PROJETO CUSTOMIZAÇÕES

Pegar Dados do processo e enviar por e-mail

Amanda Alvim

1. Objetivo

O objetivo dessa customização é exemplificar o uso de uma integração pegando os dados de um processo como campos normais, da grid e todos os anexos de um grid e enviar por e-mail.

A integração se encontra no Projeto Base no nosso git (http://git.lecom.com.br/PSP/Projeto-Base-BPM) com o nome de **PegarDadosDocumentosEnviaEmail**, que está no package com.lecom.workflow.integracao.passagem_etapa, nesse pacote terá os exemplos que consiste em ser executado como integração de passagem de etapa no seu processo.

2. Como usar

No momento que você for colocar em um cliente essa integração você precisa ir na classe java e gerar um .jar pelo eclipse e subir no Lecom BPM (Menu Studio – Serviços – Aba Integrações) com a opção de passagem de etapa, para que seja executada no momento em que o usuário aprove/rejeite um processo.

3. Arquivo JAVA

Nesse arquivo temos exemplos de como vamos pegar os campos do processo na integração, de como gerar um zip de todos os anexos e enviar um e-mail com todas as essas informações.

PegarDadosDocumentosEnviaEmail.java

```
@IntegrationModule("PegarDadosDocumentosEnviaEmail")
@Version({1,0,0})
public class PegarDadosDocumentosEnviaEmail {

    private static final Logger logger = Logger.getLogger(PegarDadosDocumentosEnviaEmail.class);

    @SuppressWarnings("unchecked")
    @Execution
    public String enviaEmail(IntegracaoVO integracaoVO){
        logger.info("INICIO Integração de envio de email");

        try {

            logger.debug("Entrei na integração de campo");

            Map<String,String> camposEtapa = integracaoVO.getMapCamposFormulario();

            logger.debug("camposform : "+camposEtapa);

            logger.debug(" Ação     =  "+integracaoVO.getAcao());
            logger.debug(" CodCIclo = "+integracaoVO.getCodCiclo());
            logger.debug(" CodEtapa = "+integracaoVO.getCodEtapa());
            logger.debug(" CodForm  =  "+integracaoVO.getCodForm());
            logger.debug(" CodProcesso = "+integracaoVO.getCodProcesso());
            logger.debug(" CodUserEtapa = "+integracaoVO.getCodUsuarioEtapa());
            logger.debug(" CodUserIniciador = "+integracaoVO.getCodUsuarioIniciador());
            logger.debug(" Data Ini Processo =  "+integracaoVO.getDatData());
            logger.debug(" Data Finalizacao = "+integracaoVO.getDatFinalizacao());
            logger.debug(" Desfrom = "+integracaoVO.getDesFrom());
            logger.debug(" DesmailHost =  "+integracaoVO.getDesMailHost());
            logger.debug(" Desreply = "+integracaoVO.getDesReply());
            logger.debug(" NomeTabelaModelo =  "+integracaoVO.getNomeTabelaModelo());
            logger.debug(" Status =  "+integracaoVO.getStatus());
            logger.debug(" Conexão = "+integracaoVO.getConexao());


            String nome = camposEtapa.get("$NOME_USUARIO");
            String email = camposEtapa.get("$EMAIL_USER");
            String totalDespesa = camposEtapa.get("$TOTAL_DESPESAS");
```

