of an array of `status_result` elements. Each `status_result` element consists of an array of `sms_status_entry` structs. See Section 8.4 for information on `sms_status_entry`.

8.4 The `sms_status_entry`

The `sms_status_entry` reports an individual state of an individual SMS. It is used in an array to report on all the SMSs that resulted from a Message submission. See Table 8-2 for the values returned by `sms_status_entry`.

Table 8-2 Values of `sms_status_entry`

Key	Value Type	Meaning
`response_code`	integer	A 3-digit code indicating the most recent action performed on the SMS. See Table 7-4 for all legal values.
`response_text`	string	A human-readable string that corresponds to the `response_code`. This string is subject to change at any time.
`disposition_code`	integer	A 1-digit code indicating the most recent status of the SMS. See Table 7-3 for all legal values.
`disposition_text`	string	A human-readable string that corresponds to the `disposition_code`. This string is subject to change at any time.
`date`	date	The date that this action was performed on the SMS.
`time`	time	The time in GMT that this action was performed on the SMS.

9 Retrieving 2-Way Messages

Accounts that are properly configured for 2-way messaging can use Q-Caster to retrieve the 2-way messages in their queue. Contact your Quios sales representative for information on 2-way messaging.

9.1 The `get_messages` request

To access messages in the 2-way queue, the Calling Application sends a `get_messages` request. The parameters of this RPC are listed in Table 9-1. A `get_messages` request will retrieve all available messages for this account, and remove them from the queue.

Table 9-1 Parameters to `get_messages` requests

Parameter	Type	Constraints	Meaning
`username`	string	must be valid username for the submitting IP address	Provider's username for authentication.
`password`	string	must be valid password for this username	A valid password for the username; used for authentication.

9.2 The `get_messages_result` response

The response to the `get_messages` request is `get_messages_result`, an array of `get_message_entry` elements. Each `get_message_entry` represents a message that has been queued on the Q-Caster system awaiting this request. Each element of the array has the key/value pairs listed in Table 9-2.

Table 9-2 Values of `get_messages_entry` response

Key	Value Type	Meaning
`class`	integer	Indicates the SMS class of GSM0338 messages; 0 is a flash message, 1 is delivered to memory.
`udhi`	boolean	`udhi=true` shows that this message was sent with UDHI indicated. Q-Caster does not verify its UDHI compliance.
`originator`	base64	Contains the originator of the message.
`msisdn`	string	Contains the destination MSISDN (phone number) for this message.
`body`	base64	Contains the message body (message text).
`date`	date	Indicates the date that the message was received.
`time`	time	Indicates the time that the message was received.

10 Appendix A: Available GSM/UCS2 Characters and Their Encodings

http://www.unicode.org/Public/MAPPINGS/ETSI/GSM0338.TXT provides information about GSM0338 characters and how they map to Unicode characters.

For information on UCS2 characters, see The Unicode Standard at http://www.unicode.org

Characters used by ETSI GSM 03.38 default alphabet

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	@	£	$	¥	è	é	ù	ì	ò	Ç	LF	Ø	ø	C	Å	å
10	Δ	_	Φ	Γ	Λ	Ω	Π	Ψ	Σ	Θ	Ξ	es	Æ	æ	ß	É
20	spc	!	"	#	¤	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	¡	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ñ	Ü	§
60	¿	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	ä	ö	ñ	ü	à

11 Appendix B: RTTTL Specification

```
<ringing-tones-text-transfer-language> :=
<name> <sep> [<defaults>] <sep> <note-command>+

<name> := <char>+ ; maximum name length 10 characters

<sep> := ":"

<defaults> :=
<def-note-duration> := 'd'
<def-note-scale>    := 'o'
<def-beats>         := 'b'
<def-style>         := 's'
<def-looping>       := 'l'

If not specified, defaults are

4  = duration
6  = scale
63 = beats-per-minute

Valid in tone section: o, b, s

 <note-command> :=
[<duration>] <note> [<scale>] [<special-duration>] <delimiter>

<duration> :=
"1"  Full 1/1 note
"2"  1/2 note
"4"  1/4 note
"8"  1/8 note
"16"  1/16 note
"32" 1/32 note

<note> :=
"P"  pause
"C"
"C#"
"D"
"D#"
"E"
"F"
"F#"
"G"
"G#"
"A"
"A#"
"B"
"H"

<scale> :=
"4"  Note A is 440Hz
"5"  Note A is 880Hz
"6"  Note A is 1.76 kHz
"7"  Note A is 3.52 kHz

<special-duration> :=
"."  Dotted note
";"  Double dotted note
"&"  2/3 length

<delimiter> := ","
```

12 Appendix C: Examples of Perl development

Perl requirements:

1. Perl 5.4 or higher
2. SOAP::Lite v0.55

SOAP::Lite is available at

`http://www.perl.com/CPAN-local/modules/by-module/SOAP/SOAP-Lite-0.55.zip`

or

`http://www.perl.com/CPAN-local/modules/by-module/SOAP/SOAP-Lite-0.55.tar.gz`

Listing C.1 send_to_number.pl

```
#!/usr/bin/perl -w

use strict;
use SOAP::Lite soapversion => 1.1;


#grab input arguments
my ($msisdn,$body) = @ARGV;

#validate input arguments
unless ( defined($msisdn) && defined($body) ) {
  die "usage: $0 msisdn body\n";
}

## Enter your account information below
my $username = "YOURUSERNAME";
my $password = "yourpassword";

printf("Creating SOAP object and calling send_to_number...\n");

#create SOAP object
my $qcaster = SOAP::Lite->new(
           uri         => 'http://soap.ewingz.com/eWingz/SOAP/QC30',
           proxy       => 'http://soap.ewingz.com/SOAP');

#call send_to_number method
my $response=$qcaster->send_to_number(SOAP::Data->value(
             SOAP::Data->name('username')    ->type('string'  => $username),
             SOAP::Data->name('password')    ->type('string'  => $password),
             SOAP::Data->name('testmode')    ->type('boolean' => 0),
             SOAP::Data->name('notification')->type('string'  => 'handset'),
             SOAP::Data->name('type')        ->type('string'  => 'GSM0338'),
             SOAP::Data->name('class')       ->type('integer' => 1),
             SOAP::Data->name('udhi')        ->type('boolean' => 0),
             SOAP::Data->name('originator')  ->type('base64'  => 'QC30RC1'),
             SOAP::Data->name('header')      ->type('base64'  => ''),
             SOAP::Data->name('body')        ->type('base64'  => $body),
             SOAP::Data->name('footer')      ->type('base64'  => ''),
             SOAP::Data->name('msisdn')      ->type('string'  => $msisdn),
             SOAP::Data->name('uniqueid')    ->type('base64'  => '')
             ));

#test response from server
if($response->fault) {

   #request failed
   printf("Request failed.\n");
   printf("\nResponse:\n");

   #print fault information to stdout
   printf("\tFault detail: %s\n", $response->faultdetail) if
            $response->faultdetail;
   printf("\tFault code: %s\n", $response->faultcode) if
            $response->faultcode ne "";
   printf("\tFault string: %s\n", $response->faultstring) if
            $response->faultstring;

} else {

   #request succeeded
   printf("Request succeeded.\n");
   printf("\nResponse:\n");

   #print send_to_number_result data to stdout
   printf("\tuniqueid = %s\n", $response->result->{uniqueid});
   printf("\tsegments = %s\n", $response->result->{segments});
```

```
   printf("\tresponse_code = %s\n", $response->result->{response_code});
   printf("\tresponse_text = %s\n", $response->result->{response_text});
   printf("\tdisposition_code = %s\n", $response->result->{disposition_code});
   printf("\tdisposition_text = %s\n", $response->result->{disposition_text});
}
```

Listing C.2 send_to_numbers.pl

```
#!/usr/bin/perl -w

use strict;
use SOAP::Lite soapversion => 1.1;


#grab input arguments
my ($msisdn,$body) = @ARGV;

#validate input arguments
unless ( defined($msisdn) && defined($body) ) {
  die "usage: $0 msisdn body\n";
}

## Enter your account information below
my $username = "YOURUSERNAME";
my $password = "yourpassword";

printf("Creating SOAP object and calling send_to_numbers...\n");

#create SOAP object
my $qcaster = SOAP::Lite->new(
           uri         => 'http://soap.ewingz.com/eWingz/SOAP/QC30',
           proxy       => 'http://soap.ewingz.com/SOAP');

#call send_to_numbers method
my $response=$qcaster->send_to_numbers(SOAP::Data->value(
             SOAP::Data->name('username')    ->type('string'  => $username),
             SOAP::Data->name('password')    ->type('string'  => $password),
             SOAP::Data->name('testmode')    ->type('boolean' => 0),
             SOAP::Data->name('notification')->type('string'  => 'handset'),
             SOAP::Data->name('type')        ->type('string'  => 'GSM0338'),
             SOAP::Data->name('class')       ->type('integer' => 1),
             SOAP::Data->name('udhi')        ->type('boolean' => 0),
             SOAP::Data->name('originator')  ->type('base64'  => 'QC30RC1'),
             SOAP::Data->name('header')      ->type('base64'  => ''),
             SOAP::Data->name('body')        ->type('base64'  => $body),
             SOAP::Data->name('footer')      ->type('base64'  => ''),
             SOAP::Data->name('numbers')     ->value(
             [
             SOAP::Data->name('number')       ->value(
             { msisdn   => SOAP::Data       ->type('string'  => $msisdn),
             uniqueid => SOAP::Data       ->type('base64'  => '')
             })
             ])));

#test response from server
if($response->fault) {

   #request failed
   printf("Request failed.\n");
   printf("\nResponse:\n");

   #print fault information to stdout
   printf("\tFault detail: %s\n", $response->faultdetail) if
             $response->faultdetail;
   printf("\tFault code: %s\n", $response->faultcode) if
             $response->faultcode ne "";
   printf("\tFault string: %s\n", $response->faultstring) if
             $response->faultstring;

} else {

   #request succeeded
   printf("Request succeeded.\n");
   printf("\nResponse:\n");
```

```
   #loop and print send_to_numbers_result data to stdout
   for my $entry(@{$response->result}) {

      printf("\tuniqueid = %s\n", $entry->{uniqueid});
      printf("\tsegments = %s\n", $entry->{segments});
      printf("\tresponse_code = %s\n", $entry->{response_code});
      printf("\tresponse_text = %s\n", $entry->{response_text});
      printf("\tdisposition_code = %s\n", $entry->{disposition_code});
      printf("\tdisposition_text = %s\n", $entry->{disposition_text});

   }
}
```

