PROJETO FRAMEWORK – CELEPAR

CATÁLOGO DE COMPONENTES

Junho – 2006

Sumário de Informações do Documento

Tipo do Documento: Referência

Título do Documento: Catálogo de Componentes

Estado do Documento: Elaborado

Responsáveis: Filipe Lautert, João Eduardo Mikos, Cleverson Budel – Revisado por Elisabeth Hoffmann

Palavras-Chaves: Componentes, java, hibernate, struts, reuso, taglib

Resumo: Descrição dos componentes gerados para o Grupo Framework

Número de páginas: 41

Software utilizados:

Versão	Data	Mudanças
1.0'	04/02/05	Criação do documento.
1.1'	06/04/05	Criação da versão 1.0.1 dos componentes. Alterações: adicionado método Data.dataDiff . Adicionado trim em parametros específicos da tag Grid. Adicionados flush() dentro da classe HibernateUtil.
1.2'	30/09/05	Adicionado descrição do método validaForm e parâmetro decorator na tag de Grid.
1.3'	27/12/05	Adicionada descrição na tag de lista para desabilitar os botões de passagem dos itens.
1.4'	14/02/06	Adicionada descrição do método Agendador e do método setLifeTime() da classe FileGC.
1.5	21/03/06	Adicionada descrição da string de formatação de abreviaturas, revisado por Elisabeth Hoffmann.
1.6	29/06/06	Alterado documento com revisões do Framework efetuadas até sua versão 2.0, por Helio S. Piccinatto. • Excluído pacote database.log com a Classe BaseHibernateLogDAO e a Classe LogBD. • Incluido descrição das classes dos sub pacotes de taglib, Lista, Decorator e Grid. • Excluídos os métodos openSession, closeSession, getSession e setSession da classe BaseHibernateDAO. • Incluído métodos currentTransaction, commitTransaction, rollbackTransaction na classe HibernateUtil. • Incluído métodos removerAgendamento, listarTrabalhos na classe Agendador. • Incluído métodos setList, putJasperParameter, getFilterParameter(String) na classe ReportDefinition. • Incluído método getFieldValue(JRField) na classe ReportDS. • Incluído métodos abreviarString(String), verificarPreposicao(String) e abreviarTipoLogradouro(String) na classe Abreviatura. • Incluído métodos formataHoraLong(Date), formataHoraCurta(Date), mesDiff(Date, Date) e anoDiff(Date, Date) na classe Data. • Incluído método classForName(String) na classe Reflexao.
1.7	04/05/07	Cleverson Budel • Incluído método setTitulacaoLogradouroAbreviacoes na classe Abreviatura. • Criada as classes CNPJ, CPF e Modulo11. • Incluído método validarModulo11 na classe StrutsValidator. • Incluída propriedades ordenada e moverTodas na taglib de Lista.
1.8	22/08/07	Thiago Meneghello • Criada a classes GenericDAO e GenericHibernateDAO • Removida a classe BaseHibernateDAO. • Criada as classes de Upload. • Criada as classes HibernateUtil(Configuration, Factory e Interface). • Criada a taglib Upload. • Incluído métodos tentarAteAbreviarLogradouro e tentarAteAbreviarNome da classe Abreviatura. • Incluído métodos das classes de Upload. • Incluído métodos das classes HibernateUtil(Configuration, Factory e Interface). Revisado por Cíntia Evangelista.
1.9	02/04/08	Jonatas Chagas Complementada a documentação dos métodos das seguintes classes: • Agrupador • BaseDispatchAction • GenericHibernateDAO • HibernateUtil • HibernateUtilImpl • HibernateUtilInteface • Agendador • FileGC • JobAgendador • LoadImageJasper • ReportDefinition • Abreviatura • BarCode2of5 • Data • StringUtil Revisado por Cleverson Budel

SUMÁRIO

1 CATÁLOGO DE COMPONENTES....7
1.1 INTRODUÇÃO....7
2 PACOTES....7
2.1 GOV.PR.CELEPAR.FRAMEWORK.ACTION....7
2.1.1 CLASSE BASEDISPATCHACTION....7
2.2 GOV.PR.CELEPAR.FRAMEWORK.DATABASE....8
2.2.1 CLASSE GENERICDAO....8
2.2.2 CLASSE GENERICHIBERNATEDAO....8
2.2.3 CLASSE HIBERNATEUTIL....8
2.2.4 CLASSE HIBERNATEUTILCONFIGURATION....8
2.2.5 CLASSE HIBERNATEUTILFACTORY....8
2.2.6 INTERFACE HIBERNATEUTILINTERFACE....8
2.3 GOV.PR.CELEPAR.FRAMEWORK.EXCEPTION....9
2.3.1 CLASSE APPLICATIONEXCEPTION....9
2.4 GOV.PR.CELEPAR.FRAMEWORK.JMS....9
2.4.1 CLASSE QUEUELISTENER....9
2.5 GOV.PR.CELEPAR.FRAMEWORK.MENSAGEM....10
2.5.1 CLASSE MENSAGEMINTERFACE....11
2.5.2 CLASSE MENSAGEMPR....11
2.5.3 CLASSE MENSAGEMBD....11
2.5.4 CLASSE MENSAGEM....11
2.6 GOV.PR.CELEPAR.FRAMEWORK.REPORT....11
2.6.1 CLASSE AGENDADOR....11
2.6.2 CLASSE FILEGC....12
2.6.3 CLASSE JOBAGENDADOR....12
2.6.4 CLASSE LOADIMAGEJASPERHTML....12
2.6.5 CLASSE REPORTDEFINITION....12
2.6.6 CLASSE REPORTDS....12
2.7 GOV.PR.CELEPAR.FRAMEWORK.TAGLIB....12
2.7.1 CLASSE CALENDARIO....12
2.7.2 CLASSE RADIOLIST....13
2.7.3 SUBPACOTE GRID....13
2.7.3.1 CLASSE ACTION....13
2.7.3.2 CLASSE ACTIONBEAN....13
2.7.3.3 CLASSE AGRUPADOR....13
2.7.3.4 CLASSE AGRUPADORBEAN....13
2.7.3.5 CLASSE CAMPO....13
2.7.3.6 CLASSE CAMPOBEAN....13
2.7.3.7 CLASSE CHECKBOX....13

2.7.3.8 CLASSE CHECKBOXBEAN....13
2.7.3.9 CLASSE LISTA....14
2.7.3.10 CLASSE PAINEL....14
2.7.3.11 CLASSE TABLE....14
2.7.3.12 SUBPACOTE DECORATOR....14
2.7.3.12.1 CLASSE DATA....14
2.7.3.12.2 CLASSE DATAEXTENSO....14
2.7.3.12.3 CLASSE DATAHORA....14
2.7.3.12.4 CLASSE DECORATOR....14
2.7.3.10.5 CLASSE HORACURTA....14
2.7.3.12.6 CLASSE HORALONGA....14
2.7.3.10.7 CLASSE MOEDA....15
2.7.4 SUBPACOTE LISTA....15
2.7.4.1 CLASSE CAIXA....15
2.7.4.2 CLASSE OPCOES....15
2.7.4.3 CLASSE SELECT....15
2.7.5 SUBPACOTE UPLOAD....15
2.7.5.1 CLASSE PROGRESSUPLOADTAGLIB....15
2.8 GOV.PR.CELEPAR.FRAMEWORK.UPLOAD....15
2.8.1 INTERFACE LIMITUPLOADHANDLER....16
2.8.2 CLASSE PROGRESSMONITORFILEITEM....16
2.8.3 CLASSE PROGRESSMONITORFILEITEMFACTORY....16
2.8.4 INTERFACE PROGRESSOBSERVER....16
2.8.5 CLASSE PROGRESSREQUESTHANDLER....16
2.8.6 CLASSE UNLIMITEDUPLOADHANDLER....16
2.8.7 SUBPACOTE FILTER....16
2.8.7.1 CLASSE PROGRESSUPLOADFILTER....16
2.9 GOV.PR.CELEPAR.FRAMEWORK.UTIL....17
2.9.1 CLASSE ABREVIATURA....17
2.9.2 CLASSE BARCODE2OF5....17
2.9.3 CLASSE CNPJ....17
2.9.4 CLASSE CPF....17
2.9.5 CLASSE DATA....17
2.9.6 CLASSE MODULO11....18
2.9.7 CLASSE PAGINA....18
2.9.8 CLASSE REFLEXAO....18
2.9.9 CLASSE STRINGUTIL....18
2.9.10 CLASSE VALORES....18
2.10 GOV.PR.CELEPAR.FRAMEWORK.VALIDATOR....18
2.10.1 CLASSE STRUTSVALIDATOR....18
2.11 GOV.PR.CELEPAR.FRAMEWORK.XML....19
2.11.1 CLASSE ATTRIBUTE....19
2.11.2 CLASSE ELEMENT....19

