

Bookit iSMS[®]

Enterprise Server Interface Specification

Bookit Ltd

Change History

Revision	Date	Handled by	Comments
1.0	Dec 11th 2002	Jorma Korkiakoski	Phase 1 requirements only
1.3	July 29th 2004	Jorma Korkiakoski	New protocol request: Close
1.5	May 12th 2006	Jouni Takala	Update, HTML Push and DDM Interface spec combined in the same document
1.6	May 24 th 2006	Jouni Takala	References added
1.7	Nov 2 nd 2006	Jouni Takala	New ES features added with additional status codes and reply url
1.8	Nov 6 th 2006	Jouni Takala	Response element * added as special case, warning level field added to Send Reply XML
1.9	Nov 24 th 2006	Jouni Takala	CloseResult message removed because statusResult message is used in reply to closeRequest message
1.9.1	Mar 5 th 2009	Matti Ärmänen	Added reply_url to java example and clarifications to examples
1.9.2	May 26 th 2009	Guido Kohl	Minor format changes

Bookit Ltd

Table of Contents

3.1	Message Types.....	6
3.2	Operating Modes.....	6
3.3	Protocol.....	6
3.3.1	Sender Identification.....	6
3.3.1.1	Sender Name Examples:.....	7
3.3.1.2	Encoding Example.....	7
3.4	Protocol Interfaces	8
3.4.1	XML-Interface.....	8
3.4.1.1	Send Request.....	8
3.4.1.2	Send Reply.....	9
3.4.1.3	Status Request.....	9
3.4.1.4	Status Reply.....	9
3.4.1.5	Close Request.....	10
3.4.1.6	Close Reply.....	10
3.4.1.7	Send and Status Codes.....	10
3.4.2	HTML Push Interface.....	11
3.4.2.1	Parameters.....	12
3.4.2.2	Push URL.....	12
3.5	Java Client Application Programming Interface (API)	13
3.5.1	Class DDM.....	13
3.5.1.1	Methods	13
3.5.2	Class TestDDM	15
3.5.3	Class DDMAuth.....	17
3.5.3.1	Methods	17
3.5.4	Class Dialogue	17
3.5.4.1	Methods	18
3.5.5	Class DDMStatus	20
3.5.5.1	Methods	20
3.6	Document Type Definitions (DTD).....	21
3.6.1	Send Request	21
3.6.2	Send Reply.....	21
3.6.3	Status Request.....	21
3.6.4	Status Reply.....	21
3.6.5	Close Request.....	22

Bookit Ltd

1. REFERENCES

- [1] **BookIT iSMS[®] Architecture**
- [2] **BookIT iSMS[®] Functional Description**
- [3] **BookIT iSMS[®] Implementation Requirements**
- [4] **BookIT iSMS[®] Gateway Interface Specification**
- [5] **BookIT iSMS[®] Charging Interface Specification**
- [6] **BookIT iSMS[®] Administration Guide**
- [7] **BookIT iSMS[®] Installation Guide**
- [8] **BookIT iSMS[®] Hardware Configurations**
- [9] **BookIT iSMS[®] Software Components and Versions**
- [10] **BookIT iSMS[®] Monitoring**

Bookit Ltd

2. OVERVIEW

Bookit iSMS[®] platform architecture is described in reference [1]. The components (network elements) are presented in Figure 1. The platform can be divided to three functional elements; Enterprise Server, Service Center and Service Gateway. The Enterprise Server is the interface to the Enterprise applications whereas the Service Gateway provides the access to the end users. The Service Center is the actual engine to control the iSMS[®] functions. It implements the patented Dynamic Dialogue Matrix (DDM) technology, the core or the iSMS[®] system.

