



# PROJETO CUSTOMIZAÇÕES

Robô Genérico de Importação de planilha

Amanda Alvim

## 1. Objetivo

O objetivo do robô genérico de importação de planilha consiste em ter um processo no Lecom BPM que tenha um campo de anexo que o usuário irá subir uma planilha, e ao aprovar a atividade seguinte o robô irá ler essa planilha e inserir na base de dados as informações.

Para isso previamente a tabela deve já ser criada no banco auxiliar nosso, e a planilha para importação deve conter as mesmas colunas com nomes da tabela auxiliar para que funcione o mecanismo dinâmico de importação.

O Robô genérico de importação de planilha se encontra no Projeto Base no nosso git (<http://git.lecom.com.br/PSP/Projeto-Base-BPM>) com o nome de **RobolImportarPlanilha**, que está no package com.lecom.workflow.robo.satelite.generico, para funcionamento dele existe dois arquivos que são como base que são os properties: **RobolImportarPlanilha.properties** e **automatico.properties**, eles ficam no caminho (upload/cadastros/config) do projeto.

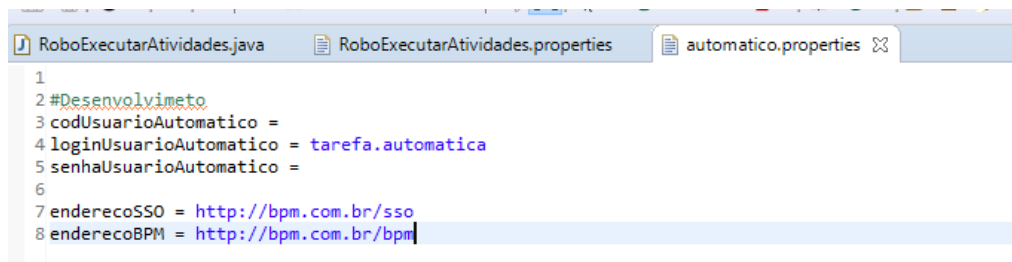
## 2. Como usar

No momento que você for colocar em um cliente esse robô você precisa ir na classe java e gerar um .jar pelo eclipse e subir no Lecom BPM (Menu Studio – Serviços – Aba Robos) e colocar o .properties no lugar correto do servidor, explicado abaixo melhor no menu de properties.

## 3. Properties

Os properties são a base para o funcionamento do robô conforme mencionado acima, nele nós iremos configurar as informações que o robô irá usar para realizar as importações das planilhas e executar a atividade, os properties são usados em várias situações no decorrer dos nossos desenvolvimentos, eles ficam diretamente no servidor do cliente no caminho /opt/lecom/app/tomcat/webapps/bim./upload/cadastros/config para ambientes anteriores a nova forma de atualização das customizações e na nova forma fica no caminho /opt/lecom/custom/web-content/config.

Vamos ver o que temos nos dois properties citados, para o arquivo automatico.properties ele contém o seguinte conteúdo:



```
1
2 #Desenvolvimeto
3 codUsuarioAutomatico =
4 loginUsuarioAutomatico = tarefa.automatica
5 senhaUsuarioAutomatico =
6
7 enderecoSSO = http://bpm.com.br/sso
8 enderecoBPM = http://bpm.com.br/bpm
```

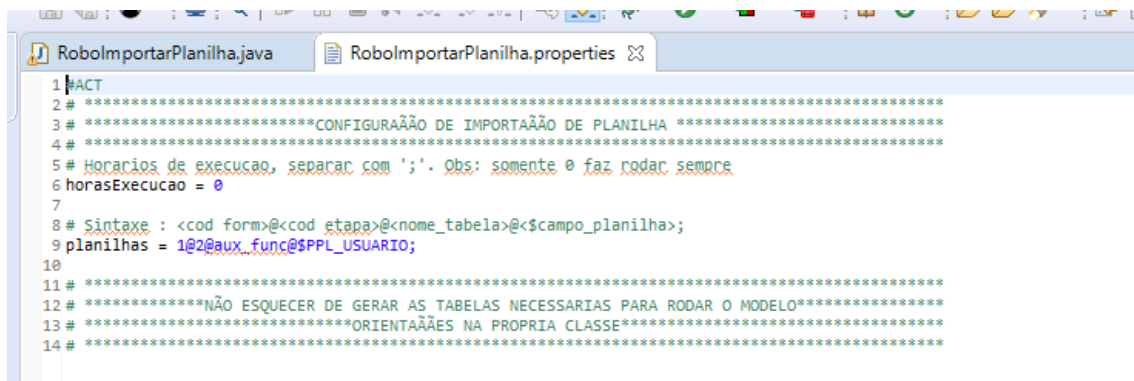
Na imagem acima temos os campos de configuração padrão para o nosso usuário automático, em todos os ambientes de clientes criamos um usuário específico denominando automático sem a opção de senha expira para que possamos conseguir deixar nossos properties com usuário e senha que somente nós consultores sabemos, sendo assim precisamos configurar:

- codUsuarioAutomatico: Nesse campo será preciso colocar o código do usuário automático criado, essa informação pegamos na tabela de usuário no banco do Lecom

BPM, precisamos dessa informação para poder fazer as buscas dos processos parados nesse usuário.

