

ORACLE®

Oracle Digital Assistant

The Complete 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 ➤ Multi language approach
- 2 ➤ Configuration
- 3 ➤ Language components
- 4 ➤ Profile variables
- 5 ➤ Translate property
- 6 ➤ Resource bundles
- 7 ➤ Custom components
- 8 ➤ Best practices

Topic agenda

1 Multi language approach

2 Configuration

3 Language components

4 Profile variables

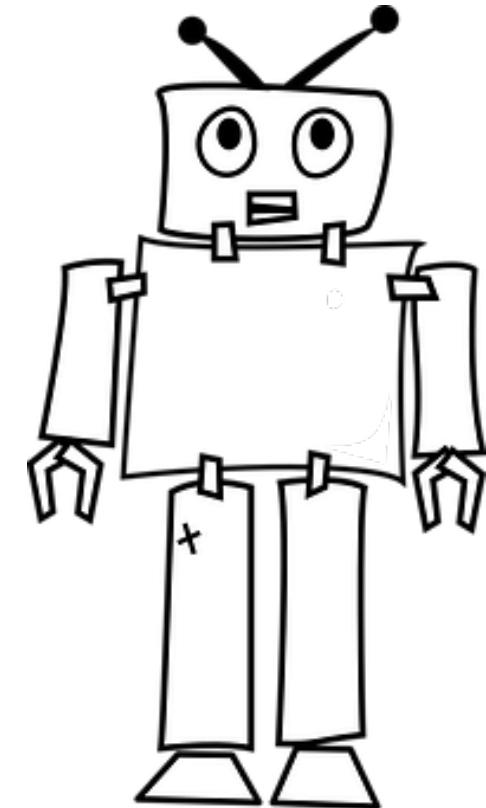
5 Translate property

6 Resource bundles

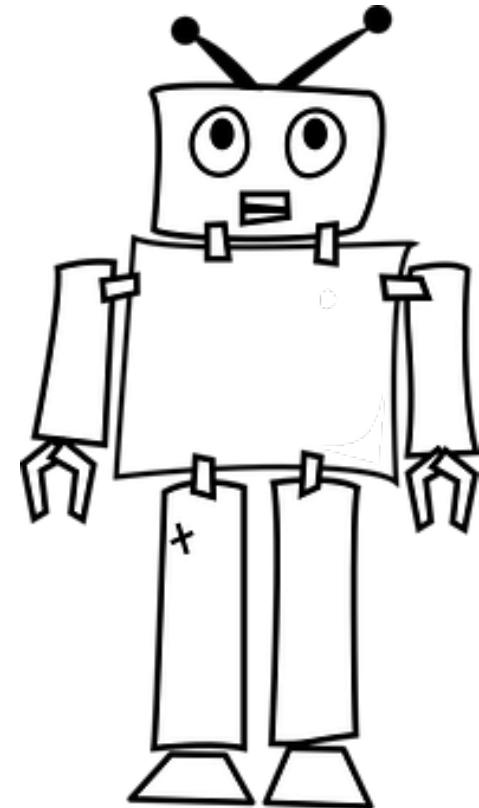
7 custom components

8 Best practices

What do you expect from a **multi
language** chatbot?



Ok, so what might our options be for building multi-language bots?



Two approaches to building multi-language bots

Native language bots

- Build the bot in the language it will be used in
- Platform has to support NLP for that (and every) language
 - Utterances
 - Entities
 - Prompts, titles, descriptions
- What happen for other languages?

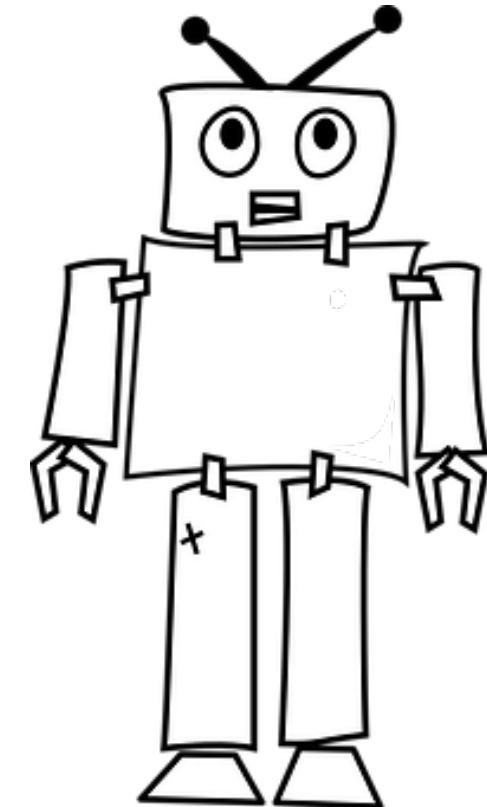
Single base-language bots

- Serve multiple languages from a single base language
- Uses translation service
 - Prompts are translated at runtime to detected user language
 - User input translated to base language
- Only one NLP engine required
- Only "appears" as native language

