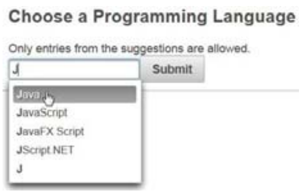
PrimeFaces: String Input Elements (Part I)

Originals of slides and source code for examples: http://www.coreservlets.com/JSF-Tutorial/primefaces/
Also see the JSF 2 tutorial – http://www.coreservlets.com/JSF-Tutorial/jsf2/
and customized JSF2 and PrimeFaces training courses – http://courses.coreservlets.com/jsf-training.html

Topics in This Section

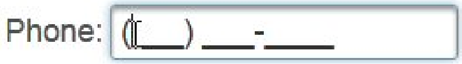
- **String input with suggestions**
  - p:autoComplete basics
  - With selections only from suggestions
  - With arbitrary selections
  - Controlling number of chars before suggestions
  - Controlling timeout before suggestions
  - Multiple selections
  - Ajax listeners



Choose a Programming Language

Only entries from the suggestions are allowed.

- **Constrained String input**
  - p:inputMask basics
  - Mask options: 9, a, *, other

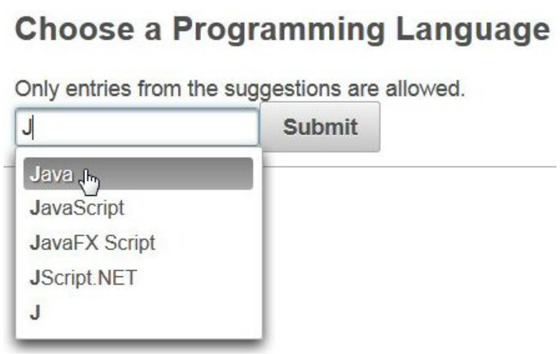


Phone: (___) ___-____

String Input with Suggestions: p:autocomplete (Basics)

p:autocomplete: Overview

- **Appearance and behavior**
  - Textfield with dropdown of suggested completions
  - Developer can choose whether user is forced to accept one of the suggestions or can also enter arbitrary text
- **Purpose: for collecting strings from user**
  - In most basic usage, value is not converted: bean property must be String.



Choose a Programming Language

Only entries from the suggestions are allowed.

p:autocomplete: Summary of Most Important Attributes

- **<p:autoComplete …/>**
  - value
    - Should point to bean property of type String.
  - completeMethod
    - A bean property referring to a server-side method that takes a String as input (the text entered so far) and returns a List<String> (the suggestions to show).
  - forceSelection (true or false [default])
    - Is user constrained to choose a selection (true), or is free text allowed (false).
  - minQueryLength (*integer* [default 1])
    - Number of chars before suggestions start.
  - queryDelay (*integer* [default 300])
    - Number of milliseconds before contacting server. Default 300.
  - multiple (true or false [default])
    - Can user select more than one choice and send it as List?

More Details on forceSelection

- **forceSelection="false" [default]**
  - User can choose a selection from the menu or type in something else. Either is accepted.
- **forceSelection="true"**
  - User can choose a selection or type something in that *exactly* matches a selection (including case). Anything else is rejected: field is cleared and it fails “required” validation rule
- **Warning when using multiple="true"**
  - forceSelection is automatically true, but in older PrimeFaces releases, the widget breaks if you actually say forceSelection="true"
    - See upcoming slides on multiple selections

Example: Constrained Input

- **Input page collects**
  - A computer programming language.
    - User can only choose a suggestion, and cannot enter languages not in the list (forceSelection="true")
    - Completions start after first character (default of 1 for minQueryLength)
    - Small delay after typing before server contacted (default of 300 for queryDelay)
- **Results page shows**
  - Confirmation of selection

Bean (Suggestion Data and Bean Property)

```
@ManagedBean
public class LanguageBean {
  // 100 most popular programming languages, according to
  // http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
  // The first half are in order of popularity, the second half
  // are in alphabetical order.
  private static final String languageString =
    "Java,C,C++,PHP,C#,Python,..."; // More in real code
  private static final String[] languageArray =
    languageString.split(",");

  private String language;

  public String getLanguage() {
    return(language);
  }

  public void setLanguage(String language) {
    this.language = language;
  }
```