- loginUsuarioAutomatico: Nesse campo será preciso colocar o login do usuário automático criado, essa informação é necessária para que possamos realizar o login por fora da ferramenta e assim fazer aprovação automática das atividades.
- senhaUsuarioAutomatico: Nesse campo será preciso colocar a senha configurada no cadastro do usuário automático, essa informação é necessária para que possamos realizar o login por fora da ferramenta e assim fazer aprovação automática das atividades.
- endereçoSSO: Nesse campo será preciso colocar a url do cliente completa com o final SSO, conforme no exemplo (<https://bpm.lecom.com.br/sso>) , essa informação é necessária para a realização do login por fora.
- endereçoBPM: Nesse campo será preciso colocar a url do cliente completa com o final BPM, conforme no exemplo (<https://bpm.lecom.com.br/bpm>) , essa informação é necessária para a realização da aprovação das atividades.

O arquivo RobolImportarPlanilha.properties tem os seguintes conteúdos, como ele é muito grande iremos destacar e explicar abaixo por partes a configuração dele:



```

1 #ACT
2 #
3 # *****CONFIGURAÇÃO DE IMPORTAÇÃO DE PLANILHA *****
4 #
5 # Horários de execução, separar com ';'. Obs: somente 0 faz rodar sempre
6 horasExecucao = 0
7
8 # Sintaxe : <cod_form>@<cod_etapa>@<nome_tabela>@<$campo_planilha>;
9 planilhas = 1@2@aux_func@$PPL_USUARIO;
10
11 # *****NÃO ESQUECER DE GERAR AS TABELAS NECESSARIAS PARA RODAR O MODELO*****
12 # *****ORIENTAÇÕES NA PRÓPRIA CLASSE*****
13 #
14 #

```

Na imagem acima temos a configuração do properties, esse arquivo de ele já vem comentado com o que deve ser preenchido, mas vamos aos detalhes aqui:

- horasExecucao: Nesse campo deve se configurar o horário de execução desse robô, pensando em horários que não atrapalhem o uso do sistema como um todo caso essa planilha for muito grande recomendado colocar para rodar de madrugada; conforme exemplo: 0 (rodará meia noite).
- planilhas: Nesse campo deve se configurar quais formulários e atividades você deseja que seja importada as planilhas e aprova a etapa automaticamente, colocando o código do seu formulário, informação que consegue pegar na listagem de modelos do Studio, o código da sua atividade a ser aprovada, informação que você pode pegar passando o mouse em cima do nome da atividade na tela do Studio na aba de etapas/atividades, o nome da tabela auxiliar, o nome do campo do tipo template onde o usuário irá incluir a planilha para ser importada, todas as configurações devem estar separados por ; conforme exemplo: 1@2@aux\_func@\$PLANILHA; (formulário 1 etapa 2 tabela aux\_func e campo \$PLANILHA).

## 4. Fonte Java

No tópico acima foi explicado como configurar os properties que serão o que de fato vocês irão configurar, mas para conhecimento e entendimento do fonte vamos passar por ele e os seus métodos para saberem o que está sendo feito.

```
47 @RobotModule("RoboImportarPlanilha")
48 @Version({1,0,0})
49 public class RoboImportarPlanilha implements WebServices{
50
51     private static final Logger logger = Logger.getLogger(RoboImportarPlanilha.class);
52     private static String configpath = Funcoes.getWRootDir() + File.separator + "upload" + File.separator + "cadastros" + File.separator + "config" + File.separator;
53     Map<String, String> parametros = null;
54
55     @Execution
56     public void executar() {
57         logger.info("[[INICIO DO ROBO IMPORTADOR DE PLANILHA]]");
58
59         try(connection conAux= DBUtils.getConnection("aux_act")){
60
61             parametros = Funcoes.getParametrosIntegracao(configpath + getClass().getSimpleName());
62
63             Calendar dataAtual = Calendar.getInstance();
64             String horaAtual = Integer.toString(dataAtual.get(Calendar.HOUR_OF_DAY));
65
66             String horasExecucao = parametros.get("horasExecucao");
67
68             boolean executarRobo = Arrays.asList(horasExecucao.split(";")).contains(horaAtual);
69
70             if (!executarRobo && !horasExecucao.equals("0")) {
71                 logger.info("==== ROTINAS NÃO DEVEREM SER EXECUTADAS AGORA =====");
72             } else {
73                 processarPlanilha(conAux);
74             }
75         } catch (SQLException e) {
76             logger.error("Erro método executar - SQLException: ", e);
77             e.printStackTrace();
78         } catch (Exception e) {
79             logger.error("Erro método executar - Call processarPlanilha: ", e);
80             e.printStackTrace();
81         }
82
83         logger.info("[[FIM DO ROBO IMPORTADOR DE PLANILHA]]");
84     }
85 }
```

Na imagem acima temos o começo da nossa classe JAVA:

