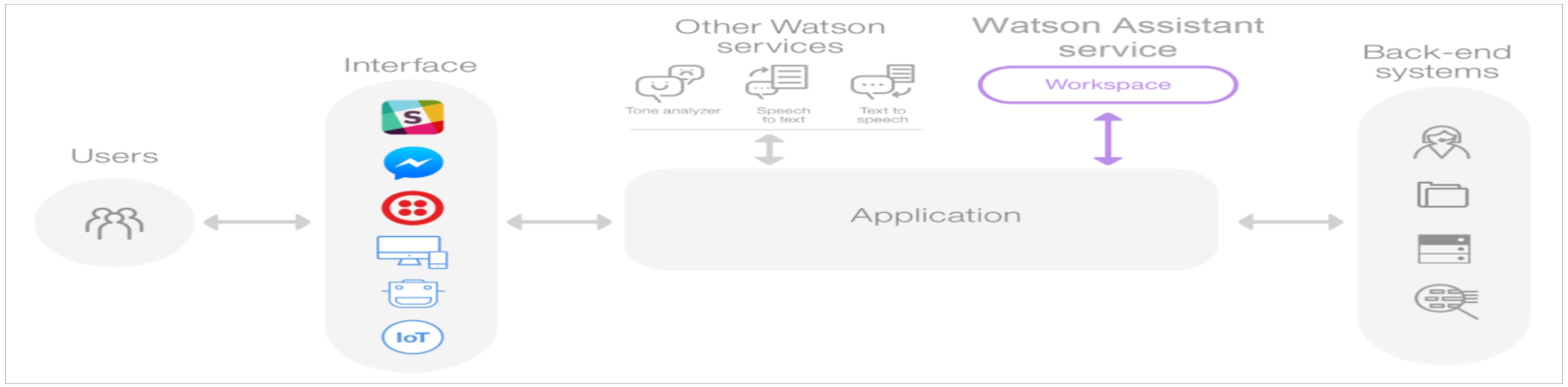
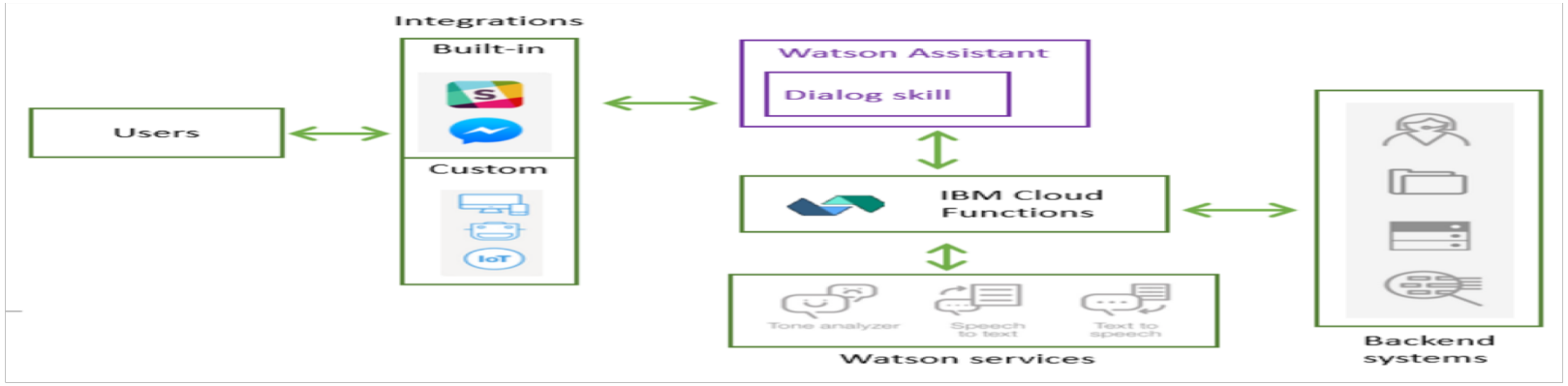
Watson Assistant
Overview

IBM **Developer**

Watson Assistant

Watson Assistant is a cloud based or on-premise platform that allows developers and non-technical users to collaborate on building conversational AI solutions.





I'm frustrated, I haven't been able to login into your online billing system...

