

ORACLE®

Oracle Digital Assistant

The Complete Training

Using the Common Response component with Composite Bag Entities

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 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

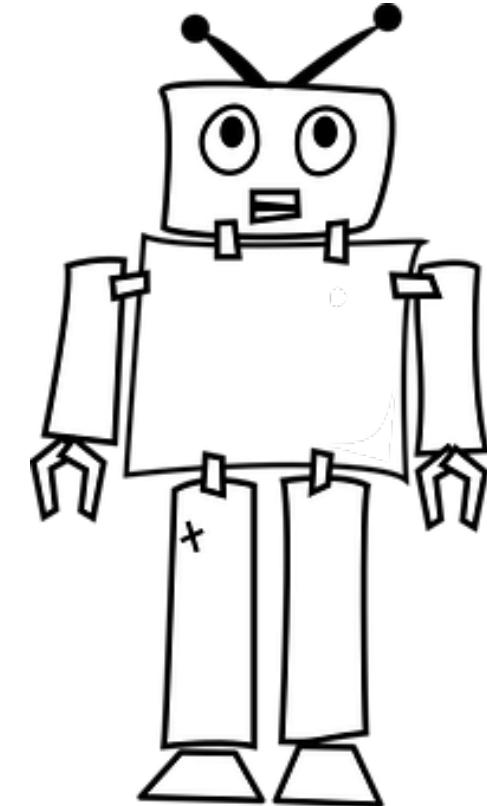
Topic agenda

- 1 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

The business behind the problem

- Intents and entities are technical terms that don't speak to business users
- Chatbots solve business problems, not technical problems
- Business understands business related objects and abstraction
 - Order, customer, meeting, product
- Composite bag entities model business objects
 - Business domain object
 - Items are resolved together
 - No need to understand dialog flow
 - Declarative configuration

Composite bag entities not only **simplify conversational design** for a skill, they also **simplify communication** between IT and business



Topic agenda

- 1 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

Two system components, one goal

- Simplification
- System.ResolveEntities and System.CommonResponse
 - Generate user interfaces for composite entity bag items
 - Resolve composite bag entities without dialog flows
 - Remove complexity from the dialog flow design
- Composite bag is the single point of truth for
 - Validation
 - error handling
 - prompts
 - message extraction

System.ResolveEntities vs. System.CommonResponse

ResolveEntities

- Resolves all entities effortlessly
- Renders lists and input text
- Allows navigation to dialog flow state after entity gets resolved
- Supports pagination
- Supports limiting the number of failed user value input attempts

CommonResponse

- Verbose OBotML
 - Requires BotML, and Apache FreeMarker skills
- Feature parity with ResolveEntities
- Allows UI customization
 - Rich UI
- Supports channel specific UI rendering

Topic agenda

- 1 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

Composite bag at runtime

Default UI using System.ResolveEntities

Configuration

Type ?

Composite Bag

Bag Items

+ Bag Item

Name	Type	Entity Name
PizzaSize	ENTITY	PizzaSize
PizzaTopping	ENTITY	PizzaTopping
DeliveryAddress	ENTITY	ADDRESS
DeliveryTime	ENTITY	TIME
PizzaDough	ENTITY	PizzaDough

What size of pizza would you like

Small

Medium

Large



What kind of pizza would you like

Cheese

Veggie

Pepperoni



Show More

Cheese

What address can we send that to?



