
ORACLE®

Oracle Intelligent Bots Advanced Training

Multi Language Support

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.

Topic Agenda

1. Oracle's Approach
2. Configuration
3. System Components
4. Message Bundles
5. Custom Components
6. Strategies & Practices

Two Approaches to Building Multi-Language Bots

Native Language Bots

- Use a specific language for
  - Utterances
  - Entities
  - Prompts, titles, descriptions
- Each bot uses language specific intent and entity recognition engine
- Bot training and testing in specific language

Single Base-Language Bots

- Serve multiple languages from a single base language
  - Utterances
  - Entities
  - Prompts, titles, descriptions
- Uses translation service
  - Prompts are translated at runtime to detected user language
  - User input translated to base language
- Only "appears" as native language

Oracle Intelligent Bot Language Support

Current Approach

- Bot is built in English
- 3rd-party Translation Services
  - Translate user input to English
  - Translate bot responses to user language
- Used for intent resolution and entity extraction
  - Utterances and entity values are in English
- Applied globally or to individual components

Why it works.

"If no mistake have you made, yet losing you are … **a different game you should play**"

- Yoda, Star Wars

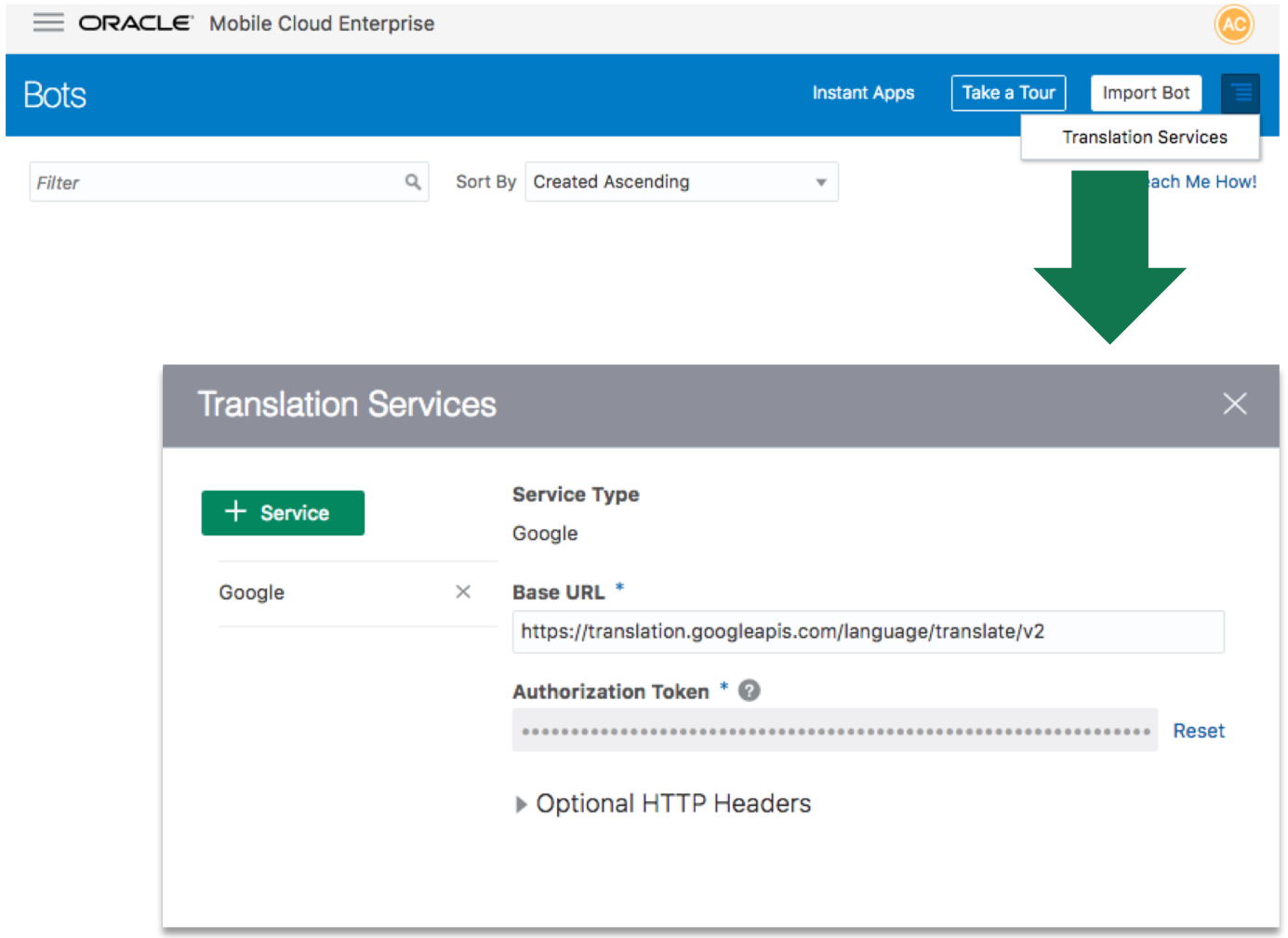
Image courtesy of Farbentek at FreeDigitalPhotos.net

