Message Logger

SQL Database & Web Server Specifications

Steffen L. Norgren
Doug Penner
Kyle Macdonald
Max Wardell
Eddie Zhang

COMP 3908 • BCIT • December 22, 2008

TABLE OF CONTENTS

Findings — SQL Database & Web Server:

While developing the server side of this project, I found relatively few obstacles with respect to creation of the database or web server side of the project. The database didn't require any undue complexity, however, I did limit the web service to only having the ability to execute stored procedures for the sake of security.

The the most part, development using C# and SOAP XML was uneventful. However, upon last-minute testing of the data being sent from the device, I found that I will need to implement some sort of conversion utility that converts Adaptive Multi-rate Coded (AMR) to a more standard WAV file.

Additionally, password hashing is not currently implemented due to there being some issues with implementing an SHA-1 hash using the BlackBerry JDE. Progress has been made on this issue and the feature will likely be implemented during the next iteration of the project. There will, however, be no issue implementing such a feature server side.

Specifications — SQL Database & Web Server:

For the test server, we used a virtual using VMware Sever 2.0 running under Ubuntu Linux. The current server specifications are as follows:

- Windows Server 2003
- SQL Server 2008
- C#.NET 3.5
- Visual Studio 2008
- Test Server: http://mlogger.trollop.org:3141

Despite the use of the versions listed above, the code I implemented is generic enough to be backwards compatible with all versions of the .NET platform.

Specifications — XML SOAP:

Currently, as there is no need for the Blackberry device to retrieve messages from the server, the XML SOAP schema is quite simple. It only consists of a UserLogin specification as well as a PostMessage specification. Each specification has a HTTP POST and response section.

UserLogin.xml

UserLogin.xml

```
POST /MLogger/MLService.asmx HTTP/1.1
Host: windev.trollop.org
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://virus-box/MLogger/UserLogin"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <UserLogin xmlns="http://virus-box/MLogger">
            <userName>string</userName>
            <password>string</password>
        </UserLogin>
    </soap:Body>
</soap:Envelope>


HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <UserLoginResponse xmlns="http://virus-box/MLogger">
            <UserLoginResult>long</UserLoginResult>
        </UserLoginResponse>
    </soap:Body>
</soap:Envelope>
```

PostMessage.xml

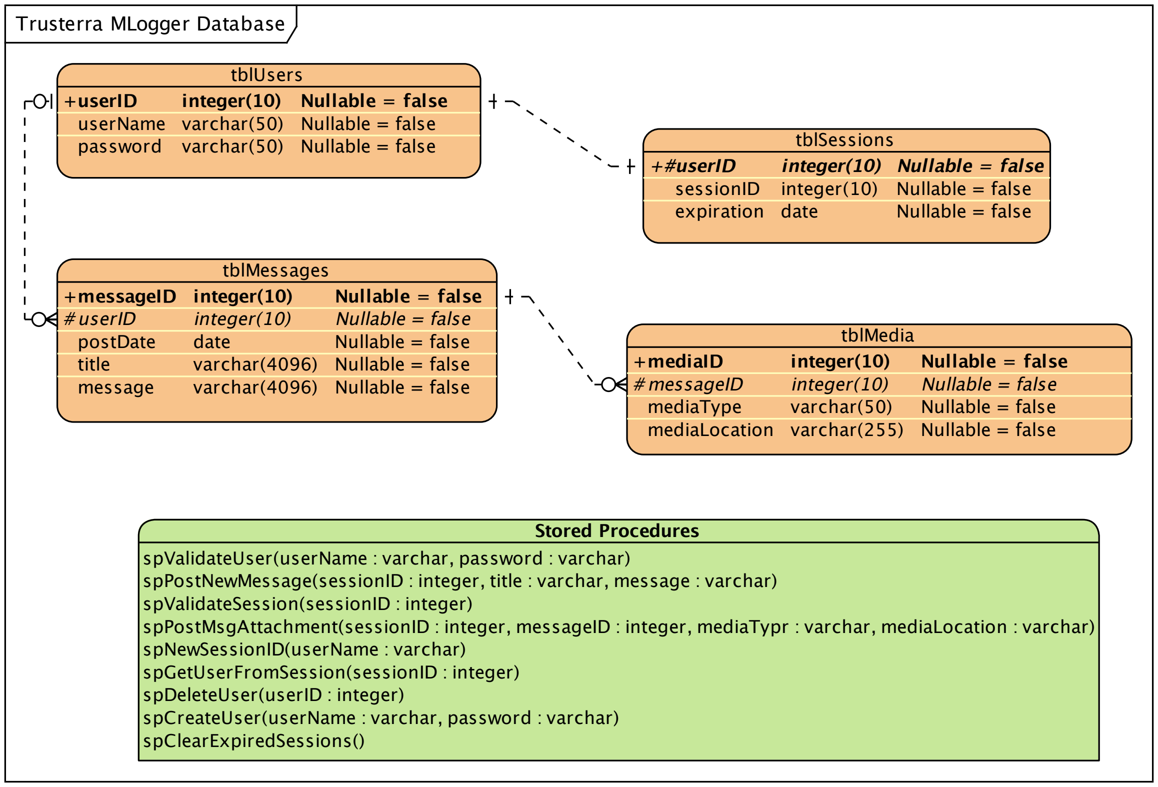
```
POST /MLogger/MLService.asmx HTTP/1.1
Host: windev.trollop.org
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://virus-box/MLogger/PostMessage"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <PostMessage xmlns="http://virus-box/MLogger">
            <postContents>
                <sessionID>long</sessionID>
                <title>string</title>
                <message>string</message>
                <!-- This section can be omitted or repeated based on the
number of attachments -->
                <dataType>string</dataType>
                <data>string</data>
            </postContents>
        </PostMessage>
    </soap:Body>
</soap:Envelope>


HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <PostMessageResponse xmlns="http://virus-box/MLogger">
            <PostMessageResult>int</PostMessageResult>
        </PostMessageResponse>
    </soap:Body>
</soap:Envelope>
```

ENTITY RELATIONSHIP DIAGRAM — SQL DATABASE:

This is the Message Logger's database entity relationship diagram. At the current stage of the project, we only require four tables.