Why single base-language bots work

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

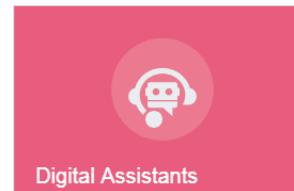
- Yoda, Star Wars



Multi language support in Oracle Digital Assistant

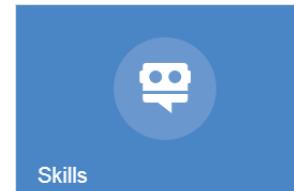
Digital assistant

- Oracle Digital Assistant supports English (19.1.5)
 - Intents, routing
- "Predominant" language allows developing single language non – English Digital Assistant

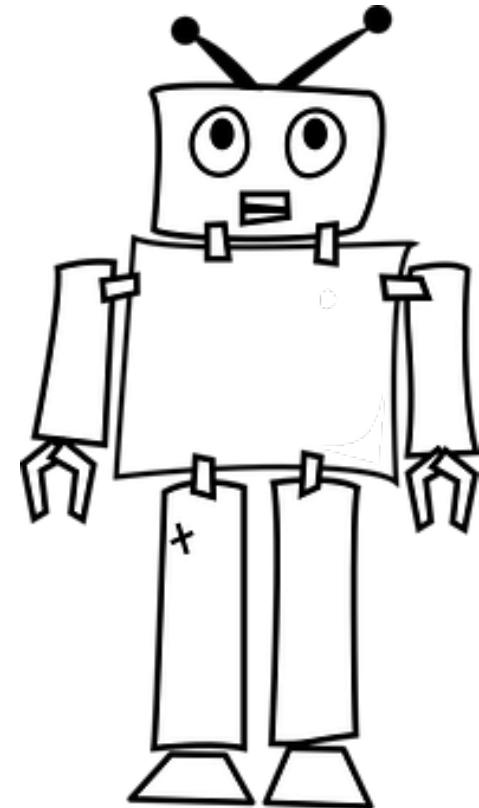


Skills

- True multi-language support
 - You need to use skill as stand alone bots
 - Expose skills directly on a messaging channel
 - Uses translation service (Microsoft, Google) to translate messages



**From here, this session will focus on
building multi language skills.**



Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

Setting up a translation service

- Use Google Translation API or Microsoft Translator Services
- Bring your own license
 - Authorization Token / Key
- Translation service is used for input and output messages at runtime
- Open kill Settings
- Select *Translation Service* in "General" tab

