

```
using System;
using System.Drawing;
using System.Diagnostics;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Lifetime;
using System.Threading;
using System.Windows.Forms;

namespace sb54_CSAI.remoteInterfaces
{
    #region RemotingObject CLASS
    /// <summary>
    /// RemotingObject is a sub class of a <see cref="System.Runtime.Remoting.
    MarshalByRefObject">MarshalByRefObject </see>
    /// that provides a remotable object across domain names. This class is used to control
    the
    /// <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain </see>
    /// <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">Embedded Media Player</see>
    /// This class also implements the <see cref="sb54_CSAI.remoteInterfaces.IClientRemoting
    ">IClientRemoting </see>
    /// interface
    /// </summary>
    public class RemotingObject : MarshalByRefObject, IClientRemoting
    {
        #region Instance Fields
        //Instance Fields
        private IServerRemoting mainForm = null;

        //Notifies a waiting thread that an event has occurred. This class cannot be
        inherited.
        private AutoResetEvent mediaResetEvent = new AutoResetEvent(false);
        private String track=null;

        #endregion
        #region Public Constructor
        /// <summary>
        /// Simply creates a new RemotingObject object
        /// </summary>
        public RemotingObject()
        {
        }
        #endregion
        #region Public Methods/Properties
        /// <summary>
        /// Set up the new lease time for this specific Remotable object
        /// </summary>
        /// <returns>An object (ILease) that the Remoting Leasetime has defined for it</
        returns>
        public override Object InitializeLifetimeService()
        {
            //get the initial ILease from base (MarshalByRefObject)
            ILease lease = (ILease)base.InitializeLifetimeService();
            //Then set up the new lease time for this specific Remotable object
            if (lease.CurrentState == LeaseState.Initial)
            {
                lease.InitialLeaseTime = TimeSpan.FromHours(12);
                lease.SponsorshipTimeout = TimeSpan.FromHours(24);
                lease.RenewOnCallTime = TimeSpan.FromHours(24);
            }
            return lease;
        }

        /// <summary>
        /// Sets the current track to be the <see cref="SB54_CSAI.MediaPlayerControl.
        uctMediaPlayer">
        /// Embedded Media Player</see>current track. If the track is different from the
        last
        /// track will release the Release the thread created at the ReMP3 Client, that
        deals with the
        /// retrieval of the current ReMP3 server media track.
    }
    #endregion
}

```

```

    /// <br></br>
    /// <br></br>
    /// See the public string getTrack() method of this object, for further details.
    /// </summary>
    /// <param name="mediaTrack">A string representing the new media item</param>
    public void setNewTrack(string mediaTrack)
    {
        //there must be a track, this would come from Media Player in real code
        if (!mediaTrack.Equals(String.Empty) && !mediaTrack.Equals(track))
        {
            //store item
            track=mediaTrack;
            //RELEASE the waiting thread by signalling it to continue
            mediaResetEvent.Set();
        }
    }

    /// <summary>
    /// Sets the mainform that this remotable object should communicate with
    /// </summary>
    public IServerRemoting theMainForm
    {
        set
        {
            mainForm = value;
        }
    }

    #endregion
    #region IClientRemoting implementation
    /// <summary>
    /// Implementation of IClientRemoting interface
    /// </summary>

    /// <summary>
    /// An initially dummy call to test that the Remoting channel has been setup
    correctly.
    /// This method is called simply to check the Remoting channel, but does not carry
    out any further
    /// functionality
    /// </summary>
    public void RemoteLoaderCall()
    {
        mainForm.DoLoaderCall();
    }

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
    call the
    /// cmdREW() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">
    Embedded Media Player</see>
    /// control
    /// </summary>
    public void RemoteExecuteREW()
    {
        mainForm.DoExecuteREW();
    }

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
    call the
    /// cmdFF() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">
    Embedded Media Player</see>
    /// control
    /// </summary>
    public void RemoteExecuteFF()
    {
        mainForm.DoExecuteFF();
    }

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to

```

```
call the
    /// cmdSTOP() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">
Embedded Media Player</see>
    /// control
    /// </summary>
public void RemoteExecuteSTOP()
{
    MainForm.DoExecuteSTOP();
}

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
call the
    /// addFiles() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer"
>Embedded Media Player</see>
    /// control. These files are added to the end of the already queued tracks of the
Media Player
    /// </summary>
    /// <param name="files">The files to add to the <see cref="SB54_CSAI.
MediaPlayerControl.uctMediaPlayer">
    /// Embedded Media Player</see></param>
public void RemoteExecuteADD_PLAY(string[] files)
{
    MainForm.DoExecuteADD_PLAY(files);
}

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
call the
    /// addFiles() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer"
>Embedded Media Player</see>
    /// control. These files will replace any queued tracks of the Media Player
    /// </summary>
    /// <param name="files">The files to add to the <see cref="SB54_CSAI.
MediaPlayerControl.uctMediaPlayer">
    /// Embedded Media Player</see></param>
public void RemoteExecuteCLEAR_PLAY(string[] files)
{
    MainForm.DoExecuteCLEAR_PLAY(files);
}

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
call the
    /// cmdPLAY() method of the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">
Embedded Media Player</see>
    /// control.
    /// </summary>
public void RemoteExecutePLAY()
{
    MainForm.DoExecutePLAY();
}

    /// <summary>
    /// Causes the <see cref="sb54_CSAI.ServerApp.FrmMain">ServerApp FrmMain</see> to
call the
    /// getTrackList() method of the <see cref="SB54_CSAI.MediaPlayerControl.
uctMediaPlayer">Embedded Media Player</see>
    /// control.
    /// </summary>
    /// <returns>A array of strings that represent the <see cref="sb54_CSAI.ServerApp.
FrmMain">ServerApp FrmMain</see>
    /// hosted <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer">Media Player</
see></returns>
public string[] RemoteGetPlayerTracks()
{
    return MainForm.DoGetPlayerTracks();
}

    /// <summary>
    /// Will be called by the <see cref="sb54_CSAI.ClientApp.FrmClient">ReMP3 Client</
see>
    /// on a special thread, that ONLY deals with the retrieval of the current media
```

```
player track from
    /// the server. The thread is created at the ReMP3 Client, then this method is
    called by using the
    /// client remotable object on the newly created thread at the client. This method
    then blocks the
    /// ReMP3 Client until such a time there is a new (different) track, which is set
    when the server
    /// sees a new mp3 within the <see cref="SB54_CSAI.MediaPlayerControl.uctMediaPlayer
">MediaPlayer</see>.
    /// This is then transmitted to the ReMP3 Client.
    /// When there is a new track the client thread is unblocked, and may retrieve the
    string name
    /// of the media player current track. The client thread, then recursively calls
    this method and the
    /// entire process starts again.
    /// </summary>
    /// <returns>A string representing the <see cref="SB54_CSAI.MediaPlayerControl.
uctMediaPlayer">MediaPlayer</see>
    /// current track</returns>
    public string getTrack()
    {
        //simply start loop
        while(true)
        {
            //AND WAIT, this will wait until there is a signal to continue, from the
            mediaResetEvent object
            mediaResetEvent.WaitOne();
            break;
        }
        return track;
    }

    #endregion
}
#endregion
}
```