All users are stored in the table tblusers. When a user logs in, a new entry is created in tblSessions with an expiration date of two hours from the initial log-in. Any time a new message is posted to tblMessages, the server verifies that the session is valid, otherwise it returns an error, causing the Message Logger application to log-in again.

A message may be posted without any attachments or as many attachments as the user desires. When there is an attachment to be posted, it is encoded on the Blackberry using base64 UTF-8 encoding and is subsequently decoded on the server and stored on the hard drive. References to the location of the stored data and its type is stored in the tblMedia table.



tblUsers

+userID	**integer(10)**	**Nullable = false**
userName	varchar(50)	Nullable = false
password	varchar(50)	Nullable = false

tblSessions

+#userID	***integer(10)***	***Nullable = false***
sessionID	integer(10)	Nullable = false
expiration	date	Nullable = false

tblMessages

+messageID	**integer(10)**	**Nullable = false**
#userID	*integer(10)*	*Nullable = false*
postDate	date	Nullable = false
title	varchar(4096)	Nullable = false
message	varchar(4096)	Nullable = false

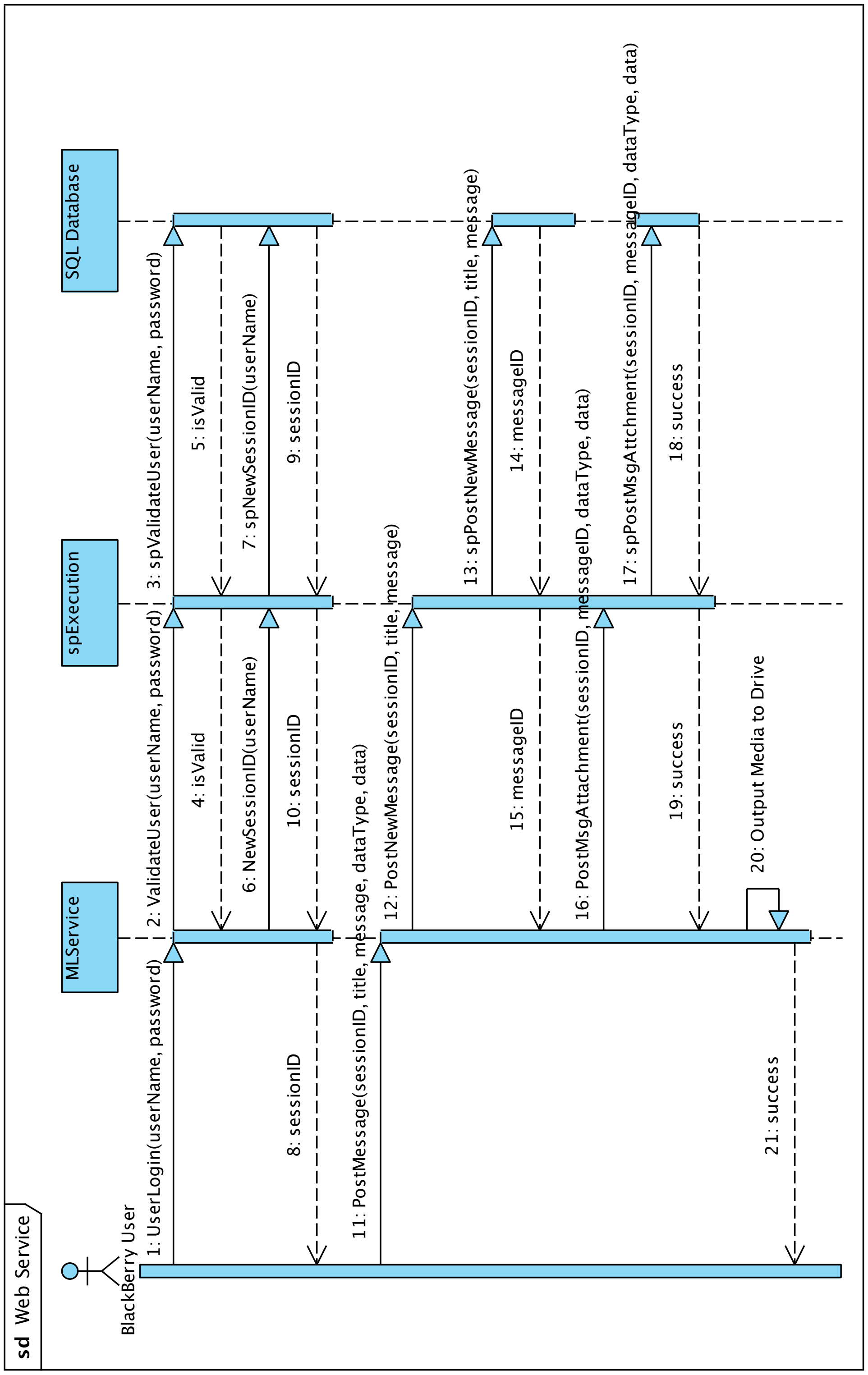
tblMedia

+mediaID	**integer(10)**	**Nullable = false**
#messageID	*integer(10)*	*Nullable = false*
mediaType	varchar(50)	Nullable = false
mediaLocation	varchar(255)	Nullable = false

