

**ORACLE®**

# Oracle Digital Assistant

## The Complete Training

### Apache FreeMarker Overview

# Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# Program agenda

- 1 ➤ Apache Freemarker – what is good for?
- 2 ➤ Built-In expressions
- 3 ➤ Support for arrays
- 4 ➤ Directives

# Program agenda

- 1 ➤ Apache Freemarker – what is good for?
- 2 ➤ Built-In expressions
- 3 ➤ Support for arrays
- 4 ➤ Introducing digital assistant and routing

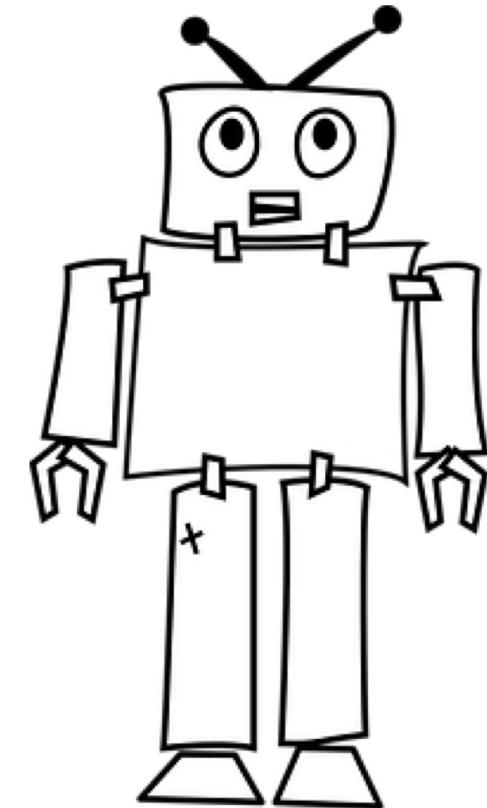
# Apache Freemarker – what is it good for?

- Access variables in OBotML
- Conditionally show or hide response items
- Find key words in a user message
- Print a list of values
- Display arrays sorted by a specific data attribute
- It's your “get out of jail” card for many problems
  - In dialog flow, AND configuration settings

## Custom components? No. Not needed!

Apache FreeMarker is the **expression language** in Oracle Digital Assistant.

You use it to **access, modify and format** data content, conditionally **show or hide components, assign values**, and many other use cases.



<https://freemarker.apache.org/docs/index.html>

TEMPLATE AUTHOR'S GUIDE

- + Getting Started
- + Values, Types
- + The Template
- + Miscellaneous

PROGRAMMER'S GUIDE

- + Getting Started
- + The Data Model
- + The Configuration
- + Miscellaneous

TEMPLATE LANGUAGE REFERENCE

- + Built-in Reference
- + Directive Reference
  - Special Variable Reference
  - Reserved names in FTL
- + Deprecated FTL constructs

# Apache FreeMarker Manual

## For Freemarker 2.3.28

### Table of Contents

- Template Author's Guide
  - Getting Started
    - Template + data-model = output
    - The data-model at a glance
    - The template at a glance
  - Values, Types
    - Basics
    - The types
  - The Template
    - Overall structure
    - Directives
    - Expressions
    - Interpolations
  - Miscellaneous
    - Defining your own directives
    - Defining variables in the template

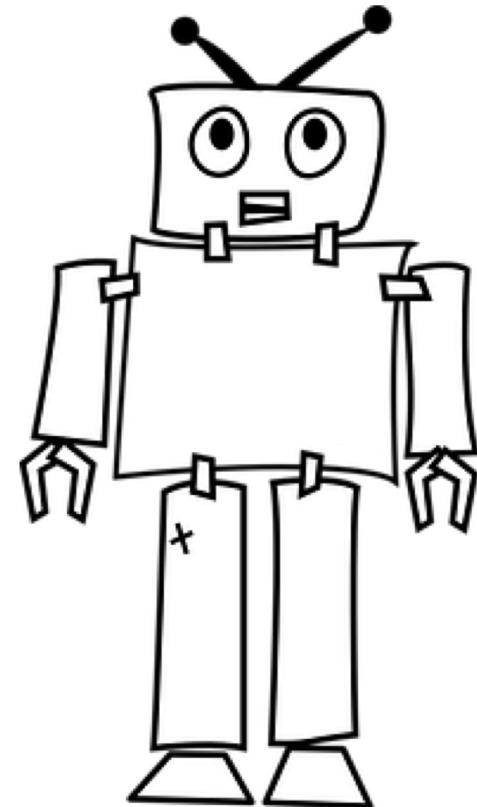
# Program agenda

- 1 ➤ Apache Freemarker – what is good for?
- 2 ➤ Built-In expressions
- 3 ➤ Support for arrays
- 4 ➤ Introducing digital assistant and routing