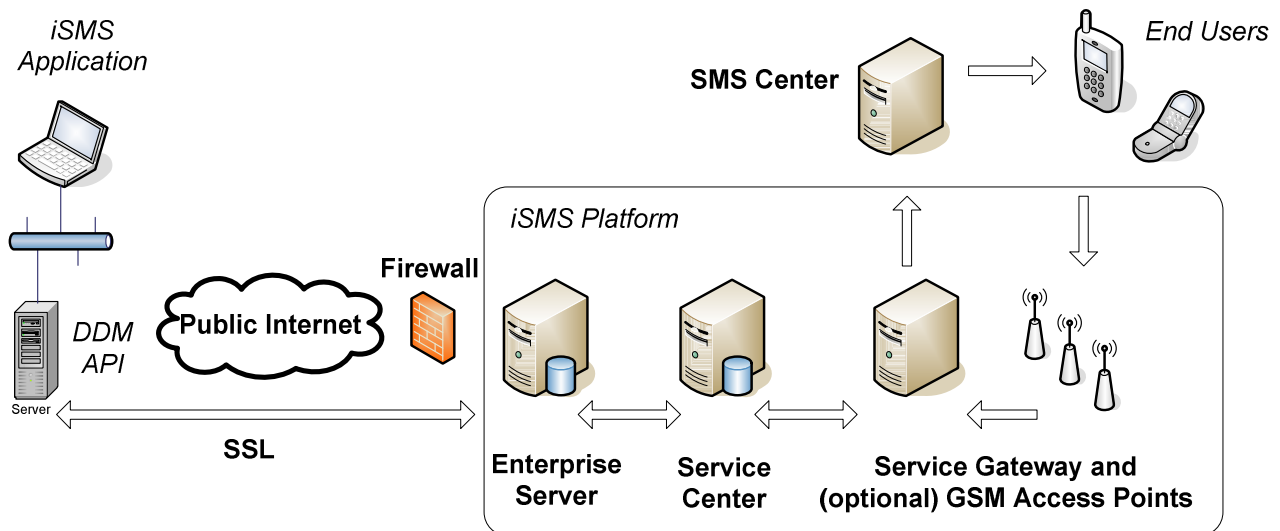


Figure 1. iSMS[®] Components

The iSMS[®] architecture consists of the iSMS[®] service, iSMS[®] clients and end users (or end-devices) receiving dialogues and replying to them.

This document is the interface specification of BookIT iSMS[®] Enterprise Server, the access point for Enterprise iSMS[®] applications. Applications (iSMS[®] clients) can send queries to end-users whose replies can be reliably and correctly matched to the questions. iSMS[®] platform offers *two-way communication with a reliable return channel*. Sent two-way messages are called *dialogues*. Moreover, end users can reply to queries in any order, resulting in greater convenience.

The iSMS[®] concept is *communications layer independent*. In the first phase SMS is supported.

3. ENTERPRISE SERVER INTERFACE

3.1 Message Types

There are two message types, Dialogue and Notification. Dialogues are two-way messages. They have sender, destination, message, one or more reply options and validity period. Each dialogue is uniquely identified.

Notifications are one-way messages, to which no reply is expected.

Note! All the fields of the xml-messages are URL-encoded. Eg. "+358" encodes to "%2B358" and "1 + 2 = 3" encodes to "1+%2B+2+%3D+3".

3.2 Operating Modes

For each sender_id, the Service Center can be configured either to forward the answer via Enterprise Server to the application or just reply to the status polls from the application. The application designer can choose the best way to be used.

3.3 Protocol

The DDM protocol is *stateless*: each request is executed independently, without any knowledge of the requests that came before it. Hence, each request must carry enough information to be complete.

Currently, there are three methods, **send**, **status** and **close**.

The **send** method is for sending new dialogues (two-way messages) or notifications (one-way messages). The DDM protocol at the messaging level is asynchronous. The sending of a dialogue (or notification) is synchronous, but the return code tells only that the dialogue (notification) has been successfully received from the DDM client. The actual sending of the dialogue may not have happened yet.

The **status** method can be used to retrieve the status of a sent dialogue. That is, the client must poll the status of a dialogue to get its status (has the client answered to it correctly) if DDM is not configured to send Push-message as the dialogue is answered. The dialogue expires at the end of the validity period. Only the sender of the dialogue can check its status.

Note: the status method cannot be used to poll the status of a sent notification.

Sometimes the sender needs to close an already sent dialogue for various reasons. In order to do this DDM offers the **close** request. It has the same parameters and restrictions as the *status* request.

3.3.1 Sender Identification

Each protocol request carries sender information, which consists of a plaintext *sender name* field and an encrypted *authentication* field.

The *sender name* field syntax is

```
sender_name = sender_organization[:application]
```

The sender organization part cannot contain semicolon (':') (or other invalid XML characters). The same applies to the application part. Last, application part is not mandatory.

The *authentication* field is used to authenticate the sender. The authentication field is formed in the following way:

```
auth_field = MD5(sender_name + shared_secret)
```

The plus sign above is the string catenation operator. For example, the sender name is "foo" and the shared secret is "bar". The authentication field is calculated by taking the MD5 hash of the string "foobar".

