VHMsg

Overview

The Virtual Human Messaging library (VHMsg) defines a protocol and provides an API wrapper around ActiveMQ. It's intended to provide an easy solution for sending and receiving messages within the Virtual Human Architecture. It supports the following programming languages:

- C/C++
- C#
- Java
- TCL
- Lisp

In order for components to send or receive messages with each other, they need to:

1. be connected to the same ActiveMQ server, which is a Windows service that needs to be running
2. be operating within the same scope on that server

The VHMsg library sets the default server to 'localhost' and scope to 'DEFAULT_SCOPE'. If you installed all 3rd party software and are running the toolkit on your local machine, there should be nothing you need to do in order for everything to work. If you want to run a distributed system on several machines simultaneously, all the machines need to point to the same ActiveMQ server (e.g., 'vh1.ict.usc.edu') and the same scope (e.g., 'TOOLKIT_SCOPE'). To do this, create two environment variables, see below.

Messages are being broadcasted, meaning there is no specific destination. As a component, **subscribe** to the messages you want to receive. For an overview of the most common messages, see the Messaging section.

When creating new modules, be sure to implement the messaging protocol as defined here. This ensures components are aware of each other's existence and that they behave correctly.

Users

Changing the ActiveMQ server or scope

Change these by creating VHMSG_SERVER and VHMSG_SCOPE environment variables.

Checking ActiveMQ server and scope

This is found in the top right corner of the Logger, or in the Launcher after going to the Advanced menu and clicking on Show Information, at the bottom.

Developers

Implementing and using VHMsg in C++

Our C++ implementation is built on top of the C++ ActiveMQ libraries called CMS (http://activemq.apache.org/cms/). Headers and libraries are provided for Visual Studio. VS2010 is the currently maintained version.

There are two implementations that can be used for C++. The simplest implementation is based off the header found in \lib\vhmsg\vhmsg-c\include\vhmsg-tt.h and will be described below.

A sample using this implementation is included in \lib\vhmsg\samples\elbench\cpp\elbench.cpp. A condensed version of this sample is shown to give an idea of what's required:

```
#include "vhmsg-tt.h"

void tt_client_callback( char * op, char * args )
{
    printf( "received - '%s %s'\n", op, args );
}

int main()
{
    // set up our callback for receiving messages
    vhmsg::ttu_set_client_callback( tt_client_callback );

    // connect to the server
    err = vhmsg::ttu_open();
    if ( err == TTU_ERROR )
    {
        printf( "Connection error!\n" );
        return -1;
    }

    // register which message we’re interested in receiving
    err = vhmsg::ttu_register( "elbench" );

    // send a message
    vhmsg::ttu_notify2( "elbench", “Hello World” );

    // poll to receive messages.  callback function is called for each message received
    while ( !_kbhit() )
    {
        err = vhmsg::ttu_poll();
        if( err == TTU_ERROR )
        {
            printf( "ttu_poll ERR\n" );
        }
    }

    // cleanup
    vhmsg::ttu_close();
}
```

There are a number of Include directories and Libraries that need to be set up in order for this sample to compile and link properly. Also, note the .dlls that are required to be alongside the executable at runtime. Please see the Visual Studio solution provided in the elbench sample to see what's required. The required .dlls are copied in the Post-Build step.

Be sure to implement the messaging protocol as defined here.

Implementing and using VHMsg in C#

Our C# implementation is built on top of the C# ActiveMQ libraries called NMS (http://activemq.apache.org/nms/).

A elbench sample is included in \lib\vhmsg\samples\elbench\cs\elbench.cs. A condensed version of the sample is shown to give an idea of what's required:

```
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
{
   elbenchcs e = new elbenchcs();
   e.Run();
}


public void Run()
{
   VHMsg.Client vhmsg;
   using ( vhmsg = new VHMsg.Client() )
   {
      vhmsg.OpenConnection();

      vhmsg.MessageEvent += new VHMsg.Client.MessageEventHandler( MessageAction );

      vhmsg.SubscribeMessage( "elbench" );

      vhmsg.SendMessage( "elbench Hello World" );


      // Run your app, messages are received via a different thread
   }
}


private void MessageAction( object sender, VHMsg.Message args )
{
   // Note: Messages are received on a different thread.  Please lock accordingly, and pay extra careful while
calling UI commands.

   //Ict.ElvinUtility eu = (Ict.ElvinUtility)sender;
   //Console.WriteLine( "Received Message '" + args.toString() + "'" );
}
```

This example uses the 'immediate' method for retrieving messages. Note that it doesn't poll from messages, and messages are received on a different thread. Please be careful when handling these messages.

A 'polling' method is available to simplify handling of the messages, if desired. Please see EnablePollingMethod() and Poll() in the VHMsg documentation. \lib\vhmsg\vhmsg-net\doc\vhmsg-net.chm.

To use this library, compile with the ActiveMQ NMS libraries. See the Visual Studio project settings in the elbench sample for how to do this.

Be sure to implement the messaging protocol as defined here.

Implementing and using VHMsg in Java

Our Java implementation is built on top of the Java ActiveMQ libraries (http://activemq.apache.org).

An elbench sample is included in \lib\vhmsg\samples\elbench\java\src\elbench.java. A condensed version of the sample is shown to give you an idea of what's required:

```
import edu.usc.ict.vhmsg.*;


public class elbench implements MessageListener
{
   public static VHMsg vhmsg;


   public elbench()
   {
      vhmsg = new VHMsg();

      boolean ret = vhmsg.openConnection();
      if ( !ret )
      {
         System.out.println( "Connection error!" );
         return;
      }

      vhmsg.enableImmediateMethod();
      vhmsg.addMessageListener( this );
      vhmsg.subscribeMessage( "elbench" );

      vhmsg.sendMessage( "elbench Hello World" );

      // Run your app, messages are received via a different thread
   }


   public void messageAction( MessageEvent e )
   {
      // Note: Messages are received on a different thread.
      //       Please lock accordingly, and pay extra careful while calling UI commands.

      //System.out.println( "Received Message '" + e.toString() + "'" );
   }


   public static void main( String[] args )
   {
      elbench elbenchObj = new elbench();
   }
}
```

This example uses the 'immediate' method for retrieving messages. Note that it doesn't poll from messages, and messages are received on a different thread. Please be careful when handling these messages.

A 'polling' method is available to simplify handling of the messages. Please see EnablePollingMethod() and Poll() in the VHMsg documentation. lib\vhmsg\samples\elbench\java\dist\javadoc\index.html.

To use this library, add vhmsg-java.jar to your project. See the Netbeans project settings in the elbench sample for how to do this.

Be sure to implement the messaging protocol as defined here.

Known Issues

FAQ

See Main FAQ for frequently asked questions regarding the installer. Please use the Google Groups emailing list for unlisted questions.