GTCP Sample

Installation

The sample was written in Visual Studio 2002.
You need to unpack provided archive and open ***GTCP 1.0*** **>** ***Project.sln*** solution file. Then rebuild the solution.
Press ***F5*** to run both client and server in Visual Studio.

Configuration

Server IP address is specified in the ***Client*** **>** ***App.config*** file. You should directly modify ***Client*** **>** ***Bin*** **>** ***Debug*** **>** ***Client.exe.config*** file if you need to change the IP address without recompiling the solution.

Design

The sample implements a simple chat. Each client chooses its name after a starting, connects to the server, subscribes to the chat event and allows users to send messages to the server which, in its turn, re-send messages to all other clients (except the sender).

Known layer contains the declaration of the chat event provider and chat message receivers. Chat room interface is declared as a separate interface; hence this sample can be easily modified for having several chat rooms.

Server provides the business object bound to the known URI. Server implements Chat and Chat Room functionality. Server sends a message to all clients except the message sender.

Client implements event receiver interface and subscribes its listener. Client builds local transparent proxy pointed to the known URI to access server's business object. Client's code uses several GTCP features such as determining server restarting and it shows Genuine Channels events. So if you close the server and the start it again, clients re-subscribe to the chat event and continue receiving chat messages.

You can find more information about this approach here:
http://www.genuinechannels.com/Content.aspx?id=28&type=1

Issue

I used Broadcast Engine was used for implementing the event. You can find more information about Broadcast Engine in Programming Guide and in articles dedicated to events:
http://www.genuinechannels.com/Content.aspx?id=27&type=1
http://www.genuinechannels.com/Content.aspx?id=73&type=1

Take a look at the implementation of the ***Server*** > ***UserFilter*** class. It implements ***IMulticastFilter*** interface to filter out the message sender from the recipients list.
This filter is forced each time the event is fired by this call:

```
// force the filter to filter out the sender
using(new DispatcherFilterKeeper(
     new UserFilter(GenuineUtility.CurrentSession["UserId"] as
                                                         string)))
{
      IMessageReceiver iMessageReceiver =
            (IMessageReceiver)
this._dispatcher.TransparentProxy;
      iMessageReceiver.ReceiveMessage(message, nickname);
}
```