Listing C.3 status.pl

```
#!/usr/bin/perl -w

use strict;
use SOAP::Lite soapversion => 1.1;


#grab input arguments
my ($uniqueid) = @ARGV;

#validate input arguments
unless ( defined($uniqueid) ) {
  die "usage: $0 uniqueid\n";
}

## Enter your account information below
my $username = "YOURUSERNAME";
my $password = "yourpassword";

printf("Creating SOAP object and calling status...\n");

#create SOAP object
my $qcaster = SOAP::Lite->new(
           uri         => 'http://soap.ewingz.com/eWingz/SOAP/QC30',
           proxy       => 'http://soap.ewingz.com/SOAP');

#call status method
my $response=$qcaster->status(SOAP::Data->value(
              SOAP::Data->name('username')->type('string' => $username),
              SOAP::Data->name('password')->type('string' => $password),
              SOAP::Data->name('uniqueid')->type('base64' => $uniqueid)
              ));

#test response from server
if($response->fault) {

   #request failed
   printf("Request failed.\n");
   printf("\nResponse:\n");

   #print fault information to stdout
   printf("\tFault detail: %s\n", $response->faultdetail) if
              $response->faultdetail;
   printf("\tFault code: %s\n", $response->faultcode) if
              $response->faultcode ne "";
   printf("\tFault string: %s\n", $response->faultstring) if
              $response->faultstring;

} else {

   #request succeeded
   printf("Request succeeded.\n");
   printf("\nResponse:\n");

   #loop and print status_result data to stdout
   for my $entry(@{$response->result}) {

      printf("\tresponse_text = %s\n", $entry->{response_text});
      printf("\tdate = %s\n", $entry->{date});
      printf("\tdisposition_text = %s\n", $entry->{disposition_text});
      printf("\tresponse_code = %s\n", $entry->{response_code});
      printf("\ttime = %s\n", $entry->{time});
      printf("\tdisposition_code = %s\n", $entry->{disposition_code});
   }

}
```

Listing C.4 history.pl

```
#!/usr/bin/perl -w

use strict;
use SOAP::Lite soapversion => 1.1;

#grab input arguments
my ($uniqueid) = @ARGV;

#validate input arguments
unless ( defined($uniqueid) ) {
  die "usage: $0 uniqueid\n";
}

## Enter your account information below
my $username = "YOURUSERNAME";
my $password = "yourpassword";

printf("Creating SOAP object and calling history...\n");

#create SOAP object
my $qcaster = SOAP::Lite->new(
           uri         => 'http://soap.ewingz.com/eWingz/SOAP/QC30',
           proxy       => 'http://soap.ewingz.com/SOAP');

#call history method
my $response=$qcaster->history(SOAP::Data->value(
             SOAP::Data->name('username')->type('string' => $username),
             SOAP::Data->name('password')->type('string' => $password),
             SOAP::Data->name('uniqueid')->type('base64' => $uniqueid)
             ));

#test response from server
if($response->fault) {

   #request failed
   printf("Request failed.\n");
   printf("\nResponse:\n");

   #print fault information to stdout
   printf("\tFault detail: %s\n", $response->faultdetail) if
           $response->faultdetail;
   printf("\tFault code: %s\n", $response->faultcode) if
           $response->faultcode ne "";
   printf("\tFault string: %s\n", $response->faultstring) if
           $response->faultstring;

} else {

   #request succeeded
   printf("Request succeeded.\n");
   printf("\nResponse:\n");

   #loop and print history_result data to stdout
   for my $entry(@{$response->result}) {

      for my $status (@{$entry}){
         printf("\tresponse_text = %s\n", $status->{response_text});
         printf("\tdate = %s\n", $status->{date});
         printf("\tdisposition_text = %s\n", $status->{disposition_text});
         printf("\tresponse_code = %s\n", $status->{response_code});
         printf("\ttime = %s\n", $status->{time});
         printf("\tdisposition_code = %s\n\n", $status->{disposition_code});
      }

   }
}
```

13 Appendix D: Examples of Java development

Java requirements:

1. Java SDK v 1.3 and higher
2. Apache Axis

Apache Axis is available at:

`http://ws.apache.org/axis/dist/1_0/ xml-axis-10.zip`

or

`http://ws.apache.org/axis/dist/1_0/ xml-axis-10.tar.gz`

Listing D.1 SendToNumber.java

```
package com.ewingz.samples.qc3;

import java.net.URL;
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.AxisFault;

import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
import org.apache.axis.encoding.*;
import org.apache.axis.encoding.ser.*;

public class SendToNumber {

  public static void main(String [] args) throws Exception {

    SendToNumber stn = new SendToNumber();

    //grab input arguments
    String msisdn = args[0];
    String body = args[1];

    //validate input arguments
    if(msisdn.length() == 0 || body.length() == 0) {
      System.out.println("Usage: send_to_number msisdn body");
      System.exit(1);
    }

    // Enter your account information below
    String username      = "YOURUSERNAME";
    String password      = "yourpassword";

    try {
      String endpointURL = "http://soap.ewingz.com/SOAP";

      System.out.println("Creating SOAP object...");

      //set up SOAP object
      Service service = new Service();
      Call call = (Call) service.createCall();
      call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
      call.setOperationName(new QName("http://soap.ewingz.com/eWingz/SOAP/QC30/",
          "send_to_number"));
      call.setUseSOAPAction(true);

call.setSOAPActionURI("http://soap.ewingz.com/eWingz/SOAP/QC30/#send_to_number");
      call.setOperationStyle("rpc");

      //set parameter types
      stn.AddParameters(call);

      //set return type
      call.setReturnType(new
QName("http://soap.ewingz.com/eWingz/SOAP/QC30/qc30/type","send_to_number_result"
),
                         SendToNumberResultBean.class);

      //declare Type Mappings
      stn.AddTypeMappings(call);

      System.out.println("Calling send_to_number...");

      try {
        //call send_to_number method
```

```
        SendToNumberResultBean stnrb = (SendToNumberResultBean) call.invoke( new
Object[] { username,
                  password, new Boolean(true), "handset", "GSM0338", new
Integer(1), new Boolean(true),
                  Base64.encode(new String("QC30RC1").getBytes()),
                  "", Base64.encode(new String(body).getBytes()), "", msisdn, ""
} );

        //request succeeded
        System.out.println("Request succeeded.");
        System.out.println("\nResponse:");

        //print send_to_number_result data to stdout
        System.out.println("\tuniqueid = " + new String(stnrb.getUniqueid()));
        System.out.println("\tsegments = " + stnrb.getSegments());
        System.out.println("\tresponse_code = " + stnrb.getResponse_code());
        System.out.println("\tresponse_text = " + stnrb.getResponse_text());
        System.out.println("\tdisposition_code = " +
stnrb.getDisposition_code());
        System.out.println("\tdisposition_text = " +
stnrb.getDisposition_text());

      } catch (Exception e) {
        //create fault object
        AxisFault fault = AxisFault.makeFault(e);
        QName qn = fault.getFaultCode();

        //request failed
        System.out.println("Request failed.");
        System.out.println("\nResponse:");

        //print fault information to stdout
        System.out.println("\tFault string: " + fault.getFaultString());
        System.out.println("\tFault code: " + qn.toString());
      }

    } catch (Exception e) {
       System.err.println(e.toString());
    }
  }

/**
 *
 * @param call
 */
  private void AddParameters(Call call) {
    //username
    call.addParameter("username",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //password
    call.addParameter("password",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //testmode
    call.addParameter("testmode",
          org.apache.axis.Constants.XSD_BOOLEAN,
          javax.xml.rpc.ParameterMode.IN);
    //notification
    call.addParameter("notification",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //type
    call.addParameter("type",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //class
    call.addParameter("class",
```

```
          org.apache.axis.Constants.XSD_INTEGER,
          javax.xml.rpc.ParameterMode.IN);
    //udhi
    call.addParameter("udhi",
          org.apache.axis.Constants.XSD_BOOLEAN,
          javax.xml.rpc.ParameterMode.IN);
    //originator
    call.addParameter("originator",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //header
    call.addParameter("header",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //body
    call.addParameter("body",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //footer
    call.addParameter("footer",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //msisdn
    call.addParameter("msisdn",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //uniqueid
    call.addParameter("uniqueid",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
  }

/**
 *
 * @param call
 */
  private void AddTypeMappings(Call call) {
    //declare serializer for SOAPStruct (contains send_to_number_result)
    call.registerTypeMapping(SendToNumberResultBean.class,
                             new QName("http://xml.apache.org/xml-
soap","SOAPStruct"),
                             BeanSerializerFactory.class,
                             BeanDeserializerFactory.class);
   //declare serializer for Base64
   call.registerTypeMapping(Base64.class,
                             new QName("http://xml.apache.org/xml-soap","Base64"),
                             Base64SerializerFactory.class,
                             Base64DeserializerFactory.class);
  }
}
```