Bean (Completion Method)

```
  // Autocompleter method
  public List<String> completeLanguage(String languagePrefix) {
    List<String> matches = new ArrayList<>();
    for(String possibleLanguage: languageArray) {
      if(possibleLanguage.toUpperCase()
                         .startsWith(languagePrefix.toUpperCase())) {
        matches.add(possibleLanguage);
      }
    }
    return(matches);
  }



  // Action controller method

  public String register() {
    return("show-language");
  }
}
```

Unlike many client-side autocompleters, there are no builtin rules as to how matches are done (front of text vs. middle of text, case sensitive, etc.). The method returns a List designating the suggestions, and the way the List is created is totally up to the developer. Here, I choose to do a case-insensitive match against the front of the user data.

Facelets Pages

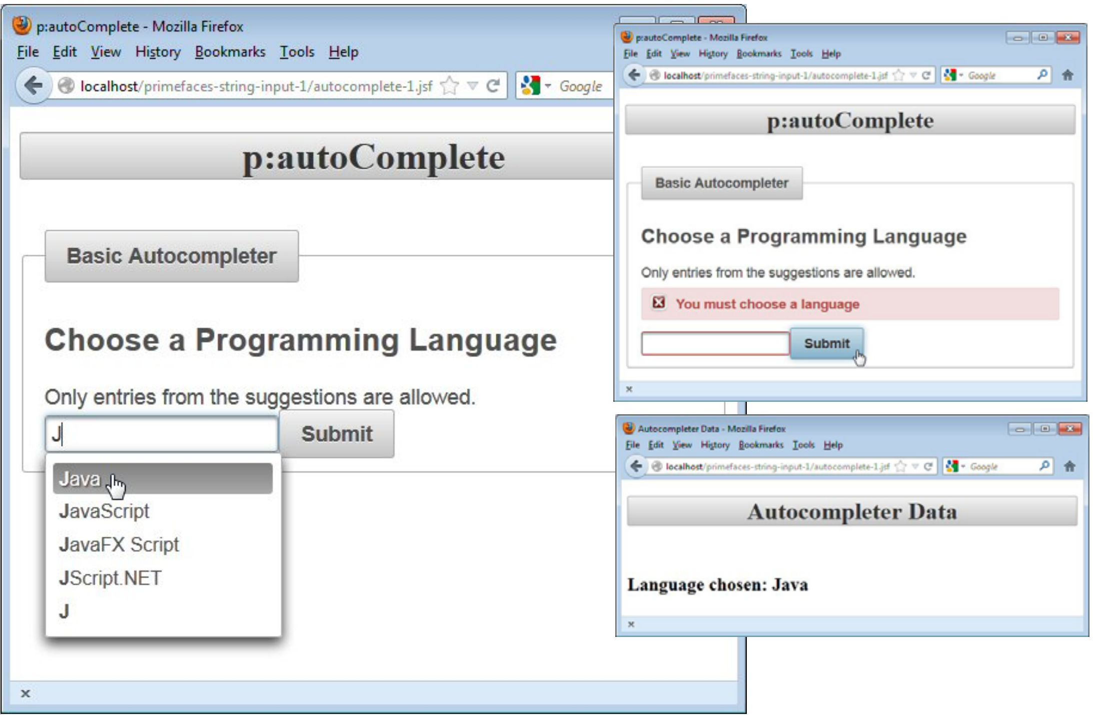
- **Input page**

```
<h:form>
  <h2>Choose a Programming Language</h2>
  Only entries from the suggestions are allowed.
  <p:messages/>
  <p:autoComplete value="#{languageBean.language}"
                  completeMethod="#{languageBean.completeLanguage}"
                  forceSelection="true"
                  required="true"
                  requiredMessage="You must choose a language"/>
  <p:commandButton action="#{languageBean.register}"
                   value="Submit" ajax="false"/>
</h:form>
```

- **Results page**

```
<h1>Language chosen: #{languageBean.language}</h1>
```

Results


p:autoComplete - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/autocomplete-1.jsf
Google
Basic Autocompleter
J
Submit
Java
JavaScript
JavaFX Script
JScript.NET
J
Autocompleter Data - Mozilla Firefox

p:autoComplete

Choose a Programming Language

Only entries from the suggestions are allowed.