Multi Language Support

Configuration

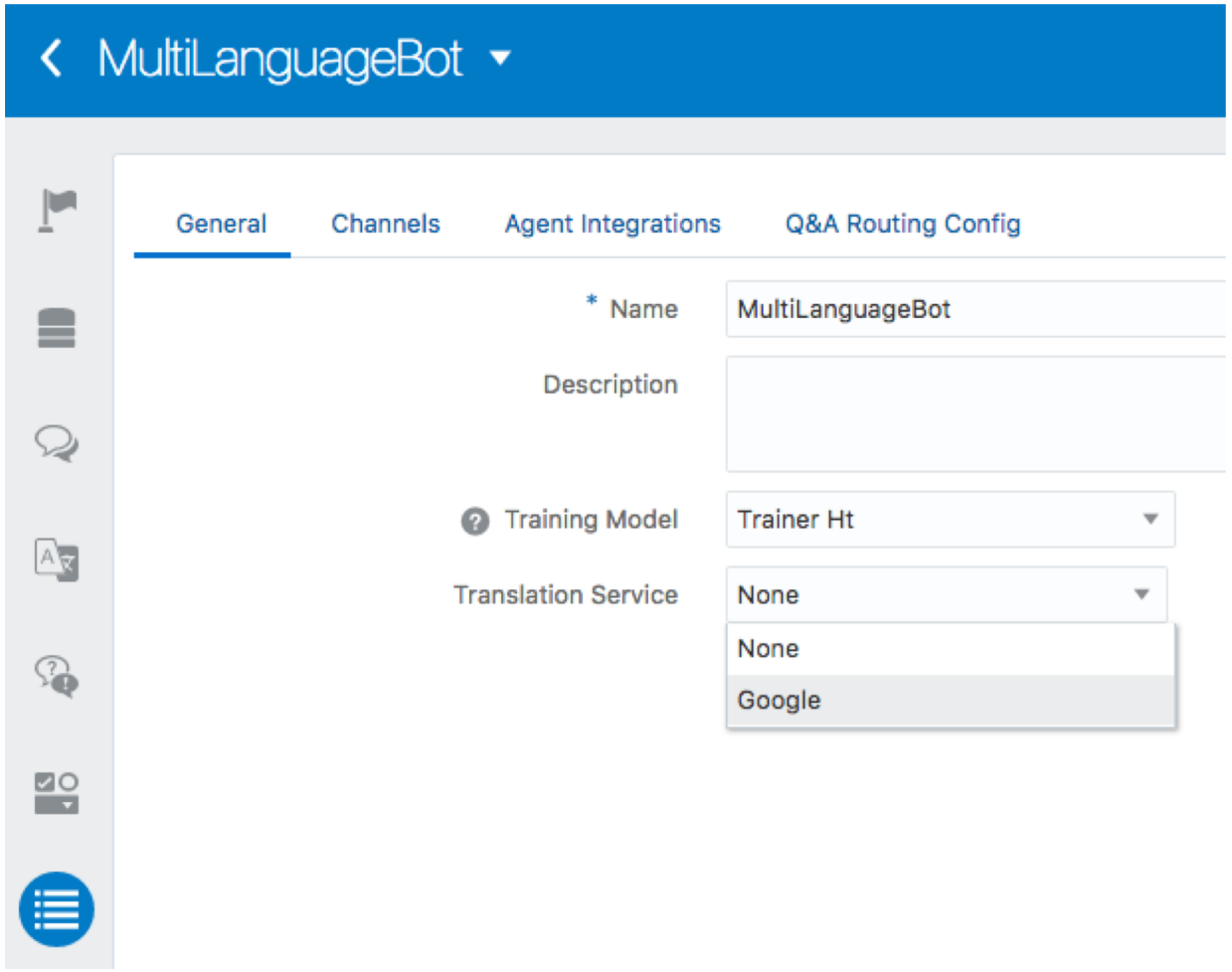
Setup

- Uses Google Translation API or Microsoft Translator Services
- Bring your own license
  - Authorization Token / Key
- Translation Service used for both runtime input and output



Enable Translation Service for Bot

- Reference a translation service configuration from a bot to use auto-translation



Enabling **auto-translation,** translates **user messages and bot responses** for the detected user language

Enabling / Disabling Auto-Translation

- Auto-translation is disabled by default
- Can be enabled / disabled at any time
  - Define "autoTranslate" context variable of type boolean
  - Set variable value to true
- When enabled
  - User messages are translated to English
  - Bot messages are translated to user language

```
variables:
  autoTranslate: "boolean"
  …
states:
  …
  enableAutotranslation:
    component: "System.SetVariable"
    properties:
      variable: "autoTranslate"
      value: true
    transitions: {}
```

Translation Strategies

Opt-in

- Disable auto-translation
- Detect user languages
  - From user message
  - From profile
- Enable auto-translation on component
- Use message bundles

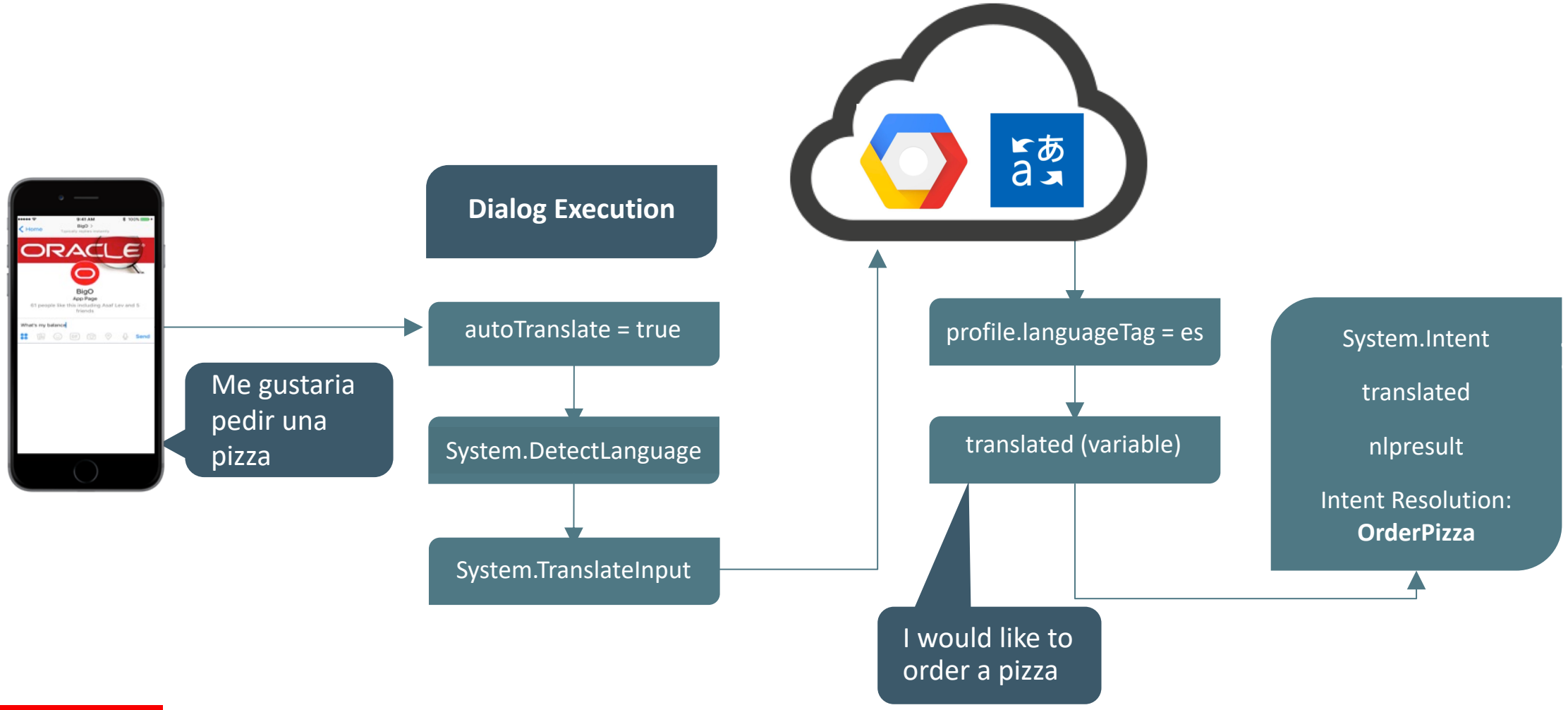
Opt-out

- Enable auto-translation
- Detect user languages
  - From user message
  - From profile
- Test bot
- Disable translation on individual components and use message bundles instead