Intent Password Reset

Entities Online Billing System

Emotional Tone Anger

Context Bill Smith, 47, Gold Member, High Value

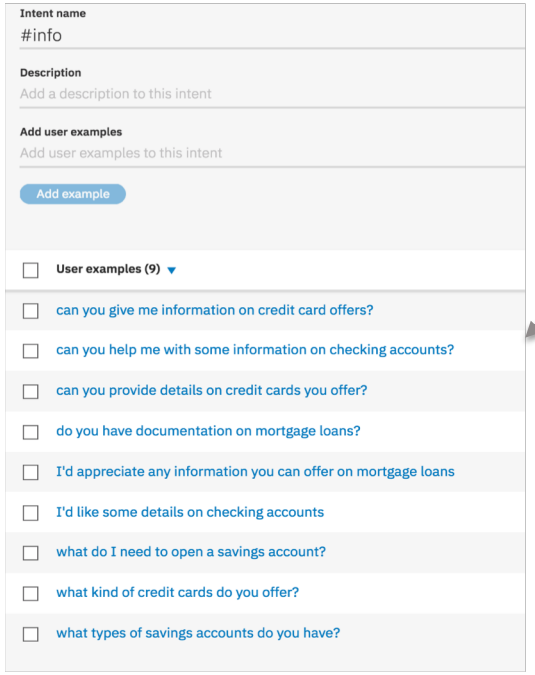
Context Mobile

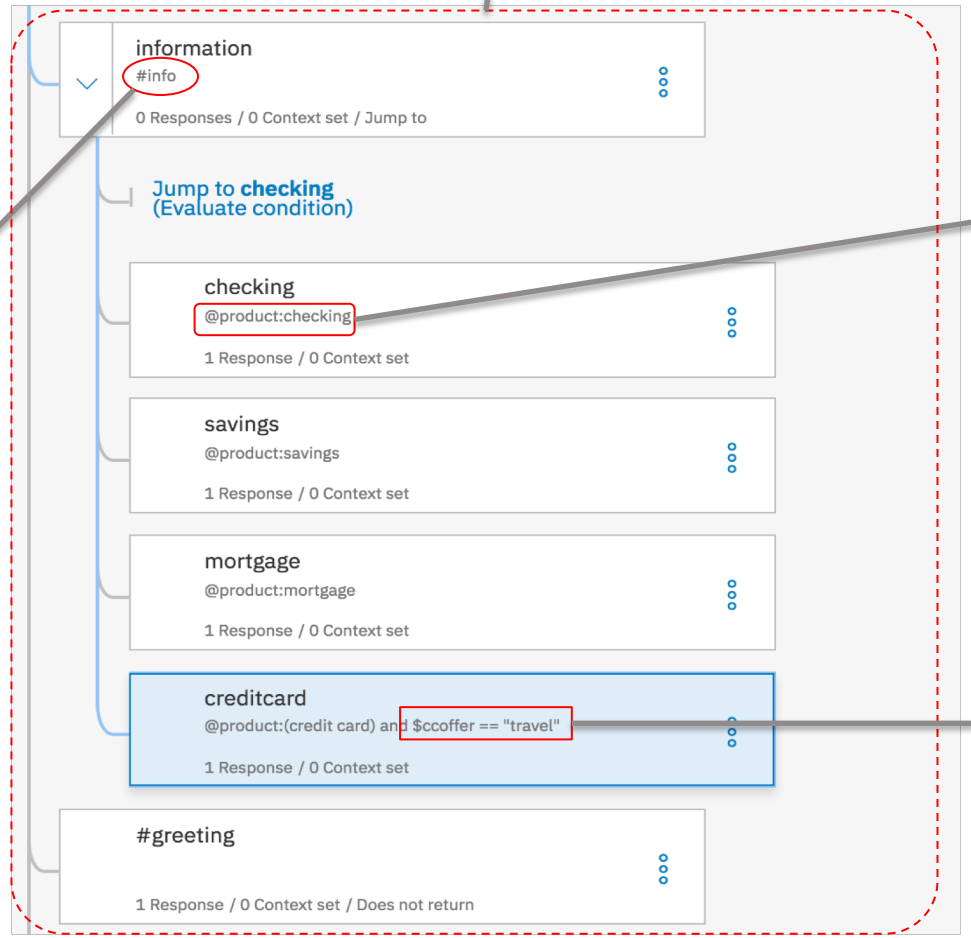
Watson Assistant Dialog Skills (Workspace)

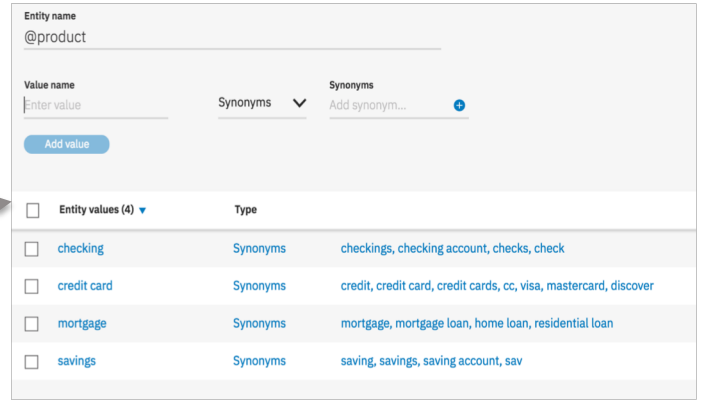
Intents: understand what users mean

Dialog: orchestrates interaction with end users based on intents, entities, and context.

Entities: terms (words or phrases) that provide clarification or context to intents







Context: offers a mechanism to communicate information between Watson Assistant service and application