You must choose a language

Autocompleter Data

Language chosen: Java

p:autocomplete: Unconstrained Input

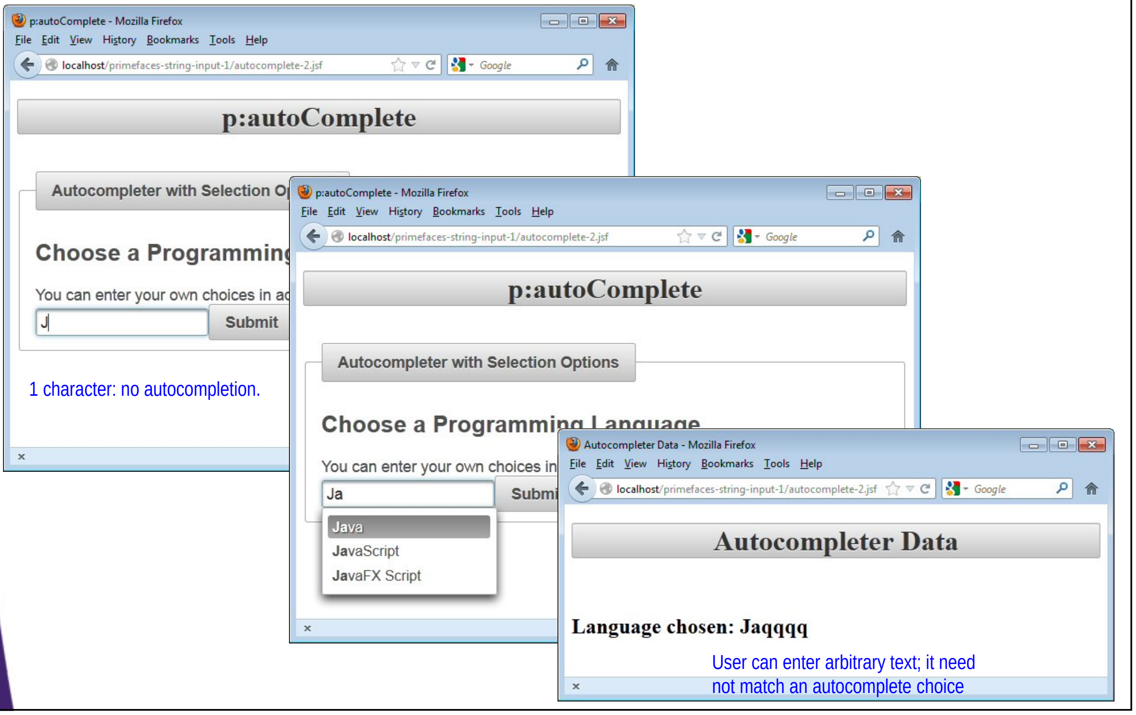
Example: Unconstrained Input

- **Input page collects**
  - A computer programming language.
    - User can either choose a suggestion or enter a language not in the list (default of false for forceSelection)
    - Completions start after second character (minQueryLength="2")
    - One second delay after typing before server contacted (queryDelay="1000")
- **Results page and bean**
  - Same as previous example, so code not repeated here

Facelets (Input Page)

```
<h:form>
  <h2>Choose a Programming Language</h2>
  You can enter your own choices in addition to choosing
  from the suggestion list.
  <p:messages/>
  <p:autoComplete value="#{languageBean.language}"
                  completeMethod="#{languageBean.completeLanguage}"
                  minQueryLength="2"
                  queryDelay="1000"
                  required="true"
                  requiredMessage="You must choose a language"/>
  <p:commandButton action="#{languageBean.register}"
                   value="Submit" ajax="false"/>
</h:form>
```

Results



p:autoComplete - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/autocomplete-2.jsf
Google
Submit
Autocompleter Data - Mozilla Firefox

p:autoComplete

Autocompleter with Selection Options

Choose a Programming Language

You can enter your own choices in

J

1 character: no autocompletion.