PegarDadosDocumentosEnviaEmail.java

```
63              String identificadorGridDespesas = "DESPESAS";
64              List<Map<String,Object>> valores = integracaoVO.getDadosModeloGrid(identificadorGridDespesas);
65
66              StringBuilder strBuilder = new StringBuilder();
67              StringBuilder htmlInicial = blocoCabecalho(strBuilder,nome);
68              StringBuilder htmlMeio = blocoMeio(htmlInicial,totalDespesa,valores);
69              StringBuilder htmlFim = blocoRodape(htmlMeio);
70
71              List<String> to = new ArrayList<>();
72              to.add(email);
73
74              String from = integracaoVO.getDesFrom();
75              logger.info(">>>> FROM : <<<<" + from);
76              logger.info(">>>> TO : <<<< " + to);
77              String assunto = "["+integracaoVO.getCodProcesso()+"]  - Despesas lançadas ";
78
79              String identificadorGridAnexos = "ANEXOS";
80              String nomeCampoAnexoGrid = "ANEXO_GERAL";
81              List<Map<String, Object>> anexosGerais = integracaoVO.getDadosModeloGrid(identificadorGridAnexos);
82              String caminhoArquivoZip = geraZipArquivos(anexosGerais,integracaoVO.getCodProcesso(),nomeCampoAnexoGrid);
83
84
85              EmailMessage emailMessage = new EmailMessage(assunto, htmlFim.toString(), from,to, true);
86              emailMessage.setAttached(caminhoArquivoZip);
87              integracaoVO.enviaEmailMessage(emailMessage);
88
89              logger.info("Fim da integração que envia email");
90              return "0|Enviado email com sucesso";
91          } catch (Exception e) {
92              logger.error("Erro ao enviar email",e);
93              return "99|Erro ao enviar o email.";
94          }
95      }
```

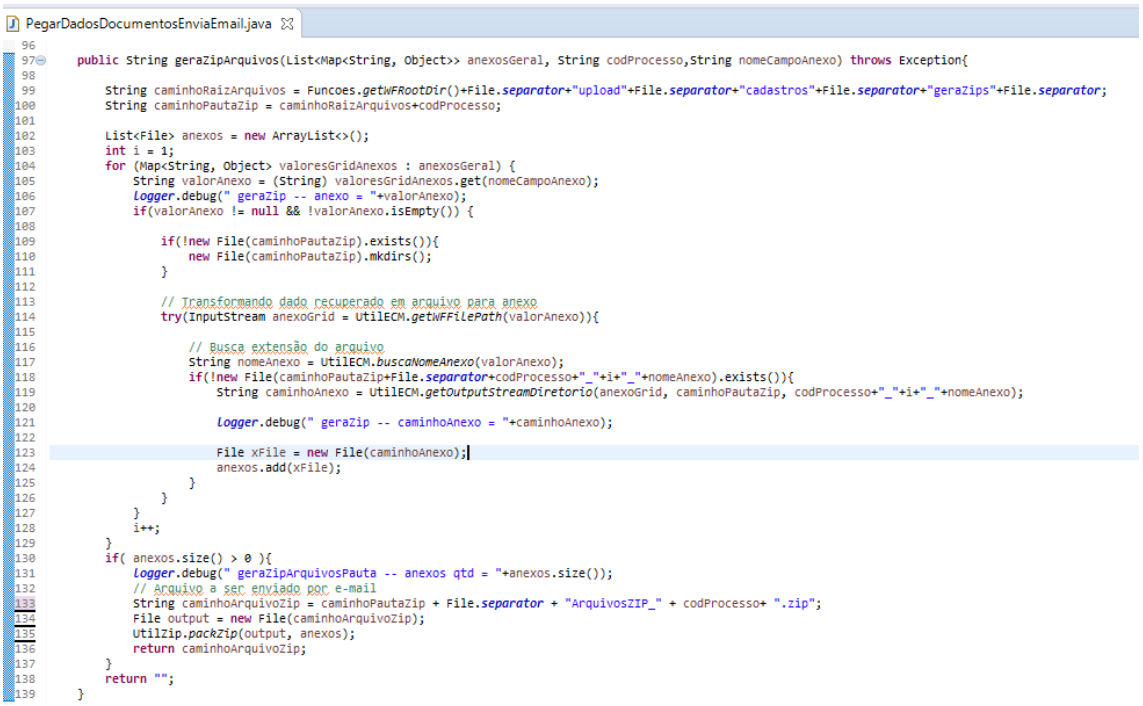
Nos dois prints acima é o essencial da classe de integração, vamos passando pelas linhas para explicar o que está sendo feito:

- Na **linha 21**, é onde informamos que essa classe é do tipo integração, colocando essa annotation @IntegrationModule(value=”Nome da integração”), dentro dela apenas colocamos um value, um nome para classe, esse nome não interfere em nada no procedimento de execução do integração, mas a anotação é necessária para que o Lecom BPM entenda que é uma integração.
- Na **linha 22,** informamos a annotation @Version({1,0,0}), nela você irá informar a versão da sua classe geralmente iniciamos em 1,0,0 e a cada subida com alguma alteração na classe altera-se o número assim você irá identificar que está subindo a versão correta da sua classe.
- Na **linha 25,** definimos a variável de log, onde utilizamos a biblioteca do log4j que o Lecom BPM já utiliza, nessa linha somente estamos definindo o nome da variável e iremos utilizar mais para baixo e sempre usamos as opções debug, info e error.
- Na **linha 28,** é onde usamos a anotação de @Execution, isso informa que que esse método é o que o Lecom BPM irá chamar primeiro, se não tiver a anotação ele não deixará subir o seu jar. O método que utilizaremos como “chamada” da integração ele deve receber o parâmetro **IntegracaoVO**, esse objeto ele tem vários valores que o Lecom BPM envia para nós, e vamos ver esses campos na sequência da classe.
- Na **linha 36** chamamos o método getMapCamposFormulario() do objeto integracaoVO, ele nos retorna um Map<String,String> que irá vir todos os campos do processo excetos os que tiverem com a configuração de oculto, vindo no map nome do campo, valor do campo na etapa configurada a integração, vamos utilizar essa variável camposEtapa mais para baixo. Na linha debaixo ele cria um debug com tudo que retornou nessa variável, assim no log você vai conseguir verificar todos os campos que veio.
- Nas **linhas 40 a 54** estamos realizando logs de todos os atributos que temos nesse objeto integracaoVO, na sequência temos a **ação**, que virá a ação que a atividade foi chamada aprovação/rejeição, o **código do ciclo**, o **código da atividade**, o **código do formulário**, o **código do processo**, o **código do usuário da atividade**, o usuário que está realizando a aprovação ou rejeição, o **código do usuário iniciador**, o usuário que iniciou esse processo, **data inicial do processo**, **data da finalização da atividade**, o **desFrom**, esse é o e-mail

cadastrado no modulo admin na parte de servidor de e-mail como desFrom ele será o e-mail de quem os usuários receberam esses e-mails enviados pelo Lecom BPM e os customizáveis, o **desMailHost**, esse é o host cadastrado no modulo admin na parte de servidor de e-mail, o **desReply**, esse é o e-mail cadastrado no modulo admin na parte de servidor de e-mail como e-mail reply ele será o e-mail caso o Lecom BPM encontre algum problema ele dispara e-mail para essa conta, o **nome da tabela do modelo**, o **status**, virá o status da atividade, **conexão**, essa conexão irá vir o objeto connection que podemos usar para fazer select no banco do Lecom BPM e/ou nas conexões auxiliares.