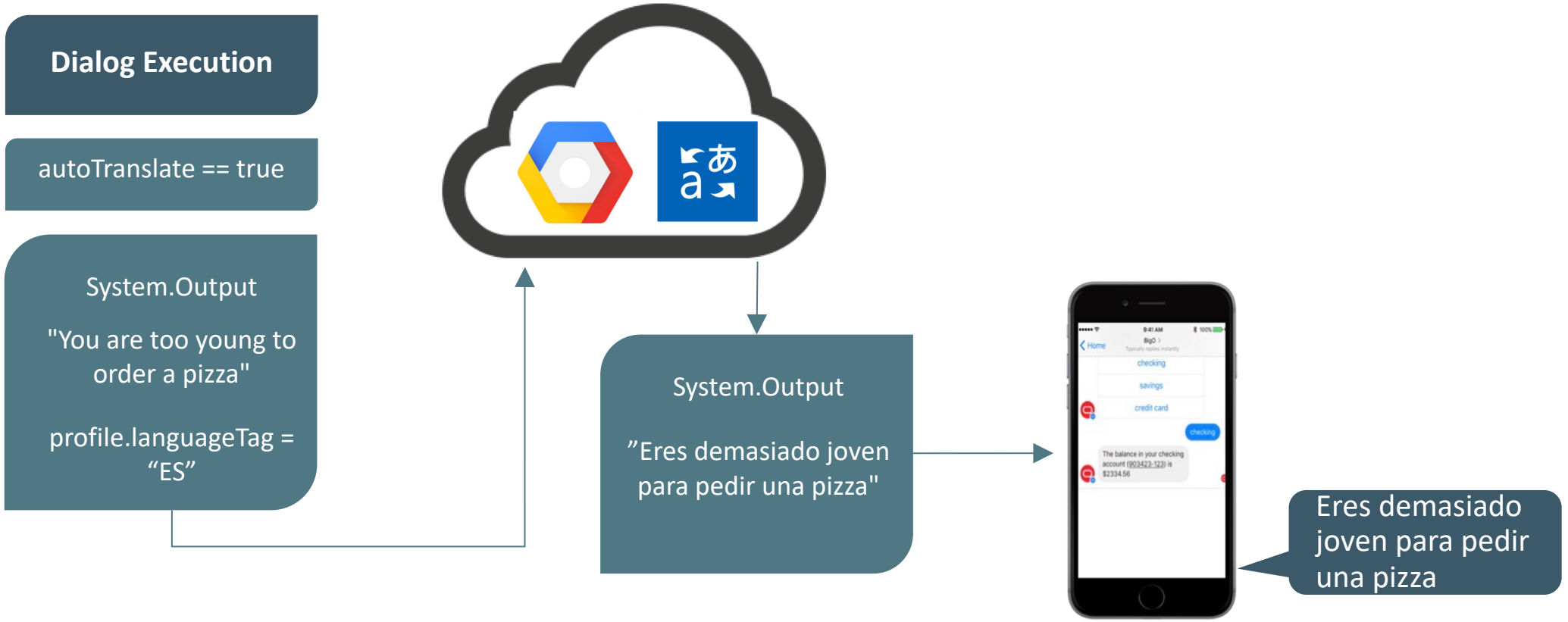
Multi Language Support

System Components

User Input Translation at Runtime



Bot Response Translation at Runtime



System.DetectLanguage

- Uses translation service to detect user language based on a provided input string
- Sets the "profile.languageTag" system variable to locale code of the detected user language
- Precedes value in "profile.locale" variable set by messenger client

Example 1

```
detectLanguageFromDirectInput:
  component: "System.DetectLanguage"
```

Example 2

```
context:
  variables:
     sourceString: “string”
 …
states:
  …
  detectLanguageFromSourceVariable:
    component: "System.DetectLanguage"
      properties:
        source: "sourceString"
```

What if the user does not enter text for the bot to detect the user language?

In this case your bot could use the language of the messenger client.

Manually setting the "profile.languageTag"

- The `profile.locale` variable holds the language setting of the messenger client
- Optional, but recommended, set the value of profile.languageTag to the value of profile.locale
- Alternatively, have the user selecting a language from a list or read it from a custom profile you maintain for users
  - Set profile.language variable

```
states:
  setLanguageToUserDefault:
    component: "System.SetVariable"
    variable: "profile.languageTag"
    value: "${profile.locale}"
```

System.TranslateInput

- Translates strings to English
  - Leverages translation service
  - **String input from direct user input**
  - String from context variable

Example 1

```
context:
  variables:
    englishString: "string"
 …
states:
  …
  translateInput:
    component: "System.TranslateInput"
    properties:
      variable: "englishString"
```

System.TranslateInput

- Translates strings to English
  - Leverages translation service
  - String input from direct user input
  - **String input from context variable**

Example 2

```
context:
  variables:
    englishString: "string"
    sourceString: "string"
 …
states:
  …
  translateInputString:
    component: "System.TranslateInput"
    properties:
      source: "sourceString"
      variable: "englishString"
```

System.TranslateInput **translates strings**, no objects.

Image courtesy of Farbentek at FreeDigitalPhotos.net

System.TranslateOutput

- Translates any string to the language detected for a user
  - profile.languageTag
  - profile.locale

Example

```
context:
  variables:
    translatedString: "string"
    sourceString: "string"
 …
states:
  …
  translateOutputString:
    component: "System.TranslateOutput"
    properties:
      source: "sourceString"
      variable: "translatedString"
```

Component "translate" Property

- Available on user interface and input components
  - Intent, Text, Output, List, CommonResponse
- Overrides the global auto-translate setting

```
statename:
  component: "System.Text"
  properties:
    …
    translate: true
    ...
```