Ja

Java
JavaScript
JavaFX Script

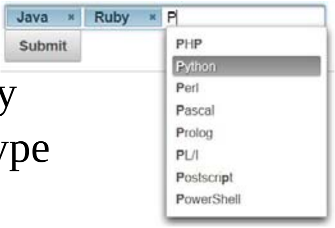
Autocompleter Data

Language chosen: Jaqqqq

User can enter arbitrary text; it need not match an autocomplete choice

p:autocomplete: Multiple Selections and Ajax Listeners

Multiple Selection



- **multiple="true"**
  - Lets user choose more than one entry
  - Corresponding bean property is of type List<String> instead of String
- **Ajax calls triggered on select/unselect**
  - <p:ajax event="itemSelect" listener="…" …/>
  - <p:ajax event="itemUnselect" listener="…" …/>
    - Ajax triggered when users adds or removes items from set of selections. Listener method takes a SelectEvent or UnSelectEvent as argument, and from that you can discover the item added or removed.
      - Instead of using listener, you could just track the List<String>, which is updated automatically. However, it is usually more convenient to use the Listener if you only care about the item that was most recently added or removed.

Example: Multiple Selections

- **Input page collects**
  - Any number of computer programming languages
    - User can only choose among suggestion, and cannot enter languages not in the list. I.e., forceSelection is automatically true for each individual entry, and this behavior cannot be changed.
- **Input page shows Ajax-updated message**
  - For every entry added to or removed from set of choices
- **Results page shows**
  - Confirmation of list of languages selected.
    - Uses ui:repeat to put them into <ul> list.

Bean (New Bean Property)

```
@ManagedBean
public class LanguageBean {
  ... // Language choices shown earlier

  private List<String> languages;

  public List<String> getLanguages() {
    return(languages);
  }

  public void setLanguages(List<String> languages) {
    this.languages = languages;
  }
```

Bean (Completion Method)

```
// Autocompleter method
public List<String> completeLanguage(String languagePrefix) {
  List<String> matches = new ArrayList<String>();
  for(String possibleLanguage: languageArray) {
    if(possibleLanguage.toUpperCase()
                          .startsWith(languagePrefix.toUpperCase())) {
      matches.add(possibleLanguage);
    }
  }
  return(matches);
}
```

This method did not change from that shown in earlier example. The completion method gives options for one choice at a time, even when you set multiple to true.

```
  // New action controller method

  public String register2() {
    return("show-languages");
  }
}
```

Bean (Ajax Listeners)

```
public void selectListener(SelectEvent event) {
  String itemSelected = event.getObject().toString();
  String message =
    String.format("Added '%s' to selections", itemSelected);
  FacesContext context = FacesContext.getCurrentInstance();
  context.addMessage(null, new FacesMessage(message));
}

public void unselectListener(UnselectEvent event) {
  String itemSelected = event.getObject().toString();
  String message =
    String.format("Removed '%s' from selections", itemSelected);
  FacesContext context = FacesContext.getCurrentInstance();
  context.addMessage(null, new FacesMessage(message));
}
```

The FacesMessages will be shown with p:message and updated on the fly with p:ajax.

