# SwingBean
## Quick Reference
### (Arquivos XML)

## Formato XML

```
<beanDescriptor>
  <line>
    <property name='xxx' ... />+
  </line>
  <line>
    <property name='xxx' ... />+
  </line>
</beanDescriptor>
```

## Tipos de Propriedades

Devem ser colocadas no atributo `type` de cada `property`. Alguns casos já são configurados como default.

**TEXT – Campo texto**
**LARGE_TEXT – Campo de área de texto**
**PASSWORD – Campo de senha**
**DATE – Campo para data**
**BOOLEAN – Checkbox**
**INTEGER – Campo para números inteiros**
**LONG – Campo para números inteiros grandes**
**FLOAT – Campo para números de ponto flutuante**
**DOUBLE - Campo para números de ponto flutuante**
**COMBO – Campo com um combo**
**DEPENDENT_COMBO – Combo dependente de outro**
**MULTIPLE_LIST – Duas listas com drag-and-drop**
**CHECKBOX_LIST – Lista de Checkbox**
**IMAGE – Campo para colocar imagem**
**COLOR – Campo para escolher uma cor**
**TREE – Uma árvore (válido para tabelas)**

## Atributos

### Gerais
`name`**: Nome da propriedade**
`label`**: Label da propriedade (def. nome da classe)**
`type`**: Tipo de campo no formulário**
`colspan`**: (F) Colunas que o campo deve ocupar**
`columnSize`**: (T) Tamanho da coluna da tabela**
`readOnly`**: (T) Coluna não editável (true/false)**
`mandatory`**: Campo obrigatório (true/false)**

### Texto (TEXT, LARGE_TEXT, PASSWORD)
`size`**: Tamanho máximo (n° de caracteres)**
`minSize`**: Tamanho mínimo (n° de caracteres)**
`pattern`**: Padrão de texto (Expressão Regular)**
`mask`**: Máscara (padrão MaskFormatter)**
`formatExample`**: Exemplo de formatação**

### Combos (COMBO)
`threadLoading`**: Carregamento em thread (o valor default é true)**
`comboList`**: Lista de valores separados por ";"**
`comboModelClass`**: Classe que implementa** `ComboBoxModel` **ou classe que possui o método para recuperação do combo.**
`comboModelMethod`**: Método que retorna um** `ComboBoxMode`**l. Se esta opção for configurada a classe deve possuir um método estático** `getInstance()` **que retorna uma instância da classe.**
`parameter`**: Parâmetro a ser passado para o método que recupera o combo (se houver).**

### Combo Dependente (DEPENDENT_COMBO)
`comboModelClass`**: Classe que implementa** `DependentComboModel`**.**
`dependentProperty`**: Valor do nome da propriedade do combo do qual este depende.**

### Imagem (IMAGE)
`showResolution`**: Resolução de vizualização**
`saveResolution`**: Resolução de retorno**

### Numérico (INTEGER, LONG, FLOAT, DOUBLE)
`max`**: Valor máximo**
`min`**: Valor mínimo**

### Imagem (IMAGE)
`showResolution`**: Resolução de vizualização**
`saveResolution`**: Resolução de retorno**

### Listas (MULTIPLE_LIST, CHECKBOX_LIST)
`threadLoading`**: Carregamento em thread (o valor default é true)**
`list`**: Lista de valores separados por ";"**
`listModelClass`**: Classe que implementa** `ListModel` **ou classe que possui o método para recuperação da lista.**
`listModelMethod`**: Método que retorna um** `ListMode`**l. Se esta opção for configurada a classe deve possuir um método estático** `getInstance()` **que retorna uma instância da classe.**
`parameter`**: Parâmetro a ser passado para o método que recupera o combo (se houver).**
`minSelected`**: Mínimo de itens selecionados.**
`maxSelected`**: Máximo de itens selecionados.**

### Arvore (TREE)
`rootName`**: Nome do nó raiz**

A hierarquia de uma árvore pode ser de três tipos:

**- Relação pai-filho por identificadores**
`idProperty`**: Propriedade Identificadora**
`parentProperty`**: Propriedade c/ id do pai**

**- Relação pai-filho por composição**
`childrenProperty`**: Propriedade c/ lista de filhos**

**- Classificação por valor de propriedades**
`classifyBy`**: Propriedades separadas por ";"**

## Carregamento do XML

O carregamento do XML costuma gerar um problema de performance. Uma das formas de acelerar isto é criando uma thread de baixa prioridade para fazer isto. Para isto basta chamar:

```
BackgroundLoadthread.loadDescriptors();
```

Este método busca o arquivo `descriptor.prop`, mas o nome pode ser alterado utilizando `setLoadFileName()`. O arquivo deve ter o seguinte formato:

```
<descriptors>
   <descriptor name='ExemploForm'
      class='org.teste.Exemplo'
      file='exemploForm.xml'/>
   <descriptor name='ExemploTable'
      class='org.teste.exemplo'
      file='exemploForm.xml'
      type='table'/>
</descriptors>
```

O nome passado como parâmetro na criação dos componentes e no arquivo serve para identificar o arquivo XML e deve ser o mesmo. Depois de carregado uma vez o descritor será reaproveitado.