2.11.3 CLASSE NODE.....19
3 MÉTODOS.....19
3.1 GOV.PR.CELEPAR.FRAMEWORK.ACTION.....19
3.1.1 MÉTODOS DA CLASSE BASEDISPATCHACTION.....19
3.2 GOV.PR.CELEPAR.FRAMEWORK.DATABASE.....20
3.2.1 MÉTODOS DA CLASSE GENERICDAO.....20
3.2.2 MÉTODOS DA CLASSE GENERICHIBERNATEDAO.....20
3.2.3 MÉTODOS DA CLASSE HIBERNATEUTIL.....21
3.2.4 MÉTODOS DA CLASSE HIBERNATEUTILCONFIGURATION.....21
3.2.5 MÉTODOS DA CLASSE HIBERNATEUTILFACTORY.....22
3.2.6 MÉTODOS DA INTERFACE HIBERNATEUTILINTERFACE.....22
3.3 GOV.PR.CELEPAR.FRAMEWORK.EXCEPTION.....22
3.3.1 MÉTODOS DA CLASSE APPLICATIONEXCEPTION.....22
3.4 GOV.PR.CELEPAR.FRAMEWORK.MENSAGEM.....23
3.4.1 MÉTODOS DA CLASSE MENSAGEM.....23
3.4.2 MÉTODOS DA CLASSE MENSAGEMINTERFACE.....23
3.4.3 MÉTODOS DAS CLASSES MENSAGEMPR E MENSAGEMBD.....23
3.5 GOV.PR.CELEPAR.FRAMEWORK.REPORT.....23
3.5.1 MÉTODOS DA CLASSE AGENDADOR.....23
3.5.2 MÉTODOS DA CLASSE FILEGC.....24
3.5.3 MÉTODOS DA CLASSE JOBAGENDADOR.....24
3.5.4 MÉTODOS DA CLASSE LOADIMAGEJASPERHTML.....24
3.5.5 MÉTODOS DA CLASSE REPORTDEFINITION.....24
3.5.6 MÉTODOS DA CLASSE REPORTDS.....25
3.6 GOV.PR.CELEPAR.FRAMEWORK.TAGLIB.....25
3.6.1 TAG CALENDARIO.....25
3.6.2 TAG RADIOLIST.....26
3.6.3 SUBPACOTE GRID.....26
3.6.3.1 TAG TABLE.....26
3.6.4 SUBPACOTE LISTA.....28
3.6.4.1 TAG LISTA.....28
3.6.5 SUBPACOTE UPLOAD.....29
3.6.5.1 TAG PROGRESSUPLOADTAGLIB.....29
3.7 GOV.PR.CELEPAR.FRAMEWORK.UPLOAD.....30
3.7.1 MÉTODO DA INTERFACE LIMITUPLOADHANDLER.....30
3.7.2 MÉTODOS DA CLASSE PROGRESSMONITORFILEITEM.....30
3.7.3 MÉTODOS DA CLASSE PROGRESSMONITORFILEITEMFACTORY.....30
3.7.4 MÉTODOS DA INTERFACE PROGRESSOBSERVER.....30
3.7.5 MÉTODOS DA CLASSE PROGRESSREQUESTHANDLER.....30
3.7.6 MÉTODOS DA CLASSE UNLIMITEDUPLOADHANDLER.....30
3.7.7 SUBPACOTE FILTER.....30
3.8 GOV.PR.CELEPAR.FRAMEWORK.UTIL.....31
3.8.1 MÉTODOS DA CLASSE ABREVIATURA.....31

3.8.2 MÉTODOS DA CLASSE BARCODE2OF5.....34
3.8.3 MÉTODOS DA CLASSE CNPJ.....34
3.8.4 MÉTODOS DA CLASSE CPF.....35
3.8.5 MÉTODOS DA CLASSE DATA.....35
3.8.6 MÉTODOS DA CLASSE MODULO11.....37
3.8.7 MÉTODOS DA CLASSE PAGINA.....37
3.8.8 MÉTODOS DA CLASSE REFLEXAO.....37
3.8.9 MÉTODOS DA CLASSE STRINGUTIL.....38
3.9 PR.GOV.CELEPAR.FRAMEWORK.VALIDATOR.....40
3.9.1 MÉTODOS DA CLASSE STRUTSVALIDATOR.....40
3.10 GOV.PR.CELEPAR.FRAMEWORK.XML.....40

1 CATÁLOGO DE COMPONENTES

1.1 Introdução

O catálogo de componentes foi criado com a finalidade de auxiliar o desenvolvedor no conhecimento dos componentes de infra-estrutura que foram criados e/ou padronizados.

Boa parte dos componentes utilizados no catálogo já existiam na empresa e foram criados por diversos autores. O serviço realizado pelo Grupo Framework foi de reuní-los, filtrar os componentes com boa possibilidade de reuso e padronizá-los, criando testes unitários para validar as suas funcionalidades.

Além dos componentes pré-existentes ao processo, foram criados alguns extras para padronizar e facilitar algumas etapas do desenvolvimento.

Este catálogo é apresentado obedecendo a divisão por pacotes e classes, abordando assim as funcionalidades de cada grupo de classes.

2 PACOTES

2.1 gov.pr.celepar.framework.action

2.1.1 Classe BaseDispatchAction

Responsável por direcionar as requisições dos usuários, especializando o tratamento de exceções e o controle de mensagens. Implementa também, métodos utilitários para todas as Actions.

2.2 gov.pr.celepar.framework.database

O pacote database auxilia nas conexões e funcionalidades básicas para acesso a banco de dados utilizando Hibernate.

2.2.1 Classe GenericDAO

Interface que define os métodos que devem existir nas classes GenericDAOs.

2.2.2 Classe GenericHibernateDAO

Classe que implementa a interface GenericDAO, conseqüentemente fornecendo vários métodos de uso genérico, da tecnologia Hibernate, para facilitar a construção de classes DAO.

2.2.3 Classe HibernateUtil

Lê arquivo de configuração do Hibernate gerando o SessionFactory e o Configuration. Cria singleton da Session para haver somente uma sessão em todo o sistema. Fornece métodos static para recuperar e encerrar a sessão e manipular transações.

2.2.4 Classe HibernateUtilConfiguration

Classe responsável em criar e armazenar o SessionFactory e o Configuration do Hibernate. Utilizada apenas pela classe HibernateUtilFactory.

2.2.5 Classe HibernateUtilFactory

Classe responsável em controlar os SessionFactorys e os Configurations, e garantir instâncias únicas por aplicações.

Cria singleton da HibernateUtilInterface por Thread.

2.2.6 Interface HibernateUtilInterface

Interface que deve fornecer métodos para recuperar a sessão e para trabalhar com transações.

2.3 gov.pr.celepar.framework.exception

2.3.1 Classe ApplicationException

Classe padrão para tratamento de exceções, usada para notificar as camadas superiores de que ocorreu um erro na aplicação.

2.4 gov.pr.celepar.framework.jms

2.4.1 Classe QueueListener

Classe responsável em criar os MessageListeners para filas JMS (Java Message Service).

Para fazer uso desta classe, basta configurar o arquivo web.xml e inserir as linhas:

```
<listener>
    <listener-class>
        gov.pr.celepar.framework.jms.QueueListener
    </listener-class>
</listener>
<context-param>
    <param-name>gov.pr.celepar.framework.jms.queue.config</param-name>
    <param-value>
        queue/nomeFila=Caminho das classes
        queue/filaExemplo=gov.pr.celepar.jms.ExemploListener
        . . .
    </param-value>
```

</context-param>

Os caminhos das classes devem conter um ou mais caminhos de classes separadas por "ponto e vírgula" (;), que serão utilizados para processar as mensagens quando essas forem criadas para as respectivas filas.

Ex:

queue/nome1=gov.pr...Classe1

queue/nome2=gov.pr...Classe1;gov.pr...Classe2;...;gov.pr...ClasseN

Podemos instanciar mais de um objeto da mesma classe para processar as mensagens, para isso, basta adicionar o caracter opcional "arroba" (@) após o caminho da classe, e complementar com um número inteiro que indica a quantidade de instancias a serem criadas dessa classe.

Ex:

queue/nome1=gov.pr...Classe1@**3**

queue/nome2=gov.pr...Classe1@**5**;gov.pr...Classe2@**10**;...;gov.pr...ClasseN@**M**

2.5 gov.pr.celepar.framework.mensagem

Este pacote provê uma maneira única para acessar mensagens do sistema, independente da forma na qual elas estejam salvas. Através de uma interface que deve ser implementada para utilizar o pacote, a classe Mensagem proporciona métodos para gerar singletons e manter a unicidade da classe de mensagens no sistema. Duas classes de mensagens implementam a interface e já estão prontas para uso: MensagemPr e MensagemBD.

2.5.1 Classe MensagemInterface

Interface que encapsula as mensagens do sistema. Para facilitar seu uso, utilizar a classe Mensagem. Classes que pretendem usar a classe Mensagem devem implementá-la.

2.5.2 Classe MensagemPr

Encapsula mensagens encontradas em arquivo de properties. Utiliza métodos do framework Struts para realizar esta tarefa. É altamente configurável, aceitando alterar o arquivo de properties e executar trocas de parâmetros nas strings.

