



PLATAFORMA DE DESENVOLVIMENTO PINHÃO PARANÁ

NEOCEP

Julho – 2006

Sumário de Informações do Documento

Tipo do Documento: Definição		
Título do Documento: Guia de uso do sistema NeoCEP		
Estado do Documento: EB (Elaboração)		
Responsáveis: João Eduardo Mikos, Fabio Gustavo Sgoda, Renan Marcel Cardoso Baggio, Filipe Lautert		
Palavras-Chaves: java, struts, jboss, cep, tomcat, javascript		
Resumo: Guia de uso para o sistema NeoCEP		
Número de páginas: 19		
Software utilizados: BrOffice 2.0		
Versão	Data	Mudanças
1.0	10/06/05	
1.0.1		Arrumada descrição do posicionamento da tag main, documentados novos campos, salientada unicidade da tag main, adicionadas funções utilitárias JavaScript
1.0.2	23/06/05	Adicionada documentação dos filtros e salientado o fato da utilização dos pacotes oficiais
1.0.4	04/07/05	Adicionada a funcionalidade de parametrização da codificação de localidade (município) a ser adotada durante as pesquisas e notificação sobre as novas propriedades opcionais mencionadas.
1.0.5	25/08/05	Adicionada a busca de endereço na perda do foco (onblur) do campo CEP, correção na função de cópia de formulários (copiaFormulario).
1.0.6	05/07/05	Correções na função resetEndereco para os casos onde os campos não obrigatórios do formulário não estiverem presentes e correção também da função onLoadCepHandler para correta manutenção dos eventos previamente definidos para os campos em casos onde hajam mais de um formulário CEP por página.
1.0.7	10/11/05	Atualização da configuração do cep-client devido a mudança do servidor de desenvolvimento sgtfdesenv01.celepar.parana para swebdesenv01.celepar.parana.
1.2.0	18/11/05	Adicionada imagem. Várias correções de javascript (municipios com cep único, recuperação do cache, removido onBlur) e alteração do SQL de busca.
1.2.1	17/01/06	Adicionada busca a cep genérico de município.
1.2.2	19/01/06	Resolvido bug de NullPointerException na busca de ceps inexistentes.
1.2.3	20/01/06	Utilização da biblioteca prototype para a comunicação assíncrona via AJAX. Adicionado controle opcional para o campo bairro (habilitado ou não, de acordo com a presença de tal informação como resultado de uma busca) e obrigatório para o campo endereço (não faz a busca por logradouros quando selecionada uma localidade com cep único). Correção em pré e post-handlers.
1.3.0	01/04/06	Mudança na base de dados, para suportar diversas codificações de localidade e busca a ceps de Agência de correio e Caixa Postal Comunitária. Adicionados parâmetros extras de retorno nas buscas.
1.3.1	01/06/06	Revisão geral do documento.
1.3.2	03/07/06	Alterado informações dos campos hidden. Incluído exemplos de retorno.
1.3.4	26/06/08	Alterado endereços do cep

SUMÁRIO

1 NEOCEP.....	4
1.1 INTRODUÇÃO.....	4
2 CONTEXTO SERVIDOR.....	4
2.1 API.....	4
2.1.1 Busca por localidades (/busca/localidades).....	4
2.1.2 Busca por CEP (/busca/cep).....	5
2.1.3 Busca por endereço parcial (/busca/endereco)	6
2.1.4 Mensagens do sistema (rpcError(id, mensagem)).....	7
2.1.5 Objeto Endereco.....	7
2.1.6 Objeto Localidade.....	8
2.1.7 Campos hidden.....	9
2.2 INSTALAÇÃO.....	9
3 CONTEXTO CLIENTE.....	9
3.1 INSTALAÇÃO.....	10
3.2 CONTEÚDO.....	10
3.3 CONFIGURAÇÃO.....	11
4 TAGLIB.....	11
4.1 INSTALAÇÃO.....	11
4.2 UTILIZAÇÃO.....	11
4.2.1 Inicialização.....	11
4.2.2 Formulários de busca.....	12
4.2.3 Campos obrigatórios.....	12
4.2.3.1 CEP.....	13
4.2.3.3 UF.....	13
4.2.3.4 Municipio.....	14
4.2.3.7 Box.....	14
4.2.4 Campos opcionais.....	15
4.2.4.1 Complemento.....	15
4.2.4.2 Filtro.....	15
4.2.5 Finalização de formulários de busca.....	16
4.2.6 Finalização.....	16
4.3 CONFIGURAÇÃO.....	16
5 EXEMPLOS.....	17
5.1 FORMULÁRIO DE INSERÇÃO.....	17
5.2 FORMULÁRIO DE ATUALIZAÇÃO.....	17
6 JAVASCRIPT.....	18
6.1 COPIAFORMULARIO(ID1, ID2).....	18
6.2 PRE E POSTHANDLERS.....	18