# SwingBean
## Quick Reference
### (Componentes e API)

## JBeanPanel

### Criação (para classe Exemplo)

```
GenericFieldDescriptor descriptor =
  XMLDescriptorFactory.getFieldDescriptor
   (Exemplo.class,
    new File("formExemplo.xml"),
    "IdentificadorForm");
JBeanPanel<Exemplo> panelCidade =
  new JBeanPanel<Exemplo>
    (Exemplo.class, descriptor);
```

### Métodos Básicos

`setBean()` **preenche campos com dados do bean.**
`populateBean()` **preenche propriedades do bean com os dados do formulário.**
`getComponent()` **retorna o componente de interface relativo a uma propriedade.**
`cleanForm()` **limpa os campos do formulário.**
`getValidationResult()` **retorna resultado da validação.**
`decorateFied()` **insere componente de interface ao lado do campo referente a uma propriedade.**
`associateAction()` **associa uma ação a alteração de um campo relativo a uma propriedade.**
`reloadList()` **recarrega lista.**
`setEnable()` **habilita e desabilita campo.**
`getPropertyValue()` **retorna o valor de um campo pelo nome da propriedade.**
`setPropertyValue()` **coloca o valor no campo relativo a uma propriedade a partir do seu nome.**

## JBeanTable e BeanTableModel

### Criação (para classe Exemplo)

```
TableFieldDescriptor tableDescriptor =
  XMLDescriptorFactory.
    getTableFieldDescriptor
      (Exemplo.class,
       new File("tableExemplo.xml"),
       "IdentificadorTable");
BeanTableModel<Exemplo> model =
  new BeanTableModel<Exemplo>
    (tableDescriptor);
JBeanTable table = new JBeanTable(model);
```

### Métodos Básicos (JBeanTable)

`associateActionToColumn()` **Associa uma ApplicationAction ao componente de edição de uma coluna.**
`addDoubleClickAction()` **adiciona ApplicationAction a ser executado no duplo clique em uma linha da tabela.**
`enableHeaderOrdering()` **habilita ordenação pelo clique no cabeçalho da tabela.**
`enableQuickEditing()` **habilita mudança de campos teclando "enter".**

### Métodos Básicos (BeanTableModel)

`getValueAt()` **recupera valor de uma propriedade em uma linha.**
`setBeanList()` **insere uma lista inteira para ser exibida e limpa todos os contadores.**
`addBean()` **adiciona um bean na lista.**
`getBeanAt()` **retorna o bean exibido em uma linha da tabela.**
`getInserted()` **retorna beans inseridos.**
`getDeleted()` **retorna beans excluídos.**
`getUpdated()` **retorna beans atualizados.**
`resetCounters()` **zera os contadores de beans inseridos, atualizados e excluídos.**
`deleteBeanAt()` **exclui o bean em uma determinada linha.**
`getValidationResult()` **retorna resultado da validação para uma linha.**
`getCompleteList()` **retorna lista completa.**
`getFilteredList()` **retorna lista exibida na tela (ordenada e filtrada).**
`orderByProperty()` **ordena lista por uma propriedade.**
`filterStartedBy()` **filtra beans com uma propriedade iniciada com determinada String.**
`filterContains()()` **filtra beans com uma propriedade contém uma determinada String.**
`filterBetween()` **filtra beans com uma propriedade entre dois determinados valores. Se um for null, somente o outro será considerado.**
`filterEquals()` **filtra beans com uma propriedade igual a um objeto.**
`getIndexStartedBy()` **retorna o numero da linha com um bean onde uma propriedade se inicie com determinada String.**
`getIndexContains()`**retorna o numero da linha com um bean onde uma propriedade contenha uma determinada String.**

## ApplicationActions

Estes componentes compõe o framework de execução de ações do SwingBean. Abaixo estão as principais classes.

**- Classes Base**
`ApplicationAction` **Classe base para a construção de ações do framework SwingBean.**
`ThreadedAction` **Ação executada em thread.**
`ActionChainFactory` **Fabrica que cria uma cadeia de ações a serem executadas em sequência.**
`ColumnAction` **Classe base para a criação de ações que atuem dentro de uma linha em uma JBeanTable.**

**- Componentes de Interface**
`JActButton` **Botão que executa uma ApplicationAction ao ser clicado.**
`JActMenuItem` **Item de menu que executa uma ApplicationAction ao ser clicado.**

**- Ações Utilitárias**
`ValidationAction` **Executa a validação de um formulário ou tabela e exibe os problemas. Caso existam problemas a próxima ação não é executada.**
`MessageAction` **Exibe mensagem em popup.**
`CheckBoxEnableAction` **Habilita e desabilita campos com base no valor de um checkbox.**
`ReloadListAction` **Atualiza os valores de uma lista (ComboBox ou List) buscando-os novamente no método especificado no XML.**

## Aparência

Para alterar a aparência de todos os componentes deve ser chamado `LookProvider.setLook()`. Para implementar as customizações, basta herdar a classe `DefaultLookDescriptor` ou implementar a interface `LookDescriptor`.