2.5.3 Classe MensagemBD

Encapsula mensagens do sistema encontradas em uma tabela especificada, que deve conter os campos com as propriedades id e valor dessa classe. Utiliza por default o método HibernateUtil.getSession para receber a sessão Hibernate, possibilitando ainda que sejam especificadas outras sessões do Hibernate através de seus métodos.

2.5.4 Classe Mensagem

Fornece métodos para manipular classes de mensagens, que implementem a MensagemInterface, e mantenham-nas únicas em todo o sistema. Provê métodos static para recuperar a classe de mensagens e utilizá-la com maior facilidade.

2.6 gov.pr.celepar.framework.report

Conjunto de classes utilizadas na geração, visualização e agendamento de relatórios.

2.6.1 Classe Agendador

Classe utilizada para agendar relatórios. Utiliza a ferramenta Quartz para realizar agendamentos de relatórios, tanto relatórios que devam ocorrer somente uma vez, como relatórios semanais e de outras periodicidades.

2.6.2 Classe FileGC

Essa classe é um job do Quartz que verifica se existem arquivos PDF, resultantes de agendamentos que existem a mais de 15 dias e os remove, bem como seus jobs (se eles não estiverem agendados para rodar novamente).

2.6.3 Classe JobAgendador

A classe JobAgendador executa um relatório Jasper e salva seu resultado no filesystem. Gera logs do horário de início e fim, para totalização e visualização de quais relatórios consomem muito tempo para execução.

2.6.4 Classe LoadImageJasperHtml

Servlet utilizado para carregar as imagens do relatório jasper em HTML.

2.6.5 Classe ReportDefinition

Classe que serve como base para o desenvolvimento de relatório utilizando JasperReports, nesta classe já esta implementada toda a parte do controle dos objetos que serão passados para o Jasper e geração dos buffers para o output dos relatórios.

2.6.6 Classe ReportDS

Classe que funciona como um DataSource, operando de maneira genérica a todos os reports, onde conterá e controlará os dados que serão requisitados pelo relatório.

2.7 gov.pr.celepar.framework.taglib

Classes de taglibs que visam facilitar a criação da parte HTML do sistema. Automatizam partes mais repetitivas acelerando assim o processo de desenvolvimento.

2.7.1 Classe Calendario

Componente que gera calendário a partir do mês e ano informado, gerando links nos botões do calendário. Para utilizar este componente deve ser implementado um arquivo css

contendo os estilos dos botões do calendário.

2.7.2 Classe RadioList

RadioList básica. Pode tanto ser usada normalmente como estendida para criar novas tags de Radiolist. A radioList é composta de vários botões do tipo radio com o mesmo nome, gerando uma lista do tipo “escolha um”.

2.7.3 SubPacote Grid

Desenha tabela de dados. Altamente configurável, recebe uma collection e pode gerar os controles de incluir, alterar, excluir e criar links ou links javascripts para as páginas desejadas. Gera a tabela no novo padrão da CELEPAR.

2.7.3.1 Classe Action

Adiciona coluna de ação na tabela.

2.7.3.2 Classe ActionBean

Representa uma coluna de alguma ação na tabela.

2.7.3.3 Classe Agrupador

Adiciona um agrupador de linhas na tabela.

2.7.3.4 Classe AgrupadorBean

Representa um Agrupador de linhas na tabela.

2.7.3.5 Classe Campo

Acrescenta uma coluna de campo na tabela.

2.7.3.6 Classe CampoBean

Representa uma coluna de campo na tabela.

2.7.3.7 Classe CheckBox

Adiciona uma coluna de checkbox na tabela.

2.7.3.8 Classe CheckBoxBean

Representa uma coluna de campo na tabela.

2.7.3.9 Classe Lista

Adiciona uma collection com o conteúdo que deve ser mostrado na tabela.

2.7.3.10 Classe Painel

Desenha o painel de navegação dos registros na tabela.

2.7.3.11 Classe Table

Desenha o grid depois de todas as informações necessárias a serem colocadas por suas tags complementares (subtags).

2.7.3.12 SubPacote Decorator

2.7.3.12.1 Classe Data

Implementação Decorator para formatar um Object (java.util.Date) em uma String no formato dd/MM/yyyy.

2.7.3.12.2 Classe DataExtenso

Implementação Decorator para formatar um Object (java.util.Date) em uma String da data por extenso (ex: 04 de março de 1997).

2.7.3.12.3 Classe DataHora

Implementação Decorator para formatar um Object (java.util.Date) em uma String no formato dd/MM/yyyy HH:mm:ss.

2.7.3.12.4 Classe Decorator

Interface a ser implementada para suportar o parâmetro decorator da tag "campo" da tag de grid "table".

2.7.3.10.5 Classe HoraCurta

Implementação Decorator para formatar um Object (java.util.Date) em uma String no formato HH:mm.

2.7.3.12.6 Classe HoraLonga

Implementação Decorator para formatar um Object (java.util.Date) em uma String no

formato HH:mm:ss.

2.7.3.10.7 Classe Moeda

Implementação Decorator para formatar um Object (Double, Float, Long) em uma String que representa o número na formatação monetária.

2.7.4 SubPacote Lista

Gera duas listas que podem ser permutadas utilizando setas existentes entre elas. Usadas normalmente para atribuir e retirar pessoas de grupos, permissões de grupos, etc.

2.7.4.1 Classe Caixa

Gera uma lista de opções.

2.7.4.2 Classe Opcoes

Gera a lista de opções a ser colocada dentro de uma caixa.

2.7.4.3 Classe Select

Desenha a região com as duas caixas de lista.

2.7.5 SubPacote Upload

2.7.5.1 Classe ProgressUploadTagLib

Cria um onde serão atualizadas as estatísticas de upload, enquanto o arquivo está sendo enviado para o servidor.

2.8 gov.pr.celepar.framework.upload

Pacote com classes utilizadas no componente responsável em mostrar estatísticas na tela, como: nome, tamanho total, quantidade que já foi feito de upload, tempo total, tempo estimado para término, porcentagem de conclusão; enquanto o upload de um arquivo está sendo realizado.

2.8.1 Interface LimitUploadHandler

Interface que pode ser implementada pela aplicação, deve retornar o tamanho máximo de arquivo que pode ser feito upload.

2.8.2 Classe ProgressMonitorFileItem

Classe Interna do framework responsável em monitorar a quantidade de dados que foi enviada para o servidor.

2.8.3 Classe ProgressMonitorFileItemFactory

Classe Interna do framework utilizada para criar novas instâncias de ProgressMonitorFileItem.

2.8.4 Interface ProgressObserver

Interface Interna do framework para atualizar as informações de upload de arquivo.

2.8.5 Classe ProgressRequestHandler

Classe responsável em tratar as requisições do tipo multipart/form-data para o struts. Caso a aplicação desejar utilizar o componente de estatística de upload, essa classe deve ser configurada no arquivo struts-config.xml e inserida na tag controller da seguinte forma:

```
<controller
   . . .
   multipartClass="gov.pr.celepar.framework.upload.ProgressRequestHandler"/>
```

2.8.6 Classe UnlimitedUploadHandler

Classe que implementa a interface LimitUploadHandler e torna o tamanho máximo do arquivo, para upload ilimitado.

2.8.7 SubPacote filter

2.8.7.1 Classe ProgressUploadFilter

Classe utilizada como filtro, responsável em atualizar a página de estatística quando estamos fazendo o upload de um arquivo. Caso a aplicação necessite utilizar o componente de estatística de upload, essa classe deve ser configurada no arquivo web.xml e inserida na tag de