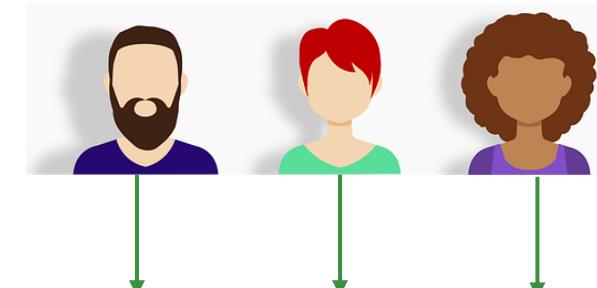
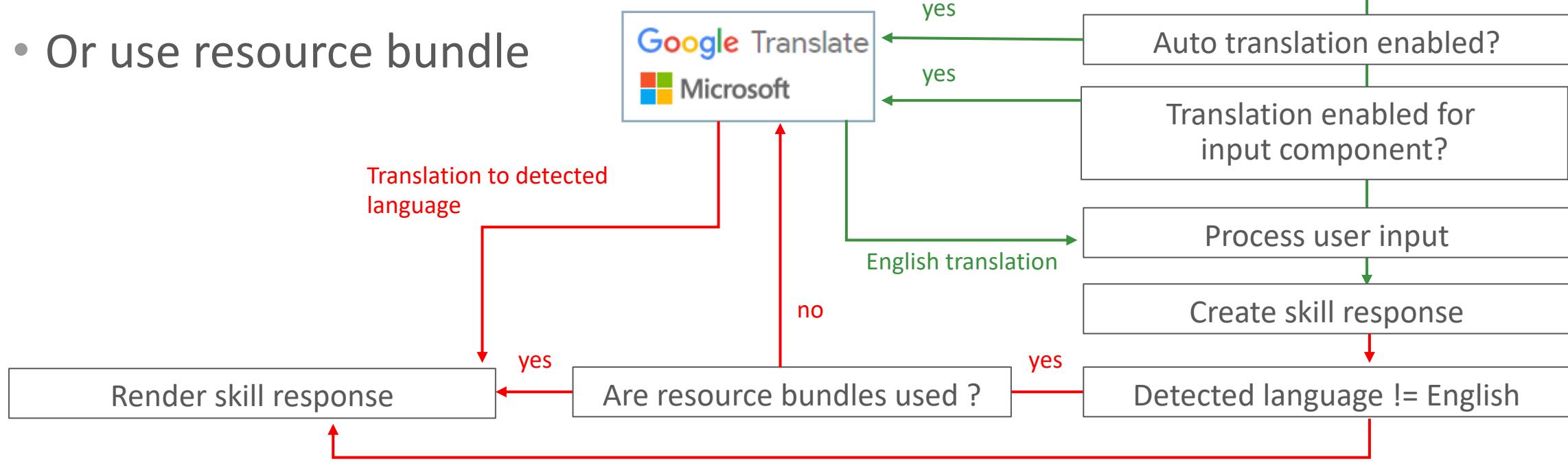
The screenshot shows the Oracle Kill interface. On the left, there's a sidebar with icons for Home, Development, Analytics, Settings (which is selected), Authentication Services, Translation Services (highlighted with a red box), Data Management, Downloads..., and Documentation... A secondary window titled "New Translation Service" is open, prompting for "Service Type" (Google), "Base URL" (URL from Google), and "Authorization Token" (token from Google). Below these, there's a section for "Optional HTTP Headers". At the bottom right of the secondary window is a "Create" button. In the main pane, a "General" tab is selected, showing fields for "Display Name" (adtv24hrsflowers_ReferenceBot), "Name" (adtv24hrsflowers_ReferenceBot), "Version" (1.0), "Category" (Category your Skill bot falls under), "One-Sentence Description" (Advanced Training - 24 hours flowers star), "Detailed Description" (This description appears on the skill's De), "Training Model" (Trainer Ht), "Translation Service" (Google, highlighted with a blue box), and "Enable Insights" (toggle switch). A red box highlights the "Translation Service" dropdown. On the far left of the main pane, there's a vertical toolbar with icons for General, Configuration, Digital Assistant, Events, and Q&A Routing Config.

Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

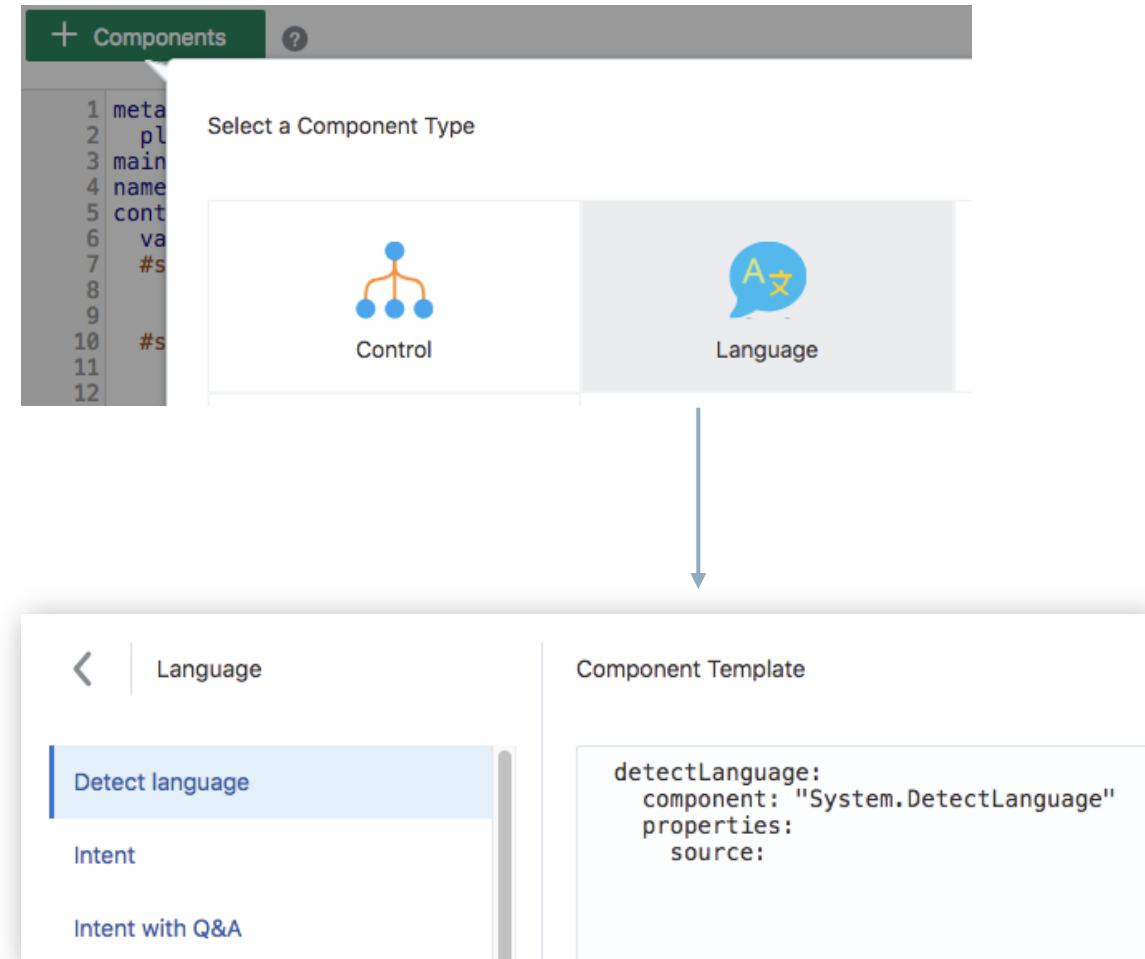
The workflow of language translation

- Detect the user's language
- Store detected language in profile variable
- Translate using translation service
- Or use resource bundle



System.DetectLanguage

- Detects language from user input
 - Uses translation service
 - Detects language from user message
 - Optional 'source' property can be used to read user message from variable
- Sets `profile.languageTag` variable
 - User language saved as 2 character code
 - E.g. "fr", but not "fr-ca"



Enabling / disabling auto-translation

- Enabled / disabled auto-translation
 - Define "autoTranslate" context variable of type boolean
 - Set variable value to true to enable auto-translation
- When enabled
 - User messages are translated to English
 - skill messages are translated to user language

variables:

autoTranslate: "boolean"

...

states:

...

enableAutotranslation:

component: "System.SetVariable"

properties:

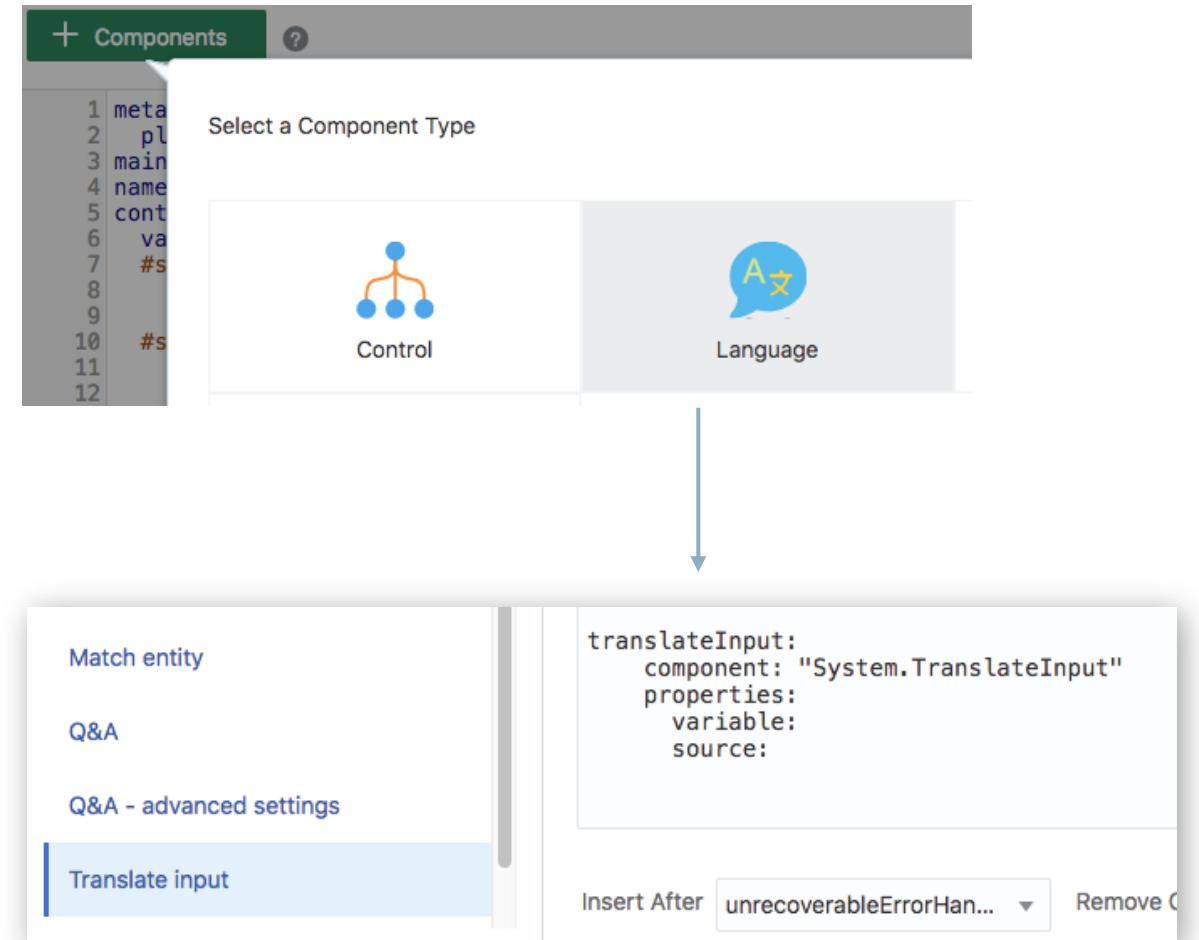
variable: "**autoTranslate**"