The shared secret is allocated by the iSMS[®] service administration on *per sending organization basis*. The sending organization is typically a company, but can also be a subdivision, or any other suitable entity.

Binding the shared secret to the sender organization – as opposed to the whole sender name field – offers easier maintenance while not sacrificing security. This stems from the fact that organizations using DDM can add new sending applications (bookkeeping and logging easier) using the same authentication credentials. No extra administrative overhead is incurred at the DDM service or sender side. However, nothing prevents from using empty application name part in the sender name and creating a distinct sending organization per application. Or, the same sender can be used always, but the application part is unique identifier on the sender systems.

3.3.1.1 Sender Name Examples:

```
com.company.division1.marketing:CRM
com.company.division1.marketing:test
com.company.support:sales
com.company.support:app1
```

3.3.1.2 Encoding Example

These examples calculate the MD5 authentication field based on the sender name and the shared secret for the sender organization, "com.company.support":

Sender Name	com.company.support:app1
Shared Secret	SharedSecret
Authentication Field	002B47A6A989F5FA1AF448525DB76D7E
Sender Name	com.company.support:app2
Shared Secret	SharedSecret
Authentication Field	D362AA267D0B8E843133D50249E6C2DB

Table 1. MD5 encoding example

Bookit Ltd

3.4 Protocol Interfaces

Two interfaces are offered: An unrestricted and well-known XML interface (XML-POST) is suitable for a plethora of heterogeneous applications, and a HTML interface (HTML-GET) is supported to push responses back to applications when polling is not practical. In addition, a convenient Java API is supplied for easy linkage and use for Java-based business applications. It implements the XML interface.

3.4.1 XML-Interface

Note! The field contents shall be url-encoded with the character set indicated in the xml-header.

3.4.1.1 Send Request

```
<?xml version="1.0" encoding="UTF-8" ?>
<sendRequest>
  <version>1.0</version>
  <sender>
    <id>Sender Id</id>
    <enc>MD5-encrypted string using the shared secret</enc>
  </sender>
  <coreMsg>CoreMsg</coreMsg>
  <recipient>Recipient ID</recipient>
  <!-- Expiration options, (value in minutes), if missing, 1440 used →
  <expiry>
    <relative>1440</relative>
  </expiry>
  <!-- preformatted bit, optional:
    1 = coreMsg is the preformatted message
    which is sent to the recipient.
    0 = the message will be formatted by DDM, see
    chapter 3.5.2.
    Note: if the tag is missing the bit is set to 0. →
  <preformatted>0_or_1</preformatted>
  <!-- if options is missing, treated as one way notification →
  <options>
    <option>
      <response>RESPONSE</response> <!-- e.g "K","Y" →
      <description>DESC</description>
    </option>
    <!-- more option tags... →
  </options>
  <reply_url>URL for sending answers</reply_url>
</sendRequest>
```

Version is 1.0.

Special case: If there is only one response-element and it's value is *, all answers are delivered to the application independent of the actual response. This allows the use of freeform answers.

Bookit Ltd

3.4.1.2 Send Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<sendResult>
  <id>ID</id>
  <status>
    <code>statuscode</code>
    <message>status message</message>
  </status>
  <!-- Optional field if ES load limiter is over warning level →
  <warn_level>1</warn_level>
</sendResult>
```

Note ! The maximum length of the id-field is **20 digits**.

3.4.1.3 Status Request

The status request has only two logical parameters: the dialogue id and the sender (identification). A sender can only request the status of a dialogue that he has sent.

```
<?xml version="1.0" encoding="UTF-8"?>
<statusRequest>
  <id>ID</id>
  <sender>
    <id>Sender Id</id>
    <enc>MD5-Encrypted stuff w\ the shared secret</enc>
  </sender>
</statusRequest>
```

3.4.1.4 Status Reply

The request to the status request includes the dialogue id, status code and message, and, optionally, the client response and response number (numbering starts from 1) in the node answer. The node is sent, obviously, only when the recipient has replied to the dialogue denoted by the `statusRequest/id` node with a valid option and within the expiration time.