- Na **linha 47** é onde informamos que essa classe é do tipo robô, colocando essa annotation `@RobotModule(value="Nome Robô")`, dentro dela apenas colocamos um value, um nome para classe, esse nome não interfere em nada no procedimento de execução do robô.
- Na **linha 48** 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 51** 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 52** estamos definindo o caminho a qual irá estar o arquivo de properties (esse caminho completo está mencionado no começo do documento)
- Na **linha 59** estamos iniciando uma conexão com o banco de dados do Lecom BPM, utilizando uma classe do produto (DBUtils) que nos retorna a conexão com o bpm.
- Na **linhas 61** estamos transformando esses properties que foram explicados acima em um `Map<String, String>` para que seja fácil de pegar os parâmetros que configuramos.
- Na **linha 68** está sendo validado se o horário preenchido no properties contém o horário atual
- Na **linha 70** está realizando o if para verificar se vai executar ou não nesse momento o robô, se estiver na hora ele irá executar o método `processarPlanilha`.

```
RobolImportarPlanilha.java RobolImportarPlanilha.properties
107
108 private void processarPlanilha(Connection conAux) {
109     Logger.info("[[ENTRANDO NO MÉTODO PROCESSAR PLANILHA]]");
110
111     Map<String, String> parametros;
112     try {
113         parametros = Funcoes.getParametrosIntegracao(configpath + getClass().getSimpleName());
114         String planilhas = parametros.get("planilhas");
115
116         if (!"".equals(planilhas)) {
117
118             Logger.debug("planilhas: " + planilhas);
119
120             for (String plan : planilhas.split(";")) {
121
122                 Logger.debug("plan: " + plan);
123
124                 String[] planilhaFormEtapa = plan.split("@");
125
126                 Logger.debug("planilhas: " + planilhas);
127
128                 String codFormAnalise = planilhaFormEtapa[0];
129                 String codEtapaAnalise = planilhaFormEtapa[1];
130                 String tablename = planilhaFormEtapa[2];
131                 String campo = planilhaFormEtapa[3].trim();
132
133                 Logger.debug("codFormAnalise: " + codFormAnalise);
134                 Logger.debug("codEtapaAnalise: " + codEtapaAnalise);
135                 Logger.debug("tablename: " + tablename);
136                 Logger.debug("campo planilha: " + planilhaFormEtapa[3].trim());
137
138                 // insere na tabela
139                 try(Connection cnbpm = DBUtils.getConnection("workflow")){
140                     getProcessData(cnbpm, conAux, codFormAnalise, codEtapaAnalise, tablename, campo);
141                 } catch (SQLException e) {
142                     // TODO Auto-generated catch block
143                     e.printStackTrace();
144                 }
145             }
146         } else {
147             Logger.error("Parametro PLANILHA vazio");
148         }
149     } catch (Exception e) {
150         // TODO Auto-generated catch block
```

Na imagem acima temos o print do método **processarPlanilha** que irá realizar o procedimento de importação, os parâmetros passados para esse método é:

- Conexão (a conexão que foi instanciada na linha 59)

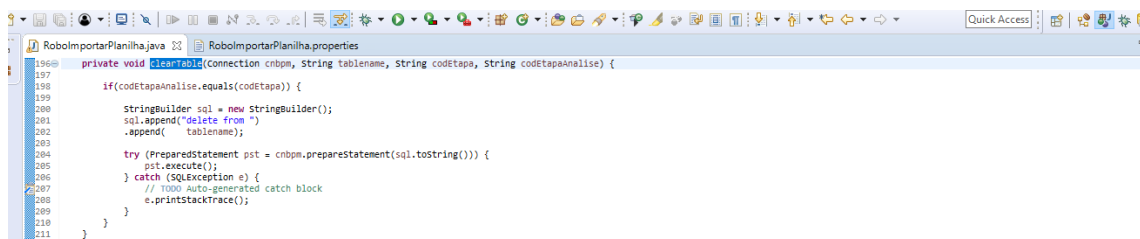
No restante do método ele separa os parâmetros que configuramos, para cada variável diferente e instancia na linha 139 nova conexão com a base auxiliar, para assim enviar esses parâmetros separados ao método **getProcessData**.

```
RobolImportarPlanilha.java RobolImportarPlanilha.properties
155
156 private void getProcessData(Connection cnbpm, Connection conAux, String codFormAnalise, String codEtapaAnalise, String tablename, String campo) {
157     Logger.info("[[ENTRANDO NO MÉTODO GET PROCESS DATA]]");
158
159     String codProcesso = "";
160     String codEtapaAtual = "";
161     String codCiclo = "";
162     String codForm = "";
163
164     StringBuilder consultaProcesso = new StringBuilder();
165     consultaProcesso.append("SELECT p.cod_processo, p.cod_etapa_atual, p.cod_ciclo_atual FROM PROCESSO p");
166     .append(" JOIN PROCESSO_ETAPA pe ON pe.cod_processo = p.cod_processo")
167     .append(" AND pe.cod_etapa = p.cod_etapa_atual")
168     .append(" AND pe.cod_ciclo = p.cod_ciclo_atual")
169     .append(" WHERE pe.ide_status = 'A'")
170     .append(" AND p.cod_form = ?")
171     .append(" AND p.cod_etapa_atual = ?");
172     try(PreparedStatement pst = cnbpm.prepareStatement(consultaProcesso.toString())){
173         Logger.info("codFormAnalise: " + codFormAnalise);
174         Logger.info("codEtapaAnalise: " + codEtapaAnalise);
175         pst.setString(1, codFormAnalise);
176         pst.setString(2, codEtapaAnalise);
177         try(ResultSet rs = pst.executeQuery()){
178             while(rs.next()) {
179                 codProcesso = rs.getString("cod_processo");
180                 codEtapaAtual = rs.getString("cod_etapa_atual");
181                 codCiclo = rs.getString("cod_ciclo_atual");
182                 codForm = getCodForm(cnbpm, codProcesso);
183                 clearTable(conAux, tablename, codEtapaAtual, codEtapaAnalise);
184                 importarPlanilha(cnbpm, conAux, codProcesso, codEtapaAtual, codCiclo, codFormAnalise, codEtapaAnalise, campo, tablename);
185                 registraAlteracoes(conAux, tablename, codProcesso, codEtapaAtual);
186                 executaAtividade(cnbpm, codProcesso, codEtapaAtual, codCiclo);
187             }
188         }
189     } catch (SQLException e) {
190         Logger.error("Erro método getProcessData - SQLException: ", e);
191         e.printStackTrace();
192     }
193 }
194
195 }
```