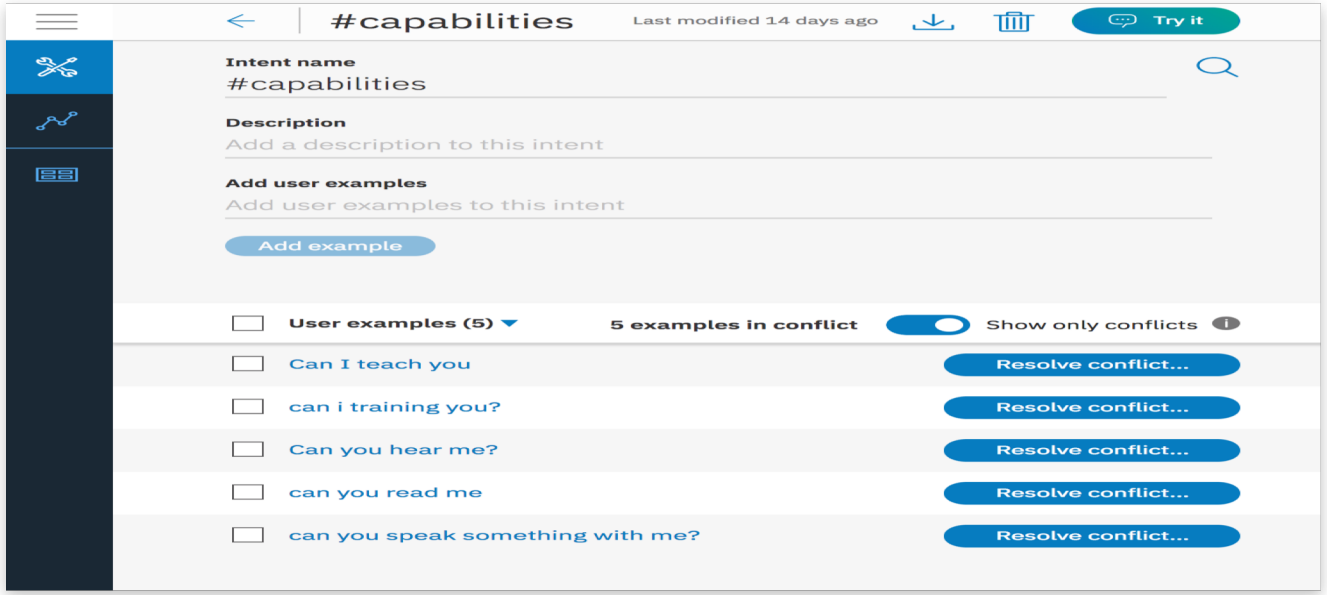
Intents

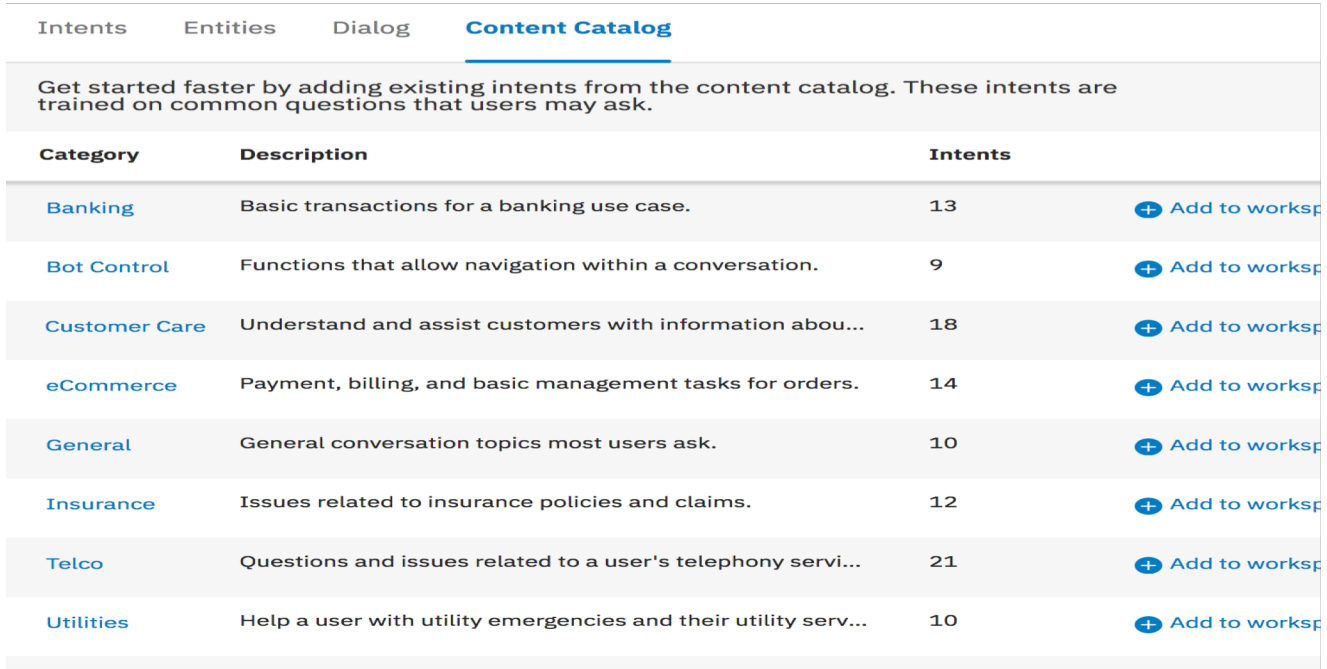
An **intent** is the goal or purpose of the user's input. Adding examples to intents helps your virtual assistant understand different ways in which people would say them.

- Provide at least five examples for each intent.

Features:

- Content Catalog
- Conflict Resolution*



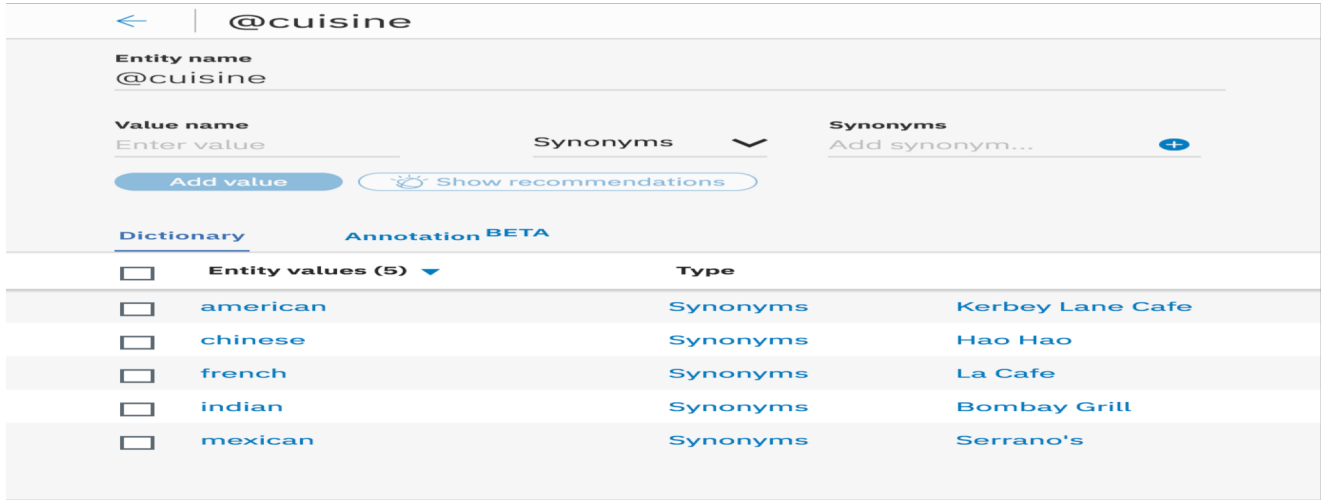


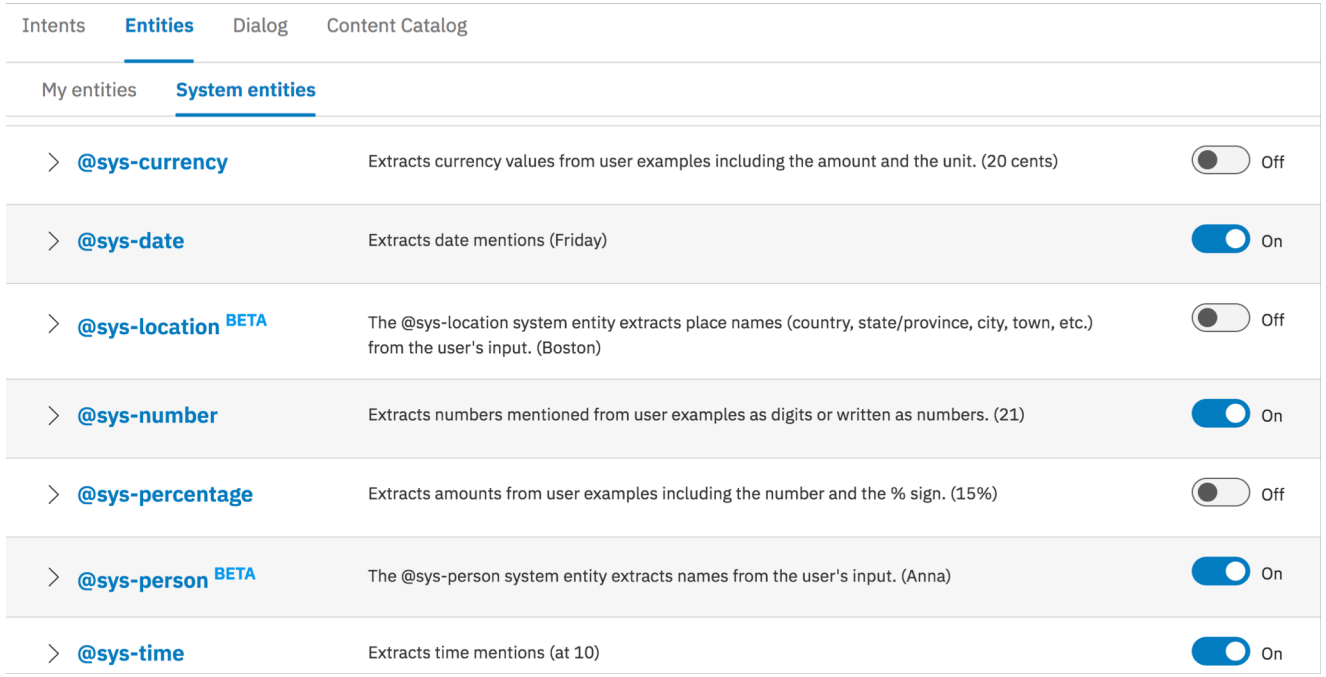
Entities

An **entity** is a portion of the user's input that you can use to provide a different response to a particular intent. Adding values and synonyms to entities helps your virtual assistant learn and understand important details that your users mention.

Features:

- System Entities.
- Contextual Entities.
- Synonym Recommendations.



