

# mGw2 Client for .Net User Guide

## CONTENTS

1	Introduction .....	3
2	Service Managers.....	3
2.1	Manager Types.....	3
2.2	Manager Common Properties.....	3
2.3	Manager Instantiation .....	4
2.3.1	Constructor Arguments .....	4
2.3.2	Setting Properties.....	4
2.3.3	Mixed .....	4
2.4	Manager Reuse and Threading Issue .....	4
3	Messaging Services .....	5
3.1	Sending Messages .....	5
3.1.1	Sending SMS message .....	6
3.1.2	Sending MMS Message .....	6
3.1.3	Sending Logo Message .....	6
3.1.4	Sending RingTone Message .....	7
3.1.5	Sending SMPP Message .....	7
3.1.6	Sending WAP Push SI Message .....	7
3.1.7	Extensions for Sending Messages.....	7
3.2	Acquire Messages Delivery Status .....	8
3.3	Retrieve Messages (Pull Model).....	8
3.4	Receive Messages.....	10
3.5	Notification Web Service Hosted on IIS .....	11
3.5.1	Notification Web Service Started Locally in Application .....	12
3.6	Registering Message Listener .....	13
4	Charging Services .....	13
4.1	Service Limitations .....	13
4.2	Charging Examples .....	14
4.2.1	Charge Amount.....	14
4.2.2	Reserve, Commit or Release Charge.....	14
5	Location Services .....	15
5.1	Finding User Location.....	15
6	Manager Exceptions .....	16
7	Logging .....	16
8	Adding mGw Client Assemblies From GAC (Global Assembly Cache) .....	17
9	Additional Notes .....	17

## 1 INTRODUCTION

---

Message Gateway (mGw) provides various functionality through its remoting services. These services use standardized or custom protocols for communication. Examples of such protocols are SOAP for web services, SMPP and many more. They can become too complex and cumbersome to use. mGw client is user-friendly library for communicating with mGw remoting services.

## 2 SERVICE MANAGERS

---

### 2.1 Manager Types

Different services are abstracted via `Managers`. There are three `Manager`, each for every service exposed by mGw remoting interfaces:

- Message Manager – sending and receiving messages
- Charging Manager – charging funds to user
- Location Manager – finding user location

#### Manager Interfaces and Implementations

Interface	Implementation	Used for
MessageManager2	MessageManager2Impl	Sending and receiving messages
ChargingManager2	ChargingManager2Impl	Charging funds to user
LocationManager2	LocationManager2Impl	Finding user location

### 2.2 Manager Common Properties

Every type of `Manager` uses SOAP for web services to communicate with mGw. There are some common properties that each `Manager` must set before it can communicate with mGw.

- mGw service URL

To be able to communicate with mGw, `Manager` must specify URL on which various mGw services are exposed.

- Authentication

To call mGw service, `Manager` must authenticate itself. `Manager` authenticates itself with username/password pair. Every `Manager` can set default username/password pair used for every call. There is also copy of every call in which new username/password pair can be specified overriding default values.

- Timeouts

Since every call is in effect network communication there are plenty of reasons why mGw can't be reached (no network connection, firewall blocking traffic, ...). Every `Manager` have timeout property which define maximum time in milliseconds waiting for mGw to send response, after which call is abandoned and exception is thrown.

- Receive threading model

When `Manager` receive message from mGw it has to send message to registered listeners for processing. Since processing can take long time there are two possibilities. To spawn a new thread and return response to mGw immediately or to wait until processing is done and then return response to mGw. Both

possibilities have their weaknesses and strengths. It is up to developer to decide which approach is better for him. The default operation is to spawn new thread for processing and returning response to mGw immediately.

### Common Manager Properties

Property	Description
serviceURL	mGw service url
username	mGw service authentication username
password	mGw service authentication password
timeout	connection timeout in milliseconds (default is 30,000)
threadedListener	new received message spawn new thread for processing (default is true)

*NOTE: To obtain connection parameters (URL, username/password pair) contact your mGw administrator or your mobile operator contact person.*

## 2.3 Manager Instantiation

Before any service call can be made appropriate `Manager` must be instantiated. There are several ways `Manager` can be instantiated.