Na imagem acima temos o print do método **getProcessData** que irá buscar os processos do formulário configurado e parado na atividade, os parâmetros passados para esses métodos são:

- Conexão do bpm (a conexão que foi instanciada na linha 59)
- Conexão do auxiliar (a conexão que foi instanciada na linha 139)
- Código do formulário
- Código da atividade
- Nome da tabela
- Nome do campo

No restante do método é feito um select para identificar todos os processos do formulário configurado que estão parados na atividade configurada, para cada processo retornado é passada as informações para os métodos nas sequencias realizando a limpeza da tabela aux, importando a planilha nova, registra alteração e executa a atividade.



```

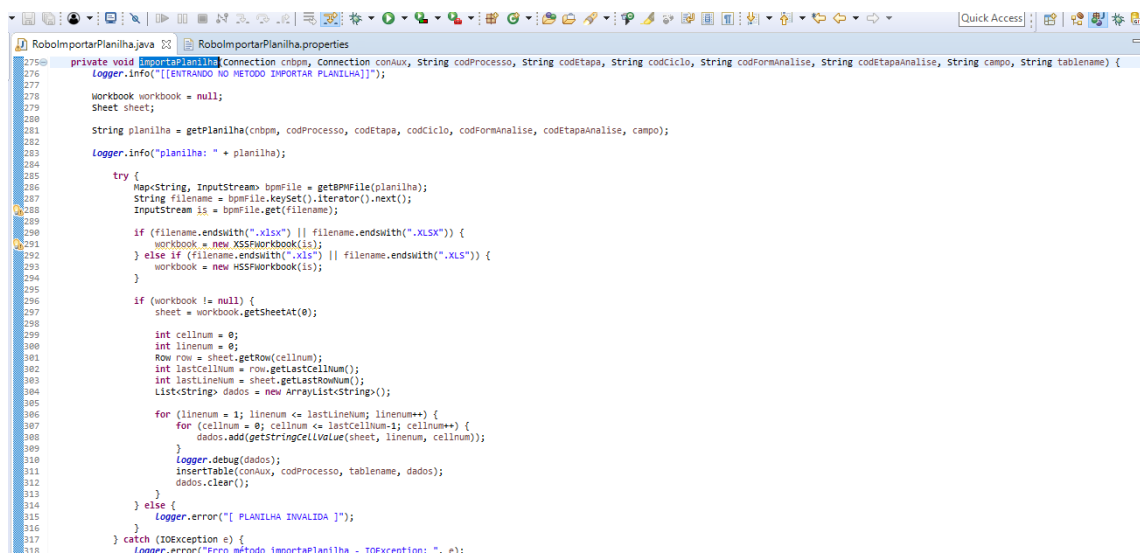
196 private void getProcessData(Connection cnbpm, String tablename, String codEtapa, String codEtapaAnalise) {
197     if(codEtapaAnalise.equals(codEtapa)) {
198         // Limpeza da tabela auxiliar
199         String sql = new StringBuilder();
200         sql.append("delete from ");
201         sql.append(tablename);
202         try (PreparedStatement pst = cnbpm.prepareStatement(sql.toString())) {
203             pst.execute();
204         } catch (SQLException e) {
205             // Auto-generated catch block
206             e.printStackTrace();
207         }
208     }
209 }

```

Na imagem acima temos o print do método **clearTable** os parâmetros passados para esses métodos são:

- Conexão do auxiliar (a conexão que foi instanciada na linha 139)
- Nome da tabela
- Código da atividade (retornada do select)
- Código da atividade configurada no properties