value: true

transitions: {}

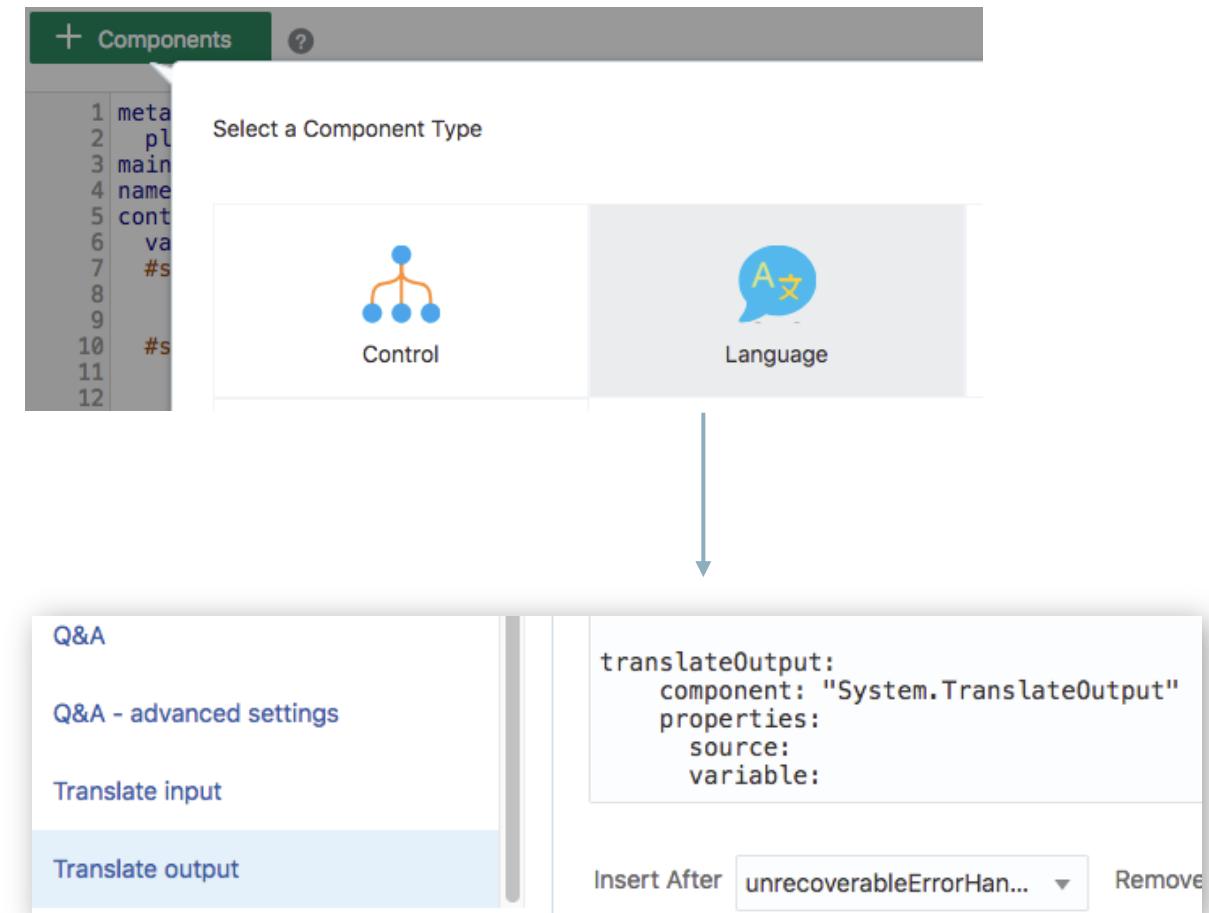
System.TranslateInput

- Translates user message from detected language to English
 - Uses translation service
- Translates user entered messages
 - Optional '*source*' property used to reference variable holding string to translate
- '*variable*' property references dialog flow variable to store the translated string