# String built-in

- boolean
  - capitalize
  - contains
  - date, time, datetime
  - ends\_with
  - ensure\_ends\_with
  - ensure\_starts\_with
  - index\_of
- 
- keep\_after
  - keep\_after\_last
  - keep\_before
  - keep\_before\_last
  - last\_index\_of
  - left\_pad
  - length
  - lower\_case / upper\_case
  - number
  - match
- 
- replace
  - right\_pad
  - remove\_beginning
  - remove\_ending
  - slice
  - split
  - starts\_with
  - string
  - switch
  - trim

Apache FreeMarker built-ins are added to an expression using the '?' character

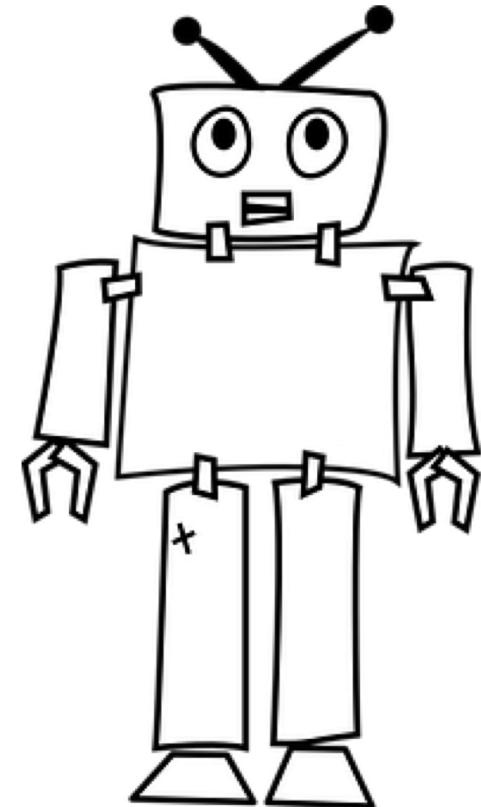


# String built-in examples

```
setStringValue:  
    component: "System.SetVariable"  
    properties:  
        variable: "textVar"  
        value: "Hello World"
```

FreeMarker Expression	Result
<code> \${textVar.value?length}</code>	11
<code> \${textVar.value?replace('World', 'Friends')}</code>	Hello Friends

Apache FreeMarker built-in  
expressions can be **chained** together



# String built-in examples

```
setStringValue:  
  component: "System.SetVariable"  
  properties:  
    variable: "flight"  
    value: "UA 1234"
```

FreeMarker Expression	Result
<code> \${flight.value?contains('ua')?string('UA flight','other')}</code>	UA flight
<code> \${flight.value?trim?lower_case?remove_beginning('ua ')}"</code>	1234

# Number built-in

- abs
  - returns positive number
- round
  - rounds to the nearest whole number
  - $\geq n.5$  rounded upwards
- floor
  - rounds the number downwards
- ceiling
  - rounds the number upwards
- lower\_abc
  - converts 1, 2, 3, etc., to "a", "b", "c"
- upper\_abc
  - converts 1, 2, 3, etc., to "A", "B", "C"
- string
  - converts a number to string
- <number>?string['###.##']
  - formatted string output
  - rounds to nearest neighbor
- Use utf-8 for currency
  - ?string['###.##\u00A4'] \$
  - ?string['###.##\u20AC'] €
  - ?string['###.##\u00A3'] £

# Number examples

```
setNegativeValue:  
    component: "System.SetVariable"  
    properties:  
        variable: "negativeValue"  
value: -2.512
```

FreeMarker Expression	Result
<code> \${negativeValue.value?abs}</code>	2.512
<code> \${negativeValue.value?round}</code>	- 2
<code> \${negativeValue.value?floor}</code>	- 3
<code> \${negativeValue.value?string['###.##']}</code>	- 2.51
<code> \${negativeValue.value?abs?string['###.##\u00A3']}</code>	2.51 £