No restante do método é feito um delete de toda tabela auxiliar caso seja a mesma atividade do select e do properties.



```

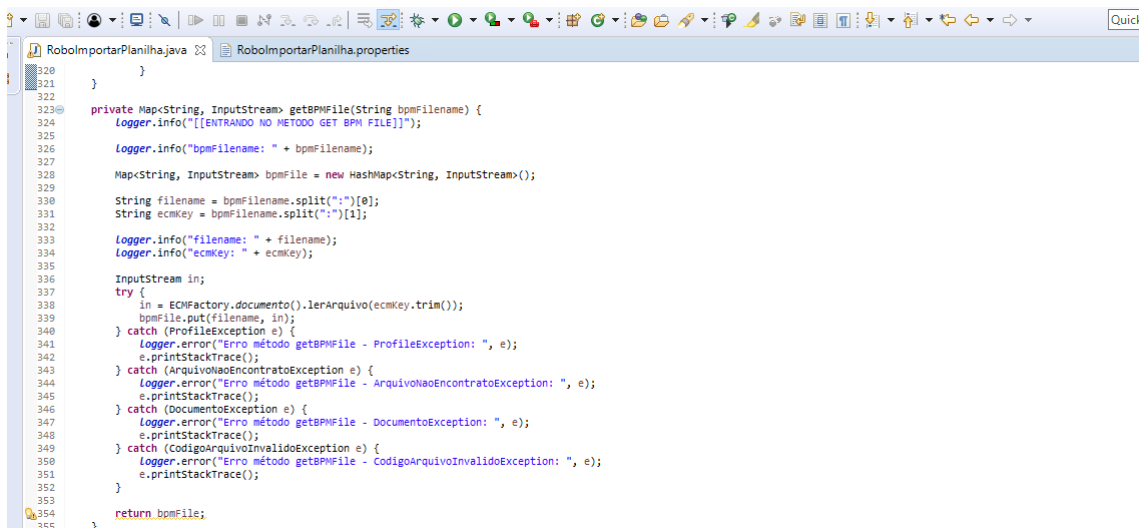
275 private void clearTable(Connection cnbpm, Connection conAux, String codProcesso, String codEtapa, String codCiclo, String codFormAnalise, String codEtapaAnalise, String campo, String tablename) {
276     Logger.info("[Entrando no método Importar Planilha]");
277     Workbook workbook = null;
278     Sheet sheet;
279     String planilha = getPlanilha(cnbpm, codProcesso, codEtapa, codCiclo, codFormAnalise, codEtapaAnalise, campo);
280     Logger.info("planilha: " + planilha);
281     try {
282         Map<String, InputStream> bpmFile = getBPMFile(planilha);
283         String filename = bpmFile.keySet().iterator().next();
284         InputStream is = bpmFile.get(filename);
285         if (filename.endsWith(".xlsx") || filename.endsWith(".xls")) {
286             workbook = new XSSFWorkbook(is);
287         } else if (filename.endsWith(".xls")) {
288             workbook = new HSSFWorkbook(is);
289         }
290         if (workbook != null) {
291             sheet = workbook.getSheetAt(0);
292             int cellnum = 0;
293             int linenum = 0;
294             Row row = sheet.getRow(cellnum);
295             int lastCellnum = row.getLastCellNum();
296             int lastLinenum = sheet.getLastRowNum();
297             List<String> dados = new ArrayList<>();
298             for (linenum = 1; linenum <= lastLinenum; linenum++) {
299                 for (cellnum = 0; cellnum <= lastCellnum-1; cellnum++) {
300                     dados.add(getStringCellValue(sheet, linenum, cellnum));
301                 }
302                 Logger.debug(dados);
303                 insertTable(conAux, codProcesso, tablename, dados);
304                 dados.clear();
305             }
306         } else {
307             Logger.error("[ PLANILHA INVALIDA ]");
308         }
309     } catch (IOException e) {
310         Logger.error("Erro método ImportarPlanilha - IOException: ", e);
311     }
312 }

```

Na imagem acima temos o print do método **importaPlanilha**, os parâmetros passados para esse método é:

- Conexão do bpm (a conexão que foi instanciada na linha 59)
- Conexão do auxiliar (a conexão que foi instanciada na linha 139)
- Código do processo
- Código da atividade
- Código do ciclo
- Código do formulário
- Código da atividade configurada
- Nome do campo
- Nome da tabela

Nesse método é chamado o método **getPlanilha**, que buscara o valor do campo passado na configuração, e partir desse valor ele fará a leitura do arquivo importado no método **getBPMFile**, com o retorno desse método ele fara a utilização da biblioteca poi para leitura de um arquivo xls, e assim ele lê cada linha cada coluna e insere na tabela no método **insertTable**.



```
320 }
321 }
322
323 private Map<String, InputStream> getBPMFile(String bpmFilename) {
324     Logger.info("[[ENTRANDO NO METODO GET BPM FILE]]");
325
326     Logger.info("bpmFilename: " + bpmFilename);
327
328     Map<String, InputStream> bpmFile = new HashMap<String, InputStream>();
329
330     String filename = bpmFilename.split(":")[0];
331     String ecmKey = bpmFilename.split(":")[1];
332
333     Logger.info("filename: " + filename);
334     Logger.info("ecmKey: " + ecmKey);
335
336     InputStream in;
337     try {
338         in = EcmFactory.documento().lerArquivo(ecmKey.trim());
339         bpmFile.put(filename, in);
340     } catch (ProfileException e) {
341         Logger.error("Erro método getBPMFile - ProfileException: ", e);
342         e.printStackTrace();
343     } catch (ArquivoNaoEncontradoException e) {
344         Logger.error("Erro método getBPMFile - ArquivoNaoEncontradoException: ", e);
345         e.printStackTrace();
346     } catch (DocumentoException e) {
347         Logger.error("Erro método getBPMFile - DocumentoException: ", e);
348         e.printStackTrace();
349     } catch (CodigoArquivoInvalidoException e) {
350         Logger.error("Erro método getBPMFile - CodigoArquivoInvalidoException: ", e);
351         e.printStackTrace();
352     }
353
354     return bpmFile;
355 }
```