Input Page

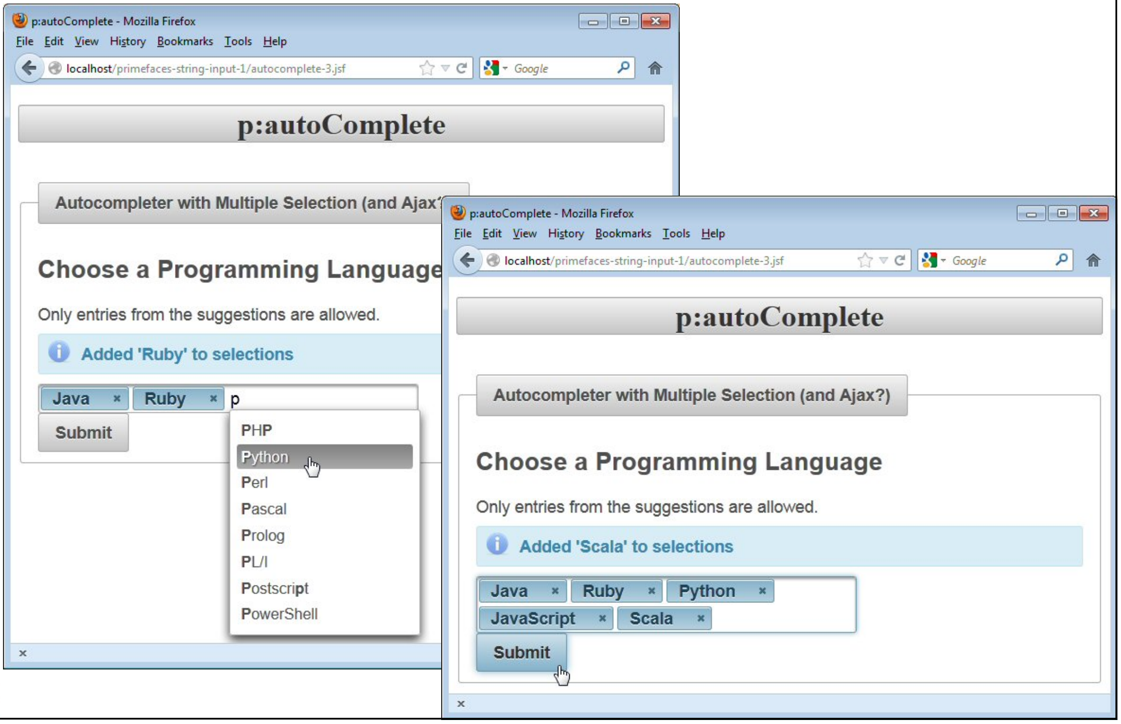
```
<h:form>
   <h2>Choose a Programming Language</h2>
   Only entries from the suggestions are allowed.
   <p:messages id="messages"/>
   <p:autoComplete value="#{languageBean.languages}"
                   completeMethod="#{languageBean.completeLanguage}"
                   required="true"
                   requiredMessage="You must choose a language"
                   multiple="true">
     <p:ajax event="itemSelect"
             listener="#{languageBean.selectListener}"
             update="messages"/>
     <p:ajax event="itemUnselect"
             listener="#{languageBean.unselectListener}"
             update="messages"/>
   </p:autoComplete>
   <p:commandButton action="#{languageBean.register2}"
                    value="Submit" ajax="false"/>
</h:form>
```

Reminder: do not use forceSelection="true" with multiple="true". Forced selections are automatic, but auto completer will not work properly if you explicitly say forceSelection="true".

Results Page

```
...
<ul>
<ui:repeat var="language"
           value="#{languageBean.languages}">
  <li>#{language}</li>
</ui:repeat>
</ul>
...
```

Results (Input Page)


p:autoComplete - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/autocomplete-3.jsf
Google

p:autoComplete

Autocompleter with Multiple Selection (and Ajax?)

Choose a Programming Language

Only entries from the suggestions are allowed.

Added 'Ruby' to selections

Java × Ruby × p

PHP
Python
Perl
Pascal
Prolog
PL/I
Postscript
PowerShell
Submit

p:autoComplete - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/autocomplete-3.jsf
Google

p:autoComplete

Autocompleter with Multiple Selection (and Ajax?)