System.TranslateOutput

- Translates English strings to detected user language
- Uses translation service
- '*source*' property references variable holding the English string to translate
- variable referenced in component '*variable*' property gets updated with translated string



Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

Language profile variables

- `profile.locale`
 - Set by the messenger client based on user setting
 - `${profile.locale}`
- `profile.languageTag`
 - Holds language detected at runtime
 - Set by `System.DetectLanguage` component
 - Can be set manually using `System.SetVariable`
 - Precedes `profile.locale` setting
 - `${profile.languageTag}`
- Determine the language used by QnA

Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

Component 'translate' property

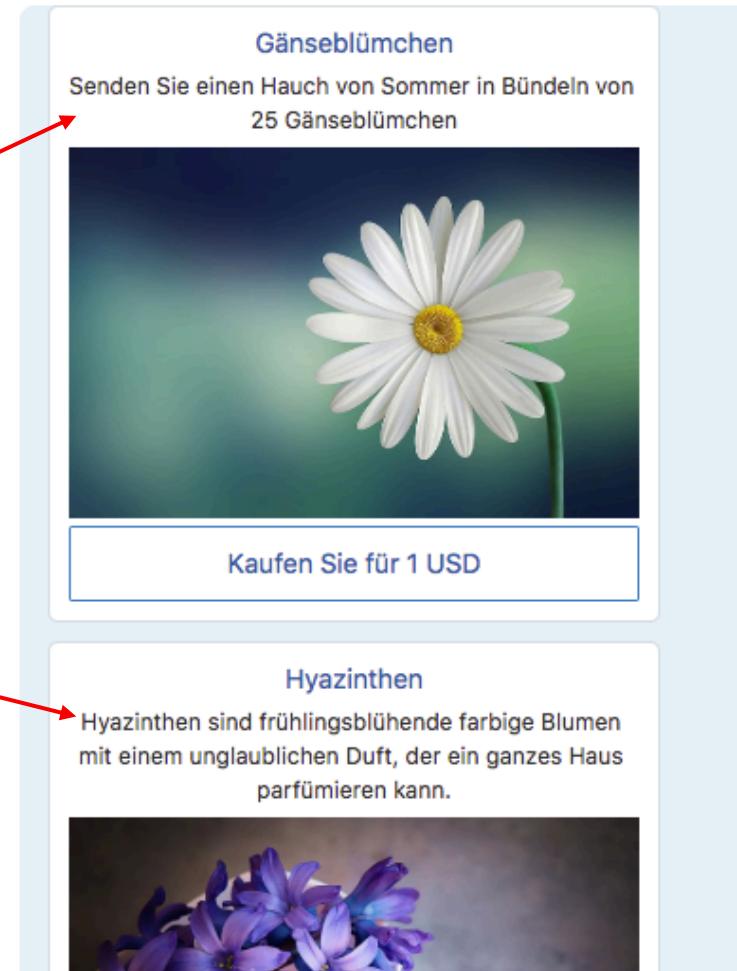
- Boolean property
 - Enables / disables auto-translation for components
 - Set to true
 - If auto-translation is not enabled but component should use translation service
 - If component input is not expected to be English and auto-translation is disabled
 - Always on System.Intent components to be able to resolve intents from non-English language
 - Set to false for components that use resource bundles
- Requires System.DetectLanguage to be used early in dialog flow

Ich möchte Blumen kaufen

'translate' property in action

```
showFlowersMenu:  
  component: "System.CommonResponse"  
  properties:  
    variable: "flowersName"  
    nlpResultVariable: "iResult"  
    processUserMessage: true  
    translate: true  
    metadata:  
      responseItems:  
        - type: "text"  
          text: "${rb.orderFlowersMenuPrompt}"  
        - type: "cards"  
          cardLayout: "vertical"  
          cards:  
            - title: "${menu.title}"  
              description: "${menu.description}"  
              imageUrl: "${advtImagesHost.value}${menu.image}"  
              iteratorVariable: "menu"  
              rangeStart: "${orderMenuRangeIndex.value}"  
              rangeSize: "${orderMenuRangeSize.value}"  
            actions:  
              - label: "${rb.orderBuyFor} ${menu.price} USD"  
                type: "postback"  
                payload:  
                  action: "copyValueAction"  
                  variables:  
                    flowersName: "${menu.title}"  
                    flowerCost: "${menu.price}"  
      ...
```