Na imagem acima temos o print do método **getBPMFile** que irá realizar a leitura do arquivo inserido no campo configurado, os parâmetros passados para esse método é:

- Valor do campo (Nome do arquivo:uniqueiddoECM)

Nesse método ele pega o valor do campo e realiza um split a partir do :, para pegar o valor do uniqueid do ECM, com esse valor ele chama o método da api do ECM de **lerArquivo(documento().lerArquivo())**, esse método retorna um objeto do tipo **InputStream**, que será utilizado no método anterior para realizar a importação da planilha.

```
RobolImportarPlanilha.java RobolImportarPlanilha.properties
356
357 private void insertTable(Connection conAux, String codProcesso, String tablename, List<String> dados) {
358     Logger.info("[[ENTRANDO NO METODO INSERT TABLE]]");
359
360     Logger.debug("tablename: " + tablename);
361
362     int idx = 0;
363     StringBuilder values = new StringBuilder();
364
365     for (String value : dados) {
366         idx++;
367
368         if(value.toString().equals("")) {
369             values.append("null");
370         } else {
371             values.append("'");
372             values.append(value);
373             values.append("'");
374         }
375         if(dados.size() != idx) {
376             values.append(", ");
377         }
378     }
379
380     StringBuilder sql = new StringBuilder();
381     sql.append("insert into " + tablename)
382     .append(" values (")
383     .append(    values.toString() )
384     .append(")");
385
386     Logger.debug("INSERT DADOS: " + sql.toString());
387
388     try (PreparedStatement pst = conAux.prepareStatement(sql.toString())) {
389         pst.execute();
390     } catch (SQLException e) {
391         Logger.error("Erro método insertTable - SQLException: ", e);
392         e.printStackTrace();
393     }
394 }
```

Na imagem acima temos o print do método **insertTable** que irá realizar a ação de inserção dos dados para cada linha do excel importado, recebendo os seguintes parâmetros:

- Conexão auxiliar
- Código do processo
- Nome da tabela auxiliar
- Lista de String dos dados da linha

Nesse método é realizado o insert na tabela, montando corretamente a forma que deve ser inserido a partir da lista de String com o dado daquela linha do excel.

```
RobolImportarPlanilha.java RobolImportarPlanilha.properties
395
396 private void registraAlteracoes(Connection cnbpm, String tablename, String codProcesso, String codEtapa) {
397     Logger.info("[[ENTRANDO NO METODO REGISTRA ALTERACOES]]");
398
399     StringBuilder sql = new StringBuilder();
400     sql.append("insert into ")
401     .append(    tablename)
402     .append(" alteracao ( ")
403     .append("    cod_processo_bpm ")
404     .append("    , data_processamento ")
405     .append("    , usuario_resp_etapa ")
406     .append("    )")
407     .append(" values(?, now(), ?)");
408
409     try (PreparedStatement pst = cnbpm.prepareStatement(sql.toString())) {
410         pst.setString(1, codProcesso);
411         pst.setString(2, codEtapa);
412         pst.execute();
413     } catch (SQLException e) {
414         Logger.error("Erro método registraAlteracoes - SQLException: ", e);
415         e.printStackTrace();
416     }
417 }
```

Na imagem acima temos o print do método **registraAlterações**, recebendo os seguintes parâmetros:

- Conexão auxiliar
- Nome da tabela auxiliar
- Código do processo
- Código da atividade

Nesse método é realizado o insert na tabela de alterações, registrando quando foi feito a importação.



```

RobolImportarPlanilha.java RobolImportarPlanilha.properties
416 }
417
418 private String execucaoAtividade(Connection cnbpm, String codProcesso, String codEtapa, String codCiclo) {
419     Logger.info("[[ENTRANDO NO METODO EXECUTA ATIVIDADES]]");
420
421     String retAprovacao = "";
422
423     try {
424         Map<String, String> parametros = Funcoes.getParametrosIntegracao(configpath + "tarefa.automatica.properties");
425         String login = parametros.get("loginusuarioAutomatico");
426         String senha = parametros.get("senhausuarioAutomatico");
427         String codusuario = parametros.get("codusuarioAutomatico");
428
429         String sso = parametros.get("enderecoSso");
430         String bpm = parametros.get("endereçoBPM");
431         String formApp = parametros.get("endereçoFormAPP");
432
433         String token = gerarAccessToken(login, senha, sso);
434         String teste = modoTeste(codProcesso);
435
436         DadosProcesso dp = new DadosProcesso("P");
437
438         DadosProcessoAbertura procOrigemUtil = new DadosProcessoAbertura();
439         procOrigemUtil.setProcessInstanceId(codProcesso);
440         procOrigemUtil.setCurrentActivityInstanceId(codEtapa);
441         procOrigemUtil.setCurrentCycle(codCiclo);
442         procOrigemUtil.setModoTeste(teste ? "true" : "false");
443
444         AprovaProcesso aprovaProcesso = new AprovaProcesso(bpm, token, procOrigemUtil, dp, procOrigemUtil.getModoTeste(), codusuario);
445         retAprovacao = aprovaProcesso.aprovaProcesso();
446
447         Logger.info("RETORNO APROVACAO: " + retAprovacao);
448
449     } catch (Exception e) {
450         Logger.error("Erro método execucaoAtividade - SQLException: ", e);
451         e.printStackTrace();
452     }
453     return retAprovacao;
454 }