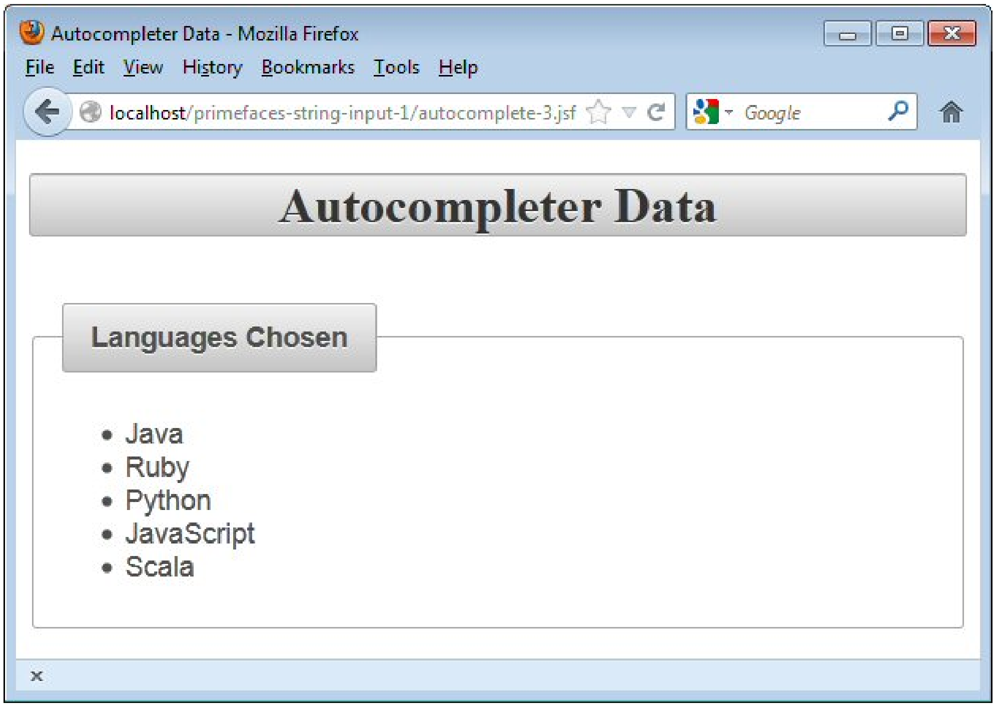
Choose a Programming Language

Only entries from the suggestions are allowed.

Added 'Scala' to selections

Java × Ruby × Python × JavaScript × Scala ×

Results (Results Page)


Autocompleter Data - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/autocomplete-3.jsf
Google

Autocompleter Data

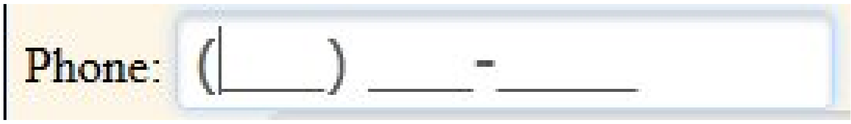
Languages Chosen

- Java
- Ruby
- Python
- JavaScript
- Scala

Constrained String Input: p:inputMask

p:inputMask: Overview

- **Appearance and behavior**
  - Textfield with template text that constrains the values that the user can enter.
- **Purpose: for collecting strings from user**
  - Value is not converted. Bean property must be string. Spaces, dashes and other template text are sent with the string and must be parsed on server.
-



Phone: (___) ___-____

Documentation for p:inputMask

- **Very, very sparse in official User's Guide**
  - At least as of 5.1
  - *No* description of the interpretation of the special characters (9, a, *, and ?)
- **Underlying library is well documented**
  - http://digitalbush.com/projects/masked-input-plugin/
- **Covered well in *PrimeFaces Cookbook 2/e***
  - See page 54 and following

The mask Attribute

- **<p:inputMask mask="…" … />**
  - Each character has five possible values
    - 9. Permits only a number to be entered there.
    - a. Permits only a letter (upper or lower case) to be entered there.
    - *. Permits a letter or a number to be entered there.
    - ?. Indicates that everything following is optional
    - Anything else. Literal text that is displayed to the user and is not editable. This text does become part of value sent to server.

The value and slotChar Attributes

- **<p:inputMask mask="…"**
  value="…" slotChar="…"/>
  - value
    - Should point to bean property of type String. Literal text in textfield (e.g., parens and dashes in phone number) is part of value sent to server, so server method must parse it
  - slotChar
    - The placeholder text. Defaults to underscore.