Stored Procedures

```
spValidateUser(userName : varchar, password : varchar)
spPostNewMessage(sessionID : integer, title : varchar, message : varchar)
spValidateSession(sessionID : integer)
spPostMsgAttachment(sessionID : integer, messageID : integer, mediaTypr : varchar, mediaLocation : varchar)
spNewSessionID(userName : varchar)
spGetUserFromSession(sessionID : integer)
spDeleteUser(userID : integer)
spCreateUser(userName : varchar, password : varchar)
spClearExpiredSessions()
```

Sequence Diagram — Web Services


sd Web Service
BlackBerry User
MLService
spExecution
SQL Database
1: UserLogin(userName, password)
2: ValidateUser(userName, password)
3: spValidateUser(userName, password)
4: isValid
5: isValid
6: NewSessionID(userName)
7: spNewSessionID(userName)
8: sessionID
9: sessionID
10: sessionID
11: PostMessage(sessionID, title, message, dataType, data)
12: PostNewMessage(sessionID, title, message)
13: spPostNewMessage(sessionID, title, message)
14: messageID
15: messageID
16: PostMsgAttachment(sessionID, messageID, dataType, data)
17: spPostMsgAttchment(sessionID, messageID, dataType, data)
18: success
19: success
20: Output Media to Drive
21: success

Source Code — SQL Database

spClearExpiredSessions.sql

```
C:\Documents and Settings\Administrator\...\Database Schema\clearExpiredSessions.sql    1
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spClearExpiredSessions') IS NOT NULL
    DROP PROCEDURE spClearExpiredSessions
GO

USE MLogger
GO

-- Clear expired sessions from the Session table
CREATE PROCEDURE spClearExpiredSessions AS
    DELETE FROM tblSessions WHERE expiration < GETDATE()
GO
```

spCreateUser.sql

```
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spCreateUser') IS NOT NULL
    DROP PROCEDURE spCreateUser
GO

USE MLogger
GO

-- Stored procedure for creating a new user
-- Returns: 1 if created
--          0 if user name existed and not created
CREATE PROCEDURE spCreateUser(@userName varchar(50), @password varchar(50)) AS
    DECLARE @count      INT
    DECLARE @maxUserID  INT

    SELECT @count = COUNT(*) FROM tblUsers

    IF @count > 0
        SELECT @maxUserID = MAX(userID + 1) FROM tblUsers
    ELSE
        SET @maxUserID = 0

    IF NOT EXISTS (SELECT * FROM tblUsers WHERE userName = @userName)
    BEGIN
        INSERT INTO tblUsers(userID, userName, [password]) VALUES(@maxUserID, @userName, @   ↙
    password)
        RETURN 1
    END
    ELSE
        RETURN 0
GO
```

spDeleteUser.sql

```
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spDeleteUser') IS NOT NULL
    DROP PROCEDURE spDeleteUser
GO

USE MLogger
GO

-- Stored procedure for user deletion
-- Returns: 1 if existed and deleted
--          0 if didn't exist
CREATE PROCEDURE spDeleteUser(@userID INT) AS
    IF EXISTS (SELECT * FROM tblUsers WHERE userID = @userID)
    BEGIN
        DELETE FROM tblUsers WHERE userID = @userID
        RETURN 1
    END
    ELSE
        RETURN 0
GO
```

spGetUserFromSession.sql

```sql
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spGetUserFromSession') IS NOT NULL
    DROP PROCEDURE spGetUserFromSession
GO

USE MLogger
GO

-- Stored procedure for posting new message
-- Returns: userName
CREATE PROCEDURE spGetUserFromSession(@sessionID int) AS
    DECLARE @userID      INT
    DECLARE @userName    VARCHAR(50)

    -- Retrieve the UserID associated with the current session
    SELECT @userID = userID FROM tblSessions WHERE sessionID = @sessionID

    -- Retrieve the User Name associated with the UserID
    SELECT @userName = userName FROM tblUsers WHERE userID = @userID

    SELECT @userName AS 'UserName'
GO
```

spNewSessionID.sql

```
C:\Documents and Settings\Administrator\... L. Norgren\Database Schema\newSessionID.sql       1
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spNewSessionID') IS NOT NULL
    DROP PROCEDURE spNewSessionID
GO

USE MLogger
GO

-- Creates a new session ID and stores it in the Session table
-- Returns: Session ID
CREATE PROCEDURE spNewSessionID(@userName varchar(50)) AS
    DECLARE @sessionID  INT
    DECLARE @userID     INT

    -- Grab the user's ID
    SELECT @userID = userID FROM tblUsers WHERE userName = @userName

    -- Set the new session ID
    SELECT @sessionID = ABS(CAST(CAST(NEWID() AS VARBINARY) AS INT));

    -- Clear all previous sessions for the user
    DELETE FROM tblSessions WHERE userID = @userID

    -- Insert the new userID/session ID pair into the table with a 2-hour expiration
    INSERT INTO tblSessions(userID, sessionID, expiration) VALUES(@userID, @sessionID,        ↙
    DATEADD(hh, 2, GETDATE()))

    -- Returns the session ID
    RETURN @sessionID
GO
```

spPostMsgAttachment.sql

```
C:\Documents and Settings\Administrator\...Norgren\Database Schema\postMsgAttachment.sql      1
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spPostMsgAttachment') IS NOT NULL
    DROP PROCEDURE spPostMsgAttachment
GO

USE MLogger
GO

-- Stored procedure for posting new message
CREATE PROCEDURE spPostMsgAttachment(@sessionID int, @messageID int, @mediaType varchar(50), ↙
    @mediaLocation varchar(255)) AS
    DECLARE @count      INT
    DECLARE @maxMediaID INT
    DECLARE @userID     INT

    SELECT @count = COUNT(*) FROM tblMedia

    -- Set the new message ID
    IF @count > 0
        SELECT @maxMediaID = MAX(mediaID + 1) FROM tblMedia
    ELSE
        SET @maxMediaID = 0


    INSERT INTO tblMedia(mediaID, messageID, mediaType, mediaLocation) VALUES(@maxMediaID, @ ↙
    messageID, @mediaType, @mediaLocation)
GO
```