- Nas **linhas 59 a 61**, estou pegando os valores dos campos do meu processo, nesse caso de campos que não são do tipo grid precisamos pegar usando aquela variável **camposEtapa**, e dando um **get** passando o nome do campo do processo com **$ na frente**, nesse exemplo pegando Nome do usuário, e-mail do usuário e total das despesas.
- Nas **linhas 63 e 64**, estou definindo o nome do identificador da minha grid e abaixo utilizamos o método getDadosModeloGrid passando o identificador, ele retorna para nós uma list de map de String, object, e assim vem todas as linhas da grid que foi preenchido no processo, nós iremos fazer o for dessa lista pegando os valores no método blocoMeio que iremos ver mais para baixo.
- Nas **linhas 66 a 69** estamos montando uma StringBuilder como todo o HTML com os dados do processo a ser enviado, os métodos usados para montar o html será explicado mais abaixo um por um.
- Nas **linhas 71 e 72**, estamos adicionando a uma lista de String o e-mail que pegamos do campo EMAIL_USUARIO na **linha 59**, usaremos essa variável **to** para ser quem irá receber o e-mail quando formos enviar, pode ser mais de um destinatário.
- Na **linha 74** estamos atribuindo o desFrom a uma variável para utilizarmos no método de envio do e-mail.
- Na **linha 77** estamos montando o assunto que chegará para o usuário na conta de e-mail dele.
- Nas **linhas 79 a 82** estamos pegando os valores da grid ANEXOS, e encaminhando essa lista para o método **geraZipArquivos**, onde será gerado um zip com todos os arquivos anexados nessa grid, esse método será explicado também individualmente mais abaixo.
- Nas **linhas 85 a 87** estamos utilizando a api que o Lecom BPM já nos disponibiliza pra realizar o envio de e-mail via integração, precisamos instanciar a classe EmailMessage e no construtor passar as informações geradas nas linhas acimas que mencione, o assunto, o conteúdo a ser enviado, o from, o to, e o último parâmetro é passado true geralmente para que o componente transforme em HTML o conteúdo e consiga enviar formatado e bonito o e-mail. Na **linha 86** setamos no objeto emailMessage o caminho do arquivo zip que foi gerado no método geraZipArquivos para que seja enviado como anexo no e-mail. Na **linha 87** usamos o objeto IntegracaoVO para passar como parâmetro no método enviaEmailMessage o objeto instanciado e preenchido acima, internamente nesse método é feito o envio do e-mail ao usuário preenchido para que nós não nos preocupássemos em ter que fazer todo mecanismo de envio de e-mail via java.
- Na **linha 90** é onde realizamos o retorno para o Lecom BPM de que a integração foi executada com sucesso, para isso precisamos retornar uma String contendo um número, que pode ser 0 ou 99, e a mensagem a qual queremos que apareça na tela do Lecom BPM para o usuário. Quando colocamos o retorno 0 significa que estamos falando que deu sucesso na execução, se retornarmos qualquer número diferente de 0 geralmente utilizamos 99 significa que deu algum erro em algum momento da execução da integração.

• Sempre nos códigos de integrações e outros desenvolvimento é bom utilizar os try/catch para tratar alguma exceção que possa ocorrer na execução do fonte, no caso desse exemplo nas **linhas 91 a 94** está sendo colocado o log do erro para conseguirmos visualizar onde deu problema e o retorno conforme explicado no item acima.



```
PegarDadosDocumentosEnviaEmail.java
96
97  public String geraZipArquivos(List<Map<String, Object>> anexosGeral, String codProcesso,String nomeCampoAnexo) throws Exception{
98
99      String caminhoRaizArquivos = Funcoes.getWFRootDir()+File.separator+"upload"+File.separator+"cadastros"+File.separator+"geraZips"+File.separator;
100     String caminhoPautaZip = caminhoRaizArquivos+codProcesso;
101
102     List<File> anexos = new ArrayList<>();
103     int i = 1;
104     for (Map<String, Object> valoresGridAnexos : anexosGeral) {
105         String valorAnexo = (String) valoresGridAnexos.get(nomeCampoAnexo);
106         logger.debug(" geraZip -- anexo = "+valorAnexo);
107         if(valorAnexo != null && !valorAnexo.isEmpty()) {
108
109             if(!new File(caminhoPautaZip).exists()){
110                 new File(caminhoPautaZip).mkdirs();
111             }
112
113             // Transformando dado recuperado em arquivo para anexo
114             try(InputStream anexoGrid = UtilECM.getWFFilePath(valorAnexo)){
115
116                 // Busca extensão do arquivo
117                 String nomeAnexo = UtilECM.buscaNomeAnexo(valorAnexo);
118                 if(!new File(caminhoPautaZip+File.separator+codProcesso+"_"+i+"_"+nomeAnexo).exists()){
119                     String caminhoAnexo = UtilECM.getOutputStreamDiretorio(anexoGrid, caminhoPautaZip, codProcesso+"_"+i+"_"+nomeAnexo);
120
121                     logger.debug(" geraZip -- caminhoAnexo = "+caminhoAnexo);
122
123                     File xFile = new File(caminhoAnexo);
124                     anexos.add(xFile);
125                 }
126             }
127         }
128         i++;
129     }
130     if( anexos.size() > 0 ){
131         logger.debug(" geraZipArquivosPauta -- anexos qtd = "+anexos.size());
132         // Arquivo a ser enviado por e-mail
133         String caminhoArquivoZip = caminhoPautaZip + File.separator + "ArquivosZIP_" + codProcesso+ ".zip";
134         File output = new File(caminhoArquivoZip);
135         Utilzip.packZip(output, anexos);
136         return caminhoArquivoZip;
137     }
138     return "";
139 }
```