1 NEOCEP

1.1 Introdução

O novo sistema de busca de CEP foi criado com o principal objetivo de abandonar a tecnologia proprietária. Esta solução permite utilizar os novos recursos apresentados pela plataforma adotada, o JBoss, possuidor de um mecanismo avançado de cache que pode ser utilizado com o Hibernate para melhorar a velocidade das consultas.

Além disso, foram criadas facilidades para o desenvolvedor, como um contexto cliente do sistema e uma taglib para o formulário.

2 CONTEXTO SERVIDOR

O contexto servidor destina-se aos servidores de produção e desenvolvimento e é responsável pela consulta direta ao banco. O servidor possui uma API acessível via HTTP que serve para a realização das mesmas.

2.1 API

O contexto servidor sempre estará disponível no servidor de aplicações no prefixo **“/cep”**. Para realizar as queries, é acessado o **“/busca”** dentro do servidor. As buscas dividem-se em 3 tipos (Cep, Endereço e Localidade). Todas as funções são realizadas como queries HTTP, podendo ser feitas pelo browser, e retornando funções JavaScript que devem ser implementadas para que seja atingida a funcionalidade desejada, funções estas sempre tendo o primeiro argumento sendo o id informado na consulta. Tomaremos como referência para os exemplos um servidor rodando em <http://www.neocep.pr.gov.br/cep/>. Os parâmetros de cada busca **não** precisam estar em ordem.

2.1.1 Busca por localidades (/busca/localidades)

A busca por localidades retorna a lista de localidades (municípios) de determinada UF ou uma mensagem informando que nenhum elemento foi

encontrado.

Parâmetros:

- **UF**: sigla da UF.
- **id**: id da tag que realiza a busca.
- **codificacao**: identificador da codificação de municípios a ser utilizada.

Exemplo:

- <http://www.neocep.pr.gov.br/cep/busca/localidades?id=4&UF=PR&codificacao=SERPRO>

Retorno:

rpcLocalidadesDone(4, <array de objetos Localidade>);

2.1.2 Busca por CEP (/busca/cep)

A busca por cep retorna um array de objetos Endereco contendo um único elemento: o logradouro referenciado por aquele CEP ou uma mensagem informando que nenhum elemento foi encontrado. A forma da busca depende do nível de refinamento desejado:

0 – busca na tabela de logradouros e na tabela de localidades para verificar se não é o caso de uma cidade com apenas 1 CEP.

1 – busca na tabela de logradouros, na tabela de localidades e na tabela de grandes usuários, para encontrar registros que possam ser de grandes usuários de CEPs, como prédios, indústrias, etc.

Parâmetros:

- **CEP**: CEP do logradouro desejado.
- **id**: id da tag que realiza a busca.
- **codificacao**: identificador da codificação de municípios a ser utilizada.
- **refinamento**: nível de refinamento para a pesquisa (0/1). Pode ser emitido sendo então assumido como “0” , porém no contexto cliente o padrão é 1.

Exemplo:

- <http://www.neocep.pr.gov.br/cep/busca/cep?CEP=81230418&id=4&codificacao=CORREIOS&refinamento=1>

Retorno:

- `rpcCEPDone(4, <array de objetos Endereco>);`

2.1.3 Busca por endereço parcial (/busca/endereco)

A busca por endereço parcial retorna um array de objetos Endereco contendo as possibilidades retornadas pela consulta ou uma mensagem informando que nenhum elemento foi encontrado.