Bitte wählen Sie eine Option aus dem
Menu



Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

Use resource bundle for prompts and skill messages

- Ensures appropriate language and tone presented to user
- Doesn't require a translation service
- Set component '*translate*' property to false if auto translation is enabled for a skill (opt-out)
 - Property can be set dynamically at runtime

```
getUserIntent:  
  component: "System.Intent"  
  properties:  
    variable: "iResult"  
    qnaSkipIfIntentFound: true  
    qnaEnable: false  
  optionsPrompt: "${rb.IntentSelectListPrompt}"  
  optionsQnaLabel: "${rb.mainIntentOptionsQnALabel}"  
  translate: "${useTranslationService.value}"  
  
transitions:  
  next: "showMenu"  
actions:  
  OrderFlowers: "startOrderFlowers"  
  RequestAgentSupport: "startHumanAgent"  
  TrackOrders: "startTrackOrders"  
  OpenFranchise: "startOpenFranchise"  
  FileComplaint: "startFileComplaint"  
  Welcome: "startWelcome"  
  unresolvedIntent: "resetiResult"  
  qna: "qna"
```

Creating resource bundles

The screenshot shows a user interface for creating a resource bundle entry. On the left, there is a vertical sidebar with seven icons: a person, a gear, a bar chart, a document with text, a question mark, and a checkmark. The main area has a title "Create Entry" with a close button "X". A message on the left says "You haven't defined an" and "You can localize your sl". The form fields are as follows:

- Language ***: default
- Key ***: welcome_msg
- Text ***: Good to see you. {0}, what can I do for you today?

A green "Create Entry" button is at the bottom right.

Creating a translation

- Select "+ Language"
- Add or select a two letter language code
 - "de", "fr", "es" etc.
- Select a key
- Provide a translation string for the English message string

View strings by key or by language

The screenshot shows a localization management interface. On the left, there's a vertical toolbar with icons for different operations like creating new entries, managing keys, and settings. The main area has a table listing a single entry:

Key	Language	Message
welcome_msg	default	Good to see you. {0}, what can I do for you today?

A modal window titled "Create Entry" is open in the foreground, prompting for new data:

Key *: welcome_msg
Language *: de
Text *: Schön, daß Du wieder da bist. {0}, was kann ich für Dich tun?

At the top right of the main interface, there's a dropdown menu set to "View By Key". A red arrow points from the text "View strings by key or by language" to this dropdown.

Defining variables in a resource string

The screenshot shows a software application interface for managing resource strings. On the left, there's a sidebar with various icons. The main area displays a list of keys, with 'AskBalancesAccountPrc' being the current item selected. A modal window titled 'Create Entry' is open, prompting for a 'Language *' (set to 'default'), a 'Key *' (set to 'DisputeResponse'), and a 'Text *' (containing the placeholder 'Successfully filed dispute, your reference number is {0} and reason is {1}').

Key *

AskBalancesAccountPrompt

View By

+ Key

AskBalancesAccountPrc

AskFromAccountType

AskPaymentAmount

AskToAccount

AskTxnsAccountType

AskTxnsType

Page 1 of 1 | K <

Create Entry

Language *

default

Key *

DisputeResponse

Text *

Successfully filed dispute, your reference number is {0} and reason is {1}

Create Entry

is {0} and reason is {1}

Using resource bundles for a skills

- Add variable of type "resourcebundle"
- Access resource string with no parameters
 - \${rb('message_key')} or \${rb.message_key}
- Access resource string with single parameter
 - \${rb('message_key','optional_parameter')}
- Access resource string with multiple parameter
 - \${rb('message_key','optional_parameter','...','...')}

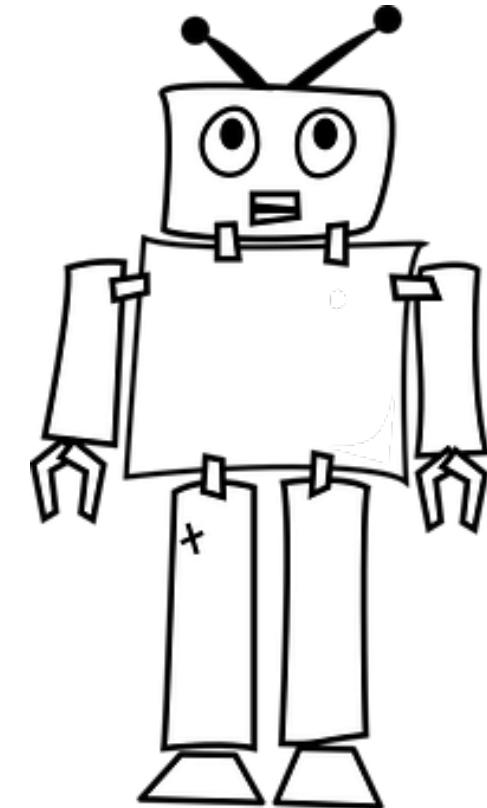
```
context:  
variables:  
rb: "resourcebundle"  
  
getUserIntent:  
component: "System.Intent"  
properties:  
variable: "iResult"  
qnaSkipIfIntentFound: true  
qnaEnable: false  
optionsPrompt: "${rb.IntentSelectListPrompt}"  
optionsQnaLabel: "${rb.mainIntentOptionsQnALabel}"  
translate: "${useTranslationService.value}"  
transitions:  
next: "showMenu"  
actions:  
OrderFlowers: "startOrderFlowers"  
RequestAgentSupport: "startHumanAgent"  
TrackOrders: "startTrackOrders"  
OpenFranchise: "startOpenFranchise"  
FileComplaint: "startFileComplaint"  
Welcome: "startWelcome"  
unresolvedIntent: "resetiResult"  
qna: "qna"
```

```
confirmOrderAndQuantity:  
component: "System.Output"  
properties:  
text: "${rb('orderConfirmOrderQuantity','${orderQuantity.value.number}', '\"${orderProductName.value}\"')}"  
keepTurn: true  
transitions:  
next: "askDeliveryOption"
```

Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

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



Options to return a custom component message response

- Custom component saves data in a dialog flow variable
 - Variable referenced from output component (e.g. System.CommonResponse)
 - Component's translation setting determines whether saved data gets auto-translated
- Custom component sends response directly to the messenger
 - Does not require system components
 - skill stays out of the loop and thus cannot help with translations
 - Translation must be part of the custom component design

Example: translating data saved in a dialog flow 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
```

Topic agenda

- 1 Multi language approach
- 2 Configuration
- 3 Language components
- 4 Profile variables
- 5 Translate property
- 6 Resource bundles
- 7 Custom components
- 8 Best practices

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

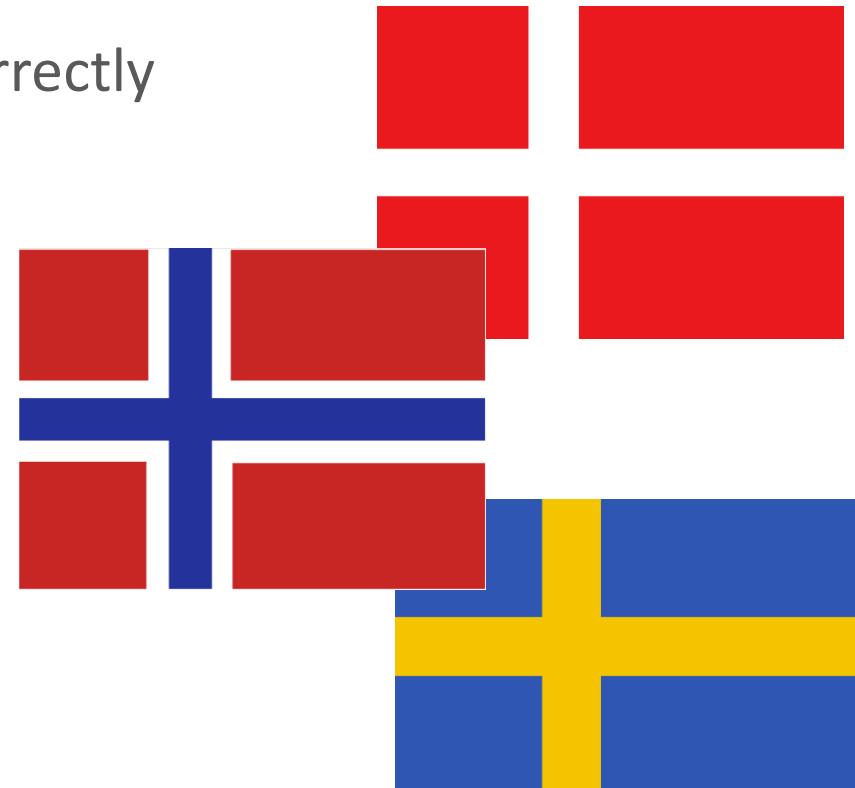
- Enable auto-translation
- Detect user languages
 - From user message
 - From profile
- Test skill
- Disable translation on individual components and use message bundles instead

Ensure good entity recognition

- Back-and-forth test the translation service
 - Translate an English string into a foreign language and then translate it back to English
 - Use synonyms in entities where the translation service deviates from the original
- Consider "blind testing" testing
 - skill developers know about the utterances
 - Good testing aims for skills 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

- Be aware of closely related languages
 - Translation service may fail to detect language correctly
- For example: "Good morning my friend"
 - Swedish: "God morgen min vän!"
 - Danish: "God morgen min ven!"
 - Norwegian: "God morgen min venn!"
- Ask user if in doubt



Control the languages to support

- Using a translation service your skill probably understands more languages than you need
- It does not make sense to support languages you don't speak or for which you have no expertise in house
- 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 but have the user selecting a preferred language

Integrated Cloud Applications & Platform Services

ORACLE®



Oracle Digital Assistant Hands-On

TBD