spPostNewMessage.sql

```
C:\Documents and Settings\Administrator\...L. Norgren\Database Schema\postNewMessage.sql     1
```

```
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spPostNewMessage') IS NOT NULL
    DROP PROCEDURE spPostNewMessage
GO

USE MLogger
GO

-- Stored procedure for posting new message
-- Returns: messageID
CREATE PROCEDURE spPostNewMessage(@sessionID int, @title varchar(MAX), @message varchar(MAX))↙
     AS
    DECLARE @count          INT
    DECLARE @maxMessageID   INT
    DECLARE @userID         INT

    SELECT @count = COUNT(*) FROM tblMessages

    -- Set the new message ID
    IF @count > 0
        SELECT @maxMessageID = MAX(messageID + 1) FROM tblMessages
    ELSE
        SET @maxMessageID = 0

    -- Retrieve the UserID associated with the current session
    SELECT @userID = userID FROM tblSessions WHERE sessionID = @sessionID

    INSERT INTO tblMessages(messageID, userID, postDate, title, [message]) VALUES(@          ↙
    maxMessageID, @userID, GETDATE(), @title, @message)

    RETURN @maxMessageID
GO
```

spValidateSession.sql

C:\Documents and Settings\Administrator\.... Norgren\Database Schema\validateSession.sql 1

```
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spValidateSession') IS NOT NULL
    DROP PROCEDURE spValidateSession
GO

USE MLogger
GO

-- Stored procedure to check if the session ID is valid
-- Returns: 1 if valid
--            0 if invalid
CREATE PROCEDURE spValidateSession(@sessionID int) AS
    IF EXISTS (SELECT * FROM tblSessions WHERE sessionID = @sessionID AND expiration >=      ↙
    GETDATE())
        RETURN 1
    ELSE
        RETURN 0
GO
```

spValidateUser.sql

```
C:\Documents and Settings\Administrator\... L. Norgren\Database Schema\validateUser.sql      1
-- Drop the stored procedure if it exists
IF OBJECT_ID ('spValidateUser') IS NOT NULL
    DROP PROCEDURE spValidateUser
GO

USE MLogger
GO

-- Stored procedure for user validation
-- Returns: 1 if valid
--          0 if invalid
CREATE PROCEDURE spValidateUser(@userName varchar(50), @password varchar(50)) AS
    IF EXISTS (SELECT * FROM tblUsers WHERE userName = @userName AND [password] = @password)
        RETURN 1
    ELSE
        RETURN 0
GO
```