Parâmetros:

- **id**: id da tag que realiza a busca.
- **endereco**: Nome completo ou parcial do logradouro desejado. Nessa versão inicial, o nome não deve possuir os prefixos “Rua”, “Avenida” e outros.
- **localidade**: código do município.
- **localidadeCorreio**: código do Correio corrigido para a localidade.
- **uf**: sigla da UF.
- **codificacao**: identificador da codificação de municípios a ser utilizada.

Exemplo:

- <http://www.neocep.pr.gov.br/cep/busca/endereco?endereco=am%C3%A1lia&localidade=6015&uf=PR&id=4&codificacao=Serpro>

Retorno:

- `rpcLogradourosDone(4, <array de objetos Endereco>);`

2.1.4 Mensagens do sistema (rpcError(id, mensagem))

O sistema retorna mensagens de erro por meio da função `rpcError`, contendo

o id passado para a consulta falha e a mensagem sobre qual erro ocorreu.

Exemplo:

- <http://www.neocep.pr.gov.br/cep/busca/endereco?endereco=naoexiste&localidade=6015&uf=PR&id=4>

Retorno:

- rpcError(4, 'Não encontrado nenhum logradouro')

2.1.5 Objeto Endereco

O objeto Endereco possui as seguintes propriedades, acessíveis da maneira **objeto.<propriedade>**.

Lista de propriedades:

- **cep**: número do cep.
- **nome**: nome do logradouro.
- **tipo**: tipo do logradouro.
- **complemento**: complemento necessário.
- **bairro**: nome do bairro.
- **localidade**: chave da localidade.
- **uf**: sigla da uf.
- **num_ini**: número inicial.
- **num_fim**: número final.
- **tipoCep**: nível de codificação de CEP(s) ('D' – Distrito | 'R' - Região | 'M' – Município | 'P' – Povoado | 'A' – Agência de Correio | 'C' – Caixa Postal Comunitária | 'G' – Grandes Usuários). Como esta propriedade também existe no objeto localidade (necessário por questões de performance do contexto cliente) e pode conter um valor destoante, recomenda-se que o seu valor seja recuperado a partir de um campo hidden associado (vide item **2.1.7**), que recupera o valor a partir do objeto correto após cada pesquisa.

- **isCepGenerico** informa se é um CEP genérico ('true' | 'false' | null), ou seja, um CEP que referencia toda uma localidade, mas que não se trata de um CEP único. Reiterando: um cep único NÃO É um cep genérico. Recomenda-se a utilização de um campo hidden (vide item **2.1.7**) associado a esta propriedade para recuperar seu valor, que transforma os retornos null em 'false'.
- **situacao** situação da localidade ('0' = cep único para localidade | '1' = município com cep codificado para logradouro | '2' = distrito ou povoado com cep codificado para logradouro).

2.1.6 Objeto Localidade

O objeto Localidade possui as seguintes propriedades, acessíveis da maneira **objeto.<propriedade>**.

Lista de propriedades:

- **chave**: código da localidade na codificação definida.
- **nome**: nome da localidade.
- **uf**: sigla da uf.
- **situacao**: situação da localidade ('0' = cep único para localidade | '1' = município com cep codificado para logradouro | '2' = distrito ou povoado com cep codificado para logradouro).
- **ChaveCorreios**: código da localidade na codificação base (correios).
- **cep**: cep único da localidade, se houver ('0', caso contrário).
- **tipoCep**: nível de codificação de CEP(s) desta localidade ('D' – Distrito | 'R' - Região | 'M' – Município | 'P' – Povoado). Observe que os tipos 'A', 'C' e 'G' (definidos na propriedade de mesmo nome e semântica do objeto Endereco) não são definidos em nível de localidade. Recomenda-se que o valor desta propriedade seja recuperado a partir de um campo hidden (vide item **2.1.7**) associado, o qual tem seu valor definido a partir do objeto correto.