Na imagem acima temos o conteúdo do método **geraZipArquivos** mencionado nos item acima, vamos ver o que ele está fazendo linha a linha:

• Na **linha 99,** estamos inicializando e colocando o caminho a qual ele vai gerar esses arquivos temporários e assim gerar o zip de todos os arquivos anexados na grid, lembrando que esse caminho sempre deve ser criado na pasta upload/cadastro que fica no servidor do cliente que fica no caminho /opt/lecom/app/tomcat/webapps/bpm/upload/cadastros/ para ambientes anteriores a nova forma de atualização das customizações e na nova forma fica no caminho /opt/lecom/custom/web-content

• Na **linha 100** adicionamos a variável do caminho da pasta que irá salvar todos os anexos o código do processo, para que fique separado as pastas para cada instancia que chegar nessa fase de enviar um e-mail.

• Na **linha 102** inicializamos uma lista de file para salvar cada arquivo gerado.

• Na **linha 103** inicializamos um inteiro com o código 1 para que durante o for da lista de grid ele itere para o nome do arquivo gerado.

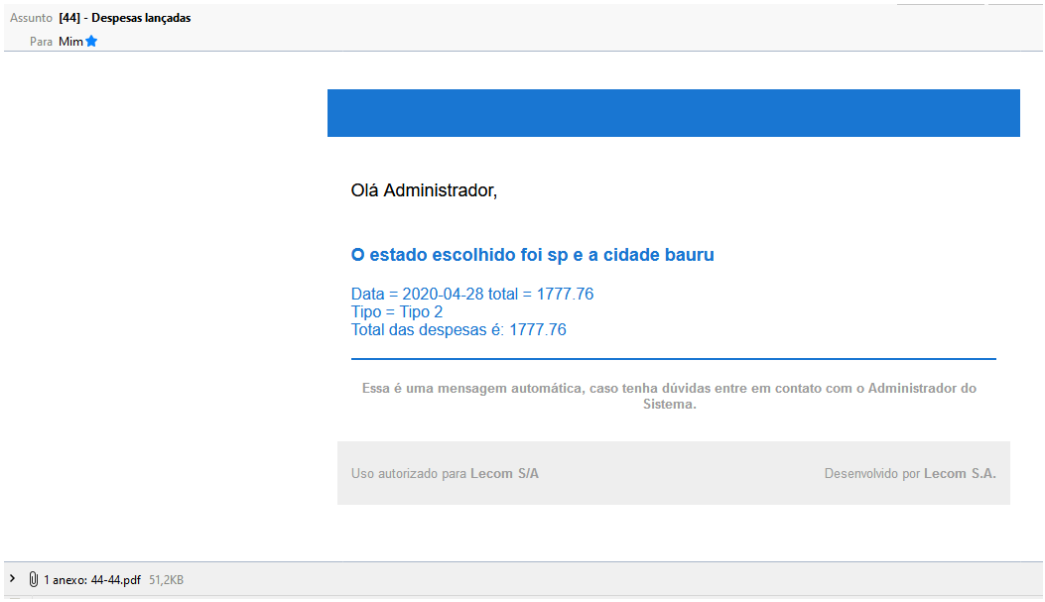
• Nas **linhas 104 a 129** está sendo feito a iteração da lista da grid e para cada linha da grid realizaremos as ações de pegar valor do campo da grid (**linha 105)** que retornará o nome do arquivo:unique id do ECM.

• Nas **linhas 109 a 111** estamos validando se a pasta com o código do processo já existe no servidor se não ele realiza a criação nesse momento fazendo a chamada mkdirs.

• Na **linha 114** estamos usando uma chamada de uma classe externa **UtilECM,** essa classe irá fazer as ações que precisamos com a api do ECM, nessa linha por exemplo chamamos o método **getWFFilePath** passando o valor do campo, ele internamente fará a chamada da api do ecm document().lerArquivo() passando para ela o unique id do ecm e retornando um objeto do tipo **InputStream** conforme está sendo atribuído nessa linha, nela também está

sendo utilizado um recurso do Java try resource onde criamos um try e assim o próprio Java trata de abrir e fechar esses objetos sem a necessidade de fazermos esses fechamentos manual.