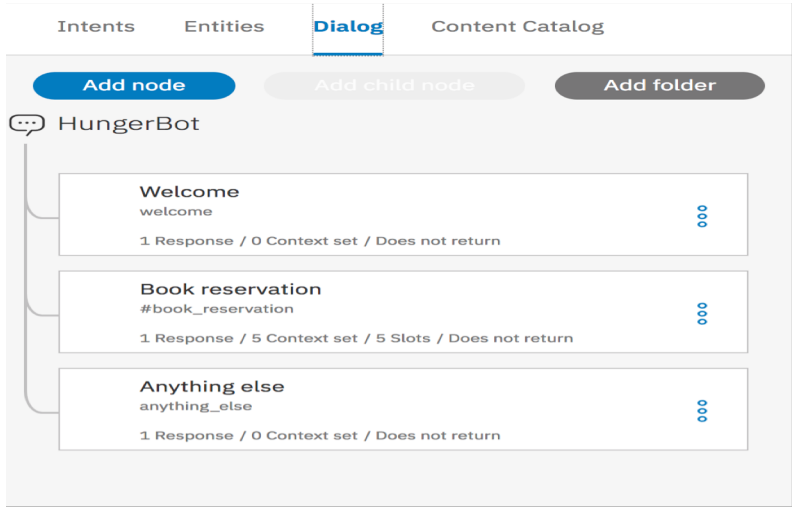


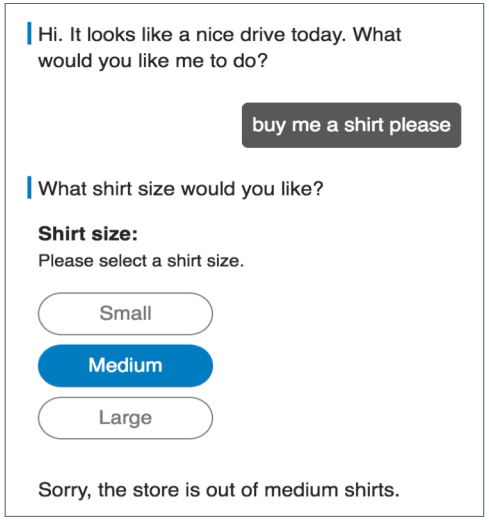
Dialog

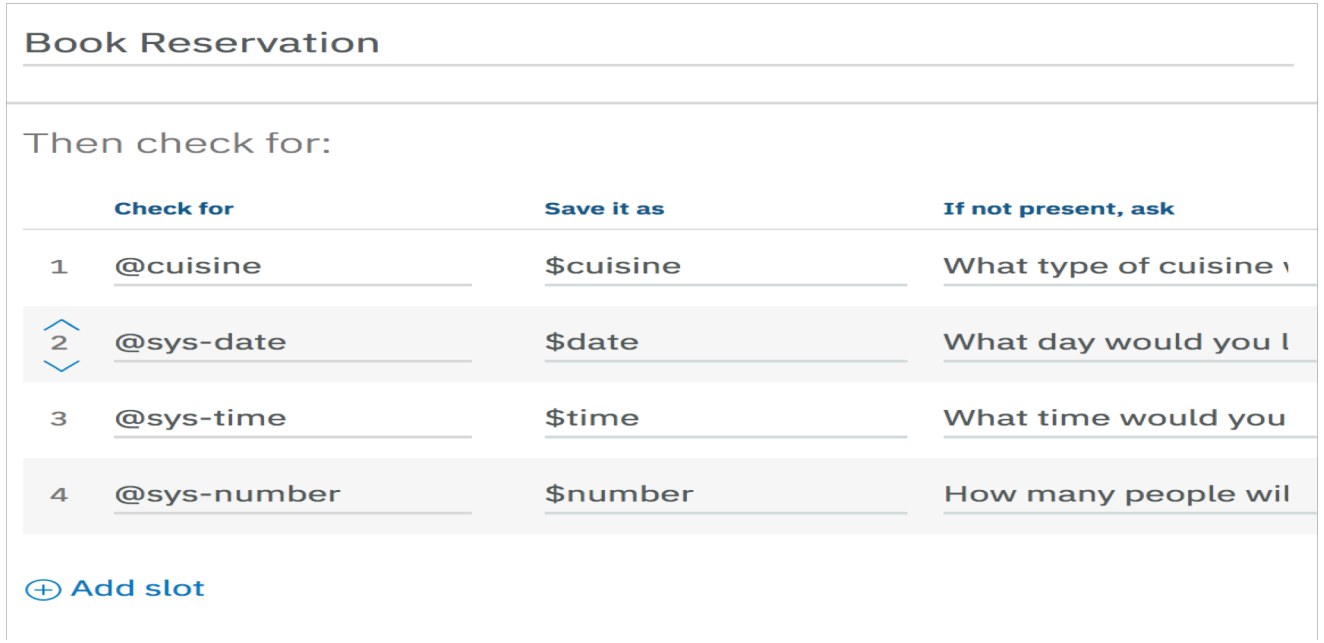
A **dialog** uses intents, entities, and context from your application to define a response to each user's input.

Features:

- Slot based dialog nodes.
- Multiple response types.
  - Buttons, Images, Videos, Pause delays
- Digressions.
- Disambiguation.







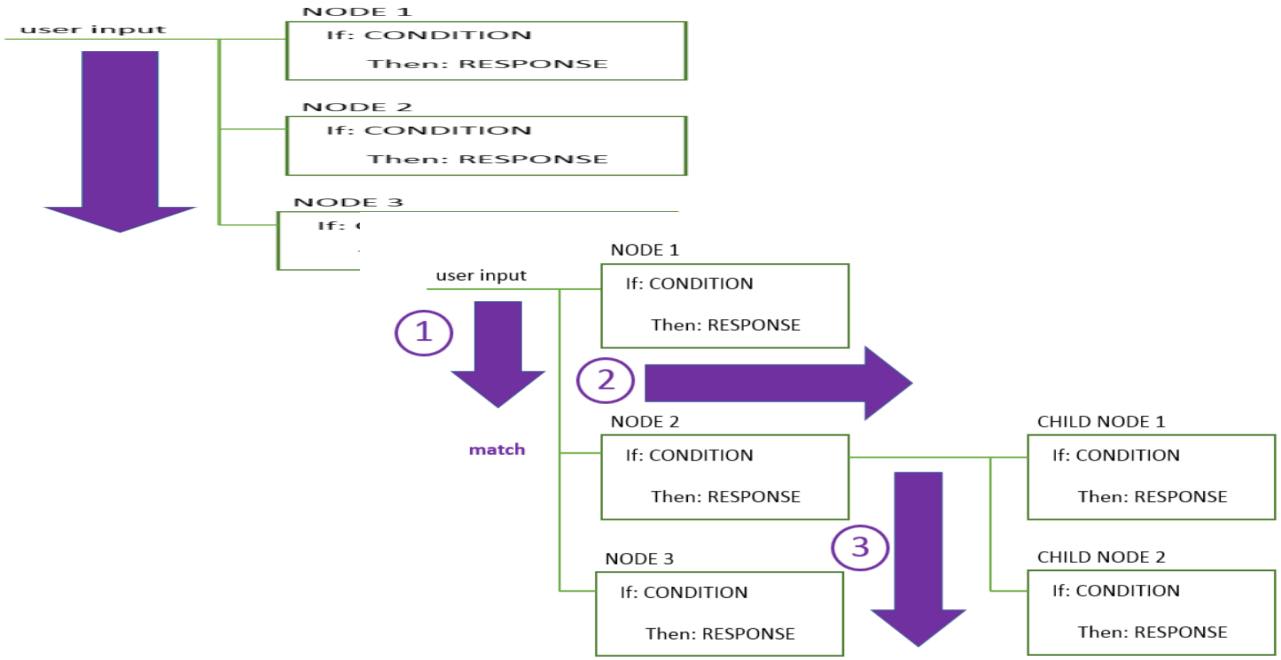
Dialog: Evaluation and Flow

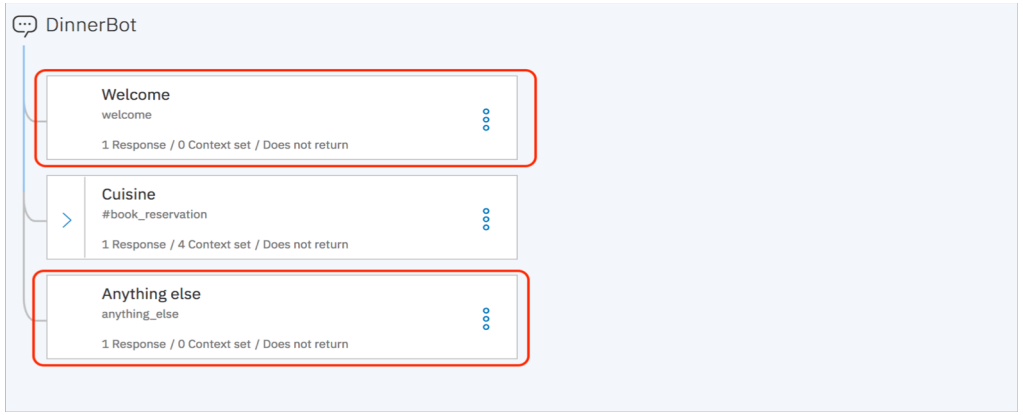
Order matters in dialog evaluation:

- Dialog nodes evaluated in order they appear in UI, top to bottom.
- If a match is found at the top level, child nodes are evaluated recursively until last leaf node is processed
- “Jump to” can be used to alter default flow, to route the flow and/or simplify dialog logic.

Special Condition Nodes

- “Welcome” node is only invoked when calling app sends no user input and no context.
  - Should reply with capabilities of your chatbot
- “Anything else” node is invoked when there are no matches for user input at the top level.
  - Should give helpful message including examples of valid input





DEMO

Runtime APIs (v1)

Request payload includes input for the conversation workspace to process in JSON format.

- Minimum includes input.text value which represents the users input.
- Context for prior / existing sessions of the conversation chat.
- alternative_intents flag to instruct conversations service to return all intent values.

Request

PATH PARAMETERS	
workspace_id * string	Unique identifier of the workspace.
QUERY PARAMETERS	
version * string	Release date of the API version you want to use. Specify dates in YYYY-MM-DD format. The current version is `2018-09-20`. **Constraints:** matches regular expression `^\d[4]-\d[2]-\d[2]$`
nodes_visited_details boolean	Whether to include additional diagnostic information about the dialog nodes that were visited during processing of the message. Default: `false`
REQUEST BODY MessageRequest	The message to be sent. This includes the user's input, along with optional intents, entities, and context from the last response. Example: View
input > InputData	The user input.
alternate_intents boolean	Whether to return more than one intent. Set to `true` to return all matching intents. Default: `false`
context > Context	State information for the conversation. To maintain state, include the context from the previous response.
entities > RuntimeEntity[]	Entities to use when evaluating the message. Include entities from the previous response to continue using those entities rather than detecting entities in the new input.
intents > RuntimeIntent[]	Intents to use when evaluating the user input. Include intents from the previous response to continue using those intents rather than trying to recognize intents in the new input.
output > OutputData	An output object that includes the response to the user, the dialog nodes that were triggered, and messages from the log.

input ∨ InputData	The user input.
text * string	The text of the user input. This string cannot contain carriage return, newline, or tab characters, and it must be no longer than 2048 characters.
Additional properties	
alternate_intents boolean	Whether to return more than one intent. Set to `true` to return all matching intents. Default: `false`
context ∨ Context	State information for the conversation. To maintain state, include the context from the previous response.
conversation_id string	The unique identifier of the conversation.
system > object	For internal use only.
metadata > MessageContextMetadata	Metadata related to the message.
Additional properties object	

Authoring APIs (v1 only)

Intents — Show/Hide | List Operations | Expand Operations

GET	/v1/workspaces/{workspace_id}/intents	List intents
POST	/v1/workspaces/{workspace_id}/intents	Create intent
DELETE	/v1/workspaces/{workspace_id}/intents/{intent}	Delete intent
GET	/v1/workspaces/{workspace_id}/intents/{intent}	Get intent
POST	/v1/workspaces/{workspace_id}/intents/{intent}	Update intent

Examples — Show/Hide | List Operations | Expand Operations

GET	/v1/workspaces/{workspace_id}/intents/{intent}/examples	List user input examples
POST	/v1/workspaces/{workspace_id}/intents/{intent}/examples	Create user input example
DELETE	/v1/workspaces/{workspace_id}/intents/{intent}/examples/{text}	Delete user input example
GET	/v1/workspaces/{workspace_id}/intents/{intent}/examples/{text}	Get user input example
POST	/v1/workspaces/{workspace_id}/intents/{intent}/examples/{text}	Update user input example

Entities — Show/Hide | List Operations | Expand Operations

GET	/v1/workspaces/{workspace_id}/entities	List entities
POST	/v1/workspaces/{workspace_id}/entities	Create entity
DELETE	/v1/workspaces/{workspace_id}/entities/{entity}	Delete entity
GET	/v1/workspaces/{workspace_id}/entities/{entity}	Get entity
POST	/v1/workspaces/{workspace_id}/entities/{entity}	Update entity

Values

GET	/v1/workspaces/{workspace_id}/entities/{entity}/values
POST	/v1/workspaces/{workspace_id}/entities/{entity}/values
DELETE	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}
GET	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}
POST	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}

Synonyms — Sho

GET	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}/synonyms
POST	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}/synonyms
DELETE	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}/synonyms/{synonym}
GET	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}/synonyms/{synonym}
POST	/v1/workspaces/{workspace_id}/entities/{entity}/values/{value}/synonyms/{synonym}

Workspaces — Show/Hide | List Operations | Expand Operations

GET	/v1/workspaces	List workspaces
POST	/v1/workspaces	Create workspace
DELETE	/v1/workspaces/{workspace_id}	Delete workspace
GET	/v1/workspaces/{workspace_id}	Get information about a workspace
POST	/v1/workspaces/{workspace_id}	Update workspace

Dialog nodes — Show/Hide | List Operations | Expand Operations

GET	/v1/workspaces/{workspace_id}/dialog_nodes	List dialog nodes
POST	/v1/workspaces/{workspace_id}/dialog_nodes	Create dialog node
DELETE	/v1/workspaces/{workspace_id}/dialog_nodes/{dialog_node}	Delete dialog node
GET	/v1/workspaces/{workspace_id}/dialog_nodes/{dialog_node}	Get dialog node
POST	/v1/workspaces/{workspace_id}/dialog_nodes/{dialog_node}	Update dialog node

Runtime APIs (v2)

Request payload input for the Assistant to process in JSON format.