2.1.7 Campos hidden

O componente provê à aplicação informações relacionadas ao CEP recuperado na busca através de campos hidden com armazenamento de valor na forma de vetor, sendo o índice relacionado à tag de busca associada. Seguem os campos definidos:

- **tipoCep**: nível de codificação do CEP ('D' – Distrito | 'R' - Região | 'M' – Município | 'P' – Povoado | 'A' – Agência de Correio | 'C' – Caixa Postal Comunitária | 'G' – Grandes Usuários).
- **situacaoCep**: situação da localidade ('0' = localidade com cep atribuído | '1' = localidade sem cep atribuído (provavelmente o município tem cep codificado para logradouro) | '2' = distrito ou povoado sem cep atribuído (provavelmente o distrito ou povoado tem cep codificado para logradouro)).
- **isCepGenerico**: informa se é um CEP genérico, ou seja, um CEP que referencia toda uma localidade, mas que não se trata de um CEP único ('true' | 'false').

Exemplos:

- Quando uma localidade tem cep atribuído: situacaoCep = 0, isCepGenerico = false;
- Quando logradouro tem cep atribuído: situacaoCep = 1, isCepGenerico = false;
- Quando variacao_localidade tem cep atribuído: situacaoCep = 1, isCepGenerico = true;

2.2 Instalação

Para a instalação do contexto servidor, é necessário que o war referente ao mesmo seja nomeado como “**cep.war**”, e que, junto com o datasource “**neocep-{desenv|prod}-ds.xml**” devidamente configurado, seja copiado para o diretório de deploy do servidor JBoss 4.

É importante salientar que o contexto servidor **funciona apenas no servidor JBoss, não tendo suporte oficial para outro servidor de aplicação.**

3 CONTEXTO CLIENTE

A instalação local do contexto servidor não é recomendada para o desenvolvedor, considerando que por um uso extensivo de caches há uma ocupação de recursos valiosos da máquina, fato este que pode tornar onerosa para o desenvolvedor a utilização da estação para desenvolvimento.

Em vista disso, para uso das equipes de desenvolvimento, foi criado um contexto cliente com o intuito de realizar a ponte das requisições para um servidor central de desenvolvimento, este sim rodando um contexto servidor. Esse contexto cliente é uma simples aplicação Struts que rodará independente do sistema em desenvolvimento, seja ele qual for, eliminando assim um ponto de falha e facilitando a vida do programador.

Por ser uma aplicação simples, pode ser rodada em qualquer servidor que suporte o framework Struts, consideração esta tomada em vista de que por falta de recursos na máquina, muitos desenvolvedores realizam suas tarefas utilizando o Tomcat ao invés do Jboss.

3.1 Instalação

A instalação do contexto cliente é simples. Basta que o seu “.war” seja baixado e que o deploy seja feito no servidor de aplicação desejado, podendo este ser qualquer servidor que suporte o framework Struts. Oficialmente, o suporte é oferecido apenas para os servidores **JBoss** e **Tomcat**.

→ É **extremamente** recomendável que o desenvolvedor utilize-se dos pacotes Debian oferecido pela equipe (tanto clientes quanto servidores), disponíveis no repositório oficial corporativo, garantindo assim sempre a presença de uma versão atualizada com os últimos bugfixes de todos os itens instalados.

3.2 Conteúdo

O contexto cliente fornece:

- o caminho “/cep” necessário para o funcionamento da taglib desenvolvida.
- folha de estilo CSS padrão, que pode ser copiada para o projeto em desenvolvimento.
- imagem padrão de botões de consulta.
- pacote “.jar” da taglib de formulário de consulta.
- exemplo de uso da taglib de consulta.

3.3 Configuração

A única configuração necessária no contexto cliente é a alteração no arquivo “web.xml” do mesmo das propriedades referentes ao endereço do servidor de CEP e porta do mesmo, caso seja necessário modificar a configuração padrão.

