Thrift & Eclipse & JUnit with TServlet

If you haven't downloaded thrift yet: http://thrift.apache.org/download/ # Download {{thrift-}}_{{version}}_{{.tar.gz}} and extract it somewhere, eg. "thrift-_version_" (from here referred to as {{$thrift-}}_{{version}}_). # For windows, download {{thrift-version.exe}}. and copy-paste it to {{C:\Windows}} then rename it {{thrift.exe}}. # For Linux: [ThriftInstallation] # At the command prompt, go to {{thrift-}}_{{version}}_{{/lib/java}} and execute {{ant}} to compile the source code into the generated {{build}} folder. \\ {toc} h2. Eclipse Editor # Help{{ -\-> }}Software Updates{{ -\-> }}Add Site... # URL: http://thrift4eclipse.sourceforge.net/updatesite/ # Tick: T``hrift4Eclipse # Click Install NOTE: It may take some time to install (5mins) h2. Standalone Example # In Eclipse: File{{ -\-> }}New{{ -\-> }}Other... (Ctrl+N){{ -\-> }}Dynamic Web Project... # Project name: {{ThriftExample}} # In {{$thrift-}}_{{version}}_{{/lib/java/build}} copy {{libthrift-version.jar}} to {{ThriftExample/WebContent/WEB-INF/lib}} # In {{$thrift-}}_{{version}}_{{/lib/java/build/lib}} copy all the jars into {{ThriftExample/WebContent/WEB-INF/lib}} # In Eclipse, under {{ThriftExample/Java Resources/src}} create the file {{example.thrift}} with the following content: \\ h3. example.thrift {{{#!thrift namespace java example struct [BeanExample] \{ 1: bool booleanPrimive; 2: byte bytePrimive; 3: i16 shortPrimive; 4: i32 intPrimive; 5: i64 longPrimive; 6: double doublePrimive; 7: string stringObject; 8: binary byteArray; //ByteBuffer \} service [ServiceExample] \{ [BeanExample] getBean(1: i32 anArg; 2: string anOther) \} }}} At the command prompt, got to folder containing {{example.thrift}} (eg. {{C:\workspace\ThriftExample\src}}) then execute the following command: {noformat} thrift --gen java -out . example.thrift {noformat} # In Eclipse "refresh" the {{ThriftExample}} project to show the newly created package {{example}} below {{src}}. # In the package {{example}} create the following files: \\ \\ h3. [ServiceExampleImpl].java {code:language=java} package example; import java.nio.ByteBuffer; import org.apache.thrift.TException; public class ServiceExampleImpl implements ServiceExample.Iface { @Override public BeanExample getBean(int anArg, String anOther) throws TException { return new BeanExample(true, (byte) 2, (short) 3, 4, 5, 6.0, "OK", ByteBuffer.wrap(new byte[] { 3, 1, 4 })); } } {code} h3. [ClientExample].java {code:language=java} package example; // import org.apache.thrift... etc.; public class ClientExample { private static final int PORT = 7911; public static void main(String[] args) { try { TTransport transport = new TSocket("localhost", PORT); Protocol protocol = new TBinaryProtocol(transport); ServiceExample.Client client = new ServiceExample.Client(protocol); transport.open(); BeanExample bean = client.getBean(1, "string"); transport.close(); System.out.println(bean.getStringObject()); } catch (TTransportException e) { e.printStackTrace(); } catch (TException e) { e.printStackTrace(); } } } {code} h3. [ServerExample].java {code:language=java} package example; // import org.apache.thrift... etc.; public class ServerExample implements Runnable { private static final int PORT = 7911; @Override public void run() { try { TServerSocket serverTransport = new TServerSocket(PORT); ServiceExample.Processor processor = new ServiceExample.Processor(new ServiceExampleImpl()); TServer server = new TThreadPoolServer(new TThreadPoolServer.Args(serverTransport).processor(processor)); System.out.println("Starting server on port " + PORT); server.serve(); } catch (TTransportException e) { e.printStackTrace(); } } public static void main(String[] args) { new Thread(new ServerExample()).run(); } } {code} # Execute the server (right-click {{ServerExample.java}}{{ -\-> }}Run as{{ -\-> }}Java Application) to see the message: {{Starting server on port 7911}} # Execute the client (right-click {{ClientExample.java}}{{ -\-> }}Run as{{ -\-> }}Java Application) to see the message: {{OK}} NOTE: To stop the server you'll need to kill the process via the console. h2. Unit Test # In Eclipse right-click the {{ThriftExample}} Project{{ -\-> }}New{{ -\-> }}Source Folder{{ -\-> }}Folder name: {{test}} # In {{test}} create the package {{example}} then in there create the following file: \\ \\ h3. [TestExample].java {code:language=java} package example; // imports... see last example for correct imports public class TestExample { private static final int PORT = 7911; @BeforeClass @SuppressWarnings({ "static-access" }) public static void startServer() throws URISyntaxException, IOException { // Start thrift server in a seperate thread new