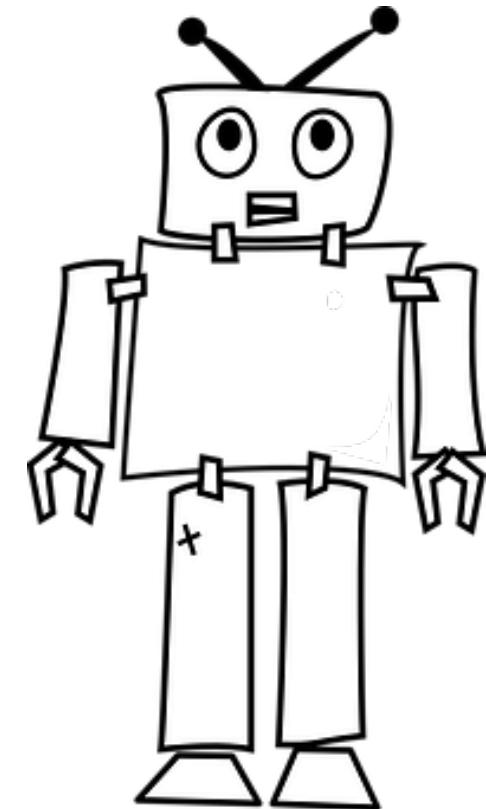
Common response component template

The screenshot shows the Oracle Dialog Flow interface. In the top left, there's a code editor window with some JSON-like metadata. Below it, a sidebar lists components: Control, Language, Variables, and User Interface. The User Interface section is expanded, showing categories like Agent conversation, Agent initiation, Common response - attachment, Common response - card, and Common response - composite bag. The 'Common response - composite bag' item is highlighted with a red box. To the right, a modal window titled 'Component Template' displays the following JSON code:

```
resolveCompositeBag:  
  component: "System.CommonResponse"  
  properties:  
    # set processUserMessage to true if the dialog flow should return  
    # to this state after receiving user message  
    processUserMessage: true  
    # set keepTurn (true/false) to true if the dialog flow should  
    # transition to the next state without waiting for user input. Only  
    # applicable when processUserMessage is false  
    keepTurn: false  
    # variable should refer to a composite bag entity variable. For  
    # each entity in the bag, the user will be prompted for a value. If all  
    # entities in the bag have a value, the dialog flow transitions to the  
    # next state.  
    variable:  
      # optionalVariable (optional) is only applicable when the
```

Below the code, there are buttons for 'Insert After' (set to 'startOrderPizza'), 'Remove Comments' (with a toggle switch), and 'Apply'.

Developer has full access to the entity bag item that is processed at a time



system.entityToResolve variable

- System variable exposing information about the current entity to resolve
 - \${system.entityToResolve.value.<attribute>}
- Used in Common Response component to
 - Determine when to show list if values
 - Conditionally show/hide buttons and response items
- Displayed in skill bot conversation tester
- Commonly used attributes
 - Name, prompt, enumValues, needShowMoreButton rangeStartVar, nextRangeStartVar