O conteúdo a ser modificado é o seguinte:

```
<!-- parâmetros de configuração do cliente de CEP -->
<context-param>
    <param-name>servidor</param-name>
    <param-value>nic.celepar.parana</param-value>
</context-param>
<context-param>
    <param-name>porta</param-name>
    <param-value>8080</param-value>
</context-param>
```

4 TAGLIB

Por fim, a parte final do sistema NeoCEP é composta pela taglib, componente utilizado para inserir o formulário de busca de CEP na página.

4.1 Instalação

Para efetuar a instalação da taglib, apenas é necessário que o seu arquivo “.jar” seja copiado para o diretório de libs (bibliotecas) do projeto.

4.2 Utilização

A taglib de busca de CEP é seccionada em várias tags, tendo o procedimento de uso especificado a seguir.

4.2.1 Inicialização

Para utilizar a taglib, é necessário que primeiramente seja inserido no topo da página a chamada que a importa para a página:

```
<%@taglib prefix="cep" uri="http://celepar.pr.gov.br/taglibs/cep-1.0" %>
```

Feito isso, é preciso inserir a taglib <cep:main>, que inicia os parâmetros necessários para a renderização do conteúdo. Isto é necessário para que possam ser utilizados múltiplos formulários por página:

```
<cep:main css="/css/cep.css">
```

A tag “**main**” possui um atributo opcional “**css**”, que aponta para a localização do arquivo de CSS com a configuração local do projeto para o estilo dos componentes.

→ A tag “**main**” é única para todos os formulários inseridos na página. Deve ser iniciada antes deles e finalizada depois de todos.

→ Utilizando com o Struts, a tag “**main**” deve ser colocada depois do <html:form> e finalizada antes do </html:form>.

4.2.2 Formulários de busca

A taglib trabalha com o conceito de regiões de formulário, de maneira similar ao HTML, por isso o início de um novo formulário de busca de CEP é feito com a seguinte diretiva:

```
<cep:form findOnType="true" codificacao="SERPRO"
textoBusca="Procurando...">
```

A propriedade “**findOnType**” ativa a busca de endereços enquanto o usuário digita o nome parcial, exibindo uma tela com os resultados para seleção dentro da própria página, sem a necessidade de um popup externo.

A propriedade “**codificacao**” informa qual o tipo de codificação de localidades (municípios) a ser adotado durante a pesquisa. Os possíveis valores

atualmente são CORREIOS e SERPRO. Como a propriedade não é obrigatória caso nada seja informado ou seja informado algum valor diferente dos aceitos a codificação CORREIOS é adotada (codificação padrão).

A propriedade “**textoBusca**” define o texto que será exibido durante a realização da busca. Não é obrigatória.

4.2.3 Campos obrigatórios

Existem alguns campos obrigatórios que **devem estar presentes** para o correto funcionamento da interface visual.

4.2.3.1 CEP

```
<cep:cep name="cep" value="81230418" size="70"/>
```

A tag “**cep**” mostra um campo com o número do CEP e um botão para busca. A busca também pode ser realizada apertando-se <enter>.

Suporta as seguintes propriedades opcionais: **value**, **size**, **disabled**, **onclick**, **ondblclick**, **onmouseover**, **onmouseout**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **style**, **styleClass**, **maxlength**, **readonly**.

4.2.3.2 Endereco

```
<cep:endereco name="endereco" value="Rua Amália Ludwig Dubard" size="70"/>
```

A tag “**endereco**” mostra um campo com o nome da rua, avenida, etc e um botão para busca. A busca também pode ser realizada apertando-se <enter>.

Suporta as seguintes propriedades opcionais: **value**, **size**, **disabled**, **maxlength**, **onclick**, **ondblclick**, **onmouseover**, **onmouseout**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **readonly**, **style**, **styleClass**.

Caso a propriedade **findOnType** desse formulário (ver 4.2.2) seja definida como “**true**”, a busca enquanto o usuário digita é realizada.

4.2.3.3 UF

```
<cep:uf name="uf" value="PR" />
```

A tag “**uf**” mostra uma combo com as UFs existentes. Cada seleção de nova UF provoca uma busca pelos municípios referentes àquela UF.