```
<?xml version="1.0" encoding="UTF-8"?>
<statusResult>
  <id>ID</id>
  <status>
    <code>statuscode</code>
    <message>status message</message>
  </status>
  <answer> <!-- optional →
    <response>RESPONSE</response>
    <no>response number</no>
  </answer>
  <!-- Optional field if ES load limiter is over warning level -->
  <warn_level>1</warn_level>
</statusResult>
```

Bookit Ltd

Note! The status code is not answered although answer is received if the matcher allowing multiple matches is used.

3.4.1.5 Close Request

The close request has only two logical parameters: the dialogue id and the sender (identification). A sender can only close a dialogue that he has sent.

```
<?xml version="1.0" encoding="UTF-8"?>
<closeRequest>
  <id>ID</id>
  <sender>
    <id>Sender Id</id>
    <enc>MD5-Encrypted stuff w\ the shared secret</enc>
  </sender>
</closeRequest>
```

3.4.1.6 Close Reply

The close reply message is statusResult described above with status 5.

3.4.1.7 Send and Status Codes

When sending a dialogue or notification, the DDM will return two codes: the dialogue id and the status code. Additionally, a status message field is also returned as a string. When sending a notification, the DDM will return 0 as the dialogue id, if the sending was successful. There is no need to check for the status code (which is `DDM.ONGOING`, see the table below). If the notification sending failed, the dialogue id < 0 (and the status id = dialogue id). The reason DDM sets dialogue id as 0 when the notification sending was successful is the fact that there is (currently) no need to check the status of a sent notification because the customer is not allowed to reply to it.

When sending a dialogue, the DDM will return a positive integer (> 0) as the dialogue id if the sending was successful. The status code will be `DDM.ONGOING`, as when sending a notification. If the dialogue sending failed, the dialogue id < 0 (and the status id = dialogue id). Exception is the load control functionality of the Enterprise Server. It will return status -190 if the warning level set to the sender is exceeded. The message is delivered to the recipient in spite of the negative status code.

The status codes (integers) and the static member fields ("constants") in the class DDM in the java Client Interface are listed in Table 2.

Status Code	Constant	Comment
1	<code>DDM.ONGOING</code>	The dialogue or notification has been successfully sent.
2	<code>DDM.ANSWERED</code>	The dialogue has been answered.
3	<code>DDM.EXPIRED</code>	The dialogue was not answered during the validity period.

4	DDM.PUSHED	The dialogue was successfully pushed to the sender.
5	DDM.CLOSED	The dialogue was closed by the sender (Only ongoing dialogues can be closed).
-2	DDM.INVALID_DIALOGUE_ID	The status query failed – the dialogue id and the sender id did not match.
-3	DDM.DUPLICATE_OPTIONS	The dialogue contained duplicate options. (The options are case-insensitive.)
-4	DDM.AUTHENTICATION_FAILED	The sender authentication failed.
-5	DDM.NO_HOST_FOUND	The DDM host was not found.
-6	DDM.MESSAGE_TOO_LONG	The message is too long (> 459 chars = 3 concatenates SMS messages).
-7	DDM.INVALID_PROTOCOL	The XML protocol is invalid. The Java Client returns this if the DDM Service reply is invalid, i.e. not well-formed. Similarly, the DDM Service will return this error code if the client XML is not valid.
-8	DDM.INVALID_SENDER	The sender is too long (>255) or empty.
-9	DDM.MATRIX_FULL	There were not enough resources to allocate a dialogue. Existing dialogue has to expire or it must be closed.
-10	DDM.INVALID_ARGUMENTS	One or more of the arguments supplied was invalid. Typical example: empty option (the field <options><option><response>).
-100	DDM.INTERNAL_ERROR	Can happen for various reasons, e.g. dialogue id allocation failed (due to a database error). Can indicate lack of resources. Check the status message for a description of the actual error.

Additional status codes (integers) from the **iSMS® Enterprise Server**:

<i>Status Code</i>	<i>Constant</i>	<i>Comment</i>
-210		Error in servlet
-220		Error in load control filter
-221		Error in validation filter
-230		Error connecting to database
-231		SQL exception
-240		Global load limit exceeded, message blocked
-241		Sender load limit exceeded, message blocked

Table 2. Status Codes

3.4.2 HTML Push Interface

The standard HTML Push module in DDM 2.0 uses a well-defined, easy-to-understand HTML GET method for sending dialogue replies to the dialogue sender's backend systems. By default, SSL is used to encrypt the HTTP traffic.

Bookit Ltd

3.4.2.1 Parameters

<i>Parameter name</i>	<i>Comment</i>
Recipient	The mobile phone number that the dialogue is sent to. The number is in the internationalized format (e.g. +358501234567).
Sender	The sender id of the dialogue.
Enc	The encoded hash based on the sender id (the parameter sender above) and the shared secret associated with the sender organisation.
dialogue_id	The matched dialogue id (> 0).
Status	The status of the dialogue. 1, if OK, see the interface specification for more detail.
Reply	The user reply.
reply_no	The matched reply option number, starting from 1.
reply_time	Timestamp denoting time the user reply was received and processed in the DDM service. The format is the default java date and timestamp format, like Fri Mar 28 10:30:10 EEST 2005

Table 3. HTML Push Parameters

All the parameters are URL encoded.

3.4.2.2 Push URL

In order to deploy HTML Push, the (base) push URL is needed from the customer. All the parameters listed in the previous chapter are added to the end of this URL.

Example:

The base URL is