- Requires Assistant ID & Session ID.
- Minimum includes input.text value which represents the users input.
- Option to return context.

Create a session

Create a new session. A session is used to send user input to a skill and receive responses. It also maintains the state of the conversation.

Request

PATH PARAMETERS

assistant_id * string	Unique identifier of the assistant. You can find the assistant ID of an assistant on the **Assistants** tab of the Watson Assistant tool. For information about creating assistants, see the documentation. **Note:** Currently, the v2 API does not support creating assistants.

QUERY PARAMETERS

version * string	Release date of the API version you want to use. Specify dates in YYYY-MM-DD format. The current version is `2018-11-08`. **Constraints:** matches regular expression `^\d{4}-\d{2}-\d{2}$`

Response

RESPONSE BODY
SessionResponse

session_id * string	The session ID.

Send user input to assistant

Send user input to an assistant and receive a response.

There is no rate limit for this operation.

Request

PATH PARAMETERS

assistant_id * string	Unique identifier of the assistant. You can find the assistant ID of an assistant on the **Assistants** tab of the Watson Assistant tool. For information about creating assistants, see the documentation. **Note:** Currently, the v2 API does not support creating assistants.
session_id * string	Unique identifier of the session.

QUERY PARAMETERS

version * string	Release date of the API version you want to use. Specify dates in YYYY-MM-DD format. The current version is `2018-11-08`. **Constraints:** matches regular expression `^\d{4}-\d{2}-\d{2}$`

REQUEST BODY
MessageRequest

The message to be sent. This includes the user's input, along with optional content such as intents and entities.
Example: View

input > MessageInput	The user input.
context > MessageContext	State information for the conversation.

```
POST /v2/assistants/{assistant_id}/sessions/{session_id}/message
```

input ∨ MessageInput	The user input.
message_type string	The type of user input. Currently, only text input is supported. **Possible values:** [`text`] Default: `text`
text string	The text of the user input. This string cannot contain carriage return, newline, or tab characters, and it must be no longer than 2048 characters.
options ∨ MessageInputOptions	Optional properties that control how the assistant responds.
debug boolean	Whether to return additional diagnostic information. Set to `true` to return additional information under the `output.debug` key. Default: `false`
restart boolean	Whether to start a new conversation with this user input. Specify `true` to clear the state information stored by the session. Default: `false`
alternate_intents boolean	Whether to return more than one intent. Set to `true` to return all matching intents. Default: `false`
return_context boolean	Whether to return session context with the response. If you specify `true`, the response will include the `context` property. Default: `false`

Developing Your Application

Develop application by calling conversation service using REST API or Watson Developer Cloud SDK.

Assistant V1:

- Specify authentication credentials and API version
- Send Input
- Process Output
  - Carry out actions
- Repeat
  - Maintaining state

Assistant V2**:

- Specify authentication credentials and API version
- Create Session
- Send Input
- Process Output
  - Carry out actions
- Repeat
- Delete Session

```javascript
// Example 4: implements app actions.

var prompt = require('prompt-sync')();
var ConversationV1 = require('watson-developer-cloud/conversation/v1');

// Set up Conversation service.
var conversation = new ConversationV1({
  username: 'USERNAME', // replace with username from service key
  password: 'PASSWORD', // replace with password from service key
  path: { workspace_id: 'WORKSPACE_ID' }, // replace with workspace ID
  version_date: '2016-07-11'
});

// Start conversation with empty message.
conversation.message({}, processResponse);

// Process the conversation response.
function processResponse(err, response) {
  if (err) {
    console.error(err); // something went wrong
    return;
  }

  var endConversation = false;

  // Check for action flags.
  if (response.output.action === 'display_time') {
    // User asked what time it is, so we output the local system time.
    console.log('The current time is ' + new Date().toLocaleTimeString());
  } else if (response.output.action === 'end_conversation') {
    // User said goodbye, so we're done.
    console.log(response.output.text[0]);
    endConversation = true;
  } else {
    // Display the output from dialog, if any.
    if (response.output.text.length != 0) {
        console.log(response.output.text[0]);
    }
  }

  // If we're not done, prompt for the next round of input.
  if (!endConversation) {
    var newMessageFromUser = prompt('>> ');
    conversation.message({
      input: { text: newMessageFromUser },
      // Send back the context to maintain state.
      context : response.context,
    }, processResponse)
  }
}
```

```javascript
var AssistantV2 = require('watson-developer-cloud/assistant/v2');

// Set up Assistant service wrapper.
var service = new AssistantV2({
  iam_apikey: '{apikey}', // replace with API key
  version: '2018-09-20'
});

var assistantId = '{assistant_id}'; // replace with assistant ID
var sessionId;

// Create session.
service.createSession({
  assistant_id: assistantId
}, function(err, result) {
  if (err) {
    console.error(err); // something went wrong
    return;
  }
  sessionId = result.session_id;
  sendMessage(); // start conversation with empty message
});

// Send message to assistant.
function sendMessage() {
  service.message({
    assistant_id: assistantId,
    session_id: sessionId
  }, processResponse);
}
```

Chatbot Frameworks

Frameworks simplify / abstract messaging channel and backend integration.

- Botkit, Botmaster.ai, Cordova, botbuilder, botpress, etc

Botkit is an open source Node JS framework for chatbot development & integration.

- Supports integration to multiple messaging platforms (FB Messenger, Slack, Twilio, Microsoft Teams, Cisco Spark, etc)
- Event driven conversation management with end user
  - Channel dependent events (DM, channel join, etc)
  - User driven messages

Controller object created per channel.