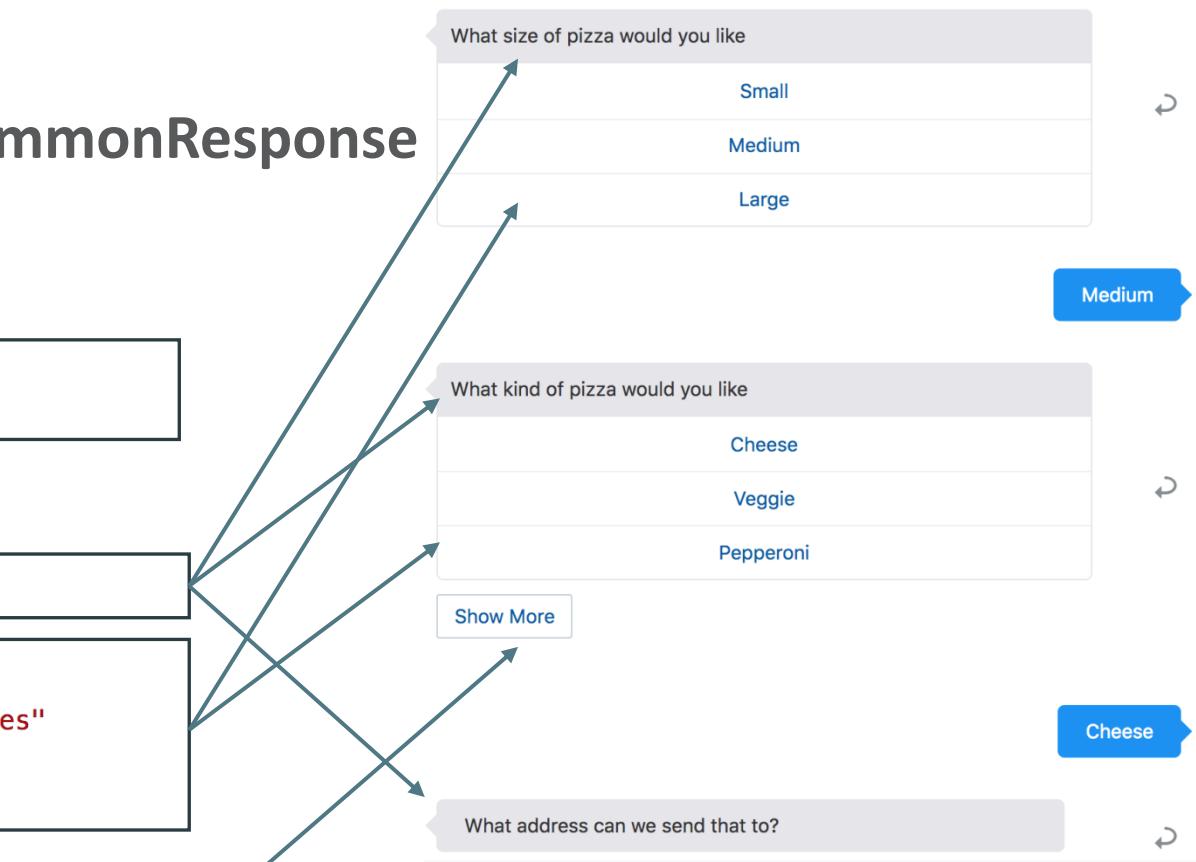
Variables [View All](#)

- ▶ pizza
- ◀ system
 - entityToResolve
 - nextRangeStart: 0
 - updatedEntities
 - needShowMoreButton: false
 - outOfOrderMatches
 - rangeStartVar: system.state.resolveEntities.PizzaToppingRangeStart
 - promptCount: 1
 - transitionedAfterMatch: false
 - name: DeliveryTime
 - validationErrors
 - allMatches
 - resolvingField: DeliveryTime
 - userInput: tomorrow 2pm
 - skippedItems
 - disambiguationValues
 - prompt: When can we deliver that for you?
 - enumValues
 - config
 - postbackActions
 - processedUserMessage: true

Composite bag at runtime

Same UI as ResolveEntities but using CommonResponse

```
resolveEntities:  
  component: "System.CommonResponse"  
  properties:  
    processUserMessage: true  
    keepTurn: false  
    variable: "pizza"  
    nlpResultVariable: "iResult"  
    cancelPolicy: "immediate"  
    transitionAfterMatch: "false"  
    maxPrompts: 3  
    metadata:  
      responseItems:  
        - type: "text"  
          text: "${system.entityToResolve.value.prompt}"  
      actions:  
        - label: "${enumValue}"  
          type: "postback"  
          iteratorVariable: "system.entityToResolve.value.enumValues"  
          payload:  
            variables:  
              pizza: "${enumValue}"  
      globalActions:  
        - label: "Show More"  
          type: "postback"  
          visible:  
            expression: "${system.entityToResolve.value.needShowMoreButton}"  
            payload:  
              variables:  
                ${system.entityToResolve.value.rangeStartVar}: "${system.entityToResolve.value.nextRangeStart}"  
  transitions:  
    ...
```