    <p:inputText mask="(999) 999-9999"/>

    - User sees “(___) ___-____” after clicking in field

    <p:inputText mask="(999) 999-9999" slotChar="X"/>

    - User sees “(XXX) XXX-XXXX” after clicking in field

Example: Collecting Data

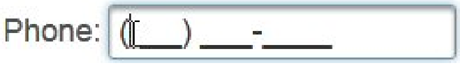
- **Input page collects**
  - Phone number (no extension)
  - Phone number with extension
  - Social Security Number
  - Product key
  - License plate value
  - Arbitrary message (with p:inputText) for comparison
  - Arbitrary message (with h:inputText) for comparison
    - Incomplete values will be treated as missing values, and requiredMessage will be shown
- **Results page shows**
  - User values

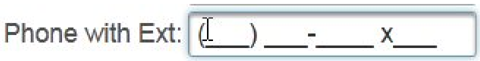
Bean

```
@ManagedBean
public class MaskBean {
  private String phone, phoneWithExt,
                 ssn, productKey, license;

  // Getters and setters for each
}
```

Input Page (Part I)

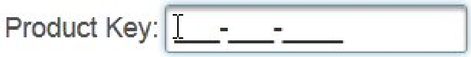
```
<h:form>
<h:panelGrid columns="3" class="formTable">
  Phone:
  <p:inputMask mask="(999) 999-9999"
               value="#{maskBean.phone}"
               required="true" id="phone"
               requiredMessage=
                 "Missing or incomplete phone number"/>
  <p:message for="phone"/>

  Phone with Ext:
  <p:inputMask mask="(999) 999-9999 x999"
               value="#{maskBean.phoneWithExt}"
               required="true" id="phoneWithExt"
               requiredMessage=
    "Missing or incomplete phone number with extension"/>
  <p:message for="phoneWithExt"/>
```



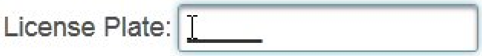


Input Page (Part II)

```
Product Key:
<p:inputMask mask="aaa-999-a999"
             value="#{maskBean.productKey}"
             required="true" id="productKey"
             requiredMessage=
               "Missing or incomplete product key"/>
<p:message for="productKey"/>
```

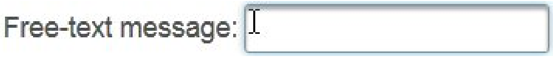

Product Key: ___-___-____

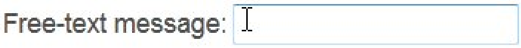
```
License Plate:
<p:inputMask mask="******"
             value="#{maskBean.license}"
             required="true" id="license"
             requiredMessage=
              "Missing or incomplete license plate"/>
<p:message for="license"/>
```


License Plate: ____

Input Page (Part III)

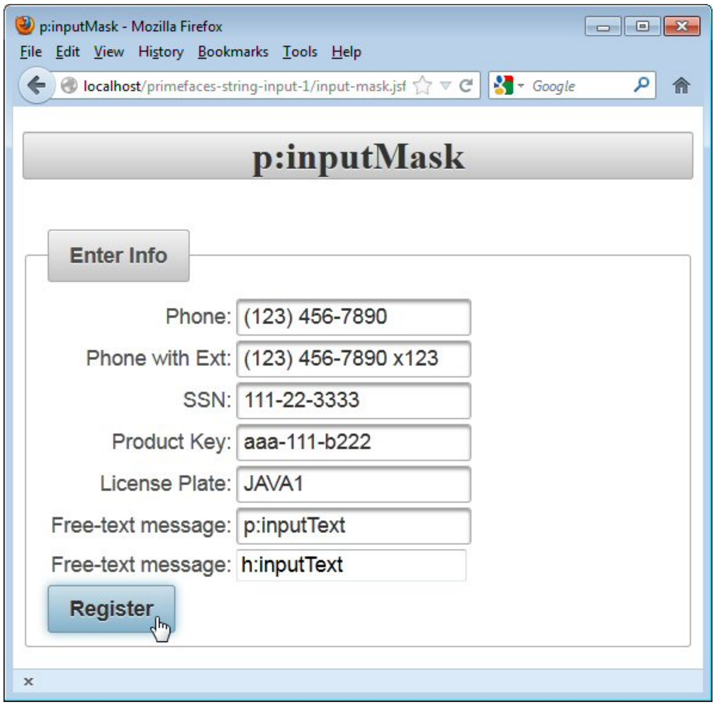
```
  Free-text message:
  <p:inputText value="#{maskBean.message1}"
               required="true" id="message1"
               requiredMessage="Missing message"/>
  <p:message for="message1"/>

  Free-text message:
  <h:inputText value="#{maskBean.message2}"
               required="true" id="message2"
               requiredMessage="Missing message"/>
  <p:message for="message2"/>
</h:panelGrid>
  <p:commandButton action="#{maskBean.register}"
                   value="Register" ajax="false"/>
</h:form>
```


Free-text message:


Free-text message:

The ordinary textfields at the end are just for the sake of comparison.