```
https://my.site.com/push/here
```

Upon a successful match the DDM service appends the base URL with the following parameters:

- sender=foo
- enc= B1ADCEA16453D1ECA9C20F32488932AB
- recipient=%2B358501234567

Bookit Ltd

- reply=OK
- reply_no=2
- status=1
- reply_time=Fri Mar 28 10:30:10 CST 2003
- dialogue_id=689011

The resulting URL is:

https://my.site.com/push/here?sender=foo&enc=B1ADCEA16453D1ECA9C20F32488932AB&recipient=%2B358501234567&reply=OK&reply_no=2&status=1&reply_time=Fri+Mar+28+10%3A30%3A10+CST+2003&dialogue_id=689011

3.5 Java Client Application Programming Interface (API)

For easy linking into business applications, a Java interface is supplied. It uses XML over HTTP(S) underneath. J2SE JDK 1.4 is used. The Java interface is supplied as a jar file (`ddm-client-version.jar`). The current version is `ddm-client-2.0.jar`.

3.5.1 Class DDM

The class name is DDM. It has three basic methods, *send*, *status* and *close*.

3.5.1.1 Methods

3.5.1.1.1 Constructor

In the constructor the sender gives the URL of the DDM Service Host.

```
public DDM(String DDMHost);
```

The `DDMHost` parameter format is the following:

```
DDMHost ::= protocol "://" host:port
protocol ::= "http" | "https"
```

Examples:

```
DDM ddm1 = new DDM("http://ddm.bookit.net");
DDM ddm2 = new DDM("https://ddm.bookit.net:443");
DDM ddm3 = new DDM("http://localhost:8080");
```

3.5.1.1.2 Send

The `send` method gets as parameters an instance of `DDMAuth`, which is used to authenticate the sender at the DDM service. The second parameter is an instance of `Dialogue`, initialized by the sender, which contains all the attributes of a dialogue to be sent.

```
long send(DDMAuth auth, Dialogue dialogue);
```

If the sending was successful, the method returns the allocated dialogue id, which is a positive long integer. However, if a dialogue with no options was sent (it is valid), a successful invocation returns 0. Hence, you cannot query the status of such a dialogue.

Example:

```
DDM ddm = new DDM("http://ddm.bookit.net");
Dialogue dialogue = new Dialogue();
dialogue.setRecipient("+358471234567");
dialogue.setCoreMsg("A meeting proposal tomorrow at 11pm.");
dialogue.setOption("Y", "Yes"); // Option no 1.
dialogue.setOption("N", "No"); // Option no 2.
dialogue.setExpiryTime(2880); // minutes
dialogue.setReplyUrl("http://www.company.com/receive?source=isms");
// get replies to this url

DDMAuth auth = new DDMAuth("net.bookit.rd:test_app", getSharedSecret());
long dialogueId = ddm.send(auth, dialogue);
if (dialogueId > 0) {
    sentOk(dialogueId);
} else {
    throw new IOException ("Problem: " + dialogue.getStatus());
}
```

3.5.1.1.3 Status

```
DDMStatus status(DDMAuth auth, long dialogueId)
```

Example:

```
/* Example skeleton code for retrieving the dialogue status.
   For DDM version 2.0. */

import net.bookit.ddm.*;