# Date examples

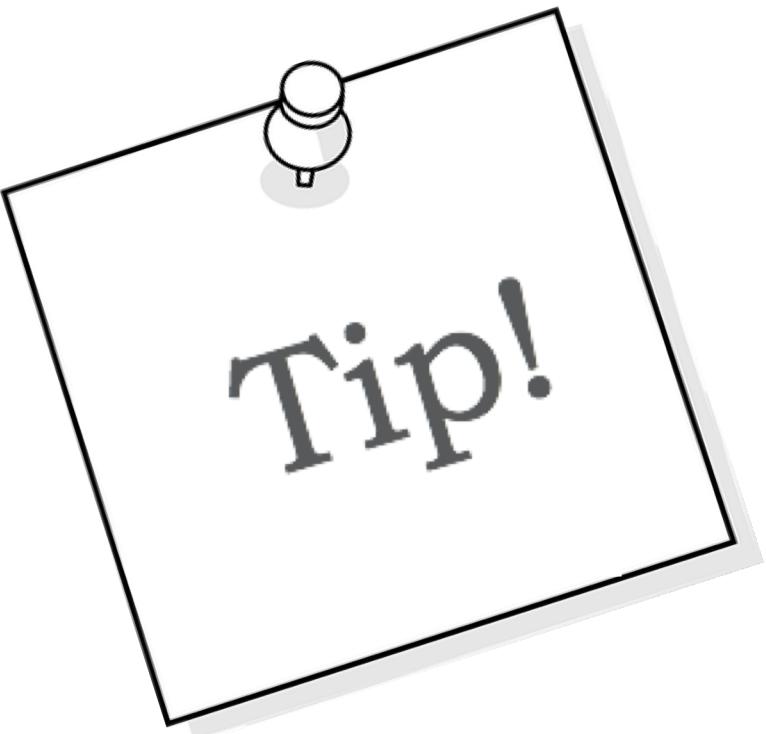
```
variables:  
  dateVar: "DATE"  
  
...  
getNLPDate:  
  component: "System.SetVariable"  
  properties:  
    variable: "dateVar"  
    value: "${iResult.value.entityMatches['DATE'][0]}"
```

FreeMarker Expression	Result
<code> \${dateVar.value.date}</code>	Date in milliseconds UTC
<code> \${dateVar.value.date?long?number_to_date?string.short}</code>	4/13/18
<code> \${dateVar.value.date?long?number_to_date?string.medium}</code>	April 13, 1:29:54 PM
<code> \${dateVar.value.date?long?number_to_date?string.long}</code>	April 13, 1:29:54 PM UTC

# Date examples

```
variables:  
    dateVar: "DATE"  
  
...  
getNLPDate:  
    component: "System.SetVariable"  
    properties:  
        variable: "dateVar"  
        value: "${iResult.value.entityMatches['DATE'][0]}"
```

FreeMarker Expression	Result
<code>\${dateVar.value.date?number_to_date?string['dd.MM.yyyy, HH:mm']}</code>	13.04.2018, 13:29
<code>\${dateVar.value.date?long?number_to_datetime?iso_local}</code>	2018-04-13T13:29:54Z
<code>\${dateVar.value.date?number_to_datetime?time}}</code>	1:29:04 AM/PM



You can use Apache FreeMarker  
To test if a variable or an object  
has content



# Handle variables with no value

- Solution works with all variable types, as well as variable attributes
- Print 'unknown' if a variable does not have a value
  - \${firstName.value! 'unknown'}
- Setting a default value if no value is set
  - Using System.SetVariable
  - \${subscriptionStart.value?has\_content?then(subscriptionStart.value, .now)}
- Print 'hello stranger' if variable has no content
  - <#if firstName.value?has\_content>hello \${varName.value}<#else>hello stranger<\#if>

# Program agenda

- 1 ➤ Apache Freemarker – what is good for?
- 2 ➤ Built-In expressions
- 3 ➤ Support for arrays
- 4 ➤ Introducing digital assistant and routing

# Arrays in Apache Freemarker

- System arrays
  - \${iResult.value.entityMatches['name of entity']}
  - \${iResult.value.intentMatches.summary}
- Variable arrays
  - Context variable of type string
    - Custom Components
    - Apache FreeMarker Template Expression

# Array built-in

- `first.<attribute>, last.<attribute>`
  - Returns first or last element's attribute
- `seq_index_of(value)`
  - Works with simple arrays
- `seq_last_index_of(value)`
  - Works with simple arrays
- `join('delimiter_char')`
  - Returns simple array as strings delimited by delimiter\_char
- `seq_contains(value)?string('yes','no')`
  - returns yes or no string
- `min, max`
  - For simple arrays of type number or datetime
  - Returns min or max value
- `sort`
  - Sorts simple array
- `sort_by(attribute)`
  - Sorts object array by named attribute