Multi Language Support

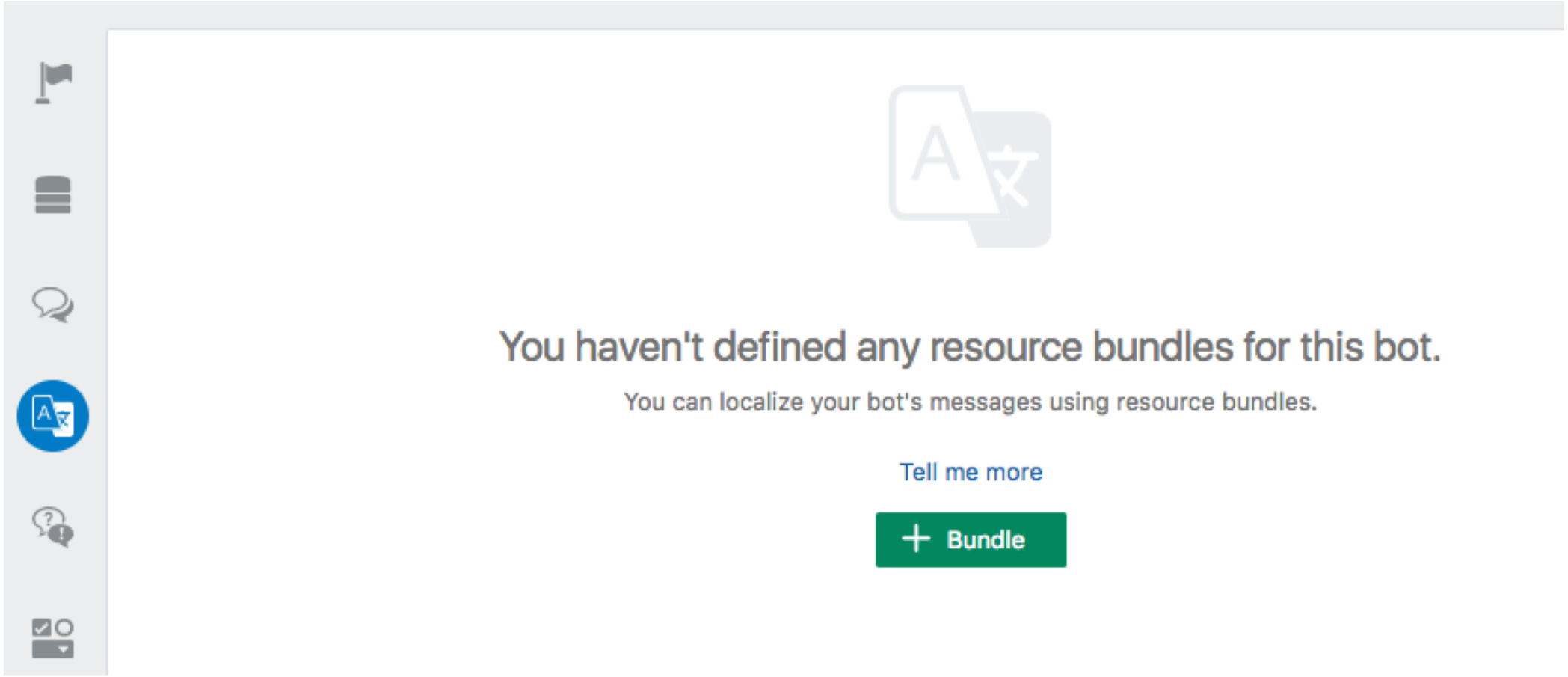
Resource Bundles

Resource bundles in Oracle Intelligent Bots **don't require a translation service.**

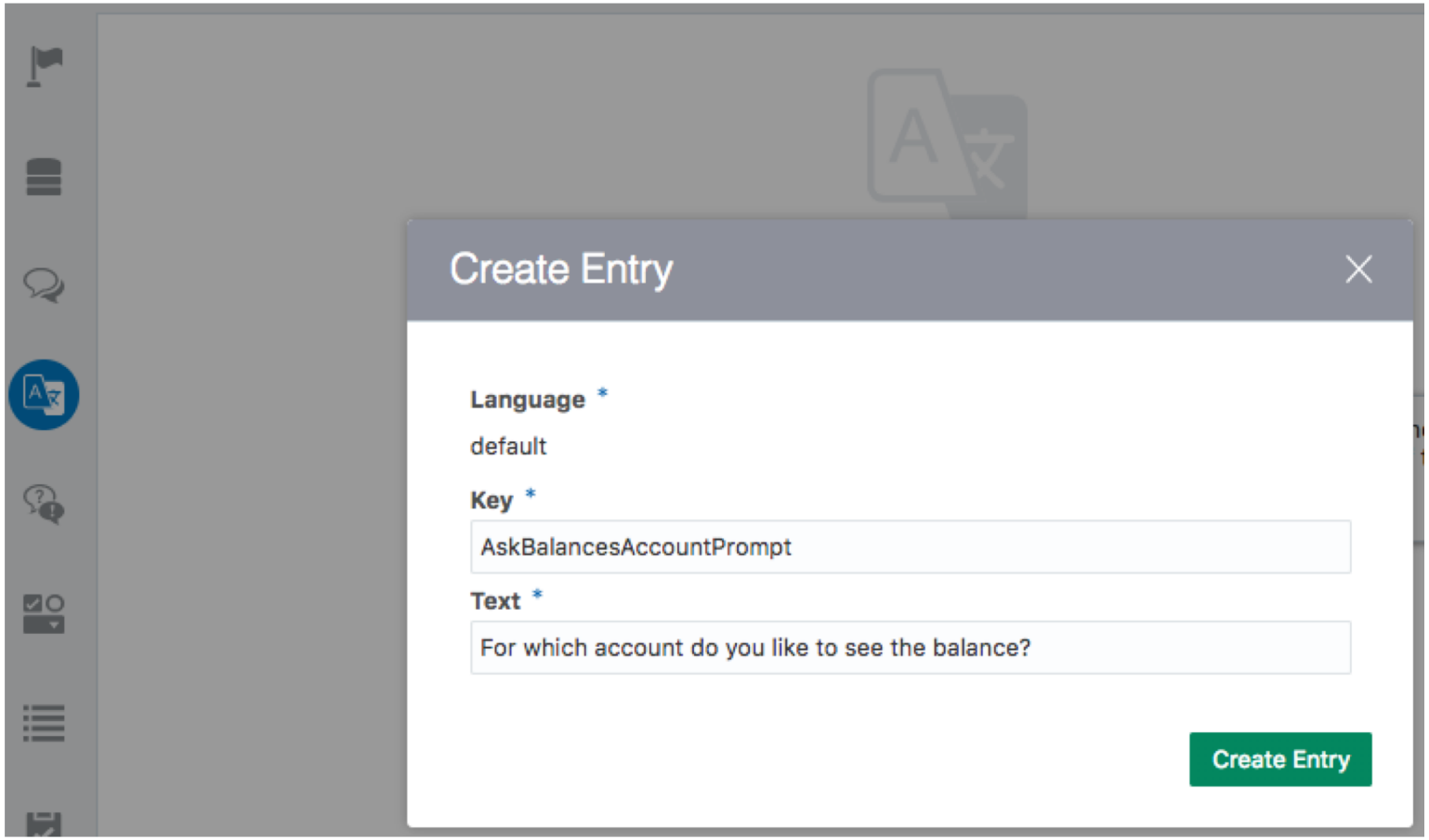
User Profile Variables

- profile.locale
  - Set by the messenger client used
- profile.languageTag
  - Usually set by System.DetectLanguage component
  - Can be set manually using System.SetVariable
  - Precedes profile.locale setting