MLogger.sql — Entire Database SQL Dump

```
USE [MLogger]
GO
/****** Object:  User [mlogger]    Script Date: 12/21/2008 19:18:08 ******/
CREATE USER [mlogger] FOR LOGIN [mlogger] WITH DEFAULT_SCHEMA=[dbo]
GO
/****** Object:  FullTextCatalog [Full-Text Index]    Script Date: 12/21/2008 19:18:08 ******↙
    /
CREATE FULLTEXT CATALOG [Full-Text Index]
WITH ACCENT_SENSITIVITY = ON
AS DEFAULT
AUTHORIZATION [dbo]
GO
/****** Object:  Table [dbo].[tblUsers]    Script Date: 12/21/2008 19:18:12 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblUsers](
    [userID] [int] NOT NULL,
    [userName] [varchar](50) NOT NULL,
    [password] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
    [userID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS↙
      = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
INSERT [dbo].[tblUsers] ([userID], [userName], [password]) VALUES (0, N'ironix', N'trustno1')
INSERT [dbo].[tblUsers] ([userID], [userName], [password]) VALUES (1, N'foo', N'bar')
/****** Object:  StoredProcedure [dbo].[spValidateUser]    Script Date: 12/21/2008 19:18:20 *↙
    *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for user validation
-- Returns: 1 if valid
--          0 if invalid
CREATE PROCEDURE [dbo].[spValidateUser](@userName varchar(50), @password varchar(50)) AS
    IF EXISTS (SELECT * FROM tblUsers WHERE userName = @userName AND [password] = @password)
        RETURN 1
    ELSE
        RETURN 0
GO
/****** Object:  StoredProcedure [dbo].[spDeleteUser]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for user deletion
-- Returns: 1 if existed and deleted
--          0 if didn't exist
CREATE PROCEDURE [dbo].[spDeleteUser](@userID INT) AS
    IF EXISTS (SELECT * FROM tblUsers WHERE userID = @userID)
    BEGIN
        DELETE FROM tblUsers WHERE userID = @userID
        RETURN 1
    END
    ELSE
        RETURN 0
GO
/****** Object:  StoredProcedure [dbo].[spCreateUser]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for creating a new user
-- Returns: 1 if created
--          0 if user name existed and not created
CREATE PROCEDURE [dbo].[spCreateUser](@userName varchar(50), @password varchar(50)) AS
    DECLARE @count      INT
    DECLARE @maxUserID  INT

    SELECT @count = COUNT(*) FROM tblUsers

    IF @count > 0
        SELECT @maxUserID = MAX(userID + 1) FROM tblUsers
    ELSE
        SET @maxUserID = 0

    IF NOT EXISTS (SELECT * FROM tblUsers WHERE userName = @userName)
    BEGIN
        INSERT INTO tblUsers(userID, userName, [password]) VALUES(@maxUserID, @userName, @   ↙
    password)
        RETURN 1
    END
    ELSE
        RETURN 0
GO
/****** Object:  Table [dbo].[tblSessions]    Script Date: 12/21/2008 19:18:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblSessions](
    [userID] [int] NOT NULL,
    [sessionID] [int] NOT NULL,
    [expiration] [datetime] NOT NULL,
 CONSTRAINT [PK_Session] PRIMARY KEY CLUSTERED
(
    [userID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS↙
      = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
INSERT [dbo].[tblSessions] ([userID], [sessionID], [expiration]) VALUES (0, 1234567890, CAST ↙
    (0x00009E51003A6D2B AS DateTime))
/****** Object:  Table [dbo].[tblMessages]    Script Date: 12/21/2008 19:18:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblMessages](
    [messageID] [int] NOT NULL,
    [userID] [int] NOT NULL,
    [postDate] [datetime] NOT NULL,
    [title] [varchar](max) NULL,
    [message] [varchar](max) NULL,
 CONSTRAINT [PK_tblMessages] PRIMARY KEY CLUSTERED
(
    [messageID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS↙
      = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
INSERT [dbo].[tblMessages] ([messageID], [userID], [postDate], [title], [message]) VALUES (0,↙
     0, CAST(0x00009B79012F9322 AS DateTime), N'Memo Title', N'Memo Descrioption')
```

```
/****** Object:  Table [dbo].[tblMedia]    Script Date: 12/21/2008 19:18:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblMedia](
    [mediaID] [int] NOT NULL,
    [messageID] [int] NOT NULL,
    [mediaType] [varchar](50) NOT NULL,
    [mediaLocation] [varchar](255) NOT NULL,
 CONSTRAINT [PK_tblMedia] PRIMARY KEY CLUSTERED
(
    [mediaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS↙
      = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
INSERT [dbo].[tblMedia] ([mediaID], [messageID], [mediaType], [mediaLocation]) VALUES (0, 0, ↙
    N'gif-image', N'C:\Inetpub\MessageLogger\message-media\ironix\0\2008_25_21-06_25_16_976')
INSERT [dbo].[tblMedia] ([mediaID], [messageID], [mediaType], [mediaLocation]) VALUES (1, 0, ↙
    N'gif-image', N'C:\Inetpub\MessageLogger\message-media\ironix\0\2008_25_21-06_25_17_007')
/****** Object:  StoredProcedure [dbo].[spNewSessionID]    Script Date: 12/21/2008 19:18:20 *↙
    *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Creates a new session ID and stores it in the Session table
-- Returns: Session ID
CREATE PROCEDURE [dbo].[spNewSessionID](@userName varchar(50)) AS
    DECLARE @sessionID  INT
    DECLARE @userID     INT

    -- Grab the user's ID
    SELECT @userID = userID FROM tblUsers WHERE userName = @userName

    -- Set the new session ID