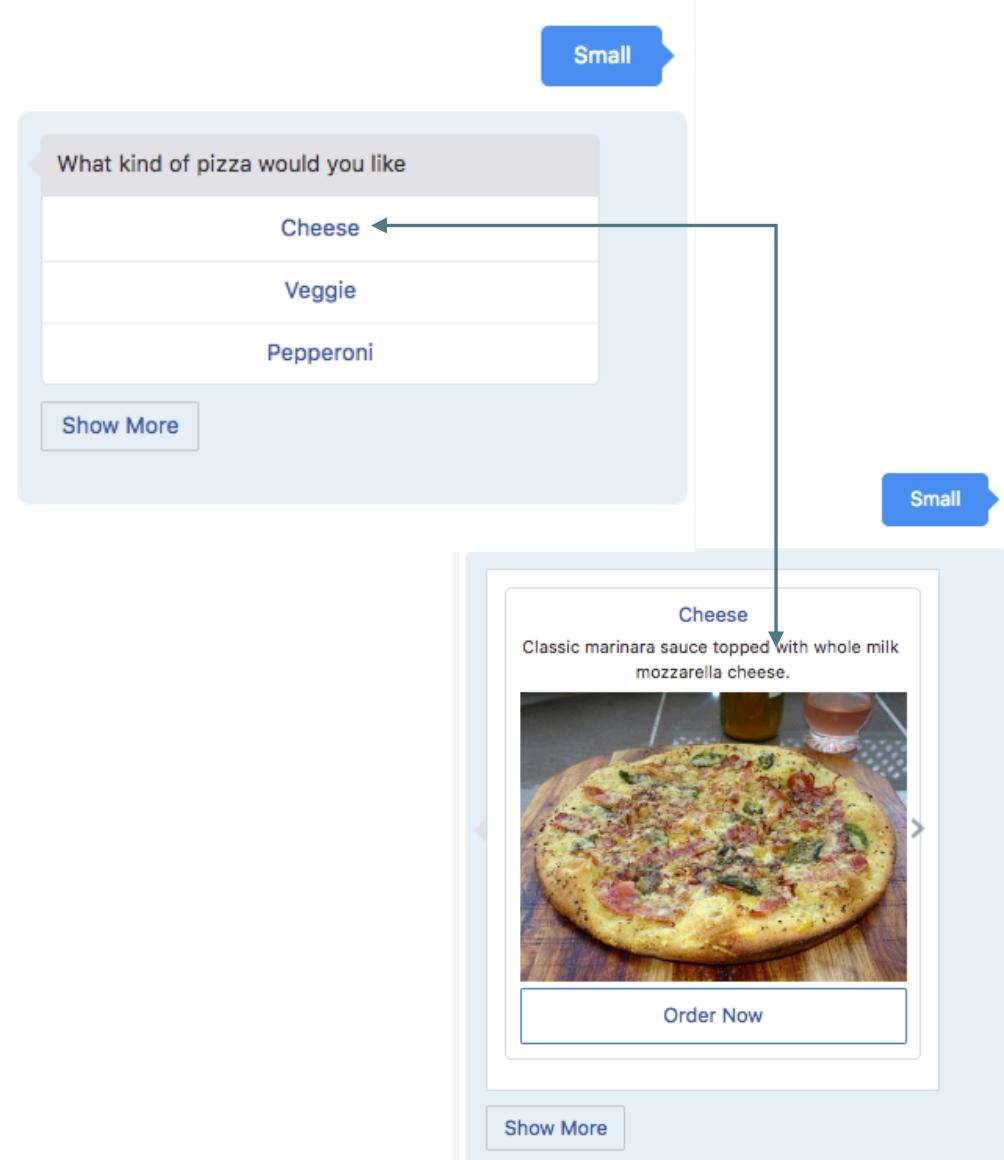


Topic agenda

- 1 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

User interface customization

- Response items can be shown or hidden dependent on entity to render
- Apache FreeMarker expressions can be used to reference variables and resource bundles



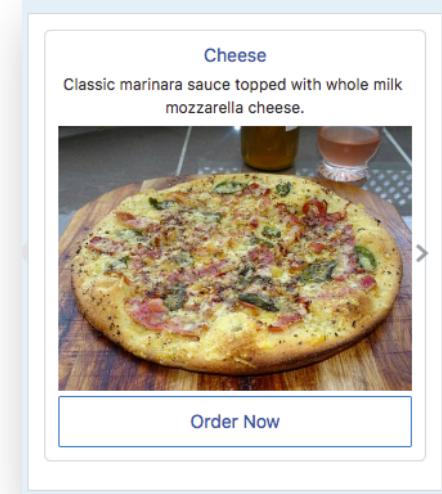
Example: building content in context variable