### 2.3.1 Constructor Arguments

Via constructor you can specify URL and username/password pair.

```
MessageManager2 manager = new MessageManager2Impl(url, user, pass);
```

### 2.3.2 Setting Properties

There is also possibility to create `Manager` without constructor arguments and set properties after.

```
MessageManager2 manager = new MessageManager2Impl();
manager.ServiceURL = url;
manager.Username = user;
manager.Password = pass;
```

### 2.3.3 Mixed

You can mix constructor arguments and setting properties. There is also possibility to override previous settings by setting properties.

```
MessageManager2 manager = new MessageManager2Impl(url);
manager.Username = user;
manager.Password = pass;
```

## 2.4 Manager Reuse and Threading Issue

Once instantiated, `Manager` can be reused many times to perform its operations. There are no limitations on reuse. As long as application is running the same

Manager can be used through application lifetime. However there are some limitation on threading issues.

### 3 MESSAGING SERVICES

To use messaging services, instantiate `MessageManager2Impl`. This is only implementation of `MessageManager2` interface that uses SOAP to call mGw messaging services.

There is also `MessageManager` which is older interface now deprecated (used for previous version of mGw). If you already have application that uses `MessageManager` it can still be used but you have to warn mGw administrator that you are using legacy interface since legacy interfaces are configured differently on mGw.

Use provided URL and username/password pair to instantiate `MessageManager2Impl`.

```
String url = <provided URL>;
String user = <provided Username>;
String pass = <provided Password>;
MessageManager2 manager = new MessageManager2Impl(url, user, pass);
```

#### 3.1 Sending Messages

After manager instantiation `sendMessage` method is used for sending messages. `sendMessage` method expected parameters are: message to send, array of destination addresses, originator address, priority and charging.

```
manager.sendMessage(message, destination, originator, priority,
charging);
```

*NOTE: See documentation for more information about `sendMessage` method on `MessageManager2` interface.*

#### Message Sending Interfaces and Implementations

<code>hr.tis.mgw.client.message.MessageManager2</code>	interface
<code>hr.tis.mgw.client.message.MessageManager2Impl</code>	implementation
<code>hr.tis.mgw.client.message.MessageManager</code>	deprecated interface
<code>hr.tis.mgw.client.message.MessageManagerImpl</code>	deprecated implementation

#### Message Types

Message Type	Use
<code>hr.tis.mgw.client.message.SmsMessage</code>	SMS message up to 460 characters
<code>hr.tis.mgw.client.message.MmsMessage</code>	MMS message with various textual and binary content
<code>hr.tis.mgw.client.message.LogoMessage</code>	Black and white logo message
<code>hr.tis.mgw.client.message.RingToneMessage</code>	Ring tone melody in RTTTL format
<code>hr.tis.mgw.client.message.SmppMessage</code>	SMPP message

hr.tis.mgw.client.message.wap. PushSIMessage	WAP Push Service Indication
---	-----------------------------

### 3.1.1 Sending SMS message

Create new `SmsMessage` containing message text. Priority is ignored for SMS message. For use of charging parameter see *Charging* section.

```
SmsMessage sms = new SmsMessage("sms message text");
manager.sendMessage(sms, destinations, originator, null, null);
```

### 3.1.2 Sending MMS Message

Create new `MmsMessage` containing `MmsParts`. `MmsMessage` holds subject of message. `MmsPart` hold textual or binary content. Attach `MmsParts` to `MmsMessage`. While sending specify priority of message. For use of charging parameter see *Charging* section.

```
byte[] bytes = ... byte array of loaded JPEG image from file ...
MmsMessage mms = new MmsMessage("subject");
MmsPart text = new MmsPart("This is text part of MMS message");
MmsPart image = new MmsPart(bytes, "image/jpeg");
mms.addPart(text);
mms.addPart(image);
manager.sendMessage(mms, destinations, originator, Priority.DEFAULT,
null);
```

#### Priority Values

Priority	Value
Default	hr.tis.mgw.client.message.Priority.DEFAULT
Low	hr.tis.mgw.client.message.Priority.LOW
Normal	hr.tis.mgw.client.message.Priority.NORMAL
High	hr.tis.mgw.client.message.Priority.HIGH

*NOTE: Binary content in this example was JPEG image. But any type of binary content can be used instead.*

### 3.1.3 Sending Logo Message

Create new `LogoMessage` containing byte array representing image. Image can be in one of following formats: jpeg, gif or png. The image will be adapted, resized to logo size and colors removed to black and white. Be careful what image resolution and how many colors it has since after adaptation result can be unrecognizable. Since various handset models have different image encoding, handset encoding format must be specified via format parameter. Priority is ignored for logo message. For use of charging parameter see *Charging* section.

```
byte[] bytes = ... byte array of loaded image from file ...
LogoMessage logo = new LogoMessage(bytes, Format.EMS);

manager.sendMessage(logo, destinations, originator, null, null);
```