Listing D.2 SendToNumberResultBean.java

```
package com.ewingz.samples.qc3;

import java.math.BigInteger;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author $Author: $
 * @version 1.0$Revision: 15 $ $Date: 10/01/02 11:44a $
 */

public class SendToNumberResultBean {

  /**
   * Default constructor to create new SendToNumberResultBean
   */
  public SendToNumberResultBean() {
  }

  private byte[]  uniqueid;
  private BigInteger segments;
  private String  response_text;
  private String  disposition_text;
  private BigInteger response_code;
  private BigInteger disposition_code;

  /**
  *
  * @return the byte[] value of uniqueid.
  */
  public byte[] getUniqueid(){
          return uniqueid;
  }

  /**
  *
  * @param aUniqueid - the new value for uniqueid
  */
  public void setUniqueid(byte[] aUniqueid){
          uniqueid = aUniqueid;
  }


  /**
  *
  * @return the BigInteger value of segments.
  */
  public BigInteger getSegments(){
          return segments;
  }

  /**
  *
  * @param aSegments - the new value for segments
  */
  public void setSegments(BigInteger aSegments){
          segments = aSegments;
  }


  /**
  *
  * @return the String value of response text.
  */
```

```
  public String getResponse_text(){
          return response_text;
  }

  /**
  *
  * @param aResponse_text - the new value for response_text
  */
  public void setResponse_text(String aResponse_text){
          response_text = aResponse_text;
  }


  /**
  *
  * @return the String value of disposition_text.
  */
  public String getDisposition_text(){
          return disposition_text;
  }

  /**
  *
  * @param aDisposition_text - the new value for disposition_text
  */
  public void setDisposition_text(String aDisposition_text){
          disposition_text = aDisposition_text;
  }


  /**
  *
  * @return the BigInteger value of response_code.
  */
  public BigInteger getResponse_code(){
          return response_code;
  }

  /**
  *
  * @param aResponse_code - the new value for response_code
  */
  public void setResponse_code(BigInteger aResponse_code){
          response_code = aResponse_code;
  }


  /**
  *
  * @return the BigInteger value of disposition_code.
  */
  public BigInteger getDisposition_code(){
          return disposition_code;
  }

  /**
  *
  * @param aDisposition_code - the new value for disposition_code
  */
  public void setDisposition_code(BigInteger aDisposition_code){
          disposition_code = aDisposition_code;
  }

}
```