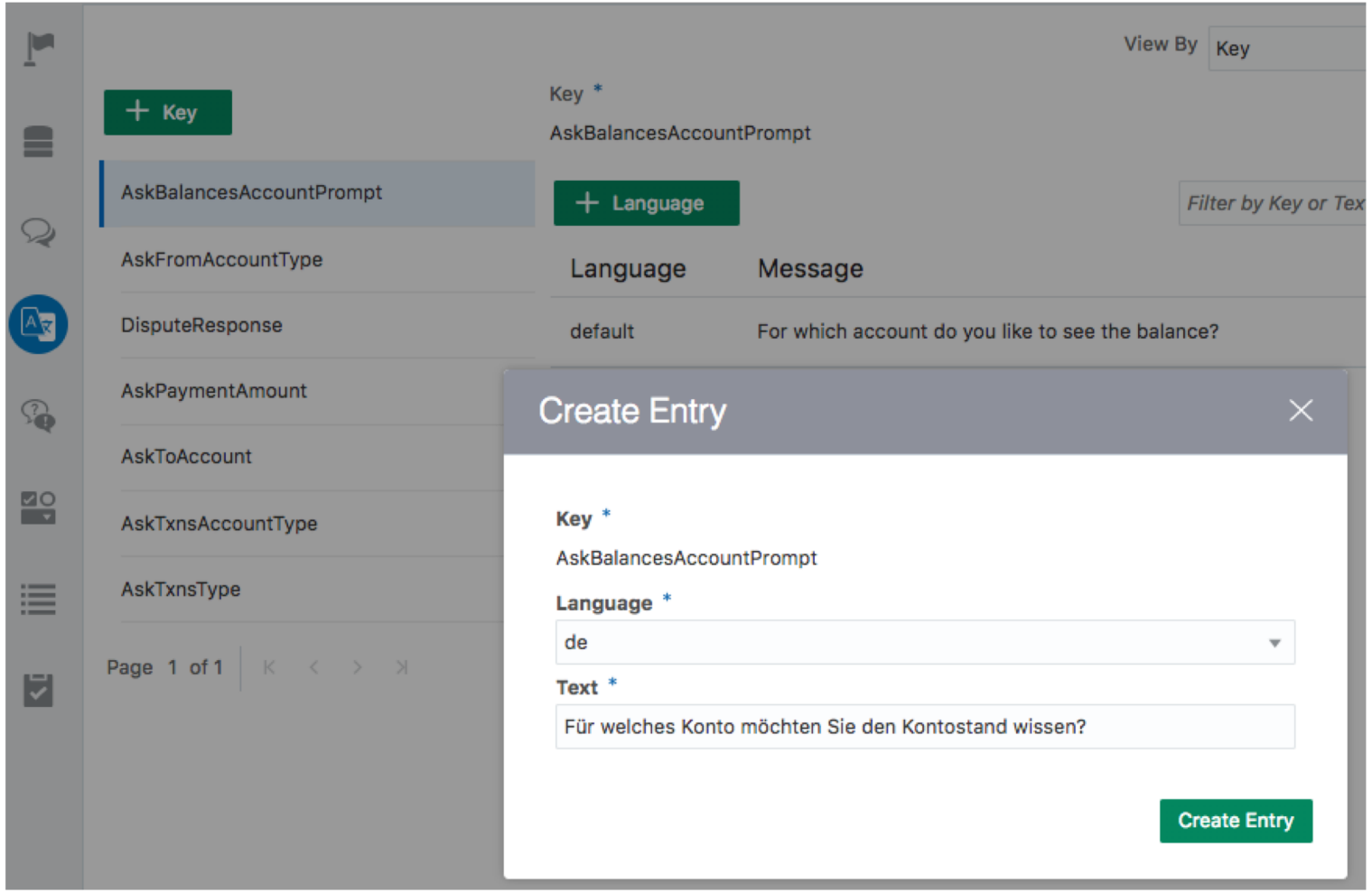
Creating a Resource Bundle



Defining a Default Language Key and String



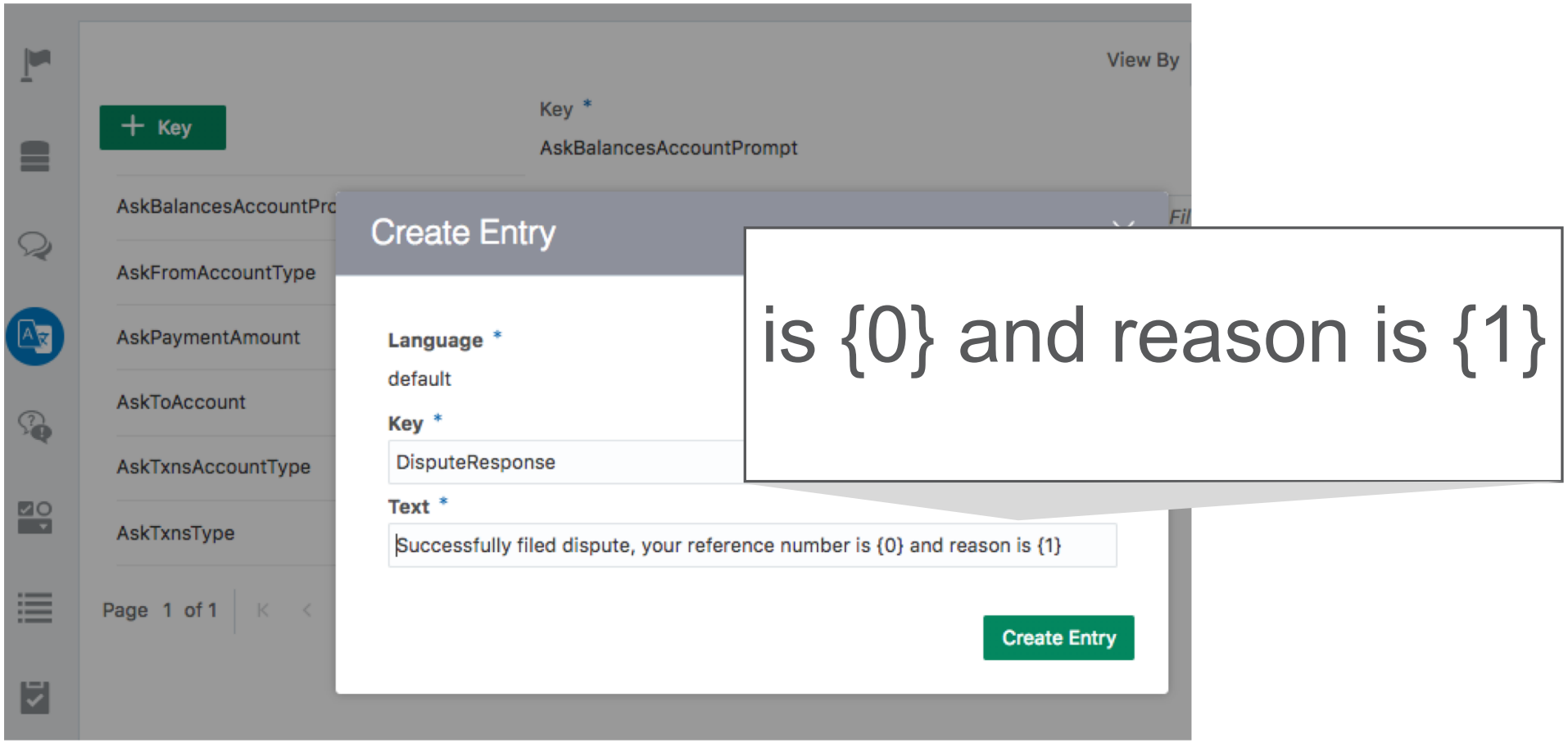
Create Translations



Resource strings can be **parameterized** allowing bot designers to **insert values referenced from context variables** at runtime

Image courtesy of Farbentek at FreeDigitalPhotos.net