## Format Values

Sms Format
<code>hr.tis.mgw.client.message.Format.EMS</code>
<code>hr.tis.mgw.client.message.Format.SMART_MESSAGING</code>

### 3.1.4 Sending RingTone Message

Create new `RingToneMessage` containing melody in RTTTL format. Since various handset models have different ring tone encoding, handset encoding format must be specified via format parameter. Priority is ignored for `RingToneMessage`. For use of charging parameter see *Charging* section.

```
String ringtone = <RTTTL melody>;
manager.sendMessage(ringtone, destinations, originator, null, null);
```

### 3.1.5 Sending SMPP Message

Create new `SmppMessage` containing smpp message as binary byte arrays. SMPP message must be split into appropriate chunks and placed inside binary byte arrays in right order as would be if message is send via smpp protocol. Originator and destination parameters are ignored. Any value specified will not have effect on message delivery. Originator and destination address are specified in SMPP message content. Also `priority` is ignored for SMPP message. For use of charging parameter see *Charging* section.

```
IList content = ... list of byte[] containing SMPP message ...
SmppMessage smpp = new SmppMessage(content);
manager.sendMessage(smpp, null, null, null, null);
```

### 3.1.6 Sending WAP Push SI Message

Create new `PushSIMessage` containing text and uri (this is minimum requirements). Priority is ignored for `PushSIMessage`. For use of charging parameter see *Charging* section.

```
PushSIMessage push = new PushSIMessage("visit Google", new
URI("www.google.com"));
```

### 3.1.7 Extensions for Sending Messages

Client API and ParlayX enables clients to send messages that might be forwarded on to messaging center (SMSC) as multiple messages. This number of messages sent to SMSC might be used for reporting and billing purposes. Also this number might be different then the number of sending requests client perform on mGw.

To resolve this issue, extensions are implemented, both in client API and on web service interface exported by mGw.

Methods take same parameters and type of messages as described before in this chapter. Only difference is in their return type.

```
manager.sendMessageWithResult(
    message, destination, originator, priority, charging);
```

Method returns `SendResult` structure.

### SendResult

Property	Description
messageId	Message ID
count	Number of messages forwarded to single destination, e.g. if long SMS sent via client is split into 3 SMS messages when sending to SMSC, and is sent to 2 destinations, value of this parameter would be 3.

*NOTE: Extensions might not be supported by some operators, contact mGw administrator to verify this.*

## 3.2 Acquire Messages Delivery Status

To acquire message delivery status `MessageId` obtained after sending message is used as parameter to `getMessageDeliveryStatus` method. Since message can be sent to many destination addresses, for every address there is one corresponding message delivery status. `DeliveryStatusInfo` contains destination address and status of message.

```
SmsMessage sms = new SmsMessage("text");
MessageId mid = manager.sendMessage(sms, destinations, originator,
null, null);

DeliveryStatusInfo[] infos =
manager.getMessageDeliveryStatus(mid);
```

### Message Status

Status	Value
Delivered	<code>hr.tis.mgw.client.message.Status.DELIVERED</code>
Delivery Uncertain	<code>hr.tis.mgw.client.message.Status.DELIVERY_UNCERTAIN</code>
Delivery Impossible	<code>hr.tis.mgw.client.message.Status.DELIVERY_IMPOSSIBLE</code>
Message Waiting	<code>hr.tis.mgw.client.message.Status.MESSAGE_WAITING</code>

*NOTE: See documentation for more information about `getMessageDeliveryStatus` method on `MessageManager2` interface.*

## 3.3 Retrieve Messages (Pull Model)

To retrieve messages call `receiveSmsMessages` to retrieve sms messages or `receiveMmsMessages` to retrieve mms messages. The maximum number of messages received is configured on mGw so to retrieve all messages call appropriate method until no more messages are retrieved. To retrieve messages via methods mentioned previously you have to specify registration identifier. It is used to select route (channel) over which message arrived.

```
ReceivedSms[] smses = manager.receiveSmsMessages(regId);
```



## ReceivedSms Properties

Property	Description
smsMessage	Received SMS
senderAddress	SMS message originator address
destinationAddress	SMS message destination address
registrationIdentifier	Route (channel) over which message arrived

```
ReceivedMms[] mmses = manager.receiveMmsMessages(regId);
```