Listing D.3 SendToNumbers.java

```
package com.ewingz.samples.qc3;

import java.net.URL;
import java.util.ArrayList;
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.AxisFault;

import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
import org.apache.axis.encoding.*;
import org.apache.axis.encoding.ser.*;

import org.apache.axis.AxisEngine;
import org.apache.axis.MessageContext;


public class SendToNumbers {

  public static void main(String [] args) throws Exception {

    SendToNumbers stn = new SendToNumbers();

    //grab input arguments
    String msisdn = args[0];
    String body = args[1];

    //validate input arguments
    if(msisdn.length() == 0 || body.length() == 0) {
      System.out.println("Usage: send_to_numbers msisdn body");
      System.exit(1);
    }

    // Enter your account information below
    String username      = "YOURUSERNAME";
    String password      = "yourpassword";

    try {
      String endpointURL = "http://soap.ewingz.com/SOAP";
      System.out.println("Creating SOAP object...");

      //set up SOAP object
      Service service = new Service();
      Call call = (Call) service.createCall();
      call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
      call.setOperationName(new QName("http://soap.ewingz.com/eWingz/SOAP/QC30/",
          "send_to_numbers"));
      call.setUseSOAPAction(true);

call.setSOAPActionURI("http://soap.ewingz.com/eWingz/SOAP/QC30/#send_to_numbers")
;
      call.setOperationStyle("rpc");

      //set PROP_DOMULTIREFS to FALSE. Otherwise we will send MultiRef array
references
       call.setOption(AxisEngine.PROP_DOMULTIREFS, new Boolean(false));

      //set parameter types
      stn.AddParameters(call);

      //set return type
      call.setReturnType(new
QName("http://soap.ewingz.com/eWingz/SOAP/QC30/qc30/type","send_to_numbers_result
"),
                         SendToNumberResultBean[].class);
```

```
      //declare Type Mappings
      stn.AddTypeMappings(call);

      //create item
      SendToNumbersItemBean stnib = new SendToNumbersItemBean();
      stnib.setUniqueid(new String("").getBytes());
      stnib.setMsisdn(msisdn);
      SendToNumbersItemBean[] numbers = new SendToNumbersItemBean[]{stnib};

      System.out.println("Calling send_to_numbers...");

      try {
        //call send_to_number method
        SendToNumberResultBean[] stnrb = (SendToNumberResultBean[]) call.invoke(
new Object[] { username,
                  password, new Boolean(true), "handset", "GSM0338", new
Integer(1), new Boolean(true),
                  Base64.encode(new String("QC30RC1").getBytes()),
                  "", Base64.encode(new String(body).getBytes()), "", numbers }
);

        //request succeeded
        System.out.println("Request succeeded.");
        System.out.println("\nResponse:");

        for(int i = 0; i < stnrb.length; i++) {
          //print send_to_number_result data to stdout
          System.out.println("\tuniqueid = " + new
String(stnrb[i].getUniqueid()));
          System.out.println("\tsegments = " + stnrb[i].getSegments());
          System.out.println("\tresponse_code = " + stnrb[i].getResponse_code());
          System.out.println("\tresponse_text = " + stnrb[i].getResponse_text());
          System.out.println("\tdisposition_code = " +
stnrb[i].getDisposition_code());
          System.out.println("\tdisposition_text = " +
stnrb[i].getDisposition_text());
        }

      } catch (Exception e) {
        //create fault object
        AxisFault fault = AxisFault.makeFault(e);
        QName qn = fault.getFaultCode();

        //request failed
        System.out.println("Request failed.");
        System.out.println("\nResponse:");

        //print fault information to stdout
        System.out.println("\tFault string: " + fault.getFaultString());
        System.out.println("\tFault code: " + qn.toString());
      }

    } catch (Exception e) {
       System.err.println(e.toString());
    }
  }

/**
 *
 * @param call
 */
  private void AddParameters(Call call) {
    //username
    call.addParameter("username",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //password
```

```
    call.addParameter("password",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //testmode
    call.addParameter("testmode",
          org.apache.axis.Constants.XSD_BOOLEAN,
          javax.xml.rpc.ParameterMode.IN);
    //notification
    call.addParameter("notification",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //type
    call.addParameter("type",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //class
    call.addParameter("class",
          org.apache.axis.Constants.XSD_INTEGER,
          javax.xml.rpc.ParameterMode.IN);
    //udhi
    call.addParameter("udhi",
          org.apache.axis.Constants.XSD_BOOLEAN,
          javax.xml.rpc.ParameterMode.IN);
    //originator
    call.addParameter("originator",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //header
    call.addParameter("header",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //body
    call.addParameter("body",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //footer
    call.addParameter("footer",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
    //item
    call.addParameter("array_of_item",
          new QName("http://xml.apache.org/xml-soap","SOAPStruct"),
          com.ewingz.samples.qc3.SendToNumbersItemBean[].class,
          javax.xml.rpc.ParameterMode.IN);
  }

/**
 *
 * @param call
 */
  private void AddTypeMappings(Call call) {
    //declare serializer for SOAPStruct (contains send_to_number_result)
    call.registerTypeMapping(SendToNumbersItemBean.class,
                             new QName("http://xml.apache.org/xml-
soap","SOAPStruct"),
                             BeanSerializerFactory.class,
                             BeanDeserializerFactory.class);

   //declare serializer for Base64
   call.registerTypeMapping(Base64.class,
                             new QName("http://xml.apache.org/xml-soap","Base64"),
                             Base64SerializerFactory.class,
                             Base64DeserializerFactory.class);

   //declare serializer for SendToNumberResultBean
   call.registerTypeMapping(SendToNumberResultBean.class,
                            new QName("http://xml.apache.org/xml-
soap","SOAPStruct"),
```

```
                            BeanSerializerFactory.class,
                            BeanDeserializerFactory.class);

    }
}
```

Listing D.4 SendToNumbersItemBean.java

```
package com.ewingz.samples.qc3;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author $Author: $
 * @version 1.0$Revision: 15 $ $Date: 10/01/02 11:44a $
 */

public class SendToNumbersItemBean {

  public SendToNumbersItemBean() {
  }

  private byte[]  uniqueid;
  private String  msisdn;


  /**
  *
  * @return the byte[] value of uniqueid.
  */
  public byte[] getUniqueid(){
          return uniqueid;
  }

  /**
  *
  * @param aUniqueid - the new value for uniqueid
  */
  public void setUniqueid(byte[] aUniqueid){
          uniqueid = aUniqueid;
  }


  /**
  *
  * @return the String value of msisdn.
  */
  public String getMsisdn(){
          return msisdn;
  }

  /**
  *
  * @param aMsisdn - the new value for msisdn
  */
  public void setMsisdn(String aMsisdn){
          msisdn = aMsisdn;
  }


}
```