DDM ddm = new DDM("http://ddm.bookit.net:443");
/* Retrieve the dialogue id from, say, a database. */
long dialogueId = retrieveMyDialogueId();
/* Ditto for the shared secret. */
String sharedSecret = getSharedSecretFromAVerySecretPlace();
String senderId = "net.bookit.ddmdemo:example";
DDMAuth auth = new DDMAuth(senderId, sharedSecret);
DDMStatus s = ddm.status(auth, dialogueId);

if (s.getStatus() == DDM.ANSWERED) {
    // Ok, hand over the result to your own method.
    myProcessTheAnswer(s.getDialogueId(), s.getRecipient(), s.getAnswer(),
    s.getAnswerNo());
} else if (s.getStatus() == DDM.EXPIRED) {
    // Ok, the recipient did not answer.
} else if (s.getStatus() == DDM.INVALID_DIALOGUE_ID) {
    // The {sender, dialogueId} tuple does not match.
    // Very suspicious.
```

Bookit Ltd

```

        panic("GOT_CAUGHT_WHILE_SNOOPING");
    } else if (s.getStatus() == DDM ONGOING) {
        // Wait and try again later.
    } else if (s.getStatus() == DDM INTERNAL_ERROR) {
        throw new IOException("error in calling DDM::status: " +
            s.getStatus() + ": " + s.getStatusMessage());
    }
}
/* End of the status example. */

```

3.5.1.1.4 Close

```
DDMStatus close(DDMAuth auth, long dialogueId);
```

Example:

```

import net.bookit.ddm.*;

DDM ddm = new DDM("http://ddm.bookit.net:443");
/* Retrieve the dialogue id from, say, a database. */
long dialogueId = retrieveMyDialogueIdTiBeClosed();
/* Ditto for the shared secret. */
String sharedSecret = getSharedSecretFromAVerySecretPlace();
String senderId = "net.bookit.ddmdemo:example";
DDMAuth auth = new DDMAuth(senderId, sharedSecret);
DDMStatus s = ddm.close(auth, dialogueId);

if (s.getStatus() == DDM.CLOSED) {
    // Ok, hand over the result to your own method.
    System.out.println("OK, dialogue " + dialogueId + " closed.");
}

```

3.5.2 Class TestDDM

This class is a subclass of `DDM`. It provides a convenient mechanism for testing the DDM process from the application's point of view. It has exactly the same interface to the DDM (which is obvious). It differs, however, from the super class by *not communicating with the DDM Service*. It merely simulates the super class behaviour internally.

The behaviour of the `TestDDM` depends on the `DDMHost` parameter given in the constructor. Whereas the client would supply the real URL of the DDM service when using the super class, for `TestDDM` the parameter denotes the expected behaviour. Note that when we are telling that a method returns a value it means the following. The `send` method returns a long value, which is either the allocated dialogue id, or an error status code. The client can also retrieve the dialogue status using the method `getStatus()`. The `status` method returns an instance of `DDMStatus` that is always guaranteed to be non null. Its status code can be retrieved using the method `getStatus()` (note that when you are sending a new dialogue, you can use the expression `dialogue.getStatus().getStatus()` – where `dialogue` is an instance of `Dialogue` – to retrieve the dialogue status code, which is a long integer). In the section below when we tell that the status method returns X we mean that the `DDMStatus.getStatus()` method returns X.

The constructor parameter values are:

`TestDDM.OK`. The methods always succeed.

- **send:** a new dialogue id is allocated. The result status is `DDM.ONGOING`.
- **status:** returns `DDM.ONGOING`.

`TestDDM.ONGOING`.

- **send:** always fails and returns a random error (see `TestDDM.FAIL`).
- **status:** returns `DDM.ONGOING`.

`TestDDM.ANSWERED`.

- **send:** always fails and returns a random error (see `TestDDM.FAIL`).
- **returns** `DDM.ANSWERED`. The fields `answer` and `answer number` are set (use `getAnswer()` and `getAnswerNo()` methods to retrieve them).

`TestDDM.EXPIRED`.

- **send:** always fails and returns a random error (see `TestDDM.FAIL`).
- **status:** returns `DDM.EXPIRED`.

`TestDDM.FAIL`.

- **send:** always fails and returns a randomly allocated status error code from the set of { `DDM.NO_HOST_FOUND`, `DDM.INTERNAL_ERROR`, `DDM.DUPLICATE_OPTIONS`, `DDM.INVALID_PROTOCOL` }
- **status:** always fails and returns a randomly allocated status error code from the set of { `DDM.NO_HOST_FOUND`, `DDM.INTERNAL_ERROR`, `DDM.INVALID_DIALOGUE_ID`, `DDM.DUPLICATE_OPTIONS`, `DDM.INVALID_PROTOCOL` }

`TestDDM.INVALID_PROTOCOL`.

- **send:** returns `DDM.INVALID_PROTOCOL`.
- **status:** returns `DDM.INVALID_PROTOCOL`.

`TestDDM.INTERNAL_ERROR`.

- **send:** returns `DDM.INTERNAL_ERROR`.
- **status:** returns `DDM.INTERNAL_ERROR`.

`TestDDM.DUPLICATE_OPTIONS`.

- **send:** returns `DDM.DUPLICATE_OPTIONS`. This status error code is also returned when there are duplicate options in the dialogue parameter (e.g. "K" and "k" qualify as the options are case-insensitive).
- **status:** see `TestDDM.FAIL` (status method cannot have duplicate options).

`TestDDM.MESSAGE_TOO_LONG`.

Bookit Ltd

- **send:** returns `DDM.MESSAGE_TOO_LONG`. This status error code is also returned when the full message length is greater than the `Dialogue.MAX_MESSAGE_LENGTH` (459 at the moment). The full message length is

$$\text{len}(\text{coreMsg}) + 1 + \sum (\text{len}(\text{option}_i) + \text{len}(\text{description}_i) + 3)$$

for messages which are not preformatted. For preformatted messages the message length is simply

- $$\text{len}(\text{coreMsg})$$
- **status:** see `TestDDM.FAIL`.

`TestDDM.NO_HOST_FOUND`.

- **send:** returns `DDM.NO_HOST_FOUND`.
- **status:** returns `DDM.NO_HOST_FOUND`.

3.5.3 Class DDMAuth

3.5.3.1 Methods

3.5.3.1.1 Constructor