filtro da seguinte forma:

```
<filter>
   <filter-name>UploadFilter</filter-name>
   <filter-class>gov.pr.celepar.framework.upload.filter.ProgressUploadFilter
   </filter-class>
   <init-param>
      <param-name>limitUploadHandler</param-name>
      <param-value>gov.pr.celepar.framework.upload.UnlimitedUploadHandler
      </param-value>
   </init-param>
   <init-param>
      <param-name>paginaErro</param-name>
      <param-value>pages/ctlr_erro_upload.jsp</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>UploadFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

2.9 gov.pr.celepar.framework.util

Pacote com classes de uso geral, que fornecem facilidades ao desenvolvedor.

2.9.1 Classe Abreviatura

Classe utilitária para realizar abreviações de nomes de pessoas e logradouros. Permite configuração do que abreviar e adição de itens a serem abreviados. A maior utilidade desta classe é para abreviar logradouros.

2.9.2 Classe BarCode2of5

Essa classe tem por objetivo a geração de códigos de barra seguindo o padrão brasileiro (FEBRABAN).

2.9.3 Classe CNPJ

Responsável pelas operações (validar, formatar, etc) referentes a CNPJ.

2.9.4 Classe CPF

Responsável pelas operações (validar, formatar, etc) referentes a CPF.

2.9.5 Classe Data

Métodos estáticos para manipulação de data. A maioria das operações necessárias e

complicadas com datas como conversões, soma de dias, comparações, entre outros estão presentes nessa classe.

2.9.6 Classe Modulo11

Responsável pelas operações referentes ao algoritmo de validação do módulo 11.

2.9.7 Classe Pagina

A classe Pagina é uma classe helper utilizada para passar à taglib de grid, uma Collection a ser iterada e parâmetros necessários para que seja realizada a paginação dos registros.

2.9.8 Classe Reflexao

Visa facilitar o uso da API Reflection através de métodos que generalizam o seu uso para beans e POJO's em geral.

2.9.9 Classe StringUtil

Utilitários para formatação/utilização de Strings, além de conversões de formato de códigos de documentos.

2.9.10 Classe Valores

Utilitários para formatação/utilização de valores (inteiros e floats).

2.10 gov.pr.celepar.framework.validator

2.10.1 Classe StrutsValidator

Classe que implementa alguns métodos para serem utilizáveis como validadores plugáveis do Struts Validator, não encontradas como default do Struts.

2.11 gov.pr.celepar.framework.xml

Pacote utilizado para criação de XML's. Facilita a geração dos XML's por conter diversas chamadas com mais e menos parâmetros para a criação dos mesmos elementos e itens.

2.11.1 Classe Attribute

Utilizado em conjunto com a classe Element para gerar páginas XML. Representa um atributo de uma tag XML.

2.11.2 Classe Element

Utilizado em conjunto com a classe Attribute para gerar páginas XML. Representa uma tag XML, com seus valores e atributos.

2.11.3 Classe Node

Classe abstract para criação de tags XML. Deve ser estendida para criar tags/atributos com características especiais.

3 MÉTODOS

3.1 gov.pr.celepar.framework.action

3.1.1 Métodos da Classe BaseDispatchAction

- execute(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse): Método sobrescrito para especializar o tratamento de exceções.
- addAlertMessage(String, HttpServletRequest): Adiciona mensagem de alerta no escopo da página.
- addAlertMessage(String, String[], HttpServletRequest): Adiciona mensagem de alerta no escopo da página, substituindo as chaves da mensagem pelos argumentos passados na Array de Strings.
- addMessage(String, HttpServletRequest): Adiciona mensagem no escopo da página.

- addMessage(String, String[], HttpServletRequest): Adiciona mensagem no escopo da página, substituindo as chaves da mensagem pelos argumentos passados na Array de Strings.
- validaForm(ActionMapping, ActionForm, HttpServletRequest): Invoca a validação do ActionForm associado, inserindo as mensagens de erro caso este não seja válido.
- getActionForward(): retorna um objeto ActionForward.
- setActionForward(ActionForward): armazena o ActionForward no contexto.
- resetActionForward(): remove o ActionForward do contexto.

3.2 gov.pr.celepar.framework.database

3.2.1 Métodos da Classe GenericDAO

- salvar(T): Salva o objeto no banco.
- alterar(T): Altera as informações do objeto no banco.
- excluir(T): Exclui o objeto no banco.
- obter(ID): Obtém o objeto através do ID.
- listar(): Lista os objetos do banco.
- listar(Integer, Integer): Lista os objetos do banco, de forma paginada.
- listar(T): Lista os objetos do banco com restrição dos campos setados no exemplo passado como argumento.
- listar(T, String[]): Lista os objetos do banco com restrição dos campos setados no exemplo passado como argumento, e ordena o resultado conforme os valores passados no segundo argumento.
- listar(T, Integer, Integer): Lista os objetos do banco com restrição dos campos setados no exemplo passado como argumento, de forma paginada.
- listar(T, String[], Integer, Integer): Lista os objetos do banco com restrição dos campos setados no exemplo passado como argumento, e ordena o resultado conforme os valores passados no segundo argumento, de forma paginada.
- buscarQtdLista(): Busca a quantidade de registros no banco.
- buscarQtdLista(T): Busca a quantidade de registros no banco com as restrições dos campos setados no exemplo passado como argumento.

3.2.2 Métodos da Classe GenericHibernateDAO

- lmplementação de todos os métodos existentes na interface **GenericDAO**.

- construtor(): Construtor normal.
- construtor(String): Construtor que utiliza uma SessionFactory passando o alias definido neste construtor.
- getHibernateUtilInterface(): retorna um objeto HibernateUtilInterface.
- getParamsMsg(): retorna um array de String que contem as informações sobre a classe em contexto.
- createCriteria(Session, T): cria e retorna um Objeto Criteria. Como parâmetros o objeto Session e o DAO.
- getPersistentClass(): retorna a classe do contexto.

3.2.3 Métodos da Classe HibernateUtil

- currentSession(): Retorna sessão do Hibernate. Se não houver sessão, cria uma nova.
- closeSession(): Fecha a sessão do Hibernate.
- currentTransaction(): Inicia uma transação se for necessário. Caso já exista uma, somente a mantém. Retorna a sessão que está nesta transação.
- commitTransaction(): Persiste transação. Somente tem efeito se a session tiver sido obtida através do método getManagedSession. Neste caso, executa um commit na transação associada caso o total de chamadas de getManagedSession tenha sido igual ao total de commits e caso não tenha sido chamado nenhum rollback. Caso algum rollback tenha sido chamado, executa roolback e lança Exception.
- rollbackTransaction(): Retorna transação ao estado inicial. Somente tem efeito se a session tiver sido obtida através do método getManagedSession. Neste caso, executa um rollback na transação associada caso o total de chamadas de getManagedSession tenha sido igual ao total de commit/rollbacks.
- getConfiguration(): Retorna o objeto Configuration usado.
- getSessionFactory(): Retorna a SessionFactory atual.

3.2.4 Métodos da Classe HibernateUtilConfiguration

- construtor(String): Cria um SessionFactory e um Configuration a partir de um arquivo de configuração definido como “hibernate.STRING.cfg.xml” , onde STRING é o argumento do método.
- getConfiguration(): Retorna o Configuration instanciado no construtor.
- getSessionFactory(): Retorna o SessionFactory instanciado no construtor.

3.2.5 Métodos da Classe HibernateUtilFactory

- getInstance(): Retorna uma implementação do HibernateUtilInterface, a partir do arquivo de configuração “hibernate.cfg.xml”.
- getInstance(String): Retorna uma implementação do HibernateUtilInterface, a partir do arquivo de configuração “hibernate.STRING.cfg.xml”, onde STRING é o argumento do método.

3.2.6 Métodos da Interface HibernateUtilInterface

- currentSession(): Retorna sessão do Hibernate. Se não houver sessão, cria uma nova.
- closeSession(): Fecha a sessão do Hibernate.
- currentTransaction(): Inicia uma transação se for necessário. Caso já exista uma, somente a mantém. Retorna a sessão que está nesta transação.
- commitTransaction(): Persiste transação. Somente tem efeito se a session tiver sido obtida através do método getManagedSession. Neste caso, executa um commit na transação associada caso o total de chamadas de getManagedSession tenha sido igual ao total de commits e caso não tenha sido chamado nenhum rollback. Caso algum rollback tenha sido chamado, executa roolback e lança Exception.
- rollbackTransaction(): Retorna transação ao estado inicial. Somente tem efeito se a session tiver sido obtida através do método getManagedSession. Neste caso, executa um rollback na transação associada caso o total de chamadas de getManagedSession tenha sido igual ao total de commit/rollbacks.
- getConfiguration(): Retorna o objeto Configuration usado.
- getSessionFactory(): Retorna a SessionFactory atual.

3.3 gov.pr.celepar.framework.exception

3.3.1 Métodos da Classe ApplicationException

A classe ApplicationException possui vários construtores que recebem várias combinações dos seus campos, que são: MessageKey, CausaRaiz e MessageArgs. Além de agir como POJO para contê-los, a classe não realiza nenhum tratamento específico.

3.4 gov.pr.celepar.framework.mensagem

3.4.1 Métodos da Classe Mensagem

- setTipo(Class): Configura a classe da mensagem que se deseja. A classe default é a MensagemPr.
- getInstance(): Retorna classe de mensagens atual. Se não existir, cria uma nova.
- getMessageKeyPadrao(): Retorna a chave da mensagem padrão.
- setMessageKeyPadrao(String): Configura a chave de mensagem padrão.

3.4.2 Métodos da Classe MensagemInterface

- setResource(Object): Configura um novo resource para as mensagens contidas na classe.
- getMessage(String): Recupera uma mensagem do banco de acordo com a chave passada.
- getMessage(String,String[]): Recupera uma mensagem do banco de acordo com a chave passada, substituindo nela os itens passados no array.
- setMessageKeyPadrao(String): Define qual o identificador da mensagem padrão, caso não seja encontrada a mensagem.
- getMessageKeyPadrao(): Retorna o identificador da mensagem padrão.

3.4.3 Métodos das Classes MensagemPr e MensagemBD

Ambas as classes implementam a Classe MensagemInterface, possuindo assim os métodos já mencionados na descrição da interface.

3.5 gov.pr.celepar.framework.report

3.5.1 Métodos da Classe Agendador

- agendador(String): cria a classe Agendador usando a string passada como nome do grupo dos trabalhos.
- agendarData(String, ReportDefinition, Date): Agenda um relatório para uma data específica.
- agendarRepeticao(String, ReportDefinition, Date, Date, long): Agenda um relatório para ser executado com uma certa freqüência.
- removerAgendamento(String): Remove agendamento com o nome recebido como parâmetro.
- recuperarRelatorio(String, boolean): Recupera um byte[] que representa um relatório no formato PDF.

- listarTrabalhos(): Retorna array com os nomes de todos os trabalhos encontrados no servidor.
- agendaGC(): Agenda dados no "GC". Caso não exista GC, cria um novo.

3.5.2 Métodos da Classe FileGC

- setLifetime(Long) configura o total de dias que os arquivos gerados podem permanecer no servidor.
- execute(JobExecutionContext), chamado automaticamente a cada execução do job.

3.5.3 Métodos da Classe JobAgendador

O único método da classe é o método execute(JobExecutionContext), chamado automaticamente a cada execução do job.

3.5.4 Métodos da Classe LoadImageJasperHtml

O único método da classe é o método service(HttpServletRequest,HttpServletRequest), chamado automaticamente a cada request feito pelo browser.

O servlet deve ser utilizado da seguinte maneira:

```
      JRHtmlExporter exporter = new JRHtmlExporter();
      exporter.setParameter(JRHtmlExporterParameter.IMAGES_URI,
"LoadImageJasperHtml?image=");
```

3.5.5 Métodos da Classe ReportDefinition

- ReportDefinition(File, String): construtor que recebe um File referente ao arquivo do tipo ".jasper".
- ReportDefinition(HashMap, File, String): construtor que recebe os dados do filtro por uma HashMap e um File referente ao arquivo do tipo ".jasper".
- getList(): Retorna a lista com as HashMaps que contém os dados do relatório.
- setList(ArrayList): Seta a lista de HashMaps que será utilizada para gerar o relatório.
- isEmpty(): Verifica se a lista de HashMaps do relatório está vazia.
- getBufferPdf(): Cria o buffer de bytes do arquivo PDF gerado pelo JasperReports.
- salvarArquivoPdf(String, String): Salva o relatório em arquivo PDF.
- getBufferHtml(HttpServletRequest): Gera string com a saída em formato HTML para o relatório.

- gerarRelatorioHtml(HttpServletRequest, OutputStream): Gera a saída em formato HTML para o relatório.
- gerarRelatorioXls(OutputStream): Cria o buffer de bytes do arquivo XLS gerado pelo JasperReports, faz a geração diretamente na stream de saída.
- getBuffetCsv(): Gera uma string com a saída em formato CSV para o relatório.
- putJasperParameter(Object,Object): Método utilizado para inserir novos registros na HashMap que irá para o JasperReports.
- getFilterParameter(String): Retorna o valor do registro da HashMap contendo os valores passados do filtro.
- execute(): Método que deve ser executado para gerar a carga dos dados para o relatório, este método deve ser implementado na classe filha e deverá conter toda a parte de código responsável em obter e colocar os valores em HashMaps contendo os dados requeridos pelo relatório.
- getPathBaseDir(): Retorna o diretório base dos relatórios gerados.
- setPathBaseDir(String): Seta o diretório base dos relatórios gerados.

3.5.6 Métodos da Classe ReportDS

- next(): verifica pela existência de uma próxima HashMap de dados de relatórios e caso exista, seta a próxima como a HashMap atual.
- setList(ArrayList): seta lista de dados.
- getObject(): Retorna um objeto do tipo HashMap com os atuais dados do DataSource.
- getFieldValue(JRField): Retorna nome do objeto campo do relatório.

3.6 gov.pr.celepar.framework.taglib

Por terem um uso diferente das classes comuns, as taglibs não terão seus métodos explicados, e sim seus parâmetros de entrada que estão especificados no arquivo tld.

3.6.1 Tag calendario

A tag calendário funciona da seguinte maneira:

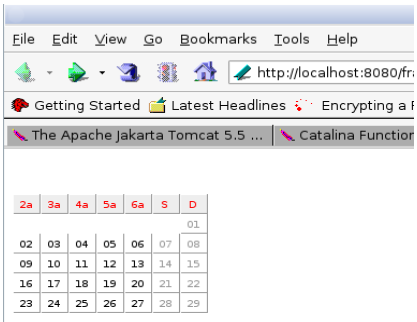
```
<h:calendario ano="2004" mes="02" estilo="calendario.css" link="pp.do"/>
```

Onde:

- ano é o ano escolhido do calendário a ser exibido.

- mês é o mês do ano a ser exibido.
- estilo é o arquivo que contêm o css para o HTML.
- link é a página para qual serão enviados os requests.

O calendário renderizado fica da seguinte forma:



3.6.2 Tag radioList

Exemplo de uso:

```
<h:radioList itens="${cl}" value="codigo" label="valor" nome="teste" selected="<%=new Integer(7)%>" />
```

Onde:

- itens é a collection a ser exibida pela radioList.
- value é o nome do atributo que deve ser recuperado para ser exibido como valor do radio.
- label é o nome do atributo que deve ser recuperado para ser exibido como texto do radio.
- nome é o nome da RadioList (e de cada elemento também).
- selected é um Object que deve ser igual ao value que deve ser selecionado.
- classe é a classe css do radio.

3.6.3 Subpacote Grid

3.6.3.1 Tag table

Exemplo de uso:

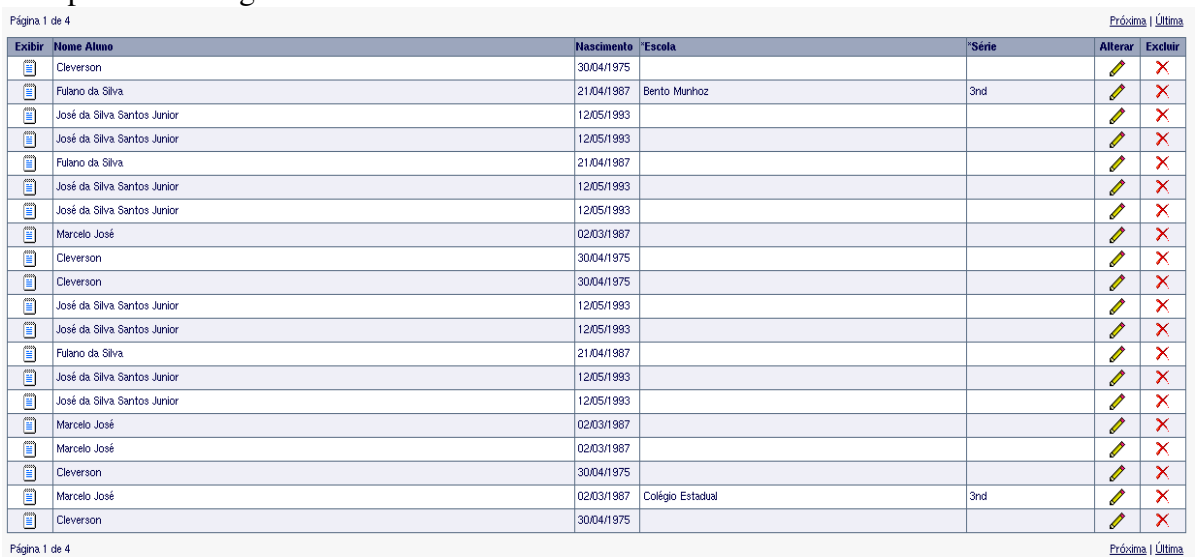
```
<ch:table classTable="list_tabela" classLinha1="list_cor_sim" classLinha2="list_cor_nao">
  <ch:lista bean="${pagina}" atributoId="idAluno,nomeAluno" />
  <ch:agrupador atributo="matricula.escola.nomeEscola" label="Escola" />
  <ch:action imagem="${icon_exibir}" link="${link_exibir_aluno}&idAluno=%1&actionType=exibir"
```