Listing C.5 History.java

```
package com.ewingz.samples.qc3;

import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.AxisFault;
import org.apache.axis.encoding.*;
import org.apache.axis.encoding.ser.*;

import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
import java.net.URL;
import java.text.SimpleDateFormat;

public class History {

  public static void main(String [] args) throws Exception {

    History h = new History();

    //grab input arguments
    String uniqueid = args[0];

    //validate input arguments
    if(uniqueid.length() == 0) {
      System.out.println("Usage: history uniqueid");
      System.exit(1);
    }

    // Enter your account information below
    String username      = "YOURUSERNAME";
    String password      = "yourpassword";

    try {
      String endpointURL = "http://soap.ewingz.com/SOAP";

      System.out.println("Creating SOAP object...");

      //set up SOAP object
      Service service = new Service();
      Call call = (Call) service.createCall();
      call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
      call.setOperationName(new QName("http://soap.ewingz.com/eWingz/SOAP/QC30/",
          "history"));
      call.setUseSOAPAction(true);
      call.setSOAPActionURI("http://soap.ewingz.com/eWingz/SOAP/QC30/#history");
      call.setOperationStyle("rpc");

      //set parameter types
      h.AddParameters(call);

      //set return type
      call.setReturnType(new
QName("http://soap.ewingz.com/eWingz/SOAP/QC30/qc30/type","history_result"),
                         SMSStatusEntryBean[][].class);

      //declare Type Mappings
      h.AddTypeMappings(call);

      System.out.println("Calling history...");

      try {
        //call status method
        SMSStatusEntryBean[][] srb = (SMSStatusEntryBean[][]) call.invoke( new
Object[] { username,
                  password, Base64.encode(new String(uniqueid).getBytes()) } );
```

```
        //create date formatter
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String t;

        //request succeeded
        System.out.println("Request succeeded.");
        System.out.println("\nResponse:");
        //loop and display
        for(int i=0;i<srb.length;i++) {
          for(int j=0;j<srb[i].length;j++) {
          //get date and format
          t = sdf.format(srb[i][j].getDate());

          System.out.println("\tresponse_text = " +
srb[i][j].getResponse_text());
          System.out.println("\tdate = " + t);
          System.out.println("\tdisposition_text = " +
srb[i][j].getDisposition_text());
          System.out.println("\tresponse_code = " +
srb[i][j].getResponse_code());
          System.out.println("\ttime = " + srb[i][j].getTime());
          System.out.println("\tdisposition_code = " +
srb[i][j].getDisposition_code());
          }
        }
      } catch (Exception e) {
        //create fault object
        AxisFault fault = AxisFault.makeFault(e);
        QName qn = fault.getFaultCode();

        //request failed
        System.out.println("Request failed.");
        System.out.println("\nResponse:");

        //print fault information to stdout
        System.out.println("\tFault string: " + fault.getFaultString());
        System.out.println("\tFault code: " + qn.toString());
      }
    } catch (Exception e) {
       System.err.println(e.toString());
    }
  }

/**
 *
 * @param call
 */
  private void AddParameters(Call call) {

    //username
    call.addParameter("username",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //password
    call.addParameter("password",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //uniqueid
    call.addParameter("uniqueid",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
  }

/**
 *
 * @param call
 */
```

```
  private void AddTypeMappings(Call call) {
    //declare serializer for SOAPStruct (contains status_result)
    call.registerTypeMapping(SMSStatusEntryBean.class,
                             new QName("http://xml.apache.org/xml-
soap","SOAPStruct"),
                             BeanSerializerFactory.class,
                             BeanDeserializerFactory.class);
   //declare serializer for Base64
   call.registerTypeMapping(Base64.class,
                             new QName("http://xml.apache.org/xml-soap","Base64"),
                             Base64SerializerFactory.class,
                             Base64DeserializerFactory.class);
  }
}
```

Listing C.6 Status.java

```
package com.ewingz.samples.qc3;

import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.AxisFault;
import org.apache.axis.encoding.*;
import org.apache.axis.encoding.ser.*;

import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
import java.net.URL;
import java.text.SimpleDateFormat;

public class Status {

  public static void main(String [] args) throws Exception {

    Status s = new Status();

    //grab input arguments
    String uniqueid = args[0];

    //validate input arguments
    if(uniqueid.length() == 0) {
      System.out.println("Usage: status uniqueid");
      System.exit(1);
    }

    // Enter your account information below
    String username      = "YOURUSERNAME";
    String password      = "yourpassword";

    try {
      String endpointURL = "http://soap.ewingz.com/SOAP";

      System.out.println("Creating SOAP object...");

      //set up SOAP object
      Service service = new Service();
      Call call = (Call) service.createCall();
      call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
      call.setOperationName(new QName("http://soap.ewingz.com/eWingz/SOAP/QC30/",
          "status"));
      call.setUseSOAPAction(true);
      call.setSOAPActionURI("http://soap.ewingz.com/eWingz/SOAP/QC30/#status");
      call.setOperationStyle("rpc");

      //set parameter types
      s.AddParameters(call);
```

```
      //set return type
      call.setReturnType(new
QName("http://soap.ewingz.com/eWingz/SOAP/QC30/qc30/type","status_result"),
                         SMSStatusEntryBean[].class);

      //declare Type Mappings
      s.AddTypeMappings(call);

      System.out.println("Calling status...");

      try {
        //call status method
        SMSStatusEntryBean[] srb = (SMSStatusEntryBean[]) call.invoke( new
Object[] { username,
                  password, Base64.encode(new String(uniqueid).getBytes()) } );

        //create date formatter
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String t;

        //request succeeded
        System.out.println("Request succeeded.");
        System.out.println("\nResponse:");
        //loop and display
        for(int i=0;i<srb.length;i++) {
          //get date and format
          t = sdf.format(srb[i].getDate());

          System.out.println("\tresponse_text = " + srb[i].getResponse_text());
          System.out.println("\tdate = " + t);
          System.out.println("\tdisposition_text = " +
srb[i].getDisposition_text());
          System.out.println("\tresponse_code = " + srb[i].getResponse_code());
          System.out.println("\ttime = " + srb[i].getTime());
          System.out.println("\tdisposition_code = " +
srb[i].getDisposition_code());
        }
      } catch (Exception e) {
        //create fault object
        AxisFault fault = AxisFault.makeFault(e);
        QName qn = fault.getFaultCode();

        //request failed
        System.out.println("Request failed.");
        System.out.println("\nResponse:");

        //print fault information to stdout
        System.out.println("\tFault string: " + fault.getFaultString());
        System.out.println("\tFault code: " + qn.toString());
      }
    } catch (Exception e) {
       System.err.println(e.toString());
    }
  }

/**
 *
 * @param call
 */
  private void AddParameters(Call call) {

    //username
    call.addParameter("username",
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //password
    call.addParameter("password",
```

```
          org.apache.axis.Constants.XSD_STRING,
          javax.xml.rpc.ParameterMode.IN);
    //uniqueid
    call.addParameter("uniqueid",
          org.apache.axis.Constants.XSD_BASE64,
          javax.xml.rpc.ParameterMode.IN);
  }

/**
 *
 * @param call
 */
  private void AddTypeMappings(Call call) {
    //declare serializer for SOAPStruct (contains status_result)
    call.registerTypeMapping(SMSStatusEntryBean.class,
                             new QName("http://xml.apache.org/xml-
soap","SOAPStruct"),
                             BeanSerializerFactory.class,
                             BeanDeserializerFactory.class);
   //declare serializer for Base64
   call.registerTypeMapping(Base64.class,
                             new QName("http://xml.apache.org/xml-soap","Base64"),
                             Base64SerializerFactory.class,
                             Base64DeserializerFactory.class);
  }
}
```

