Configurando Cliente de Web Service HTTPS com Eclipse

Sistema RDR Web Service

Versão 1.0.0 (revisão: 141)

Sumário

1 Instruções iniciais

Para esta configuração foi utilizado o Eclipse SDK Helios SR1 com o JRE versão 6 atualização 22 (1.6.0_22-b04) com sistema Windows XP Professional.

Siga a ordem dos tópicos a seguir para concluir a configuração do Cliente.

Se você tem certeza de que **nunca** importou os certificados do BACEN, necessários para acesso ao WS, pode ir direto para o passo **Importando os certificados de acesso** na página 10, antes de iniciar qualquer configuração do cliente.

Neste manual, são descritos duas maneiras de gerar um cliente de Web Services em Java:

- Implementando cliente WS com AXIS e Eclipse SDK; e

- Implementando cliente com JAX-WS (JDK 1.6).

Escolha a configuração que melhor se aplica a sua organização, e siga as orientações.

Todos os recursos do Web Service (inclusive o WSDL) são protegidos por usuário e senha de acesso. As credenciais de acesso são as mesmas utilizadas no sistema RDR/SISCAP e devem ser fornecidas no formato **<instituição><dependência>.<usuário>** (exemplo: **052377975.cef01244**). O usuário do Web Service, deve estar credenciado nas transações SSCP001 e/ou SSCP002 do SISBACEN no ambiente de homologação. Usuários que já possuem acesso ao RDR/SISCAP, automaticamente já possuem acesso também aos serviços do Web Service RDR e vice-versa.

2 Implementando cliente com JAX-WS (JDK 1.6)

JAX-WS é a especificação de Web Service padrão já inclusa no JDK 1.6 ou superior. É possível utilizar esta especificação com versões anteriores do Java, bastando incluir as bibliotecas corretas no classpath, porém este procedimento não será abordado.

Importante ressaltar que antes de iniciar este tópico, é necessário fazer a importação dos certificados (página 10 - Importando os certificados de acesso) para a JRE utilizada pelo JDK 1.6 instalado na máquina.

Para este exemplo, foi utilizada a JDK versão 1.6.0 atualização 22 (1.6.0_22).

2.1 Criando um projeto Java no Eclipse

Antes de iniciar a geração do cliente, crie um projeto simples no Eclipse, utilizando JRE ou JDK 1.6. Caso tenha alguma dúvida quanto ao procedimento vá até ao tópico “Criando um projeto Básico no Eclipse“ na página 7.

2.2 Criando arquivo de autenticação do Web Service

Na raiz do projeto que foi criado no passo anterior, crie um arquivo com o nome de “authfile.txt” com o conteúdo a seguir. Este arquivo será utilizado pela ferramenta *wsimport* do JDK para acessar o WSDL e realizar a geração das classes.

https://**usuario**:**senha**@www9.bcb.gov.br/hml/rdrws/services/InstituicaoService?wsdl

Substitua o texto em negrito pelo usuário e a senha do SISBACEN utilizada para acesso ao Web Service. Observe que a parte final refere-se ao WSDL a partir do qual será gerado as classes de acesso ao serviço.

2.3 Iniciando a geração das classes

- Abra o Prompt de Comando (Iniciar – Executar, cmd, Enter);
- Navegue até a pasta raiz do projeto que foi criado no passo 2.1 (Exemplo: cd D:\workspace\clientwsjax);

A ferramenta que faz a geração do cliente não faz parte da JRE e sim do JDK, por isso, normalmente em uma instalação padrão, o JDK não está incluso no PATH do Windows. Neste caso, recomenda-se referenciar diretamente a ferramenta que se encontra em %JAVA_HOME%\bin\wsimport, ou então criar a variável de ambiente JAVA_HOME apontando para um JDK 1.6 válido. A partir daqui, a pasta raiz do JDK será referenciada como %JAVA_HOME%, para facilitar o entendimento.

- Supondo que no passo 2.1 foi criado um projeto com as configurações padrões, então a pasta de fontes é a “src” e as classes são compiladas para a pasta “bin”.
- Digite no prompt o seguinte comando:

```
D:\workspace\clientwsjax>"%JAVA_HOME%\bin\wsimport" -extension -s src -d bin
-Xauthfile "authfile.txt" https://www9.bcb.gov.br/hml/rdrws/services/InstituicaoService?
wsdl
```

- o parâmetro *-extension* ignora os avisos durante a interpretação do WSDL;
- *-s* informa o diretório onde será colocado os arquivos .java gerados;
- *-d* informa o diretório onde sera colocado os arquivos compilados;
- -Xauthfile informa o arquivo que foi criado no passo 2.2 para autenticação.