```
Public DDMAuth(String senderId, String sharedSecret);
```

The constructor initializes the instance, which is given as a parameter to the *send* or *status* methods in the class DDM.

3.5.3.1.2 GetSender

```
Public String getSender();
```

3.5.3.1.3 GetSenderEnc

```
Public String getSenderEnc();
```

The method returns the shared secret.

3.5.3.1.4 ToString

```
Public String toString();
```

Return the MD5-encrypted sender identification.

3.5.4 Class Dialogue

The class `Dialogue` contains all the dialogue-related data and methods.

Bookit Ltd

3.5.4.1 Methods

3.5.4.1.1 Constructor

```
public Dialogue();
```

3.5.4.1.2 SetSender

Note, `DDM.send()` always overrides the sender id of the Dialogue instance after it has successfully authenticated the sender. Therefore, setting this before the call has no effect.

```
public void setSender(String sender);
```

3.5.4.1.3 SetRecipient

The recipient is a caller A identity (MSISDN) in the country normalized format. Example: "+358401234567".

```
public void setRecipient(String recipient);
```

3.5.4.1.4 SetCoreMsg

```
public void setCoreMsg(String coreMsg);
```

3.5.4.1.5 SetOption

```
public void setOption(String option, String description);
```

The method is used to set the pair of option and description. When pairs are set, they are inserted into the dialogue message in the First Come First Served order. See `clearOptions()`.

3.5.4.1.6 SetExpiryTime

```
public void setExpiryTime(int expiryTimeInMinutes);
```

The method sets the expiry time in minutes. The default expiry time is 1440 minutes. If given a invalid parameter (≤ 0) the dialogue uses the default expiry time.

3.5.4.1.7 GetId

```
public long getId();
```

The dialogue id can be retrieved meaningfully on two occasions:

1. After a successful send method invocation.
2. After a successful status method invocation (but it is the same as prior to the call).

Bookit Ltd

In other cases, the dialogue id retrieved is either the pre-status set dialogue id, or the default value, which is 0.

3.5.4.1.8 GetRecipient

```
public String getRecipient();
```

3.5.4.1.9 GetCoreMsg

```
public String getCoreMsg();
```

3.5.4.1.10 GetFullMsg

```
public String getFullMsg();
```

This method returns the full message, which consists of the core message, options, descriptions and some white space/punctuation.

3.5.4.1.11 GetOptions

```
public ArrayList getOptions();
```

This method returns an `ArrayList` object which can be used to retrieve the options (not descriptions). See `getDescriptions()`. An example:

```
// The Dialogue `dialogue' instantiated previously.
ArrayList options = dialogue.getOptions();
ArrayList descriptions = dialogue.getDescriptions();
for (int i = 0; i < options.size(); i++) {
    System.out.println "[" + i + "]: " + options.get(i)
        + ", " + descriptions.get(i) + " ";
}
}
```

3.5.4.1.12 GetDescriptions

```
public ArrayList getDescriptions();
```

This method returns an `ArrayList` object which can be used to retrieve the descriptions (not options). See `getOptions()`.

3.5.4.1.13 ClearOptions

```
public void clearOptions();
```

Clears the Dialogue options and descriptions. The next invocation of `setOption()` will add the first option, description pair into the Dialogue.

3.5.4.1.14 GetStatus

```
public DDMStatus getStatus();
```

This method returns the current `DDMStatus` instance. It always returns a non null object.

Revision 1.9.2 GKo 26.05.2009

Bookit Ltd

For dialogue status codes, see chapter 3.5.2.

3.5.4.1.15 GetSender

```
public String getSender();
```

3.5.5 Class DDMStatus

The `DDMStatus` class is conceptually tied to a `Dialogue`.

3.5.5.1 Methods

3.5.5.1.1 GetDialogueId

```
public long getDialogueId();
```

This method returns the dialogue id. The default value is 0.

3.5.5.1.2 GetRecipient

```
public String getRecipient();
```

3.5.5.1.3 GetStatus

```
public long getStatus();
```

3.5.5.1.4 GetStatusMessage

```
public String getStatusMessage();
```

3.5.5.1.5 GetAnswer

```
public String getAnswer();
```

This method (and `getAnswerNo()`) are meaningful only after a successful status method invocation, when the status code is `DDM.ANSWERED`. The method returns the dialogue reply option the recipient selected.

3.5.5.1.6 GetAnswerNo

```
public int getAnswerNo();
```

This method (and `getAnswer()`) are meaningful only after a successful status method invocation, when the status code is `DDM.ANSWERED`. The method returns, what was the dialogue reply option number the recipient selected (indexing starts from 1).

Bookit Ltd

3.6 Document Type Definitions (DTD)

3.6.1 Send Request

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT sendRequest (version+, sender, coreMsg, recipient, expiry?, preformatted,
options*, reply_url?)>
<!ELEMENT sender (id, enc)>
<!ELEMENT expiry (relative?)>
<!ELEMENT options (option+)>
<!ELEMENT option (response, description?)>
<!ELEMENT version (#CDATA)>
<!ELEMENT id (#CDATA)>
<!ELEMENT enc (#CDATA)>
<!ELEMENT coreMsg (#CDATA)>
<!ELEMENT recipient (#CDATA)>
<!ELEMENT relative (#CDATA)>
<!ELEMENT preformatted (#CDATA)>
<!ELEMENT response (#CDATA)>
<!ELEMENT description (#CDATA)>
<!ELEMENT reply_url (#CDATA)>
```

3.6.2 Send Reply

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT sendResult (id, status, warn_level?)>
<!ELEMENT status (code, message)>
<!ELEMENT id (#CDATA)>
<!ELEMENT code (#CDATA)>
<!ELEMENT message (#CDATA)>
<!ELEMENT warn_level (#CDATA)>
```

3.6.3 Status Request

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT statusRequest (id, sender)>
<!ELEMENT sender (id, enc)>
<!ELEMENT id (#CDATA)>
<!ELEMENT enc (#CDATA)>
```

3.6.4 Status Reply

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT statusResult (id, status, answer?, warn_level?)>
<!ELEMENT status (code, message)>
<!ELEMENT answer (response, no)>
<!ELEMENT id (#CDATA)>
<!ELEMENT code (#CDATA)>
<!ELEMENT message (#CDATA)>
<!ELEMENT response (#CDATA)>
<!ELEMENT no (#CDATA)>
<!ELEMENT warn_level (#CDATA)>
```

Bookit Ltd

3.6.5 Close Request

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!ELEMENT closeRequest (id, sender)>  
<!ELEMENT sender (id, enc)>  
<!ELEMENT id (#CDATA)>  
<!ELEMENT enc (#CDATA)>
```

4. OTHER INTERFACES

Interfaces from the iSMS[®] platform to the wireless network (from iSMS[®] Gateway) and Charging and Management interfaces are described in references [5] and [6].