```
label="Exibir" width="40" align="center" />
  <ch:campo atributo="nomeAluno" label="Nome Aluno" />
  <ch:campo atributo="nascimentoAluno" decorator="gov.pr.celepar.framework.taglib.grid.decorator.Data"
label="Nascimento" width="60" />
  <ch:campo atributo="matricula.escola.nomeEscola" label="*Escola" />
  <ch:campo atributo="matricula.serie.nomeSerie" label="*Série" />
  <ch:action imagem="${icon_alterar}" link="${link_editar_aluno}&idAluno=%1" label="Alterar" width="40"
align="center" />
  <ch:action imagem="${icon_excluir}" link="${link_excluir_aluno}&idAluno=%1&actionType=excluir"
label="Excluir" width="40" align="center" />
  <ch:painel pagina="${link_listar_alunos}" classe="painel" atributoIndice="indice" />
</ch:table>
```

Onde:

- classTable é o estilo CSS utilizado no corpo da tabela.
- classTitulo é o estilo CSS utilizado no título da tabela.
- classLinha1 é o estilo CSS utilizado nas linhas pares.
- classLinha2 é o estilo CSS utilizado nas linhas não pares.
- bean é um objeto Pagina contendo informações para a paginação.
- atributoId é uma lista de atributos chave separada por vírgulas, referenciados em links por %1, %2, ... %n, onde n é o número de atributos passados.
- imagem é a URL para a imagem da ação.
- link é a URL do link.
- atributo é o nome do atributo a ser recuperado do objeto.
- label é a descrição a ser colocada no título da tabela.
- pagina é o nome da página para a qual vão ser enviados os dados nos links do painel.
- classe é o estilo CSS utilizado para o painel.
- atributoIndice é o nome do atributo de índice que virá do post quando trocada a página.
- decorator é o atributo que indica uma classe que implementa a interface gov.pr.celepar.framework.taglib.grid.decorator.Decorator para formatar campos. Já foram implementadas algumas classes para formatar determinados tipos de campo: Data, DataExtenso, DataHora, HoraCurta, HoraLonga e Moeda (disponíveis no pacote gov.pr.celepar.framework.taglib.grid.decorator).
- agrupador é a tag responsável por agrupar linhas através do atributo definido. No exemplo acima a listagem de alunos será agrupada por Escola. Observação: a listagem de já deve estar agrupada pelo atributo desejado. A tag agrupador irá apenas formatar a visualização dos dados de forma agrupada.

Exemplo de tabela gerada:



Página 1 de 4 — Próxima | Última

Exibir	Nome Aluno	Nascimento	*Escola	*Série	Alterar	Excluir
	Cleverson	30/04/1975				
	Fulano da Silva	21/04/1987	Bento Munhoz	3nd		
	José da Silva Santos Junior	12/05/1993				
	José da Silva Santos Junior	12/05/1993				
	Fulano da Silva	21/04/1987				
	José da Silva Santos Junior	12/05/1993				
	José da Silva Santos Junior	12/05/1993				
	Marcelo José	02/03/1987				
	Cleverson	30/04/1975				
	Cleverson	30/04/1975				
	José da Silva Santos Junior	12/05/1993				
	José da Silva Santos Junior	12/05/1993				
	Fulano da Silva	21/04/1987				
	José da Silva Santos Junior	12/05/1993				
	José da Silva Santos Junior	12/05/1993				
	Marcelo José	02/03/1987				
	Marcelo José	02/03/1987				
	Cleverson	30/04/1975				
	Marcelo José	02/03/1987	Colégio Estadual	3nd		
	Cleverson	30/04/1975				

Página 1 de 4 — Próxima | Última

3.6.4 Subpacote Lista

3.6.4.1 Tag lista

Exemplo de uso:

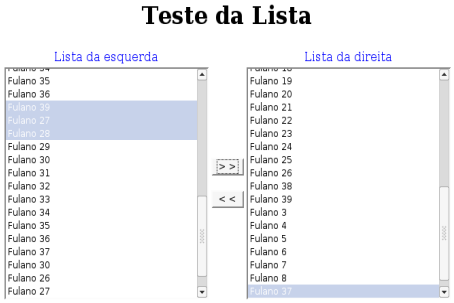
```
<lista:select classe="tabela" ordenada=”true” moverTodas=”true”>
  <lista:esquerda combo="lista1" onError="alert('Opa esquerda');" classe="lista" label="Lista da esquerda"
enable=”true”>
    <lista:opcoes colecao="lista" bean="container" nome="nome" valor="codigo" />
  </lista:esquerda>
  <lista:direita combo="lista1" onError="alert('Opa direita');" classe="lista" label="Lista da direita"
enable=”true”>
    <lista:opcoes colecao="lista" nome="nome" valor="codigo" />
  </lista:direita>
</lista:select>
```

Onde:

- classe é a Classe CSS a ser aplicada na tabela que contém os itens.
- ordenada mantém as listas ordenadas (opcional).
- moverTodas cria botão para mover todos os itens da lista (opcional).
- combo é o nome do atributo para submit que será associado a combo.
- onError é o JavaScript que define qual o erro quando tenta-se realizar um movimento de elementos entre as caixas, sendo que o elemento a ser movimentado ainda não foi selecionado.

- label é o título do conjunto de resultados exibidos.
- enable indica se o botão central para transferência dos itens deve estar habilitado ou não. Caso não seja adicionada, o default é true. Valores possíveis: true ou false.
- colecao é o nome da coleção a ser recuperada do bean, caso este seja informado ou do request caso não seja informado bean algum.
- bean é o nome do bean a ser recuperado do request para que dele seja obtida a coleção.
- nome é o nome da propriedade onde está armazenada a legenda das opções.
- valor é o nome da propriedade onde está armazenado o valor a ser passado quando dado o submit.
- colecao é nome da coleção a ser recuperada do bean, caso este seja informado ou do request caso não seja informado bean algum.

O código acima gera a seguinte tela:



3.6.5 Subpacote Upload

3.6.5.1 Tag ProgressUploadTagLib

Para se usar essa taglib, é necessário incluir o JavaScript do prototype.js na página.

A taglib de upload deve ser utilizada da seguinte forma:

```
<h:progressoUpload idForm="idForm"></h:progressoUpload>
```

Onde o atributo idForm deve ser o id do form que fará o upload do arquivo.

3.7 gov.pr.celepar.framework.upload

3.7.1 Método da Interface LimitUploadHandler

maxUploadSize(HttpServletRequest request): Este método deve retornar em bytes, o tamanho máximo do arquivo que pode ser feito upload.

3.7.2 Métodos da Classe ProgressMonitorFileItem

Como esta é uma classe interna do framework e não deve ser utilizada diretamente pela a aplicação, os seus métodos não serão explicados.

3.7.3 Métodos da Classe ProgressMonitorFileItemFactory

Como esta é uma classe interna do framework e não deve ser utilizada diretamente pela a aplicação, os seus métodos não serão explicados.

3.7.4 Métodos da Interface ProgressObserver

Como esta é uma Inteface interna do framework e não deve ser utilizada diretamente pela a aplicação, os seus métodos não serão explicados.

3.7.5 Métodos da Classe ProgressRequestHandler

Como esta é uma classe interna do framework e não deve ser utilizada diretamente pela a aplicação, os seus métodos não serão explicados.

3.7.6 Métodos da Classe UnlimitedUploadHandler

maxUploadSize(HttpServletRequest request): Este método retorna o tamanho máximo da variável do tipo long.

3.7.7 SubPacote filter

3.7.7.1 Métodos da Classe ProgressUploadFilter

Como esta é uma classe interna do framework e não deve ser utilizada diretamente pela a aplicação, os seus métodos não serão explicados.

3.8 gov.pr.celepar.framework.util

3.8.1 Métodos da classe Abreviatura

- setTipoLogradouroAbreviacoes(String): Seta quais os tipos de logradouros que serão abreviados e suas respectivas abreviações. Exemplo de sintaxe esperado pelo método: "Rua=R, Avenida=Av, Praça=P" Obs: Qualquer informação passada que fuja deste padrão de sintaxe fará com que a abreviação não funcione corretamente.
- SetTitulacaoLogradouroAbreviacoes(String): Seta quais as titulações de logradouros que serão abreviados e suas respectivas abreviações. Exemplo de sintaxe esperado pelo método: "Padre=Pe,Visconde=Visc". Obs: Qualquer informação passada que fuja deste padrão de sintaxe fará com que a abreviação não funcione corretamente.
- setPreposicoes(String): Seta as preposições que poderão ser excluídas na abreviação. Exemplo de sintaxe esperado pelo método: "a,e,das,de,nos".
- setRetiraPreposicoes(boolean): Possibilita ou não a exclusão das preposições na abreviação.
- abreviarNome(String, int): Abrevia nome até a quantidade de caracteres informada, seguindo as regras abaixo:
    - O primeiro e o último nome nunca são abreviados;
    - Caso o nome original não ultrapasse a quantidade máxima de caracteres o nome original é retornado sem nenhuma alteração;
    - Se mesmo depois de feita a abreviação o nome continuar a exceder quantidade máxima de caracteres, são feitas remoções dos sobrenomes internos, sem alterar o último nome;
    - Quando mesmo abreviado, o nome continuar a exceder a quantidade máxima de caracteres é retornado uma exceção.
- abreviarLogradouro(String, int): Abrevia logradouro até a quantidade de caracteres informada, seguindo as regras abaixo:
    - Caso o logradouro original não ultrapasse a quantidade máxima de caracteres o nome original é retornado sem nenhuma alteração;
    - O Tipo identificador de logradouro sempre é abreviado;
    - O primeiro e o último nome do logradouro nunca são abreviados, exceção feita de acordo com os parâmetros nomeLogradouroAbreviacoes e

titulacaoLogradouroAbreviacoes;

- Se mesmo depois de feita a abreviação o logradouro continuar a exceder a quantidade máxima de caracteres, são feitas remoções dos nomes internos ao logradouro, sem alterar o último nome.
- Quando mesmo abreviado, o logradouro continuar a exceder a quantidade máxima de caracteres é retornado uma exceção.

- tentarAteAbreviarLogradouro(String, int): Tenta abreviar o logradouro a partir da quantidade de caracteres informadas, até conseguir se utilizando do método abreviarLogradouro(). Invoca o método abreviarLogradouro() repassando os parâmetros recebidos, caso receba a Exception lançada pelo método abreviarLogradouro() incrementa o tamanho em um e vai repetindo esse procedimento até conseguir receber uma abreviatura para poder retornar.
- setTitulacaoLogradouroAbreviacoes(String): define o tipo de abreviação para os títulos de logradouro.
- setTipoSobrenomeAbreviacoes(String): define o tipo de abreviação para os tipos de sobrenomes.
- quebraPalavras(String): retorna um array String que contem a palavra quebrada, que foi passada por parâmetro.
- montaNome(String[]): retorna uma String contendo o nome composto pelo array passado por parâmetro.
- abreviarString(String): Abrevia a String passada como parâmetro verificando ela não é uma preposição. Se for preposição retorna uma String vazia, senão retorna a primeira letra da string informada seguida de um ponto.
  Ex: str="Marcos" - return="M".
- abreviarTitulacaoLogradouro(String): Verifica se a String passada como parâmetro é uma titulação de logradouro. Se for a titulação do logradouro é abreviado da forma informada no atributo titulacaoLogradouroAbreviacoes.
- abreviarSobrenome(String): Verifica se a String passada como parâmetro é um sobrenome abreviável. Se for, o sobrenome é abreviado de acordo com o atributo tipoSobrenomeAbreviacoes.
- verificarPreposicao(String): Verifica se a String passada como parâmetro é uma preposição. Caso for retorna um boolean com o valor true, caso contrário false.
- verificarParenteses(String): Verifica se a String passada como parâmetro contém parenteses. Caso encontrar retorna um boolean com o valor true, caso contrário false.

- verificarTitulacao(String): Verifica se a String passada como parâmetro contém titulação. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- verificarTipoLogradouro(String): Verifica se a String passada como parâmetro contém logradouro. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- isTitulacaoAbreviada(String): Verifica se a String passada como parâmetro contém titulação abreviada. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- isTipoLogradouroAbreviado(String): Verifica se a String passada como parâmetro contém logradouro abreviado. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- abreviarTipoLogradouro(String): Verifica se a String passada como parâmetro é um tipo identificador de logradouro. Se for o logradouro é abreviado da forma informada no atributo tipoLogradouroAbreviacoes.
- tentarAteAbreviarNome(String, int): Tenta abreviar o logradouro a partir da quantidade de caracteres informada até conseguir se utilizando do método abreviarNome(). Invoca o método abreviarNome() repassando os parâmetros recebidos, caso receba a Exception lançada pelo método abreviarNome() incrementa o tamanho em um e vai repetindo esse procedimento até conseguir receber uma abreviatura para poder retornar.
- setTitulacaoLogradouroAbreviacoes(String): define o tipo de abreviação para os títulos de logradouro.
- setTipoSobrenomeAbreviacoes(String): define o tipo de abreviação para os tipos de sobrenomes.
- quebraPalavras(String): retorna um array string que contem a palavra quebrada, que foi passada por parâmetro.
- montaNome(String[]): retorna uma string contendo o nome composto pelo array passado por parâmetro.
- abreviarString(String): Abrevia a String passada como parametro verificando ela não é uma preposição. Se for preposição retorna uma string vazia, senão retorna a primeira letra da string informada seguida de um ponto.
  Ex: str="Marcos" - return="M" .
- abreviarTitulacaoLogradouro(String): Verifica se a string passada como parâmetro é uma titulação de logradouro. Se for a titulação do logradouro é abreviado da forma informada no atributo titulacaoLogradouroAbreviacoes.
- abreviarSobrenome(String): Verifica se a string passada como parâmetro é um sobrenome abreviável. Se for, o sobrenome é abreviado de acordo com o atributo

tipoSobrenomeAbreviacoes.

- verificarPreposicao(String): Verifica se a String passada como parâmetro é uma preposição. Caso for retorna um boolean com o valor true, caso contrário false.
- verificarParenteses(String): Verifica se a String passada como parâmetro contém parenteses. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- verificarTitulacao(String): Verifica se a String passada como parâmetro contém titulação. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- verificarTipoLogradouro(String): Verifica se a String passada como parâmetro contém logradouro. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- isTitulacaoAbreviada(String): Verifica se a String passada como parâmetro contém titulação abreviada. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- isTipoLogradouroAbreviado(String): Verifica se a String passada como parâmetro contém logradouro abreviado. Caso encontrar retorna um boolean com o valor true, caso contrário false.
- abreviarTipoLogradouro(String): Verifica se a String passada como parâmetro é um tipo identificador de logradouro. Se for o logradouro é abreviado da forma informada no atributo tipoLogradouroAbreviacoes.

3.8.2 Métodos da Classe BarCode2of5

- BarCode2of5(): construtor, seta os dados mínimos para a construção do código de barras.
- getMinimumSize(): Retorna o tamanho mínimo do componente.
- minimumSize(): Retorna a dimensão mínima do componente.
- setSize(int, int): Seta o tamanho inicial da largura e altura de impressão do componente.
- setSize(Dimension): Seta o tamanho de impressão do componente com atributos de dimensão.
- createBarCode(String): constrói uma imagem contendo o código de barras representando o código informado no parâmetro.

3.8.3 Métodos da classe CNPJ

- validarCNPJ(String): Método que valida se determinado valor passado como parâmetro é um CNPJ válido.

- formataCNPJ(String): Recebe um CNPJ e o retorna com a máscara correta.

3.8.4 Métodos da classe CPF

- validarCPF(String): Método que valida se determinado valor passado como parâmetro é um CPF válido.

- formataCPF(String): Recebe somente os números de um CPF e o retorna com a máscara correta.

3.8.5 Métodos da Classe Data

- addAnos(int, Date): Soma ou subtrai uma quantidade de anos de uma determinada data.
- addDias(int, Date): Soma ou subtrai uma quantidade de dias de uma determinada data.
- addMeses(int, Date): Soma ou subtrai uma quantidade de meses de uma determinada data.
- calculaIdade(Date): Dada uma data de nascimento, calcula a idade.
- dataAtual(): Retorna String com o dia atual.
- formataData(Date): Converte uma String para um objeto Date.
- formataData(String): Converte um Date para uma String.
- formataDataHora(Date): Converte um Date para uma String no formato dd/mm/aaaa HH:mm:ss.
- formataHoraLonga(Date): Converte um Date para uma String no formato HH:mm:ss.
- formataHoraCurta(Date): Converte um Date para uma String no formato HH:mm.
- formataDataVoltaSQL(String): Formata data vinda no formato do SQLServer para o formato dd/MM/yyyy.
- formatExtenso(Date): Recebe um objeto Date e retorna a data por extenso (ex: 01 de Janeiro de 2000).
- getCalendar(Date): Retorna um objeto Calendar com a data fornecida como parâmetro.
- isFuturo(Date): Verifica se a data passada como parâmetro é futuro.
- isFuturo(String): Verifica se a data passada como parâmetro é futuro.
- isPassado(Date): Verifica se a data passada como parâmetro é passado.