2.4 Realizando a chamada ao Web Service

Atualize o projeto do Eclipse para onde foram geradas as classes de acesso ao Web Service, clicando sobre o mesmo e logo em seguida em “Refresh”.

Crie a classe de exemplo abaixo, dentro deste projeto (preferencialmente fora dos pacotes gerados pelo JDK).

```
import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.GregorianCalendar;
import java.util.List;

import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;

import br.gov.bcb.rdr.siscapws.objetos.xsd.DemandaOUT;
import br.gov.bcb.rdr.siscapws.objetos.xsd.SituacaoOUT;
import br.gov.bcb.rdr.siscapws.objetos.xsd.TipoPesquisaOUT;
import br.gov.bcb.rdr.siscapws.services.GetSituacoesResponse;
import br.gov.bcb.rdr.siscapws.services.GetTiposPesquisaResponse;
import br.gov.bcb.rdr.siscapws.services.InstituicaoService;
import br.gov.bcb.rdr.siscapws.services.InstituicaoServicePortType;



public class Main {
    public static void main(String[] args) throws Exception {

        //Coloque a senha de acesso ao sisbacen aqui
        Authenticator.setDefault(new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                String user = "usuario";
                String password = "senha";
                return new PasswordAuthentication(user, password.toCharArray());
            }
        });


        InstituicaoService service = new InstituicaoService();
        InstituicaoServicePortType endpoint =
service.getInstituicaoServiceHttpSoap11Endpoint();
        GetSituacoesResponse situacoes = endpoint.getSituacoes();
        List<SituacaoOUT> lista = situacoes.getReturn();
        for (SituacaoOUT situacaoOUT : lista) {
            System.out.println(situacaoOUT.getDescricao().getValue());
        }

        GetTiposPesquisaResponse tiposPesquisaResponse =
endpoint.getTiposPesquisa();
        List<TipoPesquisaOUT> tiposPesquisa = tiposPesquisaResponse.getReturn();
        for (TipoPesquisaOUT tipoPesquisaOUT : tiposPesquisa) {
            System.out.println(tipoPesquisaOUT.getCodigo().getValue() + " " +
tipoPesquisaOUT.getDescricao().getValue());
        }

        DatatypeFactory factory = DatatypeFactory.newInstance();

        XMLGregorianCalendar dataInicio = factory.newXMLGregorianCalendar(new
GregorianCalendar(2009, 5,05, 0, 0, 0));
        XMLGregorianCalendar dataFim = factory.newXMLGregorianCalendar(new
GregorianCalendar(2011, 5,05, 0, 0, 0));
```

```
        List<DemandaOUT> demandas = endpoint.getDemandas(dataInicio, dataFim,
"A", null, null);
        for (DemandaOUT demandaOUT : demandas) {
            System.out.println(demandaOUT.getIdInterno().getValue() +"
"+demandaOUT.getNumeroDemanda().getValue());
        }
    }
}
```

2.5 JAX-WS com Maven

É possível utilizar o Maven, para realizar a geração do cliente de web service, descrito nos passos anteriores. Para isso basta incluir o plugin abaixo com as configurações, no pom.xml:

```
<plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <version>1.12</version>
      <executions>
            <execution>
                  <goals>
                        <goal>wsimport</goal>
                  </goals>
                  <configuration>
                        <wsdlUrls>

      <wsdlUrl>https://www9.bcb.gov.br/hml/rdrws/services/InstituicaoService?
wsdl</wsdlUrl>
                        </wsdlUrls>
                        <extension>true</extension>
                        <verbose>true</verbose>
                        <xauthFile>authfile.txt</xauthFile>
                  </configuration>
            </execution>
      </executions>
</plugin>
```

É possível também gerar as classes do cliente utilizando todos os WSDL dos serviços suportados pelo Web Service, repetindo o elemento “wsdlUrl” para cada serviço do RDR Web Service. Mas atenção, há um problema conhecido ao ler vários WSDLs com o mesmo *namespace* – veja mais na página 16 - “There's no ObjectFactory with an @XmlElementDecl for the element“.