- You can add JSON objects to context variables
 - Apache FreeMarker
 - Custom component
- pizzaCardInfo has pizza names defined as object attributes
- Attributes are accessible from expression
 - \${pizzaCardInfo.value[pizza_name]}

```
loadPizzaCardInfo:  
component: "System.SetVariable"  
properties:  
variable: "pizzaCardInfo"  
value:  
  CHEESE:  
    description: "Classic marinara sauce topped with whole milk mozzarella cheese."  
    image: "https://cdn.pixabay.com/photo/2017/09/03/10/35/pizza-2709845_340.jpg"  
  PEPPERONI:  
    description: "Classic marinara sauce with authentic old-world style pepperoni."  
    image: "https://cdn.pixabay.com/photo/2017/08/02/12/38/pepperoni-2571392_340.jpg"  
  MEAT:  
    description: "Classic marinara sauce, authentic old-world pepperoni, all-natural\\  
      \\ Italian sausage, slow-roasted ham, hardwood smoked bacon, seasoned pork\\  
      \\ and beef."  
    image: "https://cdn.pixabay.com/photo/2017/07/22/22/51/big-2530144_340.jpg"  
  SUPREME:  
    description: "Classic marinara sauce, authentic old-world pepperoni, seasoned\\  
      \\ pork, beef, fresh mushrooms, fresh green bell peppers and fresh red onions."  
    image: "https://cdn.pixabay.com/photo/2017/07/22/22/57/pizza-2530169_340.jpg"  
  VEGGIE:  
    description: "Premium crushed tomato sauce topped with green peppers, red\\  
      \\ onions, mushrooms, Roma tomatoes and roasted spinach with our Hut Favorite\\  
      \\ on the crust."  
    image: "https://cdn.pixabay.com/photo/2017/07/22/22/57/pizza-2530169_340.jpg"  
  HAWAIIAN:  
    description: "Grilled chicken, ham, pineapple and green bell peppers."  
    image: "https://cdn.pixabay.com/photo/2017/07/22/22/51/big-2530144_340.jpg"  
  BACON:  
    description: "Garlic Parmesan sauce topped with bacon, mushrooms and roasted\\  
      \\ spinach with a salted pretzel crust."  
    image: "https://cdn.pixabay.com/photo/2017/09/03/10/35/pizza-2709845_340.jpg"  
  transitions:  
  next: "intent"
```

