PADRÃO PARA ENVIO DE E-MAIL

Versão 1.0

Fevereiro – 2006

Sumário de Informações do Documento			
Tipo do Documento: Relatório			
Título do Documento: Padrão para envio de e-mail			
Estado do Documento: Elaborado			
Responsáveis: Marlon, Edson – revisado por Elisabeth Hoffmann			
Palavras-Chaves: e-mail, padrão, smtp, serviço, JavaMail, JNDI			
Resumo: Este documento define o padrão para envio de e-mail para as aplicações que utilizam o framework Pinhão			
Número de páginas: 7			
Software utilizados: OpenOffice Writer 2.0			
Versão	Data	Mudanças	Autores
1.0	13/02/2006	Elaboração Inicial	Marlon, Edson

Sumário

1 INTRODUÇÃO

Na metodologia de desenvolvimento de software da Celepar não havia, até o momento, uma definição para a construção/configuração do envio de e-mail nas aplicações WEB. Cada aplicação poderia utilizar livremente qualquer biblioteca que suprisse os recursos essenciais para a elaboração do componente responsável por enviar os e-mails. Porém, foi identificada a necessidade de se padronizar a utilização de um serviço de e-mail (que pode ser único para todas as aplicações), bem como o componente que se encarrega de manipular e enviar os e-mail da aplicação (utilizando a biblioteca de classes JavaMail, desenvolvida pela Sun Microsystems).

Atualmente, o serviço SMTP é configurado em cada aplicação (hard-code ou em arquivo de propriedades) o que dificulta o processo de manutenção. A razão para a padronização é justamente reduzir o impacto com relação a manutenção e aumentar a produtividade no desenvolvimento. Segundo a forma proposta neste documento, todas as aplicações acessam o mesmo serviço (JNDI) de e-mail, que serão disponibilizados no servidor de aplicações (JBoss).

O objetivo deste documento é apresentar um guia para o processo de envio de e-mail nas aplicações que utilizam o Framework Pinhão. Os elementos essenciais desse guia serão explicados detalhadamente nos tópicos abaixo.

2 SERVIÇO JBOSS

Os dados do servidor SMTP são configurados como um serviço JNDI no JBoss. Este serviço é definido através de um arquivo de configurações (*mail-service.xml*) que estará ativo no JBoss para que a aplicação possa fazer uso quando necessário. O JBoss contém o arquivo *../jbossX/server/default/deploy/mail-service.xml* padrão, que deverá ser customizado conforme o modelo abaixo. Deste modo, logo que o servidor for iniciado o serviço de e-mail estará pronto para uso.

Segue abaixo o exemplo do arquivo *mail-service.xml* utilizando um servidor SMTP:

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE server>
    <server>
      <classpath codebase="lib" archives="mail.jar, activation.jar, mail-plugin.jar"/>
      <!-------------------------------------------------------- -->
      <!-- Mail Connection Factory -->
      <!-- ------------------------------------------------------ -->
      <mbean code="org.jboss.mail.MailService" name="jboss:service=Mail">
      <attribute name="JNDIName">java:/Mail</attribute>
      <attribute name="Configuration">
      <configuration>
            <!-- Change to your mail server prototocol -->
            <property name="mail.transport.protocol" value="smtp"/>
            <!-- Change to the SMTP gateway server -->
            <property name="mail.smtp.host" value="expressomx.pr.gov.br"/>
            <!-- Enable debugging output from the javamail classes -->
            <property name="mail.debug" value="false"/>
     </configuration>
     </attribute>
     </mbean>
   </server>
```

Exemplo 2.1 – Exemplo do arquivo mail-service.xml utilizando o SMTP do Expresso.

O arquivo *mail-service.xml* permite a configuração de vários servidores de envio de e-mail. Para isso basta adicionar tags <mbean> para cada configuração.

Para recuperar o serviço configurado no JBoss, a aplicação deverá utilizar a interface JNDI que será explicada no próximo tópico.

3 UTILIZANDO O SERVIÇO

Primeiramente deverá ser desenvolvido o componente para envio de e-mail da aplicação, utilizando para isto a biblioteca JavaMail desenvolvida pela *Sun Microsystems*. Abaixo o exemplo é apresentado de forma simplificada:

```
package gov.pr.celepar.consorte.action;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.naming.Context;
import javax.naming.InitialContext;

public class Mail {

   public static void enviarEmailSimples(String subject, String to,
         String from, String mensagem) throws AddressException,
         MessagingException {
            enviarEmailSimples(subject, new String[]{to}, from, mensagem);
   }

   public static void enviarEmailSimples(String subject, String[] to,
         String from, String mensagem) throws AddressException,
         MessagingException {
           Session mailSession = null;
           try{
                     Context initCtx = new InitialContext();
                     mailSession = (javax.mail.Session)initCtx.lookup("java:/Mail");
           } catch (javax.naming.NamingException e) {
                     e.printStackTrace();
           }

      InternetAddress[] destinatario = new InternetAddress[to.length];
      InternetAddress remetente = new InternetAddress(from);

      for (int i=0; i<to.length;i++){
           destinatario[i] = new InternetAddress(to[i]);
      }
      Message message = new MimeMessage(mailSession);

      message.setFrom(remetente);
      message.setRecipients(Message.RecipientType.TO, destinatario);
      message.setSubject(subject);
      message.setContent(mensagem.toString(), "text/plain");

      Transport.send(message);
   }
}
```

Exemplo 3.1 – Classe para envio de e-mail.

Neste exemplo, temos dois métodos estáticos chamados enviarEmailSimples que aceitam um e-mail (String) e um array de e-mails (String[]) respectivamente. Esta é uma sugestão de

implementação, ficando a critério do desenvolvedor adequá-la às suas necessidades. Exemplo de chamada ao componente:

```
String subject = "Assunto";
String[] to = {"emaildestino1@celepar.pr.gov.br","emaildestino2@celepar.pr.gov.br"};
String from = "emailorigem@celepar.pr.gov.br";
String mensagem = "Mensagem";
Mail.enviarEmailSimples(subject, to, from, mensagem);
```

Exemplo 3.2 – Código para a chamada ao método para envio de e-mail.

O serviço SMTP configurado no arquivo mail-service.xml é acessado através da interface JNDI (Java Naming Directory Interface). No exemplo abaixo, a classe Mail recupera esse serviço, conforme a linha em destaque:

```
public class Mail {
    ...
        Session mailSession = null;
        try{
                Context initCtx = new InitialContext();
                mailSession = (javax.mail.Session)initCtx.lookup("java:/Mail");
        } catch (javax.naming.NamingException e) {
                e.printStackTrace();
        }
    ...
}
```

Trecho do arquivo XML mail-service.xml:

```
<attribute name="JNDIName">java:/Mail</attribute>
...
<property name="mail.smtp.host" value="expressomx.pr.gov.br"/>
...
```

Exemplo 3.3 – Chamada do serviço SMTP através do JNDI (quadro superior) e Trecho do XML de configuração (quadro inferior).

Note que o nome "java:/Mail", dado ao atributo JNDIName no XML, é o mesmo invocado na classe Mail através do comando 'initCtx.lookup("java:/Mail")'. Através desse nome torna-se possível capturar o SMTP que está indicado na tag <property> do xml.