3 Implementando cliente WS com AXIS e Eclipse SDK

3.1 Criando um projeto Básico no Eclipse

Antes de mapear as classes do web services é necessário criar um projeto Java no Eclipse.

- Clique em **File / New / Java Project**;
- Forneça um nome para o projeto (sugestão: *clienttest*);
- Clique **Next**;
- Clique em **Finish** para finalizar a configuração.

3.2 Verificando a JRE utilizada pelo Eclipse

- Clique em **Window / Preferences**;
- Procure pela aba **Java / Installed JREs**. Nesta aba aparecerá todas as JVMs instaladas no sistema. A JRE padrão utilizada pelo Eclipse é a que estiver marcada.

É possível verificar também por este outro caminho:

- Clique em **Help / About Eclipse**;
- Agora em **Installation Details**;
- Clique a aba **Configuration**;
- procure pelo parâmetro -vm e logo abaixo observe o diretório da VM que o Eclipse está utilizando (Exemplo: *C:\Arquivos de programas\Java\jre6\bin\client\jvm.dll*).

Anote o caminho onde a JRE padrão do Eclipse está instalada, pois será utilizado mais tarde.

3.3 Adicionando Web Services Client no Eclipse SDK

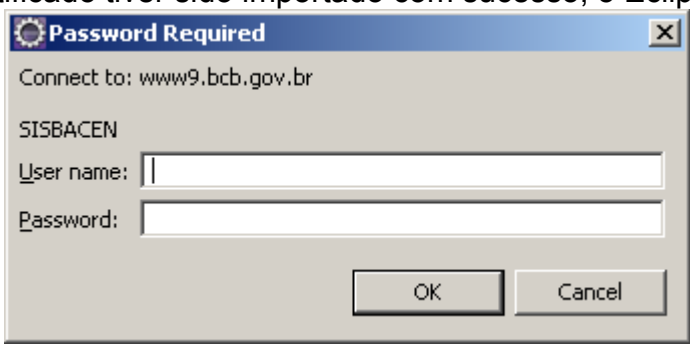
Para gerar as classes do Web Service:

- Clique em **File / New / Other...**;
- Procure pelo wizard **Web Services / Web Service Client** e clique duas vezes no mesmo;
- No indicador que aparece, arraste o mesmo até o nível **Develop client** como mostra a figura abaixo;

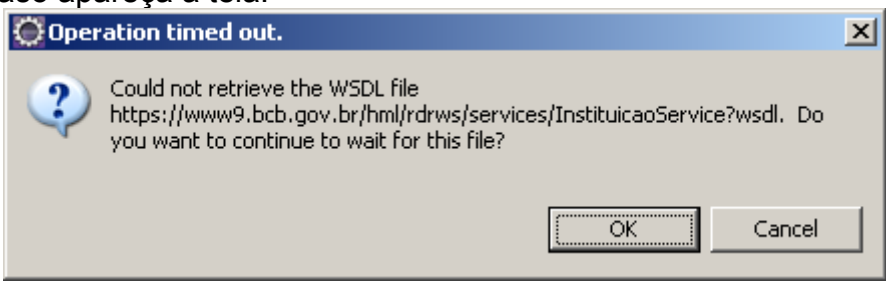
- No campo **Service definition** coloque a URL que possui o WSDL (neste exemplo use: https://www9.bcb.gov.br/hml/rdrws/services/InstituicaoService?wsdl);
- Ao terminar de digitar o endereço ou mover o foco para fora do campo, o Eclipse deverá Exibir uma tela com usuário e senha a serem informados ou a mensagem de erro **The service definition selected is invalid**; caso esta mensagem tenha sido exibida pelo Eclipse, vá para o tópico **Importando os**

certificados de acesso (página 10) e ao finalizar este passo adicional, feche o Eclipse e inicie novamente, retomando a configuração até este ponto.