## ReceivedMms Properties

Property	Description
mmsMessage	Received mms
senderAddress	MMS message originator address
destinationAddress	MMS message destination address
Priority	MMS message priority
registrationIdentifier	Route (channel) over which message arrived

*NOTE: See documentation for more information about `receiveSmsMessages` and `receiveMmsMessages` methods on `MessageManager2` interface. To find out valid Registration Identifier contact mGw administrator or your mobile operator contact person.*

Messages received this way include regular messages and delivery reports, i.e. delivery reports are encoded as `ReceivedSms` or `ReceivedMms`.

Client can use `MessageHelper` utility class with methods for recognition and creation of `DeliveryReports`.

```
ReceivedSms sms = smses[0];
if (MessageHelper.isDeliveryReport(sms)) {
    DeliveryReport deliveryReport =
        MessageHelper.createDeliveryReport(sms);
    // process delivery report
} else {
    // process sms
}
```

Same processing can be done for `ReceivedMms`.

## DeliveryReport

Property	Description
senderAddress	Delivery report sender address (equal to destination address in original MT message)
destinationAddress	Delivery report destination address (short code)
messageId	Message ID (equal to message ID received when sending original MT message)
registrationId	Registration ID.
status	Message delivery status (see <a href="#">3.2</a> ).

## 3.4 Receive Messages

There are two ways to configure message reception:

- using Internet Information Service (IIS) for hosting web service that will receive messages,
- start message listener web service directly from your application. This feature is supported only on following operating systems: Windows 98, Windows Server 2003, Windows XP Media Center Edition, Windows XP SP2, Windows XP Starter Edition

All received messages will be sent to `MessageManager2` which will decide depending on registration to which listener message is to be routed for processing. If registered listener can't be found, message will be silently rejected. Only entry in log (if enabled) will be indication of message rejection. The listener which want to receive messages has to implement interface `MessageListener2` and register itself to `MessageManager2`.

### MessageListener2 Methods

Method	Action
<code>smsReceived(ReceivedSms sms)</code>	Called when sms message is received
<code>mmsReceived(ReceivedMms mms)</code>	Called when mms message is received
<code>deliveryReportReceived(DeliveryReport report)</code>	Called when delivery report is received
<code>longSmsReceived(ReceivedLongSmsSegment seg)</code>	Called when long SMS segment is received

*NOTE: See Retrieve Messages chapter for descriptions of `ReceivedSms` and `ReceivedMms` structures or consult documentation.*

### 3.5 Notification Web Service Hosted on IIS

In order to receive messages through web service that is hosted on local IIS following steps should be done:

- Install web services defined in `NotificationService.asmx` and `SmsNotificationService.asmx` on your local IIS (in same web folder). Test web services with web browser to assure they are working properly.
- In `web.config` configuration file set property `RemoteNotificationManagerPort` to port value that will be used for communication with your application
- In remoting configuration file (or directly in your application configuration file – `appname.exe.config`) write following XML fragment.:

```
<system.runtime.remoting>
  <application>
    <lifetime leaseTime="0">
    </lifetime>
    <service>
      <wellknown mode="Singleton"
type="hr.tis.mgw.client.message.notification.RemoteNotificationManage
r, mgw2-client-1.0.6" objectUri="RemoteNotificationManager" />
    </service>
    <channels>
      <channel ref="http" port="5554"> //This section is changed from
version 1.0.5
        <serverProviders>
          <formatter ref="soap" typeFilterLevel="Full" />
        </serverProviders>
      </channel>
    </channels>
  </application>
</system.runtime.remoting>
```

- In `channel` element set `port` attribute value to port value that will be used for communication with web service (same value as in `web.config` file in `RemoteNotificationManagerPort` property).
- See `MGWClientSDKSample` sample application and its configuration file for more information.
- In your application call method `StartRemoteListeners` in class `NotificationManager2` with remoting file name as parameter to start remoting object from your application.