# Building arrays with Apache FreeMarker

## Array of Objects

```
variables:  
    fruits: "string"  
...  
states:  
  
fruitArray:  
    component: "System.SetVariable"  
    properties:  
        variable: "fruits"  
        value:  
            - name: "Apple"  
              color: "green"  
            - name: "Banana"  
              color: "yellow"
```

## Simple Array

```
variables:  
    colors: "string"  
...  
states:  
colorsArray:  
    component: "System.SetVariable"  
    properties:  
        variable: "colors"  
        value:  
            - "red"  
            - "green"  
            - "blue"  
            - "yellow"
```

# Array built-In examples

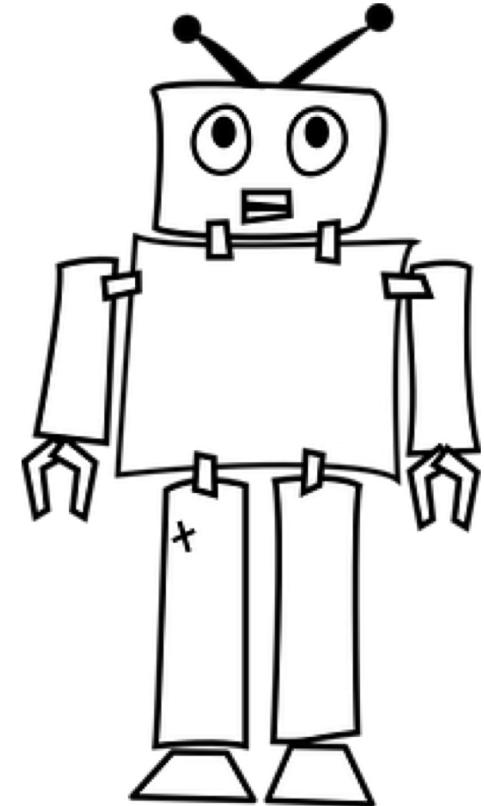
```
fruitArray:  
    component: "System.SetVariable"  
    properties:  
        variable: "fruits"  
        value:  
            - name: "Apple"  
              color: "green"  
            - name: "Banana"  
              color: "yellow"
```

FreeMarker Expression	Result
<code> \${fruits.value?size}</code>	2
<code> \${fruits.value[0].name}</code>	Apple
<code> \${fruits.value[0].name !'unknown'}</code>	Same as above, but prints "unknown" if name is missing
<code> \${fruits.value?sort_by('color')?reverse[0].name}</code>	Prints "Banana"

# Program agenda

- 1 ➤ Apache Freemarker – what is good for?
- 2 ➤ Built-In expressions
- 3 ➤ Support for arrays
- 4 ➤ Introducing digital assistant and routing

Apache FreeMarker **directives** are scripts  
that conditionally assign values or iterate  
through data arrays and data object keys  
and values



# Commonly used directives

- If-else
  - <**#if** condition> ... <**#elseif** condition2> ...<**#elseif** condition3> ... ... <**#else**> ... <**#if**>

```
actions:  
- label: "More Pizzas"  
  type: "postback"  
visible:  
  expression: "<#if cardsRangeStart?number+4 < pizzas.value?size>true<#else#if>"  
payload:  
  action: "more"  
variables:  
  cardsRangeStart: "${cardsRangeStart?number+4}"  
name: "More"
```

**MEAT LOVER**

Classic marinara sauce, authentic old-world pepperoni, all-natural Italian sausage, slow-roasted ham, hardwood smoked bacon, seasoned pork and beef.



[Order Now](#)

**SUPREME**

Classic marinara sauce, authentic old-world pepperoni, seasoned pork, beef, fresh mushrooms, fresh green bell peppers and fresh red onions.



[Order Now](#)

[More Pizzas](#)