- Se o certificado tiver sido importado com sucesso, o Eclipse mostrará a tela:



- forneça o usuário e a senha;
- Clique OK;
- Caso apareça a tela:



- Basta clicar OK e continuar normalmente;
- Na sessão de **Configuration:** selecione os seguintes itens:
  - Web service runtime: Apache Axis
  - Client project: clienttest

observe que o projeto informado deve ser o mesmo projeto criado no passo **Criando um projeto Básico no Eclipse** (página 7);

- Clique em **Next**;
- Clique em **Finish** e aguarde a criação e geração das classes do Web Service pelo Eclipse;

3.4 Realizando a primeira chamada ao Web Service

Quando a geração das classes estiver finalizada, é necessário adicionar o usuário e a senha no código gerado pelo Eclipse, antes de criar a classe de teste do Web Service.

- Procure pela classe que possui a terminação **BindingStub.java** normalmente ela possui o nome do serviço importado pelo WSDL, exemplo **InstituicaoService***Soap11BindingStub.java*;
- Procure pelo método createCall()
- Procure pelas linhas de código, dentro deste método:

```
if (super.cachedUsername != null) {
    _call.setUsername(super.cachedUsername);
}
if (super.cachedPassword != null) {
    _call.setPassword(super.cachedPassword);
}
```

e altere-as para que fiquem parecidas com:

```
_call.setUsername("052377975.cef1234");
_call.setPassword("sua senha aqui");
```

3.5 Criando classe com o método main

Crie uma nova classe no Eclipse dentro do projeto *clienttest* com o método main com a seguinte implementação:

```
    public static void main(String[] args) throws Exception {
      Calendar dataFim = Calendar.getInstance();
      Calendar dataInicio = Calendar.getInstance();
      dataInicio.set(Calendar.YEAR, 2009);
      String tipoConsulta = "A";
      Integer totalDemandas = new
InstituicaoServiceLocator().getInstituicaoServiceHttpSoap11Endpoint().getTotalDe
mandas(dataInicio, dataFim, tipoConsulta, null);
      System.out.println(totalDemandas);
  }
```

Execute a classe utilizando a mesma JRE que contém os certificados necessários. Este código é a consulta do total de demandas existentes para os parâmetros informados, este total é impresso no console.

Aqui termina a configuração do cliente, lembrando que é apenas uma configuração de teste e que podem haver particularidades do ambiente que fogem do escopo deste documento e que devem ser tratadas por conta e risco de quem está implementando.

4 Importando os certificados de acesso

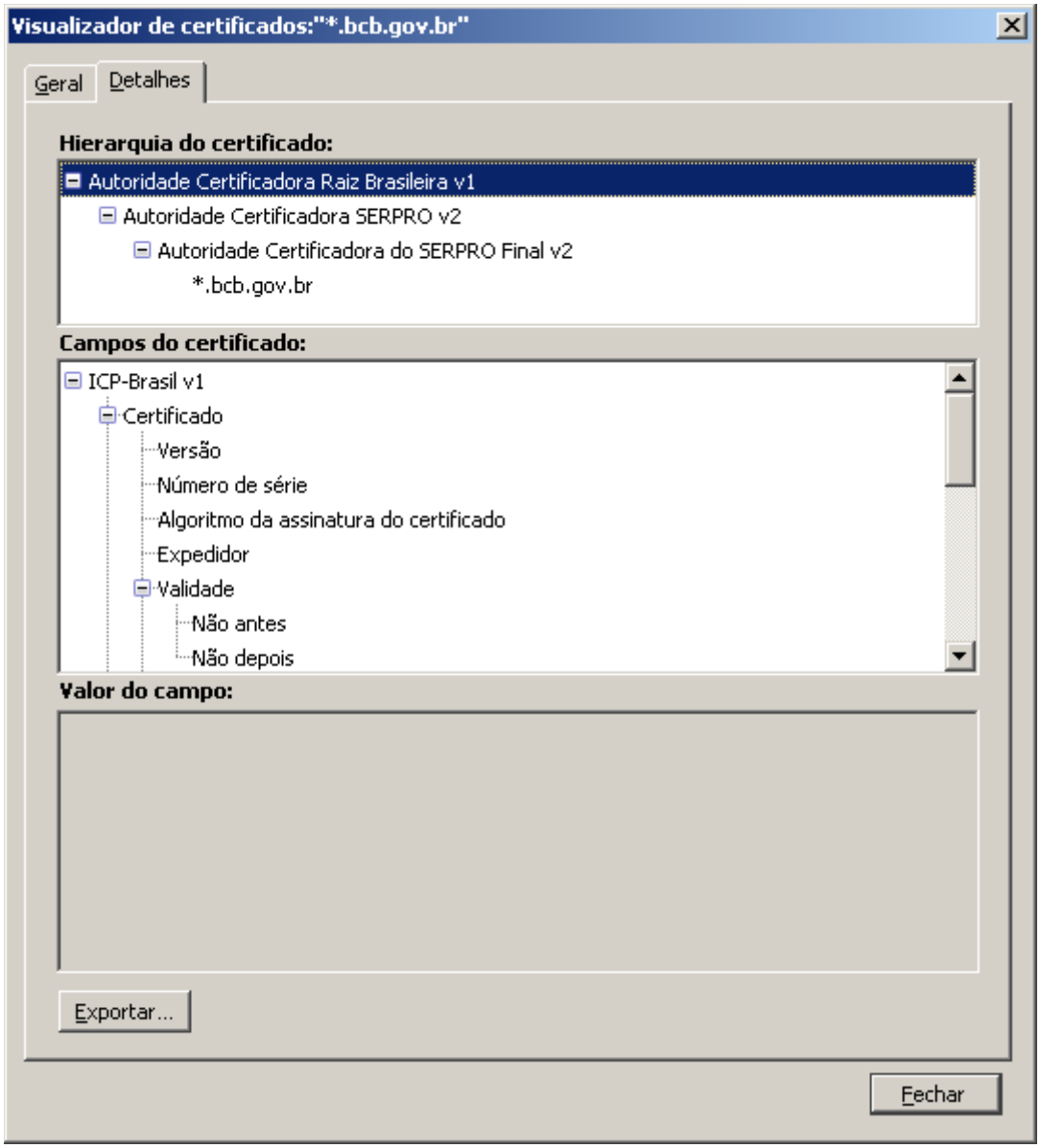
Para importar os certificados de acesso para a JVM padrão do Eclipse, siga os passos adiante, lembrando que este procedimento é necessário somente uma vez por JVM.