Suporta as seguintes propriedades opcionais: **value**, **disabled**, **onclick**, **ondblclick**, **onmouseover**, **onmouseout**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **readonly**, **style**, **styleClass**.

4.2.3.4 Municipio

```
<cep:municipio name="municipio" value="6015"/>
```

A tag “**municipio**” mostra uma lista dos municípios referentes à UF selecionada.

Suporta as seguintes propriedades opcionais: **value**, **disabled**, **onclick**, **ondblclick**, **onmouseover**, **onmouseout**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **style**, **styleClass**, **readonly**.

4.2.3.6 Bairro

```
<cep:bairro name="bairro" value="Cidade Industrial" size="70" />
```

A tag “**bairro**” mostra o nome do bairro recuperado pelo sistema.

Suporta as seguintes propriedades opcionais: **value**, **size**, **disabled**, **onclick**, **ondblclick**, **onmouseover**, **disableWhenAutomaticallyFilled**, **disabled**, **onmouseout**, **maxlength**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **readonly**, **style**, **styleClass**.

Caso a propriedade **disableWhenAutomaticallyFilled** desse formulário seja definida como “**true**”, o campo bairro é desabilitado para alterações quando uma busca retornar um valor para bairro.

4.2.3.7 Box

```
<cep:box />
```

A tag “**box**” é o campo de informação do sistema ao usuário. Serve para mostrar as mensagens retornadas pelo sistema.

Suporta as seguintes propriedades opcionais: **style**, **styleClass**.

Todos os campos possuem uma propriedade **name** obrigatória que define o nome do campo na página HTML, para que o desenvolvedor esteja livre para realizar qualquer validação JavaScript que julgue necessária.

4.2.4 Campos opcionais

Existem campos opcionais cuja presença ou ausência não são necessariamente relevantes para o funcionamento do sistema. São eles:

```
<cep:numero name="numero" value="347" size="70"/>  
<cep:telefone name="telefone" value="(041) 3333-3333" size="70"/>  
<cep:fax name="fax" value="(041) 3333-3333" size="70"/>
```

A funcionalidade fornecida pelos mesmos é auto-explicativa eliminando assim a necessidade de uma descrição mais detalhada. Os valores contidos nesses campos não são tocados pelas consultas.

4.2.4.1 Complemento

```
<cep:complemento name="complemento" value="Frente Tenda Mãe Maria/Caboclo  
Tupinambá" size="70"/>
```

A tag “**complemento**” mostra um campo com alguma informação adicional sobre o endereço desejado.

Suporta as seguintes propriedades opcionais: **value**, **size**, **disabled**, **onclick**, **maxlength**, **ondblclick**, **onmouseover**, **onmouseout**, **onmousemove**, **onmousedown**, **onmouseup**, **onkeydown**, **onkeyup**, **onkeypress**, **onselect**, **onchange**, **onblur**, **onfocus**, **readonly**, **style**, **styleClass**.

4.2.4.2 Filtro

```
<cep:filtro municipios="${beanmunicipios}" uf="${beanufs}"
erro="alert('endereço fora do escopo procurado');" />
```

A tag “**filtro**” limita as opções de UF disponíveis para procura e também os municípios listados. Caso seja buscado algum endereço cujo CEP esteja fora dos filtros, uma função de erro informada pelo usuário pode ser executada. Todos os parâmetros são opcionais.

- **municipios**: objeto java.util.List com os códigos dos municípios que podem ser mostrados, em Integer.
- **uf**: objeto java.util.List de UFs que podem ser selecionadas, em String.
- **erro**: JavaScript a ser executado caso o endereço não esteja contido no escopo.

4.2.5 Finalização de formulários de busca

Ao fim de um formulário de busca, é necessário fechá-lo, tal como uma tag “**form**” no HTML. Isso é feito da seguinte maneira:

```
</cep:form>
```

4.2.6 Finalização

Ao fim de uma página contendo um ou mais formulário de busca de CEP, faz-se necessária a inserção da tag de finalização:

```
</cep:main>
```

4.3 Configuração

A única configuração existente na taglib é aquela referente aos estilos da caixa de mensagens e ao estado do elemento na lista de seleção de resultados de busca, selecionado ou não selecionado. Tudo isto é feito alterando propriedades contidas no arquivo “**cep.css**”, que para customizações deve ser copiado para o sistema em desenvolvimento.