- Na **linha 117** ele chama a partir da classe **UtilECM** o método **buscaNomeAnexo**, que está pegando o valor do campo da grid e realizando um split por **:** e pegando o primeiro parâmetro que retorna o nome do arquivo.
- Na **linha 118** está sendo verificado se já existe no caminho passado o arquivo em questão, estamos denominando aqui que o nome do arquivo físico gerado chamará **codProcesso**, o código da instancia que está sendo executada, **i**, variável que está contando quantos anexos foram adicionados nessa grid, **nomeAnexo**, o nome do arquivo anexado.
- Na **linha 119** chamaremos um método da classe **UtilECM** chamada **getOutputStreamDiretorio** e para ele precisamos passar os seguintes parâmetros anexoGrid, objeto InputStream que retornamos da api do ecm, caminhoPautaZip, diretório do servidor mais código do processo para salvar o arquivo nesse diretório, e o nome do arquivo que iremos gerar conforme explicado no item acima, esse método nos retornará o caminho inteiro do arquivo salvo no diretório, por exemplo, /opt/lecom/app/tomcat/webapps/bpm/upload/cadastros/geraZips/2/2_1_testeAnexo.png.
- Na **linha 123** pegamos esse caminho completo e instanciamos em um objeto file, e esse objeto file será adicionado aquela lista de files criada no começo do método.
- Na **linha 130** é verificado se existe algum arquivo adicionado na lista de files.
- Na **linha 133** é criado uma variável com o nome do arquivo zip que iramos criar na pasta desse processo do servidor e abaixo instanciamos esse nome para um Objeto File, para ser criado esse arquivo no servidor.
- Na **linha 135** é chamado o método **packZip** da classe **UtilZIP**, passando para ele a variável do nome do arquivo zip que foi criado e a lista de anexos, e assim internamente ele irá ler pegar cada anexo gerado no servidor e juntar em um arquivo zip, retornando para a **linha 82** o caminho do arquivo zip gerado e preenchido.
- Nas **linhas 141 a 147** é pego a partir do caminho dessa classe de integração para chegar na raiz do bpm, esse método está retornando, /opt/lecom/app/tomcat/webapps/bpm/.
- Nas **linhas 149 a 166** é o método referente a parte inicial do e-mail customizado que será enviado, nesse método **blocoCabecalho** apenas é feito um html bem complexo em termos de tables tds, porém ele dessa forma ele fica igual aos e-mails enviados pelo Lecom BPM, para esse método apenas passo a StringBuilder que será alimentada com html e o nome do usuário, o que pegamos do valor do campo NOME_USUARIO do processo de exemplo e utilizo essa variável na **linha 163.**
- Nas **linhas 169 a 211** é o método **blocoMeio** referente a parte do meio do e-mail customizado que será enviado, nele vamos fazer a iteração da grid de DESPESAS para montar cada linha da grid como uma tr no html e com seus valores, isso é feito nas **linhas 182 a 195** onde está sendo feito um for da lista valores, e cada linha é um Map<String,Object> onde pego o valor fazendo map.get() passando o nome do campo da grid sem o $ no começo conforme vimos para campos que não são grids, nesse exemplo pego os valores do campo TIPO_DESPESA e DATA_DESPESA na **linha 186** e os valores do campo QUANTIDADE e TOTAL na **linha 192.** E no final na linha 200 trazemos o valor do total das despesas para mostrar abaixo da tabela.
- Nas **linhas 213 a 231** está o método **blocoRodape** que é o finalzinho do e-mail customizado. Vamos ver abaixo um exemplo de como fica o e-mail:



Assunto **[44] - Despesas lançadas**

Para **Mim**

Olá Administrador,

O estado escolhido foi sp e a cidade bauru

Data = 2020-04-28 total = 1777.76
Tipo = Tipo 2
Total das despesas é: 1777.76

Essa é uma mensagem automática, caso tenha dúvidas entre em contato com o Administrador do Sistema.

Uso autorizado para **Lecom S/A** Desenvolvido por **Lecom S.A.**

1 anexo: 44-44.pdf 51,2KB