    SELECT @sessionID = ABS(CAST(CAST(NEWID() AS VARBINARY) AS INT));

    -- Clear all previous sessions for the user
    DELETE FROM tblSessions WHERE userID = @userID

    -- Insert the new userID/session ID pair into the table with a 2-hour expiration
    INSERT INTO tblSessions(userID, sessionID, expiration) VALUES(@userID, @sessionID,       ↙
    DATEADD(hh, 2, GETDATE()))

    -- Returns the session ID
    RETURN @sessionID
GO
/****** Object:  StoredProcedure [dbo].[spNewSession]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Creates a new session ID and stores it in the Session table
-- Returns: Session ID
CREATE PROCEDURE [dbo].[spNewSession](@userID int) AS
    DECLARE @session INT

    -- Set the new session ID
    SELECT @session = ABS(CAST(CAST(NEWID() AS VARBINARY) AS INT));

    -- Clear all previous sessions for the user
    DELETE FROM tblSessions WHERE userID = @userID
    -- Insert the new userID/session ID pair into the table with a 2-hour expiration
GO
/****** Object:  StoredProcedure [dbo].[spGetUserFromSession]    Script Date: 12/21/2008 19: ↙
    18:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for posting new message
-- Returns: userName
CREATE PROCEDURE [dbo].[spGetUserFromSession](@sessionID int) AS
    DECLARE @userID     INT
    DECLARE @userName   VARCHAR(50)

    -- Retrieve the UserID associated with the current session
    SELECT @userID = userID FROM tblSessions WHERE sessionID = @sessionID

    -- Retrieve the User Name associated with the UserID
    SELECT @userName = userName FROM tblUsers WHERE userID = @userID

    SELECT @userName AS 'UserName'
GO
/****** Object:  StoredProcedure [dbo].[spClearExpiredSessions]    Script Date: 12/21/2008 19↙
    :18:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Clear expired sessions from the Session table
CREATE PROCEDURE [dbo].[spClearExpiredSessions] AS
    DELETE FROM tblSessions WHERE expiration < GETDATE()
GO
/****** Object:  StoredProcedure [dbo].[spValidateSession]    Script Date: 12/21/2008 19:18: ↙
    20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure to check if the session ID is valid
-- Returns: 1 if valid
--          0 if invalid
CREATE PROCEDURE [dbo].[spValidateSession](@sessionID int) AS
    IF EXISTS (SELECT * FROM tblSessions WHERE sessionID = @sessionID AND expiration >=      ↙
    GETDATE())
        RETURN 1
    ELSE
        RETURN 0
GO
/****** Object:  StoredProcedure [dbo].[spPostNewMessage]    Script Date: 12/21/2008 19:18:20↙
     ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for posting new message
-- Returns: messageID
CREATE PROCEDURE [dbo].[spPostNewMessage](@sessionID int, @title varchar(MAX), @message      ↙
    varchar(MAX)) AS
    DECLARE @count          INT
    DECLARE @maxMessageID   INT
    DECLARE @userID         INT

    SELECT @count = COUNT(*) FROM tblMessages

    -- Set the new message ID
    IF @count > 0
        SELECT @maxMessageID = MAX(messageID + 1) FROM tblMessages
    ELSE
```

```
        SET @maxMessageID = 0

    -- Retrieve the UserID associated with the current session
    SELECT @userID = userID FROM tblSessions WHERE sessionID = @sessionID

    INSERT INTO tblMessages(messageID, userID, postDate, title, [message]) VALUES(@     ↙
    maxMessageID, @userID, GETDATE(), @title, @message)

    RETURN @maxMessageID
GO
/****** Object:  StoredProcedure [dbo].[spPostMsgAttachment]    Script Date: 12/21/2008 19:18↙
    :20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Stored procedure for posting new message
CREATE PROCEDURE [dbo].[spPostMsgAttachment](@sessionID int, @messageID int, @mediaType    ↙
    varchar(50), @mediaLocation varchar(255)) AS
    DECLARE @count      INT
    DECLARE @maxMediaID INT
    DECLARE @userID     INT

    SELECT @count = COUNT(*) FROM tblMedia

    -- Set the new message ID
    IF @count > 0
        SELECT @maxMediaID = MAX(mediaID + 1) FROM tblMedia
    ELSE
        SET @maxMediaID = 0


    INSERT INTO tblMedia(mediaID, messageID, mediaType, mediaLocation) VALUES(@maxMediaID, @ ↙
    messageID, @mediaType, @mediaLocation)
GO
/****** Object:  ForeignKey [FK_tblSessions_tblUsers]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
ALTER TABLE [dbo].[tblSessions]  WITH CHECK ADD  CONSTRAINT [FK_tblSessions_tblUsers] FOREIGN↙
     KEY([userID])
REFERENCES [dbo].[tblUsers] ([userID])
GO
ALTER TABLE [dbo].[tblSessions] CHECK CONSTRAINT [FK_tblSessions_tblUsers]
GO
/****** Object:  ForeignKey [FK_tblMessages_tblUsers]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
ALTER TABLE [dbo].[tblMessages]  WITH CHECK ADD  CONSTRAINT [FK_tblMessages_tblUsers] FOREIGN↙
     KEY([userID])
REFERENCES [dbo].[tblUsers] ([userID])
GO
ALTER TABLE [dbo].[tblMessages] CHECK CONSTRAINT [FK_tblMessages_tblUsers]
GO
/****** Object:  ForeignKey [FK_tblMedia_tblMessages]    Script Date: 12/21/2008 19:18:20 ***↙
    ***/
ALTER TABLE [dbo].[tblMedia]  WITH CHECK ADD  CONSTRAINT [FK_tblMedia_tblMessages] FOREIGN   ↙
    KEY([messageID])
REFERENCES [dbo].[tblMessages] ([messageID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[tblMedia] CHECK CONSTRAINT [FK_tblMedia_tblMessages]
GO
```