Example: Results (Form)


p:inputMask - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/input-mask.jsf
Google

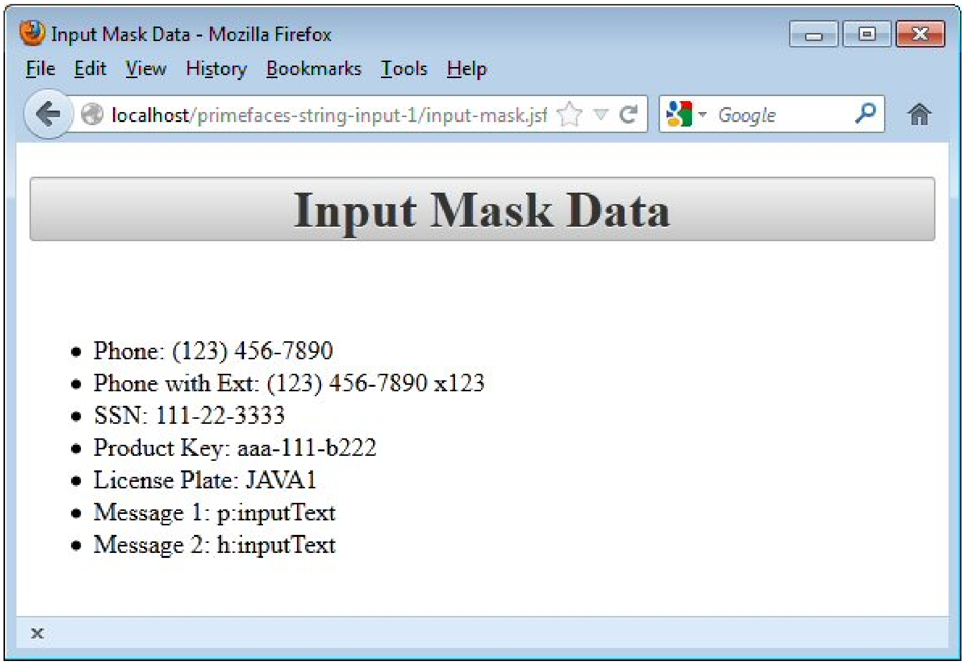
p:inputMask

Enter Info

Phone:	(123) 456-7890
Phone with Ext:	(123) 456-7890 x123
SSN:	111-22-3333
Product Key:	aaa-111-b222
License Plate:	JAVA1
Free-text message:	p:inputText
Free-text message:	h:inputText

Register

Example: Results (Results Page)


Input Mask Data - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/primefaces-string-input-1/input-mask.jsf
Google

Input Mask Data

- Phone: (123) 456-7890
- Phone with Ext: (123) 456-7890 x123
- SSN: 111-22-3333
- Product Key: aaa-111-b222
- License Plate: JAVA1
- Message 1: p:inputText
- Message 2: h:inputText

Notice that the template text (spaces, parens, dashes, etc.) becomes part of the submitted value.

Wrap-Up

Summary

- p:autocomplete
  - <p:autocomplete
    ```
    <p:autocomplete
        value="#{someBean.someStringProperty}"
        completeMethod="#{someBean.methodReturningList}"
        forceSelection="true"
        minQueryLength="2 or more"
        queryDelay="number of milliseconds"
        multiple="true"/>
    ```
    - p:ajax can take listeners for itemSelect (single or multiple) or itemUnselect (multiple)
    - Many other attributes and options – see online docs
    - If you use multiple="true", do not use forceSelection="true"
- p:inputMask

  -
    ```
    <p:inputMask value="#{someBean.string}"
                 mask="aa-99-**"/>
    ```

Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training