Listing C.7 SMSStatusEntryBean.java

```
package com.ewingz.samples.qc3;

import java.util.Date;
import java.math.BigInteger;
import org.apache.axis.types.Time;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author $Author: $
 * @version 1.0$Revision: 15 $ $Date: 10/01/02 11:44a $
 */

public class SMSStatusEntryBean {


  private String response_text;
  private Date date;
  private String disposition_text;
  private BigInteger response_code;
  private Time time;
  private BigInteger disposition_code;

             /**
             *
             * @return the String value of response_text.
             */
             public String getResponse_text(){
               return response_text;
             }

             /**
             *
             * @param aResponse_text - the new value for response_text
             */
             public void setResponse_text(String aResponse_text){
               response_text = aResponse_text;
             }


             /**
             *
             * @return the Date value of date.
             */
             public Date getDate(){
               return date;
             }

             /**
             *
             * @param aDate - the new value for date
             */
             public void setDate(Date aDate){
               date = aDate;
             }


             /**
             *
             * @return the String value of disposition_text.
             */
             public String getDisposition_text(){
               return disposition text;
             }
```

```
            /**
            *
            * @param aDisposition_text - the new value for disposition_text
            */
            public void setDisposition_text(String aDisposition_text){
              disposition_text = aDisposition_text;
            }


            /**
            *
            * @return the BigInteger value of response_code.
            */
            public BigInteger getResponse_code(){
              return response_code;
            }

            /**
            *
            * @param aResponse_code - the new value for response_code
            */
            public void setResponse_code(BigInteger aResponse_code){
              response_code = aResponse_code;
            }


            /**
            *
            * @return the Time value of time.
            */
            public Time getTime(){
              return time;
            }

            /**
            *
            * @param aTime - the new value for time
            */
            public void setTime(Time aTime){
              time = aTime;
            }


            /**
            *
            * @return the BigInteger value of disposition_code.
            */
            public BigInteger getDisposition_code(){
              return disposition_code;
            }

            /**
            *
            * @param aDisposition_code - the new value for disposition_code
            */
            public void setDisposition_code(BigInteger aDisposition_code){
              disposition_code = aDisposition_code;
            }




  public SMSStatusEntryBean() {
```

```
  }

  public SMSStatusEntryBean(Object[] obj)
  {
    System.out.println("constructor");
  }

  public SMSStatusEntryBean(Object obj)
  {
    System.out.println("constructor");
  }

}
```

14 Appendix E: Document change log

Date	Section	Description of change
20030407	Table 5-1	Corrected descriptions of `header`, `body`, and `footer`.
20030407	Section 6.8	Corrected information about multipart binary messages.
20030319	Appendix D	Added Java samples in new appendix.
20030319	Appendix C	Added Perl samples in new appendix.
20021029	Section 9	Added section describing 2-way messaging.
20021029	Table 2-7	Updated table for additional error messages.
20021029	Section 6.8	Added notes RE Nokia Smart Messaging and MCC/MCN octets.
20021029	Section 6.8	Added information about multipart binaries.
20021029	Section 6.7	Added note RE errors and RTTTL detection.
20021029	Section 2	Added reference to Nokia Smart Messaging info
20020327	all	Updated all material from 1.0 (preview) version of document.