Defining Variables in a Resource String



Requirement: Defining a Variable

```
variables:
   …
   rb: "resourcebundle"
```

Accessing a Resource Bundle String

Resource string with **no** parameters

${rb('message_key')} or ${rb.message_key}

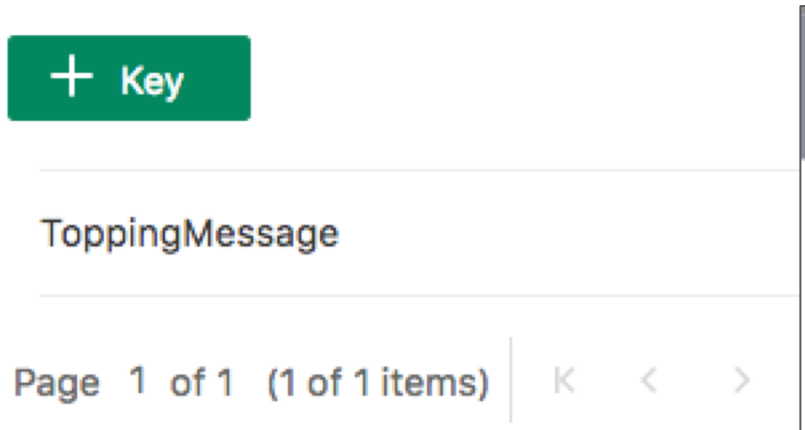
Resource string with one parameter

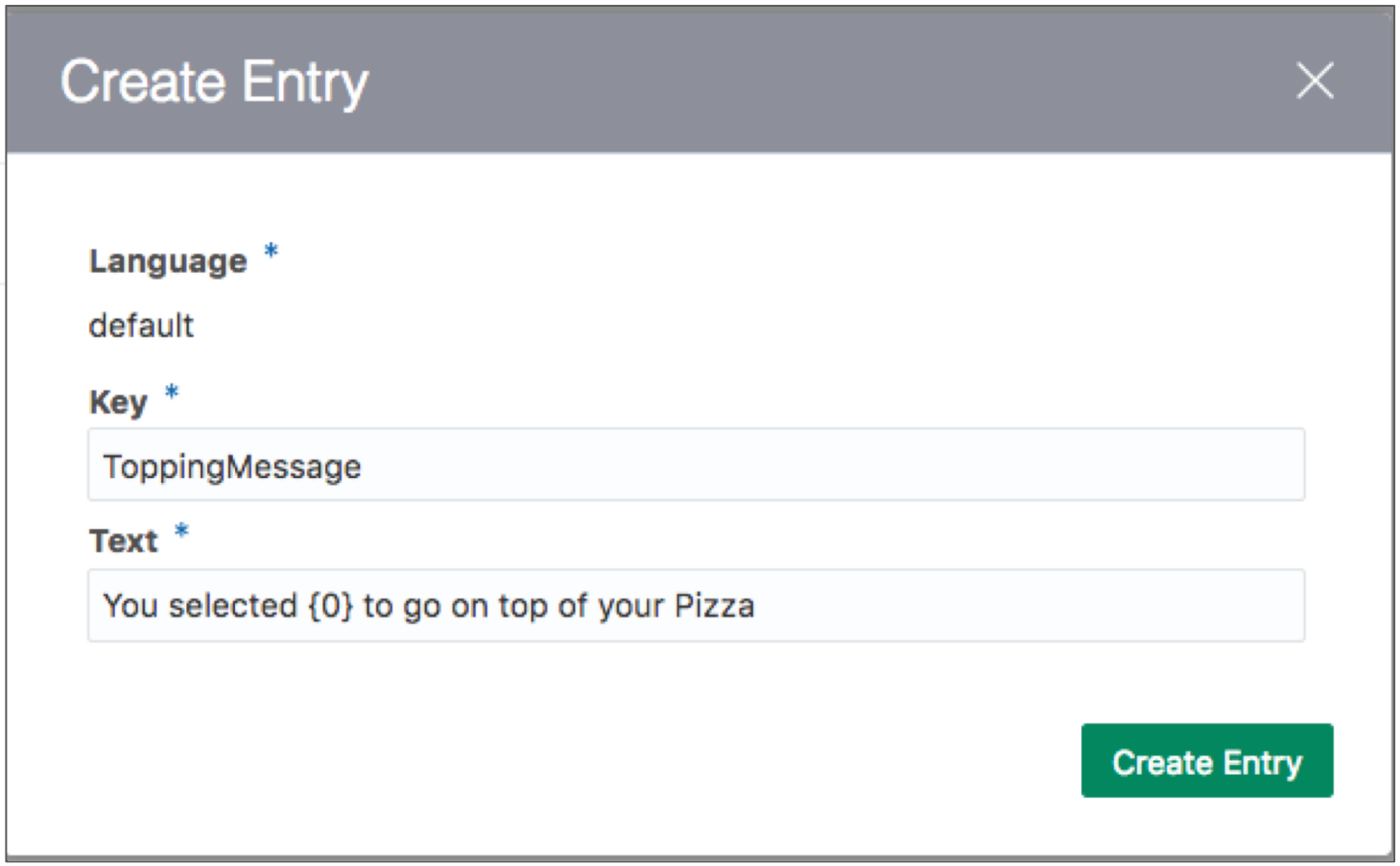
${rb('message_key','optional_parameter')}