SOURCE CODE — WEB SERVER:

MLService.cs

```
using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.IO;
using System.Collections;

namespace MessageLogger
{
    /// <summary>
    /// The main webservice class. This handles all login and message posting
    /// requests to the web server.
    /// </summary>
    [WebService(Namespace = "http://virus-box/MLogger",
        Description="Main MLogger Web Service")]
    public class MLService : System.Web.Services.WebService
    {
        /// <summary>
        /// Default constructor, does nothing.
        /// </summary>
        public MLService()
        {
        }

        /// <summary>
        /// Validates a user's login and returns a new session ID for that user
        /// </summary>
        /// <param name="userName">The user's login name</param>
        /// <param name="password">The user's password</param>
        /// <returns>0 if invalid</returns>
        [WebMethod(Description="Message Logger User Login")]
        public long UserLogin(String userName, String password)
        {
            // Create new database object
            spExecution sp = new spExecution();
            sp.SQLConnect();

            // Check if the user name & password are valid
            if (sp.ValidateUser(userName, password) == "1")
            {
                sp.SQLDisconnect(); // Disconnect from SQL
                return NewSessionID(userName);
            }
            else
            {
                sp.SQLDisconnect(); // Disconnect from SQL
                return 0L;
            }
        }

        /// <summary>
        /// Posts a new message to the database
        /// </summary>
        /// <param name="sessionID">The user's current session ID</param>
        /// <param name="title">The title of the message</param>
        /// <param name="message">The message text</param>
        /// <returns>The message ID for the new message</returns>
        private int PostNewMessage(long sessionID, String title, String message)
        {
            int messageID;

            // Create new database object
            spExecution sp = new spExecution();
            sp.SQLConnect();
            // Post new message
            messageID = sp.PostNewMessage(sessionID, title, message);

            sp.SQLDisconnect(); // disconnect from the database

            return messageID;
        }

        /// <summary>
        /// Adds a media attachment to a message post
        /// </summary>
        /// <param name="sessionID">User's current session ID</param>
        /// <param name="messageID">The message ID to which the media will be attached</
    param>
        /// <param name="dataType">The type of media being attached</param>
        /// <param name="data">The base64 media data to be written to the drive</param>
        private void PostMsgAttachment(long sessionID, int messageID, String dataType, String
     data)
        {
            String fileName = "";

            // Create new database object
            spExecution sp = new spExecution();
            sp.SQLConnect();

            // Set up the base directory for where the data will be stored
            String path = @"C:\Inetpub\MessageLogger\message-media\" + sp.GetUserFromSession
    (sessionID) + @"\" + messageID + @"\";
            Directory.CreateDirectory(path);

            // Setup the file name
            System.Globalization.CultureInfo ci = System.Globalization.CultureInfo.
    InstalledUICulture;
            fileName += System.DateTime.Now.ToString("yyyy_mm_dd-hh_mm_ss_fff", ci);

            // Disconnect and reconnect to clear the SQL command
            sp.SQLDisconnect();
            sp.SQLConnect();

            // Post the data to the database
            sp.PostMsgAttachment(sessionID, messageID, dataType, path + fileName);
            sp.SQLDisconnect();

            // Convert the base64 string to a byte array
            byte[] dataBytes = Convert.FromBase64String(data);

            // Create the appropriate filename
            if (dataType == "jpeg-image")
            {
                fileName += ".jpg";
            }
            if (dataType == "png-image")
            {
                fileName += ".jpg";
            }
            else if (dataType == "gif-image")
            {
                fileName += ".gif";
            }
            else if (dataType == "mpeg-video")
            {
                fileName += ".mpg";
            }
            else if (dataType == "wmv-video")
            {
                fileName += ".wmv";
            }
            else if (dataType == "wav-sound")
            {
                fileName += ".wav";
            }

            // Write the file to the drive
            FileStream fs = new FileStream(path + fileName, FileMode.Create, FileAccess.   ↙
    Write);
            BinaryWriter writer = new BinaryWriter(fs);

            try
            {
                for (int i = 0; i < dataBytes.Length; i++)
                {
                    writer.Write(dataBytes[i]);
                }
            }
            finally
            {
                writer.Close();
                fs.Close();
            }
        }

        /// <summary>
        /// Parses input XML and creates a new message and message attachment post.
        /// </summary>
        /// <param name="input">An XML SOAP message</param>
        /// <returns>0 for invalid session ID, 1 for success, -1 for unspecified error</   ↙
    returns>
        [WebMethod(Description="Posts a message to a user's account")]
        public int PostMessage([XmlAnyElement]XmlElement input)
        {
            long sessionID = 0;
            int messageID = 0;
            String title = "";
            String message = "";
            String dataType = "";
            String data = "";

            IEnumerator ienum = input.GetEnumerator();
            XmlNode currentNode;

            ienum.Reset();

            while (ienum.MoveNext())
            {
                currentNode = (XmlNode)ienum.Current;

                if (currentNode.Name == "sessionID")
                {
                    long.TryParse(currentNode.InnerText, out sessionID);

                    // Return -1 if the session ID is invalid
                    if (!isValidSessionID(sessionID))
                    {
                        return 0;
                    }
                }
                else if (currentNode.Name == "title")
                {
                    title = currentNode.InnerText;
                }
                else if (currentNode.Name == "message") // Here we post the new message
                {
                    message = currentNode.InnerText;
                    messageID = PostNewMessage(sessionID, title, message);
                }
                else if (currentNode.Name == "dataType")
                {
                    dataType = currentNode.InnerText;
                }
                else if (currentNode.Name == "data") // Here we post the new message ↙
    attachment
                {
                    data = currentNode.InnerText;
                    PostMsgAttachment(sessionID, messageID, dataType, data);
                }
            }

            return 1;
        }

        /// <summary>
        /// Returns a new session ID for the user
        /// </summary>
        /// <param name="userName">The user loggin in</param>
        /// <returns>The user's new session ID</returns>
        private long NewSessionID(String userName)
        {
            long sessionID;

            // Create new database object
            spExecution sp = new spExecution();
            sp.SQLConnect();

            // Create new session ID
            sessionID = sp.NewSessionID(userName);

            sp.SQLDisconnect(); // Disconnect from SQL

            return sessionID;
        }

        /// <summary>
        /// Validates a current session ID
        /// </summary>
        /// <param name="sessionID">The user's session ID</param>
        /// <returns>TRUE if valid, FALSE if not</returns>
        private bool isValidSessionID(long sessionID)
        {
            // Create new database connection object
            spExecution sp = new spExecution();
            sp.SQLConnect();

            if (sp.ValidateSessionID(sessionID))
            {
                sp.SQLDisconnect(); // Disconnect from SQL
                return true;
            }
            else
            {
                sp.SQLDisconnect(); // Disconnect from SQL
                return false;
            }
        }
    }
}
```

spExecution.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Text.RegularExpressions;

namespace MessageLogger
{
    /// <summary>
    /// This class handles all communication with the SQL server. For security
    /// we will only allow stored procedures to be executed via the web application.
    ///
    /// If you want to extend the functionality of the web application, create new
    /// stored procedures and create a new associated function in this class.
    /// </summary>
    public class spExecution
    {
        private System.Data.SqlClient.SqlConnection sqlConn;
        // Regular expression to verify string is alphanumeric and no longer than 50 chars.
        Regex reg = new Regex(@"^[a-zA-Z0-9'.\s]{1,50}$");

        /// <summary>
        /// Default constructor. This does nothing.
        /// </summary>
        public spExecution()
        {
        }

        /// <summary>
        /// Connects to the default localhost database.
        /// </summary>
        public void SQLConnect()
        {
            sqlConn = new System.Data.SqlClient.SqlConnection();
            sqlConn.ConnectionString = "server=.; database=MLogger; uid=mlogger; pwd=trustera↙
    ;";
            sqlConn.Open();
        }

        /// <summary>
        /// Overloaded SQL connection function that allows one to specify a specific server,
        /// database, user, and password.
        /// </summary>
        /// <param name="server">The server IP address or NetBIOS name.</param>
        /// <param name="database">The database we want to use</param>
        /// <param name="user">The database user</param>
        /// <param name="password">The database user's password</param>
        public void SQLConnect(String server, String database, String user, String password)
        {
            sqlConn = new System.Data.SqlClient.SqlConnection();
            sqlConn.ConnectionString = "server=" + server + "; database=" + database + "; uid↙
    =" + user + "; pwd=" + password + ";";
            sqlConn.Open();
        }

        public void SQLDisconnect()
        {
            sqlConn.Close();
        }

        /// <summary>
        /// Validates a user against the Users table.
        /// </summary>
        /// <param name="userName">User's login name</param>
        /// <param name="password">User's login password</param>
        /// <returns>1 if valid, 0 if invalid</returns>
        public String ValidateUser(String userName, String password)
        {
            userName = CleanSQL(userName);
            password = CleanSQL(password);