- Classe da caixa de mensagens:

```
.box {
  border: 1px solid black;
  padding: 2px;
}
```

- Estilo do elemento selecionado:

```
.hover {  
    background-color: lightblue;  
}
```

- Estilo do elemento não selecionado:

```
.normal {  
    background-color: white;  
}
```

- Estilo da lista de resultados:

```
.lista {  
    border: black 1px solid;  
    background-color: white;  
    width: 500px;  
}
```

5 EXEMPLOS

Finalizando, são fornecidos dois exemplos de formulários.

5.1 Formulário de inserção

```
<cep:main css="/css/cep.css">  
  
    <cep:form findOnType="true">  
        <br>  
        CEP: <cep:cep name="cep" />  
        <br>  
        Endereço: <cep:endereco name="endereco" />  
        <br>  
        UF: <cep:uf name="uf" value="PR" />  
        <br>  
        Bairro: <cep:bairro name="bairro" />  
        <br>  
        Município: <cep:municipio name="municipio" />  
        <br>  
        Número: <cep:numero name="numero" />  
        <br>  
        Complemento: <cep:complemento name="complemento" />  
        <br>  
        Telefone: <cep:telefone name="telefone" />  
        <br>  
        Fax: <cep:fax name="fax" />  
        <br>  
        <cep:box />  
    </cep:form>  
  
</cep:main>
```

5.2 Formulário de atualização

```

<cep:main css="/css/cep.css">

    <cep:form findOnType="false">
        <br>
        CEP: <cep:cep name="cep" value="81230418" size="70"/>
        <br>
        Endereço: <cep:endereco name="endereco" value="Rua Amália
Ludwig Dubard" />
        <br>
        UF: <cep:uf name="uf" value="PR" />
        <br>
        Bairro: <cep:bairro name="bairro" value="Cidade Industrial"/>
        <br>
        Município: <cep:municipio name="municipio" value="6015"/>
        <br>
        Número: <cep:numero name="numero" value="347"/>
        <br>
        Complemento: <cep:complemento name="complemento" value="Frente
Tenda Mãe Maria/Caboclo Tupinambá" />
        <br>
        Telefone: <cep:telefone name="telefone" value="(041) 3333-
3333"/>
        <br>
        Fax: <cep:fax name="fax" value="(041) 3333-3333" />
        <br>
        <cep:box />
    </cep:form>

</cep:main>

```

6 JAVASCRIPT

O sistema possui funções JavaScript que podem ser utilizadas pelo usuário que deseje obter alguma funcionalidade a mais.

6.1 copiaFormulario(id1, id2)

A função “**copiaFormulario(id1, id2)**” copia todos os dados do formulário de CEP com id correspondente a “**id1**” para o formulário de CEP com id correspondente a “**id2**”, sejam dados de campos de texto ou de combos.

6.2 pre e postHandlers

Caso deseje algum tratamento antes ou depois das requisições efetuadas, pode-se adicionar handlers antes e depois das consultas. Cada função `rpc*(id, ...)` utiliza um `_request[id]` para adicionar uma função a ser executada antes ou depois

dessas consultas. O objeto request possui duas funções, “**addPreHandler(funcão)**” e “**addPostHandler(funcão)**”, que executa essas funções fornecidas em forma de string. Por exemplo, para colocar um alert antes e depois da seleção de municípios da tag “1”, faz-se:

```
_request[1].addPreHandler("alert('AlertaAntes');");  
_request[1].addPostHandler("alert('AlertaDepois');");
```

Embora o uso dessas funções não seja explicitamente recomendado, elas estão disponíveis e documentadas para que novas funcionalidades possam ser criadas e submetidas como patches para entrarem no componente oficial.