Considere que a JVM padrão utilizada pelo Eclipse está instalada em C:\Arquivos de programas\Java\jre6\.

4.1 Baixando os certificados

- Com o browser Firefox, acesse o endereço do Web Service - https://www9.bcb.gov.br/hml/rdrws/ ;
- Clique no ícone ao lado esquerdo do endereço do WS

- Clique em **Mais informações...**;
- Na tela que aparece, clique em **Exibir certificado**;
- Clique na aba **Detalhes**;



- Exporte **todos** os certificados da Hierarquia clicando sobre o certificado e logo após no botão **Exportar...**;
- Salve o certificado no diretório *%JAVA_HOME%\lib\security*, lembrando que este JAVA_HOME deve ser a mesma JVM utilizada pelo Eclipse (Exemplo: *C:\Arquivos de programas\Java\jre6\lib\security*)
- Se os certificados necessários não tiverem sofrido nenhuma mudança, os seguintes certificados foram encontrados e baixados para sua máquina:
  - -.bcb.gov.br.crt
  - AutoridadeCertificadoradoSERPROFinalv2.crt
  - AutoridadeCertificadoraRaizBrasileirav1.crt
  - AutoridadeCertificadoraSERPROv2.crt

4.2 Importando os certificados para a JVM padrão

- Abra o prompt de comando (Iniciar / Executar, cmd, Enter). Atenção, é necessário executar o prompt de comando com privilégios de administrador.
- Navegue até o diretório da JVM padrão, digitando o comando **cd C:\Arquivos de programas\Java\jre6\bin**;

- Faça um backup copiando o arquivo original **C:\Arquivos de programas\Java\jre6\lib\security\cacerts** para outro lugar, pois este será modificado ao importar os certificados.
- Supondo que foram baixados os quatro certificados no passo anterior, repita o comando abaixo quatro vezes, uma para cada certificado, substituindo o nome em negrito pelo nome do arquivo sem a extensão:

keytool -import -trustcacerts -keystore ../lib/security/cacerts -storepass changeit -noprompt -alias **nome_do_certificado** -file ../lib/security/**nome_do_certificado**.crt

Exemplo:

C:\Arquivos de programas\Java\jre6\bin>**keytool -import -trustcacerts -keystore ../lib/security/cacerts -storepass changeit -noprompt -alias AutoridadeCertificadoraRaizBrasileirav1 -file ../lib/security/AutoridadeCertificadoraRaizBrasileirav1.crt**