            // The stored procedure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand( ↙
    "spValidateUser", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@userName", SqlDbType.VarChar).Value = userName;
            sqlCmd.Parameters.Add("@password", SqlDbType.VarChar).Value = password;

            // The return value for the stored procedure
            System.Data.SqlClient.SqlParameter pRetValue = sqlCmd.Parameters.Add("@Ret",     ↙
    SqlDbType.Int);
            pRetValue.Direction = ParameterDirection.ReturnValue;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();

            return sqlCmd.Parameters["@Ret"].Value.ToString();
        }

        /// <summary>
        /// Creates a new session ID for the user
        /// </summary>
        /// <param name="userName">The user for which the session is being opened</param>
        /// <returns>The new session ID</returns>
        public long NewSessionID(String userName)
        {
            long sessionID;
            userName = CleanSQL(userName);

            ClearExpiredSessions(); // clear all expired sessions

            // The stored procecure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand( ↙
    "spNewSessionID", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@userName", SqlDbType.VarChar).Value = userName;

            // The return value for the stored procedure
            System.Data.SqlClient.SqlParameter pRetValue = sqlCmd.Parameters.Add("@Ret",     ↙
    SqlDbType.Int);
            pRetValue.Direction = ParameterDirection.ReturnValue;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();

            // Convert to long
            long.TryParse(sqlCmd.Parameters["@Ret"].Value.ToString(), out sessionID);

            return sessionID;
        }

        /// <summary>
        /// Checks whether the session ID is valid
        /// </summary>
        /// <param name="sessionID">The user's session ID</param>
        /// <returns>TRUE if valid, FALSE if invalid</returns>
        public bool ValidateSessionID(long sessionID)
        {
            ClearExpiredSessions(); // clear all expired sessions

            // The stored procecure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand(   ↙
    "spValidateSession", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@sessionID", SqlDbType.Int).Value = sessionID;

            // The return value for the stored procedure
            System.Data.SqlClient.SqlParameter pRetValue = sqlCmd.Parameters.Add("@Ret",      ↙
    SqlDbType.Int);
            pRetValue.Direction = ParameterDirection.ReturnValue;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();

            // Verify that the session ID is valid
            if (sqlCmd.Parameters["@Ret"].Value.ToString() == "1")
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        /// Posts a new message into the database
        /// </summary>
        /// <param name="sessionID">The user's session ID</param>
        /// <param name="title">Title of the new message</param>
        /// <param name="message">Message text</param>
        /// <returns>messageID of the new message</returns>
        public int PostNewMessage(long sessionID, String title, String message)
        {
            int messageID; // the new message ID

            // The stored procecure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand(   ↙
    "spPostNewMessage", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@sessionID", SqlDbType.Int).Value = sessionID;
            sqlCmd.Parameters.Add("@title", SqlDbType.VarChar).Value = title;
            sqlCmd.Parameters.Add("@message", SqlDbType.VarChar).Value = message;

            // The return value for the stored procedure
            System.Data.SqlClient.SqlParameter pRetValue = sqlCmd.Parameters.Add("@Ret",      ↙
    SqlDbType.Int);
            pRetValue.Direction = ParameterDirection.ReturnValue;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();

            // Convert the return string to int
            int.TryParse(sqlCmd.Parameters["@Ret"].Value.ToString(), out messageID);

            return messageID;
        }

        public void PostMsgAttachment(long sessionID, int messageID, String mediaType, String↙
     mediaLocation)
        {
            // The stored procecure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand(  ↙
    "spPostMsgAttachment", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@sessionID", SqlDbType.Int).Value = sessionID;
            sqlCmd.Parameters.Add("@messageID", SqlDbType.VarChar).Value = messageID;
            sqlCmd.Parameters.Add("@mediaType", SqlDbType.VarChar).Value = mediaType;
            sqlCmd.Parameters.Add("@mediaLocation", SqlDbType.VarChar).Value = mediaLocation;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();
        }

        /// <summary>
        /// Gets the user name associated with a session ID (this is used mostly for file    ↙
    storage)
        /// </summary>
        /// <param name="sessionID">The user's current session ID</param>
        /// <returns>The user's user name</returns>
        public String GetUserFromSession(long sessionID)
        {
            // The stored procecure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand(  ↙
    "spGetUserFromSession", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Add our parameters to the stored procedure
            sqlCmd.Parameters.Add("@sessionID", SqlDbType.Int).Value = sessionID;

            // Execute the stored procedure
            System.Data.SqlClient.SqlDataReader reader = sqlCmd.ExecuteReader();

            reader.Read();

            return reader["UserName"].ToString();
        }

        /// <summary>
        /// Clears all expired sessions from the database
        /// </summary>
        private void ClearExpiredSessions()
        {
            // The stored procedure we're executing
            System.Data.SqlClient.SqlCommand sqlCmd = new System.Data.SqlClient.SqlCommand(  ↙
    "spClearExpiredSessions", sqlConn);
            sqlCmd.CommandType = System.Data.CommandType.StoredProcedure;

            // Execute the stored procedure
            sqlCmd.ExecuteScalar();
        }

        /// <summary>
        /// Verifies SQL input to protect against injection attacks.
        /// </summary>
        /// <param name="sql">The SQL variable we'll be cleaning</param>
        private String CleanSQL(String sql)
        {
            // Make sure our SQL string is alphanumeric
            if (reg.IsMatch(sql))
            {
                return sql;
            }
            else {
                return "";
            }
        }
    }
}
```