Resource string with many parameters

${rb('message_key','optional_parameter',' ...',' ...')}

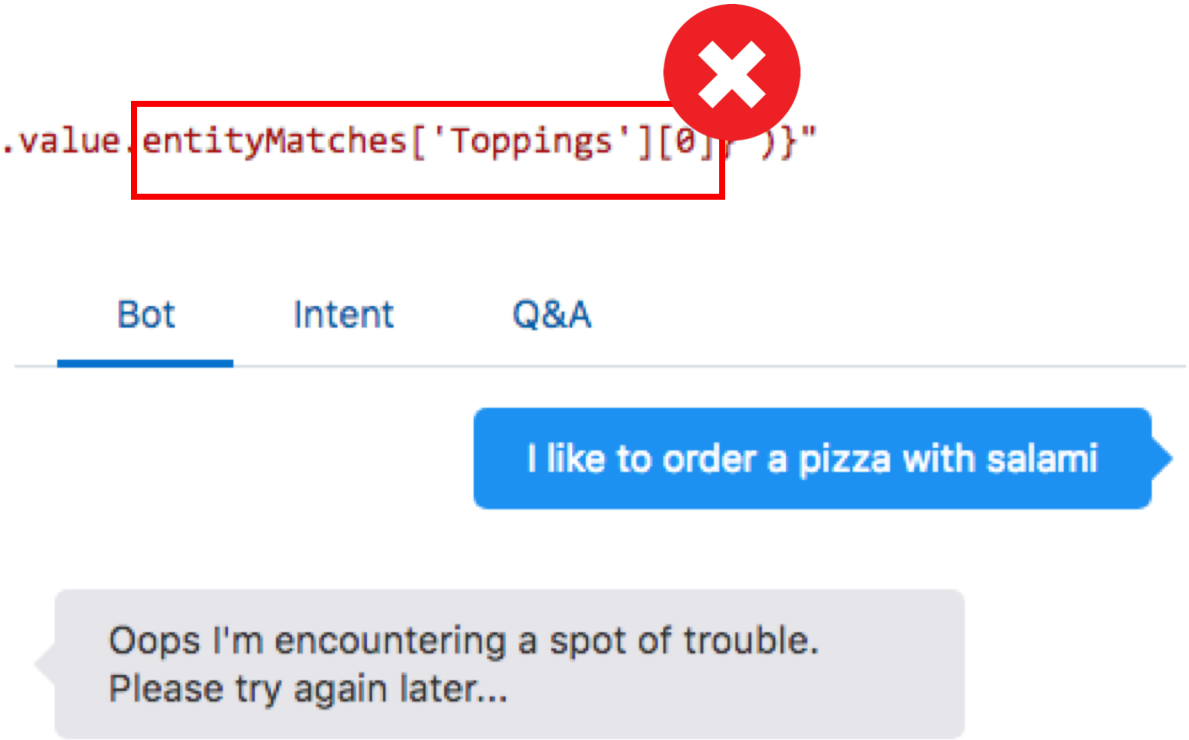
Referencing Entities in a Resource Bundle call





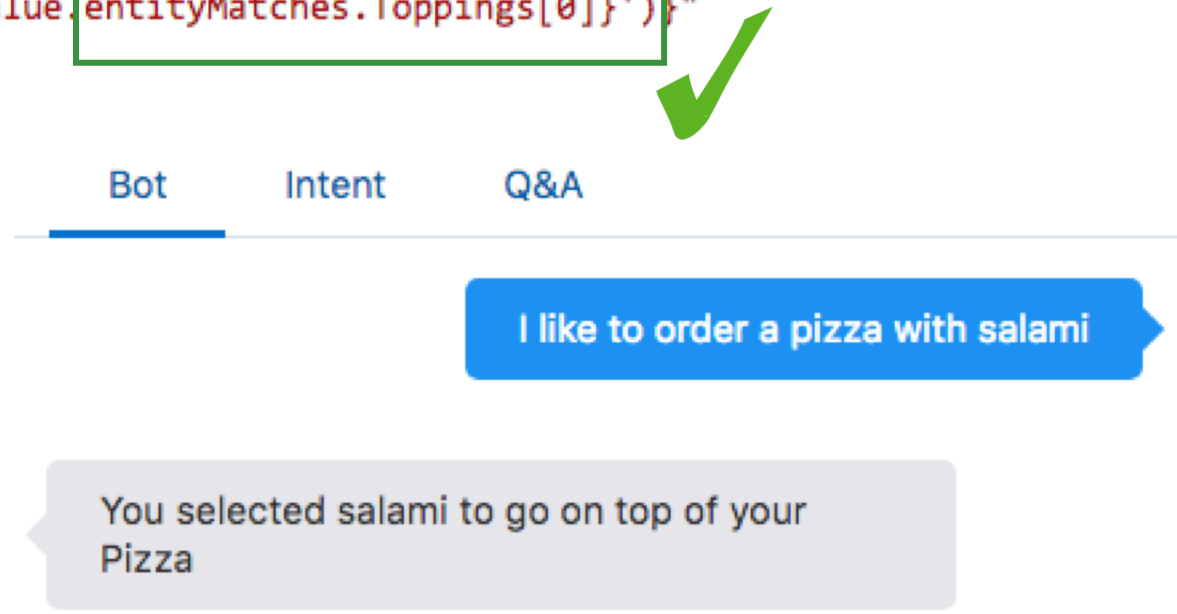
Referencing Entities in a Resource Bundle call

```
startOrder:
  component: "System.Output"
  properties:
    text: "${rb('ToppingMessage','${iResult.value.entityMatches['Toppings'][0].value}')}"
  transitions:
    return: "done"
```



Referencing Entities in a Resource Bundle call

```
startOrder:
  component: "System.Output"
  properties:
    text: "${rb('ToppingMessage','${iResult.value.entityMatches.Toppings[0]}')}"
  transitions:
    return: "done"
```



Discussion

What other solution would have worked?