```

Na imagem acima temos o print do método **execucaoAtividade** que irá realizar a ação de aprovação/rejeição dos processos, recebendo os seguintes parâmetros:

- Conexão do BPM
- Código do processo
- código da atividade
- código do ciclo

Nesse método é utilizada classes do jar RotasBPM, que é utilizado para ações do formulário novo, esse projeto foi desenvolvido pela nossa equipe de consultoria para utilizar as apis disponibilizadas pelo produto, sobre esse tema você pode ver mais afundo no documento RotasBPM, nesse método ele retorna os parâmetros do arquivo automatico e assim utiliza as classes de DadosProcessoAbertura para preenchimento das informações de código de processo, etapa, ciclo e modo teste, que são essenciais para que a classe AprovaProcesso funcione e realize a aprovação/rejeição automática do processo.

```

RobolImportarPlanilha.java RobolImportarPlanilha.properties
453 } catch (Exception e) {
454     Logger.error("Erro método execucaoAtividade - SQLException: ", e);
455     e.printStackTrace();
456 }
457 return retAprovacao;
458 }
459
460
461 private String gerarAccessToken(String loginRobo, String senhaRobo, String urlSso) throws Exception, LoginAuthenticationException {
462     Logger.info("[[ENTRANDO NO METODO GERAR ACCESS TOKEN]]");
463
464     String token = "";
465
466     try {
467         DadosLogin loginUtil = new DadosLogin(loginRobo, senhaRobo, false);
468         LoginAutenticacao loginAutenticacao = new LoginAutenticacao(urlSso, loginUtil);
469         token = loginAutenticacao.getToken();
470     } catch (Exception e) {
471         Logger.error("[[ ERRO ] : ", e);
472         System.out.println("[[ ERRO ] : " + Funcoes.exceptionPrinter(e));
473     }
474     return token;
475 }
476

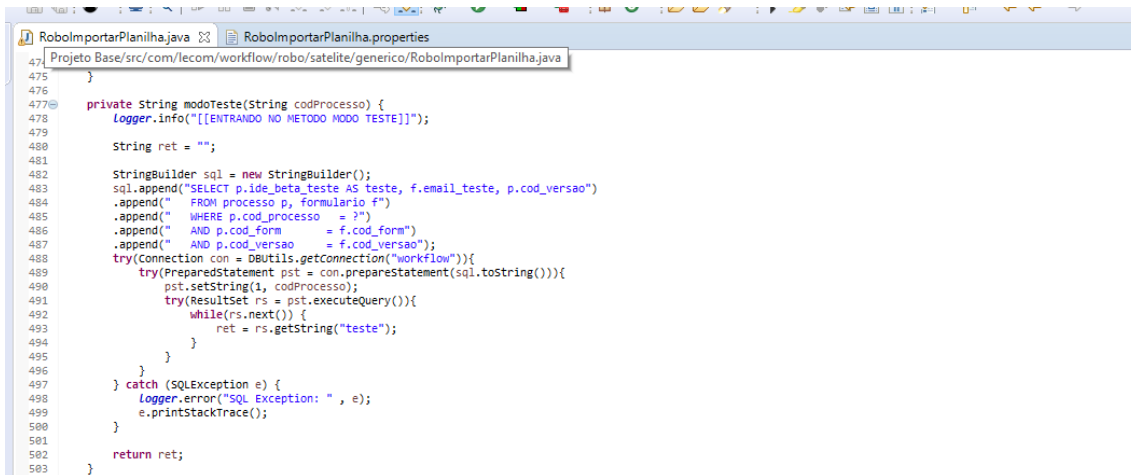
```

Na imagem acima temos o print do método **gerarAccessToken**, método utilizado para realizar o login por fora da ferramenta ele e assim ser utilizado o token retornado nos métodos de cancelamento e aprovação de atividades, ele recebe os seguintes parâmetros:

- login do usuário automatico
- senha do usuário automatico

- url do sso

Nesse método é utilizada classes do jar RotasBPM, que é utilizado para ações do formulário novo, esse projeto foi desenvolvido pela nossa equipe de consultoria para utilizar as apis disponibilizadas pelo produto, sobre esse tema você pode ver mais afundo no documento RotasBPM, ele utiliza o objeto DadosLogin para preencher as informações de login, senha do automatico e utiliza a classe LoginAutenticacao que fará o login por fora e retornará o token.



```

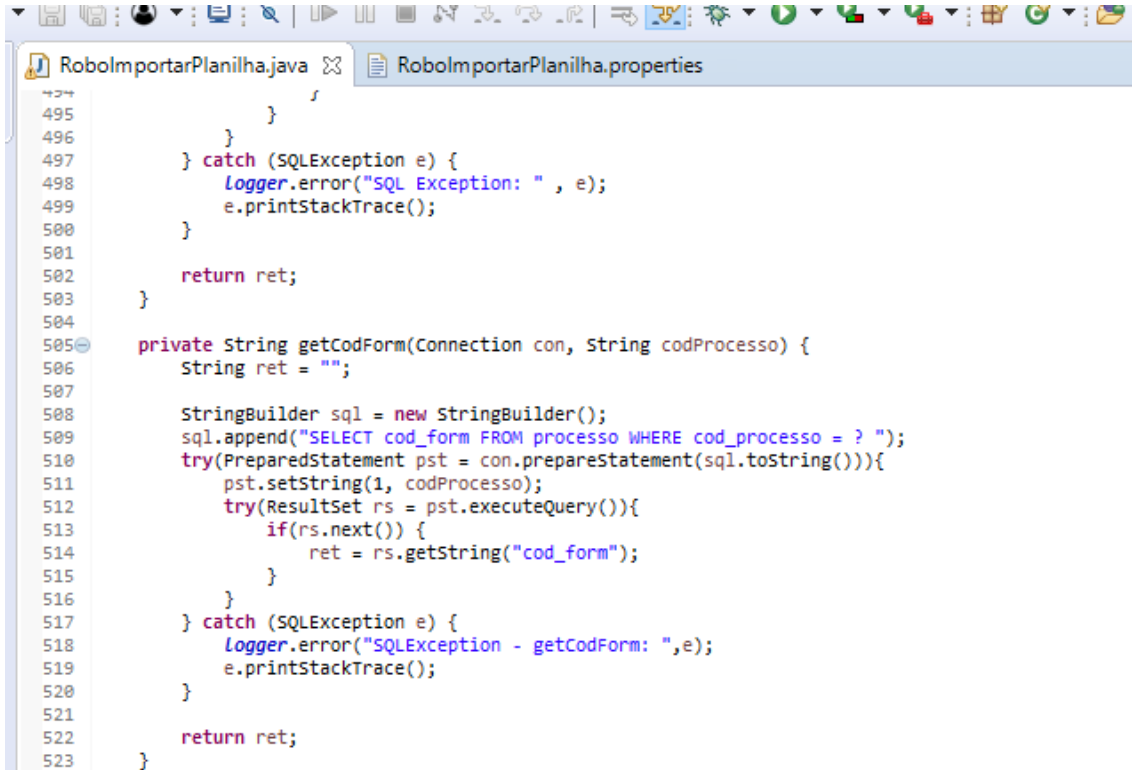
475 }
476
477 private String modoTeste(String codProcesso) {
478     logger.info("[[ENTRANDO NO METODO MODO TESTE]]");
479
480     String ret = "";
481
482     StringBuilder sql = new StringBuilder();
483     sql.append("SELECT p.ide_beta_teste AS teste, f.email_teste, p.cod_versao")
484     .append(" FROM processo p, formulario f")
485     .append(" WHERE p.cod_processo = ?")
486     .append(" AND p.cod_form = f.cod_form")
487     .append(" AND p.cod_versao = f.cod_versao");
488     try(Connection con = DBUtils.getConnection("workflow")){
489         try(PreparedStatement pst = con.prepareStatement(sql.toString())){
490             pst.setString(1, codProcesso);
491             try(ResultSet rs = pst.executeQuery()){
492                 while(rs.next()) {
493                     ret = rs.getString("teste");
494                 }
495             }
496         } catch (SQLException e) {
497             logger.error("SQL Exception: ", e);
498             e.printStackTrace();
499         }
500     }
501     return ret;
502 }
503

```

Na imagem acima temos o print do método **modoTeste**, ele recebe o seguinte parâmetros:

- Código de processo

Nesse método ele realiza um select para retornar se esse processo está em modo teste ou modo normal para utilizar na aprovação automática.



```

495     }
496   }
497   } catch (SQLException e) {
498     logger.error("SQL Exception: ", e);
499     e.printStackTrace();
500   }
501
502   return ret;
503 }
504
505 private String getCodForm(Connection con, String codProcesso) {
506   String ret = "";
507
508   StringBuilder sql = new StringBuilder();
509   sql.append("SELECT cod_form FROM processo WHERE cod_processo = ? ");
510   try(PreparedStatement pst = con.prepareStatement(sql.toString())){
511     pst.setString(1, codProcesso);
512     try(ResultSet rs = pst.executeQuery()){
513       if(rs.next()) {
514         ret = rs.getString("cod_form");
515       }
516     }
517   } catch (SQLException e) {
518     logger.error("SQLException - getCodForm: ",e);
519     e.printStackTrace();
520   }
521
522   return ret;
523 }

```

Na imagem acima temos o print do método **getCodForm**, ele recebe o seguinte parâmetros:

- Conexão do lecom BPM
- Código de processo

Nesse método ele realiza um select para retornar qual o código do formulário relacionado ao código do processo retornado no primeiro select feito nesse fonte.