# Expert tip: comparison operator encoding

Because sometimes a comparison is interpreted as the end of the script

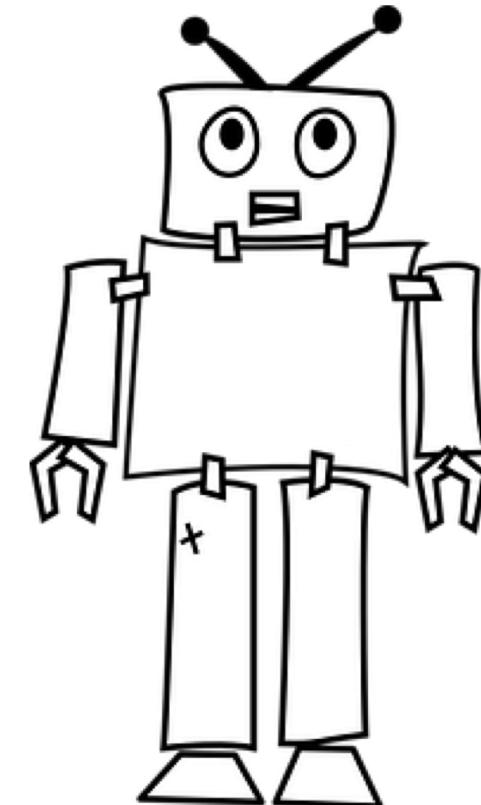
Operator	Encoding
>	&gt;
<	&lt;
>=	&gt;=
<=	&lt;=

# Commonly used directives

- List
  - <#list sequence as item> ... </#list>
  - <#list hash as key, value> ... </#list>
  - <#list sequence as item> ... <#else> executed if 0 items </#list>

Note that you don't use \${ ... } to declare expressions in the condition part of a directive. However, you do so in its body

```
<#list someArrayVariable.value as item>  
  ${item}  
</#list>
```



# Example of iterating the values in an array

createArrayOfStrings:

  component: "System.SetVariable"

  properties:

    variable: "messages"

    value:

      - "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy ..."

      - "At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd ..."

      - "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam ..."

printMessages:

  component: "System.Output"

  properties:

    text: |-

      <#list messages.value as text>\${text}



      </#list>

  transitions:

    return: "done"

  Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

  At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

  Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua

  At vero eos et accusam et justo duo dolores et ea rebum.

# Example of iterating attributes in a data object

```
createPersonDataObject:
```

```
    component: "System.SetVariable"
```

```
properties:
```

```
    variable: "data"
```

```
    value: {"FirstName":"John", "LastName":"Doe", "RegistrationId": "1234" }
```

```
print:
```

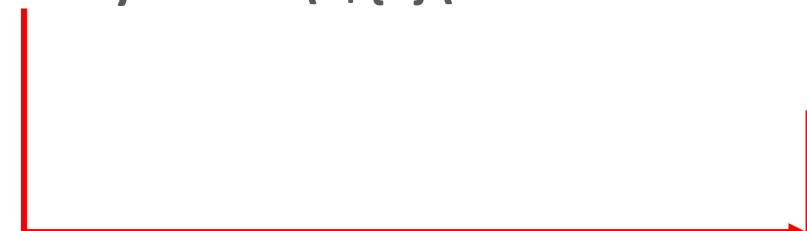
```
component: "System.Output"
```

```
properties:
```

```
    text: "<#list data.value?keys as k> \"${k}\\" has a value of \"${data.value[k]}\"\n</#list>"
```

```
transitions:
```

```
return: "done"
```



```
"FirstName" has a value of "John"  
"LastName" has a value of "Doe"  
"RegistrationId" has a value of "1234"
```

# Commonly used directives

- Switch
  - Similar statement as in other programming languages
  - <#switch value> <#case refValue1> ... <#break> <#case refValue2> ... <#break> ... <#case refValueN> ... <#break> <#default> ... </#switch>

# Example of using a switch statement

```
askMonthInYear:  
  component: "System.Text"  
  properties:  
    prompt: "Type a value between 1 - 12"  
    variable: "monthInYear"
```

```
printMonth:  
  component: "System.Output"  
  properties:  
    text: "<#switch monthInYear.value>  
      <#case 1>I love the snow in January<#break>  
      <#case 2>Too colnd in February<#break>  
      <#case 3>Sun is coming out in March<#break>  
      <#case 4>Be aware of fools day in April<#break>  
      <#case 5>It is spring time in May<#break>  
      <#case 6>Summer is close in June<#break>  
      <#case 7>Schools out in July<#break>  
      <#case 8>Its hot in August<#break>  
      <#case 9>Like the sunsets in September<#break>  
      <#case 10>Leafs show their best colors in October<#break>  
      <#case 11>Daylight is shorter in November<#break>  
      <#case 12>Christmas is coming in December<#break>  
      <#default>Not a valid month, sorry  
    </#switch>"  
  
  transitions:  
    return: "done"
```

Type a value between 1 - 12

10

Leafs show their best colors in October

# Integrated Cloud Applications & Platform Services

**ORACLE®**



## Oracle Digital Assistant Hands-On

TBD