Thread(new ServerExample()).start(); try { // wait for the server start up Thread.sleep(100); } catch (InterruptedException e) { e.printStackTrace(); } } @Test public void testExample() throws TTransportException, TException { TTransport transport = new TSocket("localhost", PORT); TProtocol protocol = new TBinaryProtocol(transport); ServiceExample.Client client = new ServiceExample.Client(protocol); transport.open(); BeanExample bean = client.getBean(1, "string"); transport.close(); Assert.assertEquals("OK", bean.getStringObject()); } } {code} Run the test by right-clicking the file{{ -\-> }}Run as{{ -\-> }}JUnit Test h2. Thrift Servlet # In Eclipse: File{{ -\-> }}New{{ -\-> }}Servlet: # Project: {{ThriftExample}} # Java package: {{example}} # Class name: {{TServletExample}} Note: This step modifies the {{web.xml}} automatically adding : {{{#!xml TServletExample TServletExample example.TServletExample TServletExample /TServletExample }}} Replace the generated code with the following: h3. TServletExample.java {code:language=java} package example; import org.apache.thrift.protocol.TCompactProtocol; import org.apache.thrift.server.TServlet; public class TServletExample extends TServlet { public TServletExample() { super(new ServiceExample.Processor(new ServiceExampleImpl()), new TCompactProtocol.Factory()); } } {code} # Right-click the {{TServletExample.java}}{{ -\-> }}Run as{{ -\-> }}Run on server (hopefully you've already got a server configured ;o) # You should get the following error (don't worry this is good!): {noformat} Error 500 javax.servlet.ServletException: org.apache.thrift.transport.TTransportException org.apache.thrift.server.TServlet.doPost(TServlet.java:86) org.apache.thrift.server.TServlet.doGet(TServlet.java:96) javax.servlet.http.HttpServlet.service(HttpServlet.java:617) javax.servlet.http.HttpServlet.service(HttpServlet.java:717) {noformat} This is because we haven't created an HTTP client to communicate with the {{TServlet}}. Note the URL if it's not the same as this one: {{http://localhost:8080/ThriftExample/TServletExample}} You'll have to replace it in the following code if it's different: h2. TServlet Unit Test Add the following method (replacing the URL if needs be): {code:language=java} @Test public void testServlet() throws TTransportException, TException { // TODO : change this URL if it's not the right one ;o) // TODO : change this URL if it's not the right one ;o) // TODO : change this URL if it's not the right one ;o) String servletUrl = "http://localhost:8080/ThriftExample/TServletExample"; THttpClient thc = new THttpClient(servletUrl); TProtocol loPFactory = new TCompactProtocol(thc); ServiceExample.Client client = new ServiceExample.Client(loPFactory); BeanExample bean = client.getBean(1, "string"); Assert.assertEquals("OK", bean.getStringObject()); } {code} Now run the unit tests again to test the {{TServlet}}: Right-click {{TestExample.java}}{{ -\-> }}Run as{{ -\-> }}JUnit Test. h2. Secure Thrift Servlet Securing a servlet is done via the web server. It's up to you to configure your web server to control the communication securely (via https rather than http). How to set that up is outside the scope of this tutorial. Please refer to your web server's documentation. Also outside the scope of this tutorial is creating the security certificates and setting-up the Java keystore. However, once you've you've got all that stuff in place. You're ready to unit test your secure* Thrift servlet (which hasn't changed!). h2. Secure Unit Test {warning} *Security Warning* You won't want to use the {{AnyCertTrustManager}} in a production environment, but it's sometimes necessary if the test server's domain differs from the one specified on the certificat issued. Trust stores and keystores are practically the same thing. {warning} Here's the whole unit test : {code:language=java} package example; import java.io.File; import java.io.FileInputStream; import java.security.KeyStore; import java.security.cert.CertificateException; import java.security.cert.X509Certificate; import javax.net.ssl.KeyManagerFactory; import javax.net.ssl.SSLContext; import javax.net.ssl.TrustManager; import javax.net.ssl.X509TrustManager; import junit.framework.Assert; import org.apache.http.HttpVersion; import org.apache.http.conn.params.ConnManagerParams; import org.apache.http.conn.params.ConnPerRouteBean; import org.apache.http.conn.scheme.Scheme; import org.apache.http.conn.scheme.SchemeRegistry; import org.apache.http.conn.ssl.SSLSocketFactory; import org.apache.http.impl.client.DefaultHttpClient; import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager; import org.apache.http.params.BasicHttpParams; import