- Controller can “hear” keyword / pattern or receive event
- Controller responds with single or multiple replies (conversation)

Middleware plugins extend core Botkit functionality

- NLP service integration (Watson, Microsoft, Google, etc)
- Storage integrations (mongo, couchdb, mysql, redis, etc)
  - Enrich message object
- Utilities (CRM, dashboard)
  - Verification of messages, logging, etc

```
var Botkit = require('botkit');

var controller = Botkit.anywhere(configuration);

controller.hears('hello','direct_message', function(bot, message) {
    bot.reply(message,'Hello yourself!');
});
```

```
controller.hears(['keyword','^pattern$'],['message_received'],function(bot,messa

  // do something to respond to message
  bot.reply(message,'You used a keyword!');

});
```

```
convo.say({text: 'I waited 3 seconds to tell you this...', delay: 3000});
```

```
var slackController = Botkit.slackbot({ clientSigningSecret: process.env.SLACK_SIGNING_SECRET });
var slackBot = slackController.spawn({
  token: process.env.SLACK_TOKEN
});
slackController.hears(['.*'], ['direct_message', 'direct_mention', 'mention'], function(bot, message) {
  slackController.log('Slack message received');
  middleware.interpret(bot, message, function() {
    if (message.watsonError) {
      console.log(message.watsonError);
      bot.reply(message, message.watsonError.description || message.watsonError.error);
    } else if (message.watsonData && 'output' in message.watsonData) {
      bot.reply(message, message.watsonData.output.text.join('\n'));
    } else {
      console.log('Error: received message in unknown format. (Is your connection with Watson Assistant
      bot.reply(message, 'I\'m sorry, but for technical reasons I can\'t respond to your message');
    }
  });
});

slackBot.startRTM();
```

Botkit Watson Middleware Plugin

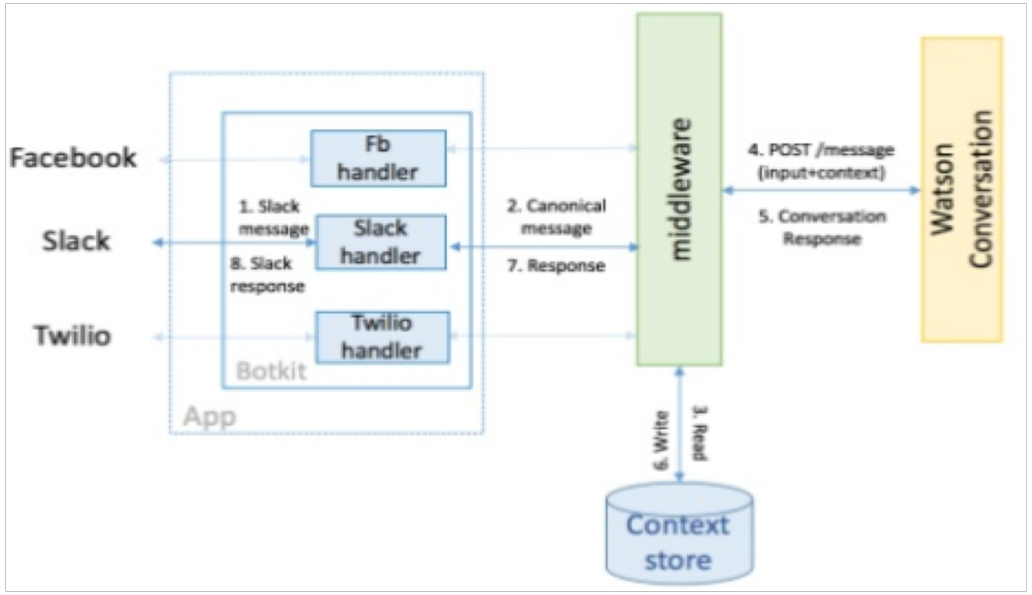
Watson Assistant Middleware Plugin available on NPM.

- Users can have simultaneous, independent conversations with Watson Assistant, through multiple social channels.
- Middleware attaches wastonData object to message

Key Functions

- Receive – tells botkit controller to use Watson Middleware for incoming messages (triggered on every message)
- Before - Code to execute before Assistant resposne
- After – Code to execute after Assistant response

Works with V1 of Watson Assistant, not V2




```
const watsonMiddleware = require('botkit-middleware-watson')({
  iam_apikey: YOUR_API_KEY,
  url: YOUR_ASSISTANT_URL,
  workspace_id: YOUR_WORKSPACE_ID,
  version: '2018-07-10',
  minimum_confidence: 0.50, // (Optional) Default is 0.75
});
```

```
slackController.middleware.receive.use(watsonMiddleware.receive);
slackBot.startRTM();
```

```
slackController.hears(['.*'], ['direct_message', 'direct_mention', 'mention'], function(bot, message) {
  if (message.watsonError) {
    bot.reply(message, "I'm sorry, but for technical reasons I can't respond to your message");
  } else {
    bot.reply(message, message.watsonData.output.text.join('\n'));
  }
});
```

https://github.com/watson-developer-cloud/botkit-middleware

Resources

Watson Assistant Documentation – https://console.bluemix.net/docs/services/assistant/getting-started.html#getting-started (updated)

Watson SDKs - https://github.com/watson-developer-cloud

- Android, Java, Node.js, Python, .NET, Salesforce APEX, Swift, Unity, Ruby

Watson Assistant Sample Apps (with code): https://ibm.biz/BdZ4kPIBM

Code Patterns: https://developer.ibm.com/code/patterns/

IBM Developer Bot Exchange: https://developer.ibm.com/code/exchanges/bots/

IBM Developer Tutorials: https://developer.ibm.com/tutorials

CognitiveClass.ai course: https://cognitiveclass.ai/courses/how-to-build-a-chatbot/

Coursera course: https://www.coursera.org/learn/how-to-build-your-own-chatbot-without-coding

Cloud Foundry: https://www.cloudfoundry.org/the-foundry/ibm-watson-conversation/

DeveloperWorks community: https://www.ibm.com/developerworks/

IBM Watson Blog: https://medium.com/ibm-watson

WordPress plugin: https://wordpress.org/plugins/conversation-watson/

..and much more.