- isPassado(String): Verifica se a data passada como parâmetro é passado.
- retornaDataBD(String): Recebe uma data formato dd/MM/yyyy e a retorna no formato

yyyy-mm-dd.

- retornaDataInversa(String): Tranforma uma String do formato dd/MM/yyyy para formato yyyymmdd.
- dataDiff(Date, Date): Compara duas datas, retornando a diferença em dias entre elas (data2 – data1).
- mesDiff(Date, Date): Compara duas datas, retornando a diferença em meses entre elas (datamaior – datamenor). Considera somente os valores de ano e mês para o cálculo (exemplo: diferença entre 31/12/2005 e 01/01/2006 é um mês). Caso a datamaior seja menor que a datamenor, retorna a diferença em valores negativos.
- anoDiff(Date, Date): Compara duas datas, retornando a diferença em anos entre elas (datamaior - datamenor). Considera somente o valor do ano para o cálculo (exemplo: diferença entre 31/12/2005 e 01/01/2006 é um ano). Caso a datamaior seja menor que a datamenor, retorna a diferença em valores negativos.
- validarIdadeReferencia(Date, int, int, Date): Valida se determinada idade é menor, maior ou igual à idade limite. A data de referência pode ser considerada a data de hoje ou uma data específica. Retorna um boolean com valor true se uma das condições(menor, maior ou igual) for atendida, senão retorna false.
- validarIdade(Date,int,int): Valida se determinada idade é menor, maior ou igual à idade limite. A data de referência pode ser considerada a data de hoje ou uma data específica. Retorna um boolean com valor true se as condições da validação forem atendidas, senão retorna false.
- validarIdadeMenor(Date,int): Valida a data de nascimento. Verifica se a idade informada na data de nascimento é menor que a idade limite. Retorna um boolean com valor true se a condição for atendida, senão retorna false.
- validarIdadeMaior(Date,int): Valida a data de nascimento. Verifica se a idade informada na data de nascimento é maior que a idade limite. Retorna um boolean com valor true se a condição for atendida, senão retorna false.
- retornarTextoData(int,int,int): Trazer o texto do dia ou mês indicado de acordo com o número. Retorna uma String.

  1 = Domingo .. 7 = Sábado

  1 = Janeiro .. 12 = Dezembro
- retornarDiaSemana(int): Trazer dia da semana de acordo com o número.

  1 = Domingo .. 7 = Sábado. Retorna uma String.
- retornarTextoDiaSemana(int,int): Trazer dia da semana de acordo com o número. O primeiro parâmetro se refere ao dia(1 = Domingo .. 7 = Sábado), e o segundo parâmetro se

refere a abreviatura(0=abreviado - 'Dom', 1=Extenso – 'Domingo'). Retorna uma String.

- retornarMes(int): Trazer texto Mês de acordo com o número(1 = Janeiro .. 12 = Dezembro). Retorna uma String.
- retornarTextoMes(int, int):Trazer mês de acordo com o número. O primeiro parâmetro se refere ao dia(1 = Janeiro .. 12 = Dezembro), e o segundo parâmetro se refere a abreviatura(0=abreviado - 'Jan', 1=Extenso – 'Dez'). Retorna uma String.

3.8.6 Métodos da classe Modulo11

CalcularDVModulo11(String, int): Calcula dígito verificador de módulo 11 aplicando o peso informado.

3.8.7 Métodos da Classe Pagina

Os métodos dessa classe são apenas construtores e getters e setters para suas propriedades.

3.8.8 Métodos da Classe Reflexao

- listaMetodosGet(Object): Lista os métodos get de um objeto passado como parâmetro.
- listaMetodosSet(Object): Lista os métodos set de um objeto passado como parâmetro.
- listaMetodos(Object): Lista os métodos de um objeto passado como parâmetro.
- listaMetodos(Object, String): Lista os métodos de um objeto passado como parâmetro que estejam dentro da expressão regular recebida como parâmetro.
- listaAtributos(Object): Lista todos os atributos declarados de um objeto passado como parâmetro.
- listaAtributos(Object, String): Lista os atributos de um objeto passado como parâmetro que estejam dentro da expressão regular recebida como parâmetro.
- invocaGet(Object, String): Invoca um método getXxxxx() para o objeto sem passar parâmetros.
- invocaGet(Object, String, Object[]): Invoca um método getXxxxx() para o objeto e o atributo informado.
- invocaSet(Object, String, Object[]): Invoca o método setXxxx para o objeto e atributo informados.
- invocaGetAninhado(Object, String): Invoca recursivamente o get para recuperar o valor de uma propriedade aninhado dentro de outra propriedade da classe. Suporta infinitos níveis de profundidade.
- classForName(String): Recupera a classe utilizando Reflection.

3.8.9 Métodos da Classe StringUtil

- formataCEP(String): Formata CEP e o retorna com máscara correta.
- formataCNPJ(String): Depreciado – utilizar o método da classe CNPJ.
- formataCPF(String): Depreciado – utilizar o método da classe CPF.
- formataRG(String): Formata RG e o retorna com máscara adequada.
- formataTamanho(String, int): Completa a String com caracteres em branco no início até atingir o tamanho especificado.
- padraoMainFrame(String): Retorna String sem acentuação e em caixa alta.
- primeiraLetraToUpperCase(String): Retorna a String informada como parâmetro modificando sua primeira letra pra letra maiúscula.
- tiraAcento(String): Remove acentuação de uma String passada como parâmetro.
- primeiraLetraToLowerCase(String): Retorna a String informada como parâmetro modificando sua primeira letra para letra minúscula.
- removeFormatacao(String): Método utilizado para remover a formatação de Strings contendo números. Retira pontos, barras, traços e espaços em branco. Retorna um String sem a formatação.
- isSameCharacter(String): Método utilizado para verificar se uma determinada String é composta sempre do mesmo caracter. Retorna um boolean contendo valor true se verdadeiro, senão retorna false.
- stringNotNull(String): Verifica se uma determinada String é não nula. Verifica se a String é diferente de null e diferente de "". Retorna um boolean contendo valor true se verdadeiro, senão retorna false.
- addCharStrPos(String, char, int, int, boolean): Adiciona caracter especificado na posição indicada (inicio/fim) da String. Primeiro parâmetro a String que será modificada, o segundo o char que deseja-se adicionar a String, o terceiro a quantidade de caracteres a ser inseridos na posição, o quarto parâmetro a posição(0 = inicio, 1 = fim) e o quinto boolean para Trim. Retorna a String modificada.
- addCharInicioStr(String, char, int): Adiciona caracter especificado n vezes no inicio da String. Os parâmetros são: a String desejada, o caracter para ser adicionado e o número de vezes. Retorna a String modificada.
- addCharFinalStr(String, char, int): Adiciona caracter especificado n vezes no final da String. Os parâmetros são: a String desejada, o caracter para ser adicionado e o número de vezes. Retorna a String modificada.
- addCharInicioStrTrim(String, char, int): Adiciona caracter especificado n vezes no inicio

da String fazendo trim no inicio. Os parâmetros são: a String desejada, o caracter para ser adicionado e o número de vezes. Retorna a String modificada.

- addCharFinalStrTrim(String, char, int): Adiciona caracter especificado n vezes no final da String fazendo trim ao final. Os parâmetros são: a String desejada, o caracter para ser adicionado e o número de vezes. Retorna a String modificada.
- delCharStrPos(String, chat, int, int, boolean): Remove o caracter especificado na posição indicada (inicio/fim) da String. Primeiro parâmetro a String que será modificada, o segundo o char que deseja-se remover da String, o terceiro a quantidade de caracteres a ser removidos na posição, o quarto parâmetro a posição(0 = inicio, 1 = fim) e o quinto boolean para Trim. Retorna a String modificada.
- delCharInicioStr(String, char): Remove caracter especificado no inicio da String. Retorna a String modificada.
- delNumCharInicioStr(String, char, int): Remove caracter especificado no inicio da String. O terceiro parâmetro representa a quantidade de caracteres a ser removidos. Retorna a String modificada.
- delCharFinalStr(String, char): Remove caracter especificado no final da String. Retorna a String modificada.
- delNumCharFinalStr(String, char, int): Remove caracter especificado n vezes do final da String. O terceiro parâmetro representa a quantidade de caracteres a ser removidos. Retorna a String modificada.
- delCharInicioStrTrim(String,char,int): Remove caracter especificado n vezes do inicio da String fazendo trim após a remoção. O terceiro parâmetro representa a quantidade de caracteres a ser removidos. Retorna a String modificada.
- delCharFinalStrTrim(String,char,int): Remove caracter especificado n vezes no final da String fazendo trim após remoção. O terceiro parâmetro representa a quantidade de caracteres a ser removidos. Retorna a String modificada.
- longToIp(Long): Transformar um endereço IP de Long para String.
- ipToLong(String): Transformar um endereço IP de String para Long.
- 3.8.10 Métodos da Classe Valores
- formataValor(float): Retorna uma String com máscara para um float.
- formataValor(String): Retorna o valor com máscara para uma String sem máscara.
- isInteger(String): Verifica se uma String é um Inteiro.
- formataTamanho(Long, int): Formata um Long para o formato alfanumérico do Natural, iniciando o Long com o valor '0' até completar tamanho.

- formataMoeda(double): Retorna um número com formatação Monetária.

3.9 pr.gov.celepar.framework.validator

3.9.1 Métodos da Classe StrutsValidator

Todos os métodos públicos aqui implementados seguem a assinatura padrão de validadores plugáveis do Struts Validator.

- validateIdentico(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest): utilizado para verificar se os valores de dois campos de um formulário são idênticos, por exemplo em uma verificação de senha onde o usuário deve informar duas vezes um mesmo valor.
- validateCPF(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest): utilizado para verificar se o valor de determinado campo de um formulário é um CPF válido, o qual pode estar formatado ou não (99999999999 ou 999.999.999-99).
- validateCNPJ(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest): utilizado para verificar se o valor de determinado campo de um formulário é um CNPJ válido, o qual pode estar formatado ou não (99999999999999 ou 99.999.999/9999-99).
- validateCPF_CNPJ(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest): utilizado para verificar se o valor de determinado campo de um formulário é um CPF ou um CNPJ válido, o qual se utiliza dos 2 métodos anteriormente descritos se comportando da mesma maneira, aceitando valores formatados ou não.
- validateModulo11(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest): utilizado para validar o dígito verificador de modulo 11 aplicando o peso informado. A cadeia numérica deve conter o dígito verificador embutido. Deve ser informado o peso a ser aplicado no cálculo do modulo (o peso deve ser sempre maior que 2).

3.10 gov.pr.celepar.framework.xml

Por se tratar de um grupo de classes que deve ser usado em conjunto, a melhor maneira para visualizar essa classe é através de um exemplo onde ocorre a interação entre as classes. O código Java descrito:

Element root = new Element ("minhaTag", "Valor da Tag");

Attribute at = new Attribute("nome", "tagTeste", root);

root.addAt(at);

Gera o seguinte código XML:

```
<minhaTag nome="tagTeste">
      Valor da tag
</minhaTag>
```