Example: referencing external content from component

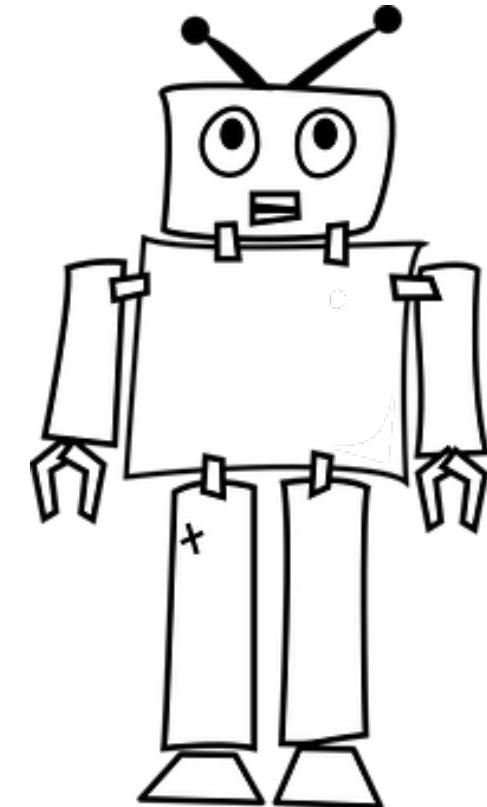
```
metadata:  
  responseItems:  
    - type: "text"  
      text: "${system.entityToResolve.value.prompt}"  
      visible:  
        entitiesToResolve:  
          exclude: "PizzaType"  
    actions:  
      - label: "${enumValue}"  
        type: "postback"  
        iteratorVariable: "system.entityToResolve.value.enumValues"  
        payload:  
          variables:  
            pizza: "${enumValue}"  
    - type: "cards"  
      cardLayout: "horizontal"  
      visible:  
        entitiesToResolve:  
          include: "PizzaType"  
    cards:  
      - title: "${enumValue}"  
        description: "<#if pizzaCardInfo.value[enumValue?upper_case]?has_content>${pizzaCardInfo.value[enumValue?upper_case].description}</#if>"  
        imageUrl: "<#if pizzaCardInfo.value[enumValue?upper_case]?has_content>${pizzaCardInfo.value[enumValue?upper_case].image}</#if>"  
        iteratorVariable: "system.entityToResolve.value.enumValues"  
      actions:  
        - label: "Order Now"  
          type: "postback"  
          payload:  
            variables:  
              pizza: "${enumValue}"
```



Topic agenda

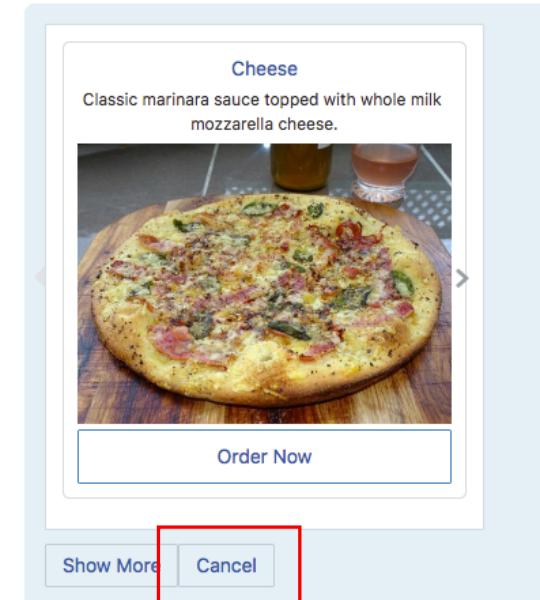
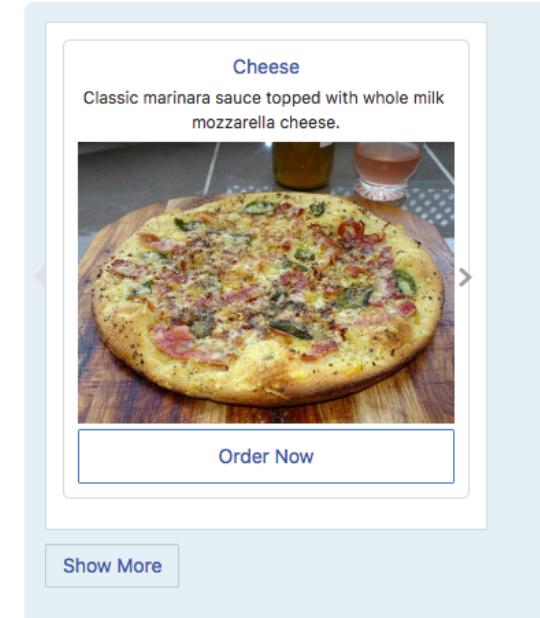
- 1 ➤ Technology doesn't speak to everyone
- 2 ➤ System.ResolveEntities vs. System.CommonResponse
- 3 ➤ Using composite bag entities
- 4 ➤ User interface customization
- 5 ➤ Providing help