C:\Arquivos de programas\Java\jre6\bin>**keytool -import -trustcacerts -keystore ../lib/security/cacerts -storepass changeit -noprompt -alias AutoridadeCertificadoraSERPROv2 -file ../lib/security/AutoridadeCertificadoraSERPROv2.crt**

C:\Arquivos de programas\Java\jre6\bin>**keytool -import -trustcacerts -keystore ../lib/security/cacerts -storepass changeit -noprompt -alias AutoridadeCertificadoradoSERPROFinalv2 -file ../lib/security/AutoridadeCertificadoradoSERPROFinalv2.crt**

C:\Arquivos de programas\Java\jre6\bin>**keytool -import -trustcacerts -keystore ../lib/security/cacerts -storepass changeit -noprompt -alias -.bcb.gov.br -file ../lib/security/-.bcb.gov.br.crt**

Quando o comando for executado corretamente a seguinte mensagem é exibida:
Certificate was added to keystore

5 Respondendo demandas com anexos

A operação a seguir está disponível no serviço AtualizarDemandaService deste Web Service, o que faz necessário gerar as classes no Eclipse para este serviço, seguindo os passos descritos na página 7 deste manual utilizando o endereço correto do serviço de atualização de demandas.

5.1 Operação responderBase64

Envia o anexo da resposta codificado em Base 64.

Todo Web Service faz uso do XML como formato de transporte dos dados entre o cliente e o servidor, porém, o XML possui algumas restrições quanto ao conteúdo que pode ser colocado; alguns caracteres quando adicionados em um XML torna-o mal formado e nenhuma ferramenta é capaz de interpretá-lo (http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char) .

Esta operação recebe o conteúdo do arquivo dentro do corpo do XML enviado ao Web Service, porém, para que não ocorra erro no processamento do XML, é necessário converter o conteúdo do arquivo para Base 64 antes de enviar a resposta. Todas as linguagens de desenvolvimento possuem API nativa para codificação e decodificação de conteúdo em Base 64, por isso esta técnica é mais fácil de implementar.

A conversão em base 64 resulta em um conteúdo baseado no conjunto de caracteres ([A-Za-z0-9], "/" e "+") que somam um total de 64 caracteres válidos para um XML. A codificação Base64 provoca um aumento global de 33% do volume dos dados a codificar, por isso pode representar uma sensível perda de performance no processamento (codificação/decodificação) e no tráfego de rede também.

Este trecho de código a seguir é um exemplo de implementação do cliente que envia a resposta para o Web Service utilizando a operação em questão.

```
public class ResponderDemandaBase64 {
    public static void main(String[] args) throws Exception {
        AtualizarDemandaServiceLocator locator = new
AtualizarDemandaServiceLocator();
        AtualizarDemandaServicePortType endpoint =
locator.getAtualizarDemandaServiceHttpSoap11Endpoint();
        String resposta = "Respondendo o encaminhamento 219216 pelo WS";
        Integer idEncaminhamento = new Integer(219216);

        File arquivo = new File("C:\\Pré-projeto.pdf");
        FileInputStream arquivoIS = new FileInputStream(arquivo);
        ByteArrayOutputStream saida = new ByteArrayOutputStream();
        copia(saida, arquivoIS);

        byte[] conteudoStringArquivo = saida.toByteArray();

        String nomeArquivo = arquivo.getName();
        String contentType = arquivo.toURL().openConnection().getContentType();
        String conteudoBase64 = Base64.encode(conteudoStringArquivo);
        if("content/unknown".equals(contentType)){
            contentType = "application/octet-stream";
        }


        endpoint.responderBase64(resposta, idEncaminhamento, nomeArquivo,
contentType, conteudoBase64);
    }
    /**
     * copiar uma stream para outra
     * @param destino
     * @param origem
     * @throws IOException
     */
    public static void copia(OutputStream destino, InputStream origem)throws
IOException {
        byte[] buffer = new byte[8192];
        int qtdeLida = 0;

        while ((qtdeLida = origem.read(buffer)) > 0) {
            destino.write(buffer, 0, qtdeLida);
        }

        destino.close();
        origem.close();
    }
}
```

6 Recuperando anexos do WS

Considerando uma plataforma em Java, os anexos de cada demanda podem ser recuperados através do componente Commons HTTP Client e Commons Codec da Apache. Logo abaixo, tem-se um exemplo de como recuperar um anexo utilizando estes componentes.