org.apache.http.params.HttpConnectionParams; import org.apache.thrift.TException; import org.apache.thrift.protocol.TBinaryProtocol; import org.apache.thrift.protocol.TCompactProtocol; import org.apache.thrift.protocol.TProtocol; import org.apache.thrift.transport.THttpClient; import org.apache.thrift.transport.TSocket; import org.apache.thrift.transport.TTransport; import org.junit.BeforeClass; import org.junit.Test; public class TestExample { private static final int PORT = 7911; private static final int PORT_SSL = 443; private static Scheme https; private static BasicHttpParams params; private static ThreadSafeClientConnManager cm; @BeforeClass @SuppressWarnings({ "static-access" }) public static void startServer() throws Exception { // Start thrift server in a seperate thread new Thread(new ServerExample()).start(); try { // wait for the server start up Thread.sleep(100); } catch (InterruptedException e) { e.printStackTrace(); } } @BeforeClass @SuppressWarnings("deprecation") public static void initConnectionManager() throws Exception { // TODO : specify where your keystore is File keystoreFile = new File(" ... /keystore.jks"); // TODO : specify your password to the key pair (not the keystore) char[] password = "yourPassword".toCharArray(); String keystoreType = KeyStore.getDefaultType(); // "jks" KeyStore keyStore = KeyStore.getInstance(keystoreType); FileInputStream instream = new FileInputStream(keystoreFile); try { keyStore.load(instream, password); } finally { try { instream.close(); } catch (Exception ignore) { } } KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm()); kmf.init(keyStore, password); // TODO : Specify un proper trust store. Don't use this in production! SSLContext sslContext = SSLContext.getInstance("TLS"); sslContext.init(kmf.getKeyManagers(), new TrustManager[] { new AnyCertTrustManager() }, null); SSLSocketFactory ssf = new SSLSocketFactory(sslContext, SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER); https = new Scheme("https", PORT_SSL, ssf); SchemeRegistry schemeRegistry = new

SchemeRegistry(); schemeRegistry.register(https); // Set up Thrift HTTP client connection parameters params = new BasicHttpParams(); params.
setParameter("http.protocol.version", HttpVersion.HTTP_1_1); params.setParameter("http.protocol.content-charset", "UTF-8"); // Disable Expect-Continue
params.setParameter("http.protocol.expect-continue", false); // Enable staleness check params.setParameter("http.connection.stalecheck", true);
HttpConnectionParams.setSoTimeout(params, 10000); // 10 secondes HttpConnectionParams.setConnectionTimeout(params, 10000); // 10 secondes
ConnManagerParams.setMaxTotalConnections(params, 20); ConnPerRouteBean connPerRoute = new ConnPerRouteBean(20); ConnManagerParams.
setMaxConnectionsPerRoute(params, connPerRoute); cm = new ThreadSafeClientConnManager(params, schemeRegistry); } private static class
AnyCertTrustManager implements X509TrustManager { @Override public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException { } @Override public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException { } @Override
public X509Certificate[] getAcceptedIssuers() { return null; } } @Test public void testExample() throws TException { TTransport transport = new TSocket
("localhost", PORT); TProtocol protocol = new TBinaryProtocol(transport); ServiceExample.Client client = new ServiceExample.Client(protocol); transport.
open(); BeanExample bean = client.getBean(1, "string"); transport.close(); Assert.assertEquals("OK", bean.getStringObject()); } @Test public void
testServlet() throws TException { // TODO : change this URL if it's not the right one ;o) // TODO : change this URL if it's not the right one ;o) // TODO :
change this URL if it's not the right one ;o) String servletUrl = "http://localhost:8080/ThriftExample/TServletExample"; THttpClient thc = new THttpClient
(servletUrl); TProtocol loPFactory = new TCompactProtocol(thc); ServiceExample.Client client = new ServiceExample.Client(loPFactory); BeanExample
bean = client.getBean(1, "string"); Assert.assertEquals("OK", bean.getStringObject()); } @Test public void testSecureServlet() throws TException { //
TODO : change this URL if it's not the right one ;o) // TODO : change this URL if it's not the right one ;o) // TODO : change this URL if it's not the right one ;
o) String servletUrl = "https://localhost:8080/ThriftExample/TServletExample"; THttpClient thc = new THttpClient(servletUrl, new DefaultHttpClient(cm,
params)); TProtocol loPFactory = new TCompactProtocol(thc); ServiceExample.Client client = new ServiceExample.Client(loPFactory); BeanExample
bean = client.getBean(1, "string"); Assert.assertEquals("OK", bean.getStringObject()); } } {code}