**Users who do not enter valid data
may need help.**



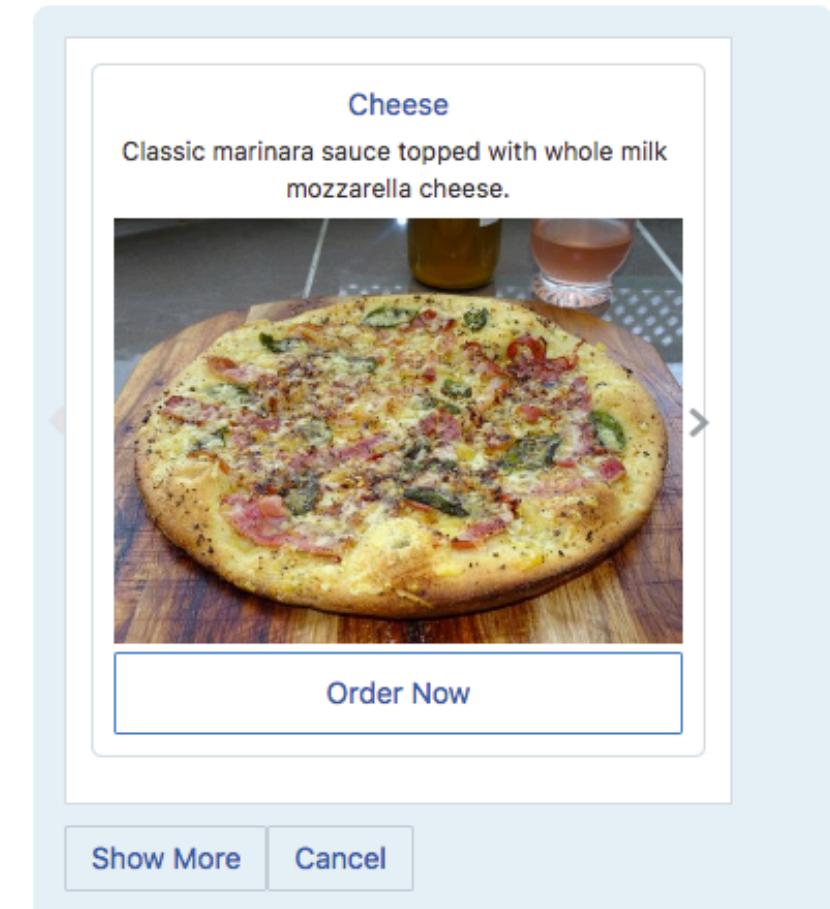
Providing help

- Composite bag entities can show alternative prompts in case of invalid data input
- CommonResponse component can also display action items
 - Cancel button
 - Help button for written help or human agent support
- No coding required!



Display cancel button after failed data input

```
cards:  
- title: "${enumValue}"  
  description: "<#if pizzaCardInfo.value[enumValue?upper_case] ? ${pizzaCardInfo.value[enumValue?upper_case]} : ${pizzaCardInfo.value[enumValue?lower_case]}>"  
  imageUrl: "<#if pizzaCardInfo.value[enumValue?upper_case] ? ${pizzaCardInfo.value[enumValue?upper_case]} : ${pizzaCardInfo.value[enumValue?lower_case]}> ${imageUrl}"  
  iteratorVariable: "system.entityToResolve.value.enumValues"  
  actions:  
    - label: "Order Now"  
      type: "postback"  
      payload:  
        variables:  
          pizza: "${enumValue}"  
  globalActions:  
    - label: "Show More"  
      type: "postback"  
      visible:  
        expression: "${system.entityToResolve.value.needShowMoreButton}"  
      payload:  
        variables:  
          ${system.entityToResolve.value.rangeStartVar}: "${system.entityToResolve.value.rangeStartVar}"  
    - label: "Cancel"  
      type: "postback"  
      visible:  
        onInvalidUserInput: true  
      payload:  
        action: "cancel"  
transitions:  
next: "setPizzaDough"  
actions:  
  cancel: "maxError"
```



Integrated Cloud Applications & Platform Services

ORACLE®



Oracle Digital Assistant Hands-On

TBD