O exemplo abaixo utilizou o commons-httpclient versão 3.1 e o commons-codec 1.3.

Lembrando que para conectar-se com sucesso ao servidor é necessário importar os certificados de acesso para a JVM a ser utilizada (página 10).

```
public class AnexosDownload {
    public static void main(String[] args) throws Exception {
        HttpClient client = new HttpClient();

        //Autenticação do WS, aqui deve ser informado o usuário e a senha para
acesso ao SISBACEN.
        client.getState().setCredentials(new AuthScope("www9.bcb.gov.br", 443,
"SISBACEN"), new UsernamePasswordCredentials("usuário sisbacen", "senha"));

        //essa URL é montada pelo WS, aqui é apenas um exemplo de url onde o
anexo está disponível
        GetMethod get = new GetMethod("https://www9.bcb.gov.br/hml/rdrws/anexos?
id=279083");
        get.setDoAuthentication(true);
        client.executeMethod(get);

        //Nome do arquivo
        String nomeArquivo = get.getResponseHeaders("Content-Disposition")
[0].getElements()[0].getParameterByName("filename").getValue();

        //Conteúdo do arquivo
        InputStream arquivo = get.getResponseBodyAsStream();

        get.releaseConnection();

    }
}
```

7 Problemas conhecidos

Configurando o Web Services client com este tutorial foram detectados alguns problemas que podem ocorrer ao realizar as chamadas. Alguns destes problemas são descritos aqui.

7.1 Validade do XML

Web Services utilizam XML como formato de intercomunicação de dados, porém, é importante observar que o conteúdo do texto deve conter apenas caracteres permitidos para que não ocorra erro no processamento do XML. Os caracteres válidos para um XML estão descritos na especificação da W3C - http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char .

7.2 Invalid element org.apache.axis.encoding.ser.BeanDeserializer.onStartChild

Existe um bug cadastrado para este erro em https://issues.apache.org/jira/browse/AXIS-2758 , mas como o AXIS 1.4 não é mais evoluído, não existe versão com o BUG corrigido. A versão 2 é a mais recente do framework AXIS, mas a forma de configuração e arquitetura são totalmente diferentes da versão utilizada neste manual, que é a primeira.

Para correção do erro é necessário alterar uma das classes geradas pelo Eclipse.

- Procure pela classe com terminação *BindingStub*;
- Esta classe possui um método de inicialização das definições dos serviços - _initOperationDesc1;
- Na implementação deste método, procure pela inicialização de operações do WS que retornem uma coleção de objetos;
- Altere a chamada ao método void org.apache.axis.description.OperationDesc.setReturnClass(Class returnClass) colocando a classe simples, e não o array da classe, como o Eclipse gerou inicialmente;

Antes:

```
oper.setReturnClass(br.gov.bcb.rdr.siscapws.objetos.xsd.DemandaOUT[].class);
```

Depois:

```
oper.setReturnClass(br.gov.bcb.rdr.siscapws.objetos.xsd.DemandaOUT.class);
```

7.3 There's no ObjectFactory with an @XmlElementDecl for the element

Este erro ocorre ao gerar todos os serviços do Web Service pelo JAX-WS (página 2). o problema ocorre ao processar vários WSDLs que apontam para o mesmo *namespace*, que é o caso do RDR Web Services, onde são três serviços que compartilham os mesmos objetos. Esta limitação está relatada em https://jax-ws.dev.java.net/issues/show_bug.cgi?id=661 .

Uma das soluções é informar um pacote separado para cada serviço informando o parâmetro “package” com um valor diferente do sugerido pelo WSDL. Esta solução é prática em nível de *build* com o Maven, mas o código gerado é repetido desnecessariamente.

Outra solução, talvez a mais aplicável para este Web Service é gerar o cliente para cada serviço, e a cada geração, copiar o arquivo “ObjectFactory.java” para outro lugar e

ao final, juntar o conteúdo dos arquivos “ObjectFactory.java” gerados para cada serviço, em apenas um. Essa estratégia foi testada, e funcionou corretamente, mantendo a coesão das classes sem ter que desnecessariamente duplicá-las.