Cheat _Sheet

Custom Actions

Action Class

Base class for all custom actions.

`Action.name`
Defines the action`s name

`Action.run(dispatcher, tracker,domain)`
Executes an action and returns a list of **`rasa_sdk.events.Event`** instances

Tracker Methods

`Tracker.current_state`
Returns the current tracker state as an object.

`Tracker.is_paused`
States whether the tracker is currently paused.

`Tracker.get_latest_entity_values`
Get entity values found for the passed entity type and optional role and group in latest message.

`Tracker.events_after_latest_restart`
Returns a list of events after the most recent restart.

`Tracker.get_slot`
Retrieves the value of a slot

Tracker Object Attributes

`> sender_id`
The unique ID of person talking to the bot.

`> slots`
The list of slots that can be filled as defined in the "ref" domains

`> latest_message`
A dictionary containing the attributes of the latest message: intent, entitites and text.

`> events`
A list of all previous events.

`> active_loop`
The name of the currently active loop

`> latest_action_name`
The name of the last action the bot executed.

Rasa SDK Event Classes

Each event returns a JSON payload describing the details of the event.

Example SlotSet payload:

```
[
  "event": "slot"
  "name": "departure_airport"
  "value": "BER"
]
```

`Slot.Set`
```
SlotSet ("slot_name", value)
```

`AllSlotsReset`
```
evt = AllSlotsReset()
```

`ReminderScheduled`
```
evt = ReminderScheduled(
    intent_name =
    "EXTERNAL_dry_plant",
    trigger_date_time =
    date_time(2020,9,15,0,36,
    0,851609),
    entities = [{"name":
    "plant", "value":"orchid"}],
    name = "remind_water
            _plants",
)
```

`ReminderCancelled`
```
evt = ReminderCancelled(name =
"remind_water_plants")
```

`ConversationPaused`
```
evt = ConversationsPaused()
```

`ConversationResumed`
```
evt = ConversationsResumed()
```

`FollowupAction`
```
evt = FollowupAction(name=
"action_say_goodbye"
```

`UserUtterancereverted`
```
evt = UseUtteranceReverted()
```

`ActionReverted`
```
evt = ActionReverted()
```

`Restarted`
```
evt = Restarted()
```

`SessionStarted`
```
evt = SessionStarted()
```

`UserUttered`
```
evt = UserUttered(text =
"Hallo bot"
```

`BotUttered`
```
evt = BotUttered(text =
"Hallo user"
```

`ActionExecuted`
```
evt = ActionExecuted(text =
"action_greet_user"
```

Dispatcher

CollectingDispatcher.utter_message returns a response to the user

Parameters

→ text
```
dispatcher.utter_message(text =
"Hey there")
```

→ image
```
dispatcher.utter_message(image =
"https://i.imgur.com/nGF1K.jpg")
```

→ template
```
dispatcher.utter_message
(template = "utter_greet")
```

→ attachment
```
dispatcher.utter_message
(attachment = "URL or file path")
```

→ json_message

A custom json payload as a dictionary. Can be used to send channel-specific responses.

→ buttons
```
dispatcher.utter_message(buttons =
    [
        {"payload": "/affirm",
        "title": "Yes"})
        {"payload": "/deny",
        "title": "No"}
    ]
```

→ elements

Facebook-specific payload

→ kwargs

Arbitrary keyword arguments that can be used to specify values for variable interpolation in response variations

Cheat Sheet

Rasa CLI

`> rasa init`
Creates a new project with example training data, actions, and config files.

`> rasa train`
Trains a model using your NLU data and stories, saves trained model in ./models.

`> rasa interactive`
Starts an interactive learning session to create new training data by chatting to your assistant.

`> rasa shell`
Loads your trained model and lets you talk to your assistant on the command line.

`> rasa run`
Starts a server with your trained model.

`> rasa run actions`
Starts an action server using the Rasa SDK.

`> rasa visualize`
Generates a visual representation of your stories.

`> rasa test`
Tests a trained Rasa model on any files starting with test_.

`> rasa data split nlu`
Performs a 80/20 split of your NLU training data.

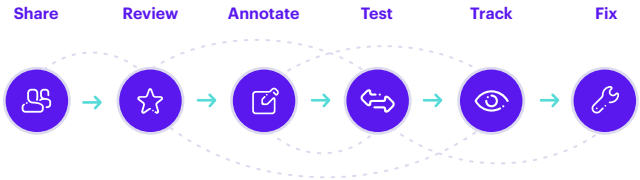
`> rasa data convert`
Converts training data between different formats

`> rasa export`
Exports conversations from a tracker store to an event broker.

`> rasa x`
Launches Rasa X locally.

`> rasa-h`
Shows all available commands.

Conversation-Driven Development



- **Share** your assistant with users as soon as possible
- **Review** conversations on a regular basis
- **Annotate** messages and use them as NLU training data
- **Test** that your assistant always behaves as you expect
- **Track** when your assistant fails and measure its performance over time
- **Fix** how your assistant handles unsuccessful conversations

Training Data

Intents and Entities

```
nlu:
- intent: greet
  examples: |
    - Hey
    - Hi
    - hey there [Sara](name)
```

Intent:
the goal behind the user's message

Examples:
a list of ways the user might express the intent

Entities:
Important key words that sjould be extracted, formatted as [entity](label)

Stories

```
stories:
- story: story with a slot
  steps:
  - intent: greet
  - slot_was_set:
    - name
  - action: utter_greet_user_by_name
  - action: restaurant_form
  - intent: chitchat
  - active_loop: restaurant_form
  - active_loop: null
  - action: utter_slots_values
```

Use stories for multi-turn conversations, where you want the assistant to learn how to generalize.

Story:
name of story, for internal reference

Steps:
the events that happen in the conversation

Slot_was_set:
value saved to memory (can affect which next action to take)

Intent:
what the user says

Action:
what the assistant should do next

Active_loop:
controls form behaviour

Rules

```
rules:
- rule: Only say `hello` if the user
  provided a name
  condition:
  - slot_was_set:
    - user_provided_name: true
  steps:
  - intent: greet
  - active_loop: utter_great
```

Use rules for single-turn interactions with fixed replies.

Rule name:
for internal reference

Steps:
the sequence of actions that should happen when triggered by a specific intent

Slot Types

When the **influence_conversation** flag is set to true, the slot will influence the next action prediction, unless it has slot type: **any**. The way the slot influences the conversation will depend on its slot type:

Text
Feature set to 1 or 0 depending on whether slot is set.

List
Feature is set to 1 if a list value is set, where the list is not empty. If no value is set, or the empty list is the set value, the feature will be 0.

Float
All values below min_value will be treated as min_value, the same happens for values above max_value.

Boolean
Checks if slot is set and if the value is true

Categorial
Used when slot can take one of N values. Creates a one-hot encoding describing which of the values matched.

Any
Not featurized - value does not affect dialogue predictions.

Not sure which slot type to choose? Start with unfeaturized until you determine that a slot should affect the dialogue flow.

Slot Mapping

`from_text`
Fills the slot with the text of the next user message

`from_entity`
Fills the slot with the value of an extracted entity

`from_intent`
Fills the slot with a provided value when the specified intent is detected.

`from_trigger_intent`
Fills the slot with a provided value when the intent that triggered the form activation is detected.

```
forms:
 restaurant_form:
    cuisine:
    - type: from_entity
      entity: cuisine
    num_people:
    - type: from_entity
      entity: number
```

Forms

Form name:
Becomes the name of the action used in stories

Slot(s):
The slots to be filled by the form

rasa.com/docs