Multi Language Support

Translation Service & Resource Bundle Summary

Translation Service and Resource Bundle Summary

Component / Feature	Description	Reads language settings from	Requires translation service
System.DetectLanguage	Detects language from user input. Sets profile.languageTag	NA	Yes
System.TranslateInput	Translates user messages to English	NA	Yes
System.TranslateOutput	Translates output messages to user language	profile.languageTag	Yes
Resource Bundle	Displays pre-defined language strings	profile.languageTag profile.locale (second)	No
Component "translate" property	Overrides global translation settings for a component	profile.languageTag profile.locale (second)	Yes

Multi Language Support

Custom Components

Custom components don't share the translation service and resource bundles configured for a bot

Options to Return a Response from Custom Component

- Save **data in a context variable**
  - Reference variable in an output component or from a label or prompt
- **Send responses directly** to the user
  - Does not require any help from system components

Example: Translating Data Saved in a Context Variable

Custom Component Code

```
…
let product = {product: "an apple", type: "fruit", origin: "Spain"} ;
conversation.variable('data_variable', product );
conversation.transition();
done();
```

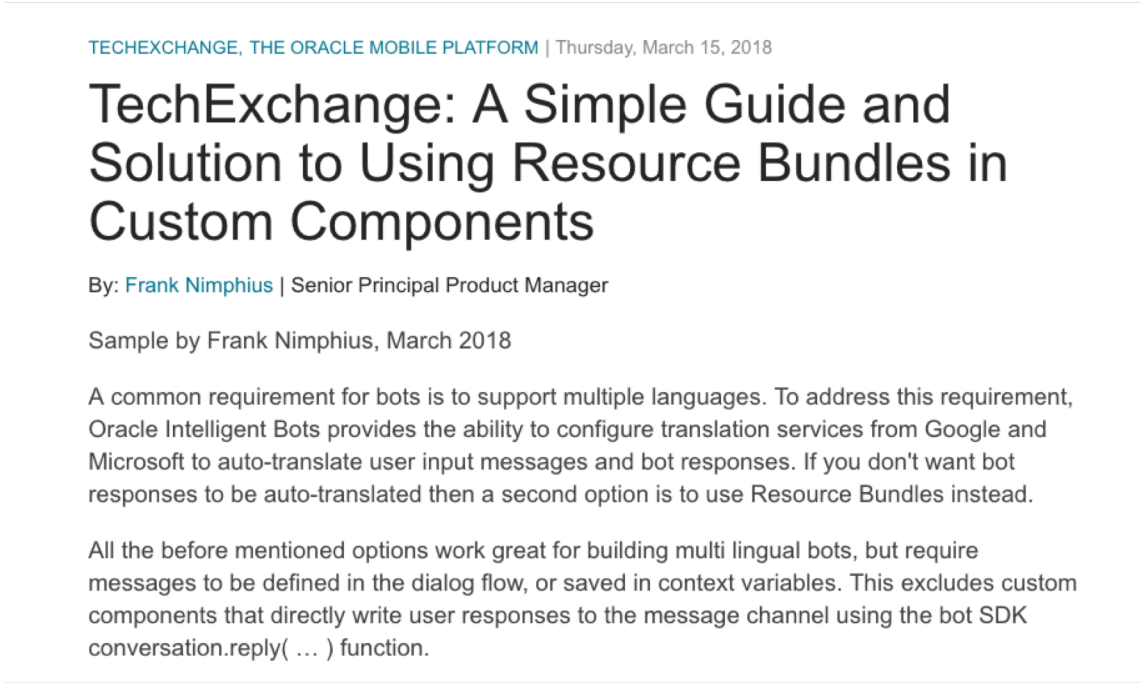
BotML

```
printProduct:
  component: "System.Output"
  properties:
    text: "The product in your cart is a ${data_variable.value.type}. It is
           ${data_variable.value.product} from ${data_variable.value.origin"
    translate: true
```

Translating Custom Component Direct Messages Responses

- Messages a custom component returns using the SDK conversation.reply() function are directly sent to a user
  - Bot stays out of the loop and thus cannot help with translation
- Messages need to be translated by the custom component
- Two Options
  - Reference a translation service before sending the message
  - Create a custom resource bundle implementation and define translations as part of the custom component

Resource Bundle Sample on Oracle TechExchange Blog



TECHEXCHANGE, THE ORACLE MOBILE PLATFORM | Thursday, March 15, 2018

TechExchange: A Simple Guide and Solution to Using Resource Bundles in Custom Components

By: Frank Nimphius | Senior Principal Product Manager

Sample by Frank Nimphius, March 2018

A common requirement for bots is to support multiple languages. To address this requirement, Oracle Intelligent Bots provides the ability to configure translation services from Google and Microsoft to auto-translate user input messages and bot responses. If you don't want bot responses to be auto-translated then a second option is to use Resource Bundles instead.

All the before mentioned options work great for building multi lingual bots, but require messages to be defined in the dialog flow, or saved in context variables. This excludes custom components that directly write user responses to the message channel using the bot SDK conversation.reply(…) function.

- JS files contain message keys and translation strings
  - One file per language
  - Messages can be parameterized
- Custom component calls resource bundle with message key and language code
  - Language code passed as input parameter from bot

Multi Language Support

Strategies & Practices

Use Resource Bundles for Prompts and Responses

- Ensures the correct business wording and tone
- Doesn't require a translation service
- Allows add content that should not be translated
  - Parameter values are not translated
  - Summer as a time of season can be translated, Summer as a female name in the US certainly not
- Set component "translate" property to false if auto translation is enabled for a bot (opt-out)

Ensure Good Entity Recognition

- back-and-forth test the translation service
  - Translate an English string into a foreign language and the translated string back into English
  - Create synonyms for entity values where the translation service deviates from the original
- Consider "blind testing" testing
  - Bot developers knows about the utterances
  - Good testing aims for bots to fail, not to succeed
- Avoid use of abbreviations or slang even if understood in a region
  - E.g. use "checking account" instead of "checking"
- Guide users
  - Use value lists whenever possible

Consider Limitations of Language Detection

- System.DetectLanguage uses the translation service to detect the user language from a sample sentence
- Be aware or closely related languages.

  "Good morning my friend"

  - **Swedish**: “God morgon min vän!”
  - **Danish**: “God morgen min ven!”
  - **Norwegian**: ”God morgen min venn!”

Control the Languages to Support by the Bot

- Using a translation service your bot probably understands more languages than you need
- Limit the languages to support to those you regularly test and that you have resource bundles for
- To limit the set of languages
  - Detect a user language and compare it to a list of supported languages
  - Don't detect the user language at all but have the user selecting a preferred language

Because **bots can detect the user language** doesn't mean they have to.

Image courtesy of Farbentek at FreeDigitalPhotos.net

Advanced Bot Training Hands-On

Lab 6: Implementing Multi Language Support

Integrated Cloud

Applications & Platform Services


ORACLE®