### 3.5.1 Notification Web Service Started Locally in Application

In order to host SMS and MMSnotification web services directly from your application (in this case IIS is not required) following steps should be done:

In your application configuration file write these XML fragments:

- In ConfigSection element write

```
<section name="microsoft.web.services3"
  type="Microsoft.Web.Services3.Configuration.WebServicesConfigurat
ion,
  Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
  PublicKeyToken=31bf3856ad364e35" />
```

- Under Configuration element write:

```
<microsoft.web.services3>
  <messaging>
    <transports>
      <add scheme="http"
type="Microsoft.Samples.HttpSys.HttpSysTransport,
  Microsoft.Samples.HttpSys" />
    </transports>
  </messaging>
</microsoft.web.services3>
```

- See MGWClientSDKSample sample application and its configuration file for more information.
- On NotificationManager2 class call this two methods: StartLocalSmsListener and StartLocalMmsListener. Parameter port for these two methods is port on which this listener will listen for message notifications from mGw. Use different port numbers for each listener. Call this methods only once during your application lifecycle.

URL of the notification web service will be <http://hostname:portForSms/SmsNotification/> for SMS messages notification and <http://hostname:portForMms/MmNotification/> for MMS messages notification.

### 3.6 Registering Message Listener

Assuming you have implemented `MessageListener2` interface, the next step is to register your message listener. Listener is registered via `registerMessageListener` method on `MessageManager2` interface. There are two methods for registering listeners, one that use default username/password pair for authentication and one where you can specify username/password pair for authentication. It is important to understand how messages are received and when and why username/password pair is used for authentication.

- **receiving SMS message**

The whole message containing message text is received when request from mGw is accepted on client side.

- **receiving MMS message**

Only message reference is received when request from mGw is accepted on client side. To get content of mms message client library must call mGw service with received reference to retrieve message content. When this call is made username/password pair is used for authentication.

To register listener you have to specify registration identifier that represents route (channel) over which message arrived. To receive messages arrived over any route (channel) use `REG_ID_ANY` value for registration identifier.

```
String regId = MessageManager2.REG_ID_ANY;
MessageListener2 listener = new YourMessageListenerImplementation();
manager.registerMessageListener(regId, listener);
```

*NOTE: See documentation for more information about `MessageListener2` interface and `receiveMmsMessages` methods on `MessageManager2` interface. To find out valid Registration Identifier contact mGw administrator or your mobile operator contact person.*

## 4 CHARGING SERVICES

To use charging services, instantiate `ChargingManager2Impl`. This is implementation of `ChargingManager2` interface that uses SOAP to call mGw charging services.

Use provided URL and username/password pair to instantiate `ChargingManager2Impl`.

```
String url = <provided URL>;
String user = <provided Username>;
String pass = <provided Password>;
ChargingManager2 manager = new ChargingManager2Impl(url, user, pass);
```

### 4.1 Service Limitations

`ChargingManager2` interface provides many methods for charging funds to user. But not all methods have to be supported by your mobile operator. If not supported method is called mGw will throw exception that will be propagated to client.

*NOTE: To find out which methods are supported contact your mGw administrator or your mobile operator contact person.*

## 4.2 Charging Examples

In following examples only commonly used methods by mobile operators will be shown. Between amount and volume charging there is little difference from the client side. Only amount charging will be shown. Some parameters to methods can be used differently than mentioned, depending on mobile operator billing system capabilities.

### 4.2.1 Charge Amount

This is most simple example that will charge funds to user. Additional parameters (billingText and referenceCode) may be left empty if not desired by mobile operator or may be used for different purpose again depending on mobile operator.

```
String user = <user>;
BigDecimal amount = <amount>;
String billingText = <text to appear on bill>;
String referenceCode = <unique request identifier>;
manager.chargeAmount(user, amount, billingText, referenceCode);
```

### 4.2.2 Reserve, Commit or Release Charge

More complex example involving reserving amount, sending message and commit or release reservation depending on successful message delivery. Additional parameters have same restrictions or use as in simple charge amount example.

```
String user = <user>;
BigDecimal amount = <amount>;
String billingText = <text to appear on bill>;
String referenceCode = <unique request identifier>;
Charging charging = chargingManager.reserveAmount(user, amount,
billingText);

String[] destination = new String[] {user};
String originator = <originator>;
SmsMessage sms = new SmsMessage("sms message text");
MessageId mid = messageManager.sendMessage(sms, destination,
originator, null, charging);
```

*NOTE: Notice usage of charging parameter in message sending. This will link charging with message sending.*

```
chargingManager.chargeAmountReservation(
    charging, amount, billingText, referenceCode);
```

Or if message was not sent successfully.

```
chargingManager.releaseAmountReservation(charging);
```

For description of all others methods on ChargingManager2 interfaces please consult documentation.

