Alan AI Cheat Sheet

`pattern` - phrase to invoke a voice/text command or a response to be played

`value` - specified value

`params` - passed parameter

`action` - action to be performed

`output` - data outcome

`[ ... ]` - optional data or parameters

Intents

Define a voice/text command to play a response
```
intent('pattern'[, 'patternN'], reply('pattern'))
```

Define a voice/text command to play a response or perform an action
```
intent('pattern'[, 'patternN'], p => { action })
```

Pattern options

Define alternatives
```
intent('phrase1|phrase2')
```

Define optional words or phrases
```
intent('pattern (optional phrase|)')
```

Response functions

Play a response (in case of multiple patterns, a response is picked at random)
```
reply('pattern'[, 'patternN'])
```

Play a response
```
p.play('pattern')
```

Define voice settings for the assistant reply: accent (*en*, *fr*, *de*, *it*, *ru*, *es*), gender (*'male'/'female'*), voice type, speaking pitch, speaking rate
```
p.play([voice(code, gender, type, pitch, rate), ]'pattern')
```

Define play options: *force:true* (execute if the button is inactive), *activate:true* (activate the button before), *deactivate:true* (deactivate the button after)
```
p.play('pattern'[, opts(options)])
```

Send a command to the client app
```
p.play({command:data})
```

Q&A AI assistant

Define a URL of a resource to be indexed
```
corpus({url: url, depth: depthLevel, maxPages: number})
```

Define text strings to be used by the AI assistant in the dialog
```
corpus(`text`)
```

User-defined slots

Define a static list of values expected in the input
```
$(SLOTNAME value1|value2) => p.SLOTNAME.value
```

Provide labels to classify or identify the slot values
```
$(SLOTNAME value1~label1|value2~label2) => p.SLOTNAME.label
```

Enable fuzzy matching to capture similar variants
```
$(SLOTNAME~ value1|value2) => p.SLOTNAME.value
```

Make a slot optional
```
$(SLOTNAME value1|value2|)
```

Capture several slot values
```
intent('$(SLOTNAME value1|value2) and
        $(SLOTNAME value1|value2 )') => p.SLOTNAME_ (array)
                                        p.SLOTNAME_[0].value
                                        p.SLOTNAME_[1].value
```

Predefined slots

Capture date values
```
$(DATE) => p.DATE.value, p.DATE.moment, p.DATE.luxon
```

Capture time values
```
$(TIME) => p.TIME.value, p.TIME.moment
```

Capture cardinal numbers
```
$(NUMBER) => p.NUMBER.value, p.NUMBER.number
```

Capture ordinal numbers
```
$(ORDINAL) => p.ORDINAL.value, p.ORDINAL.number
```

Capture locations
```
$(LOC) => p.LOC.value
```

Capture names
```
$(NAME) => p.NAME.value
```

Dynamic slots

Define a dynamic slot at the project level
```
project.name = {en: "value1|value2|value3"}
$(SLOTNAME p:name) => p.SLOTNAME.value
```

Define a dynamic slot at the dialog session level
```
p.userData.name = {en: "value1|value2|value3"}
$(SLOTNAME u:name) => p.SLOTNAME.value
```

Get data for a dynamic slot with the visual state
```
let name = ["value1|value2|value3"]
p.visual.data = {en: name};
$(SLOTNAME v:name) => p.SLOTNAME.value
```

Define a dynamic slot in a short form
```
project.name = {en: "value1|value2|value3"}
$(p:name) => p.SLOTNAME.value
```

Define labels for dynamic slots: see User-defined slots ↖

Enable fuzzy matching for dynamic slots: see User-defined slots ↖

Make a dynamic slot optional: see User-defined slots ↖

Capture several slot values: see User-defined slots ↖

RegEx slots

Capture digit and/or letter combination
```
const reg = "([A-Za-z]{1}\\s?){6}"
$(SLOTNAME* ${reg}) => p.SLOTNAME.value
```

Capture any user's input
```
$(SLOTNAME* .+) => p.SLOTNAME.value
```

Contexts

Define a context
```
let contextName = context(() => { action })
```

Activate a context
```
intent('pattern', p => {..., p.then(contextName)}
```

Pass data to the context
```
p.then(contextName, state: {data:yourData}) => p.state.data
```

Resolve a context
```
p.resolve([data:yourData])
```

Reset a context
```
p.resetContext()
```

Define intents to be matched at any time without switching the current context
```
intent(noctx, 'pattern', ...) or noContext(() => {intent(...)})
```

Play a prompt for an expected input
```
fallback('pattern1',[ 'patternN'])
```

Title a context
```
title('contextName')
```

Logs

Write info messages to Alan AI Studio logs
```
console.log(data)
```

Write error messages to Alan AI Studio logs
```
console.error(data)
```

Recognition hints

Provide a list of hints to help recognize specific terms
```
recognitionHints('hint'[, 'hintN'])
```

Predefined objects

Store static device- and user-specific data passed from the client app
```
authData.data => p.authData.data
```

Store state data to be available globally at the project scope
```
project.info = {data:yourData} => project.info.data
```

Store the intent match score
```
p.score
```

Store data to be passed between contexts
```
p.state.data
```

Store visual context data to be passed from the client app. See setVisualState ↗
```
p.visual.data
```

Store user-specific state data to be accessible during the dialog session
```
p.userData.data
```

Predefined callbacks

Define actions to be performed when the script is saved and dialog model is built
```
onCreateProject(() => { action })
```

Define actions to be performed when the dialog session starts
```
onCreateUser((p) => { action })
```

Define actions to be performed when the dialog session ends
```
onCleanupUser((p) => { action })
```

Define actions to be performed when the visual state is set
```
onVisualState((p, s) => { action })
```

Define actions to be performed when a user event is triggered in the client app: *buttonReady*, *buttonClicked*, *micPermissionPrompt*, *micAllowed*, *firstActivate*, *showPopup*, *popupCloseClicked*, *recognized*
```
onUserEvent((p, e) => { action })
```

Define actions to be performed when a context is activated
```
onEnter((p) => { action })
```

Built-in JS libraries

Make API calls
```
axios, request
```

Work with time
```
moment-timezone, luxon
```

Work with arrays, numbers, objects, strings and so on
```
lodash
```

Client API methods

Send information about the visual state from the client app to the dialog script
```
setVisualState(visualStateData:object)
```

Send data or perform actions without a voice/text command
```
projectAPI.method = function(p, param, callback) {
  p.userData.data = param.data;
  callback();
};
callProjectApi(method:string, data:object, callback:function)
```

Play a text message in the client app
```
playText(text:string)
```

Send a text message to Alan AI as the user's input
```
sendText(text:string)
```

Execute a command in the client app
```
playCommand(command:object)
```

Activate the Alan AI button programmatically
```
activate()
```

Deactivate the Alan AI button programmatically
```
deactivate()
```

Check the Alan AI button state
```
isActive()
```

Remove the Alan AI button from the parent element, page (supported on Web, Ionic)
```
remove()
```

Check the state of the wake word (supported on iOS, Android)
```
getWakewordEnabled()
```

Set the state of the wake word (supported on iOS, Android)
```
setWakewordEnabled(enabled:boolean)
```

Handlers*

Handle commands sent from the dialog script to the client app
```
onCommand: function (commandData) { action }
```

Handle Alan AI button state changes
```
onButtonState: function (e) { action }
```

Handle connection status to the AI assistant project in the Alan AI Cloud
```
onConnectionStatus: function (e) { action }
```

Handle events received from Alan AI
```
onEvent: function (e) { action }
```

* Examples are provided for the Web platform