## 5 LOCATION SERVICES

Location service contains only one method. To use location services instantiate `LocationManager2Impl`. This is implementation of `LocationManager2` interface that uses SOAP to call mGw location services.

There is also `LocationManager` which is older interface now deprecated (used for previous version of mGw). If you already have application that uses `LocationManager` it can still be used but you have to warn mGw administrator that you are using legacy interface since legacy interfaces are configured differently on mGw.

Use provided URL and username/password pair to instantiate `LocationManager2Impl`.

```
String url = <provided URL>;
String user = <provided Username>;
String pass = <provided Password>;
LocationManager2 manager = new LocationManager2Impl(url, user, pass);
```

### 5.1 Finding User Location

To find user location call `getLocation` method. Requester parameter can be left empty. Effect of accuracy values on result depends on mobile operator infrastructure. Result of method call is location which contains user longitude, latitude, accuracy of result and time of result.

```
String user = <user>;
String requester = <requester>;
Accuracy accuracy = Accuracy.MEDIUM;
Location location = manager.getLocation(user, requester, accuracy);
```

#### Accuracy Values

Accuracy	Value
Low accuracy	<code>hr.tis.mgw.client.location.Accuracy.LOW</code>
Medium accuracy	<code>hr.tis.mgw.client.location.Accuracy.MEDIUM</code>
High accuracy	<code>hr.tis.mgw.client.location.Accuracy.HIGH</code>

#### Location Properties

Property	Description
<code>longitude</code>	User location longitude
<code>latitude</code>	User location latitude
<code>accuracy</code>	Accuracy of result values
<code>time</code>	Time when measurement was made

## 6 MANAGER EXCEPTIONS

Every method in every manager throws exceptions. There is one root generic exception `GatewayException` which covers all different exceptions thrown. This way only one exception needs to be caught. If desired sub exceptions can be caught explicitly. To view all possible exceptions look at the `hr.tis.mgw.client.exceptions` package in documentation. The exceptions correspond to ParlayX exceptions. For understanding of exceptions types and usage read ParlayX for web services manual.

### Exceptions

Exception	Description
<code>MServiceException</code>	Thrown when mGw is unable to process request. See exception message for reason.
<code>MPolicyException</code>	Thrown when authentication has failed. Check username/password pair.
<code>MRemoteException</code>	Thrown when remote call to mGw failed (network problem).
<code>GatewayException</code>	Generic root exception.

## 7 LOGGING

Logging library used is Microsoft Logging Application Block (part of Microsoft Enterprise library 2006). Logging configuration must be present in your application configuration file. Logging configuration can be easily done with Enterprise Library Configuration utility that comes with Enterprise Library 3.1 installation. See `MGWClientSDKSample` sample application and its configuration file for sample configuration.

Possible logging categories are: `Trace`, `Information`, `Error`, `Exception`.

We recommend monitoring of `Information`, `Error` and `Exception` categories since library was tested thoroughly and there is no need for you to test it again

*NOTE: If updating to version 1.0.5 or later from previous versions, application's configuration file must be changed. In version 1.0.5 new version of Microsoft Enterprise Library is introduced (version 3.1 Strong-Name signed). Every element that contains reference to an older version of Microsoft Enterprise Library must be updated to reference newer version, with correct `PublicKeyToken` Attribute. For example element:*

```
<section name="loggingConfiguration"
  type="Microsoft.Practices.EnterpriseLibrary.Logging.
Configuration.LoggingSettings,Microsoft.Practices.
EnterpriseLibrary.Logging, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=null" />
```

*must be changed to:*

```
<section name="loggingConfiguration"
  type="Microsoft.Practices.EnterpriseLibrary.Logging.
Configuration.LoggingSettings, Microsoft.Practices.
EnterpriseLibrary.Logging, Version=3.1.0.0, Culture=neutral,
PublicKeyToken=f778b8875de17c4b" />
```



## 8 ADDING MGW CLIENT ASSEMBLIES FROM GAC (GLOBAL ASSEMBLY CACHE)

---

From version 1.0.5 mGw2 client is Strong-Name signed so it is possible to use client assemblies from GAC.

In order to insert mGw2 client assemblies into GAC, following steps must be performed:

- go to mGw2 client bin directory
- foreach assembly in bin folder perform following command:

```
gacutil /i <assemblyName>
```

## 9 ADDITIONAL NOTES

---

In library distribution you can find working samples that covers common usage of library. Please read README files in samples directory for instructions on how to use samples.