COTOBA DESIGN AIML Documentation

Cotobadesign

Jun 11, 2020

Contents:

CHAPTER 1

About COTOBA AIML

COTOBA AIML is a dialog language used to describe dialog scenarios in COTOBA Agent, the dialog platform of COTOBA DESIGN, Inc. Based on AIML (Artificial Intelligence Markup Language), COTOBA DESIGN, Inc. has developed its own extensions to enable dialog control using information other than COTOBA Agent, such as JSON handling, metadata processing from APIs, and dialog control by calling REST API and referencing return values.

1.1 How to describe the COTOBA AIML

This section explains how to describe a dialog scenario. Here is an example of a basic dialog scenario in which we describe the response of the dialog platform to the content of the userâĂŹs speech.

By combining the contents of the COTOBA AIML reference, you can create a variety of dialog scenarios for holding variables, using JSON, and calling REST API.

1.2 Basic Dialog Scenario

Depending on the content of the userâĂŹs utterance, the response from the dialog platform is defined and the dialog scenario is described.

The category element is the basic unit of the dialog rule of AIML. The category element contains a single dialog rule, such as pattern and template.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

    <category>
        <pattern>Good morning.</pattern>
        <template>Good morning. Let's do our best today!</template>
    </category>

</aiml>
```

Input: Good morning.

Output: Good morning. LetâĂŹs do our best today!

1.3 Branching of the Response

When creating a dialog scenario, it is often necessary to modify the response text according to the userâĂŹs previous utterance.

For example, there is a scene where you respond to a query from a dialog platform with a âĂIJyes/noâĂİ response. Although âĂIJyes/noâĂİ is used in the pattern, common phrases such as âĂIJyes/noâĂİ are used in other dialogs and need to be distinguished.

An example of a response branch in this case is that.

In the example below, the response is changed according to the results of âĂIJYesâĂİ and âĂIJNoâĂİ. However, user utterances of âĂIJYesâĂİ and âĂIJNoâĂİ can occur in other cases as well, but this response matches the previous utterance of the Bot with that, so that the response of the Bot matches only when âĂIJWould you like sugar and milk in your coffee?âĂİ.

```
<category>
    <pattern>I like coffee.</pattern>
    <template>Do you put sugar and milk in your coffee?</template>
</category>

<category>
    <pattern>Yes</pattern>
    <that>Do you put sugar and milk in your coffee?</that>
    <template>Okay.</template>
</category>

<category>
    <pattern>No</pattern>
    <that>Do you put sugar and milk in your coffee?</that>
    <template>Black coffee.</template>
</category>
```

Input: I like coffee.
Output: Do you put sugar and milk in your coffee?
Input: Yes
Output: Okay.

1.4 Extracting Dialog Content and Using Variables

To extract the content of the userâĂŹs speech, use âĂIJ*âĂİ and star. In addition, get and set are used to retain the content of the userâĂŹs speech.

Hold the type of pet in the first utterance. In this case, âĂIJ*âĂİ and wildcard are used for the pet type of the userâĂŹs speech pattern. If you want to use the wildcard part in template, use <star/>. star will be the string of the wildcard range defined by pattern.

The use of variables uses set/get, which holds the contents of the set in the name petcategory specified by the set attribute.

In the following utterances, hold the name of the userâĂŹs pet Similarly, we use set, but with a different variable name, petname.

In the following utterances, the content of the variable is included in the selection of the response from the dialog platform and the response content using the retained content.

To branch by the content of a variable, you can use condition, which is an element that compares strings with the target variable, and you can describe the process like switch-case.

In the following example, the pet type petcategory âĂIJdogâĂİ or âĂIJcatâĂİ is split by the case li of the switch-case statement. If neither is found, the unevaluated result is returned.

It also returns the content of the response, which is held by the petname.

```
<category>
    <pattern>My pet is *.</pattern>
    <template>
        <think><set name="petcategory"><star/></set></think>
        I guess you like <star/>.
    </template>
</category>

<category>
    <pattern>My pet's name is *.</pattern>
    <template>
        <think><set name="petname"><star/></set></think>
        That's a good name.
    </template>
</category>

<category>
    <pattern>Do you remember my pet?</pattern>
    <template>
        <condition name="petcategory">
            <li value="dog">Your pet is a dog <get name="petname"/>.</li>
            <li value="cat">Your pet is a cat <get name="petname"/>.</li>
            <li>I don't think you have a pet.</li>
        </condition>
    </template>
</category>
```

Input: My pet is a dog.
Output: I guess you like a dog.
Input: My petâĂŹs name is Maron.
Output: ThatâĂŹs a good name.
Input: Do you remember my pet?
Output: Your pet is a dog Maron.

1.5 BOT Collaboration

Multiple BOTs can be created and the results of each can be linked and operated. It uses the external REST API call of the sraix element for coordination. As shown below, specify the bot ID you have already created as the host name caller and set the necessary information in the body.

The return value from BOT is contained in var:__SUBAGENT_BODY__ and can be retrieved by the json element.

```
<?xml version="1.0" encoding="UTF-8"?>

<aiml version="2.0">
    <category>
        <pattern>Subagent *</pattern>
        <template>
            <think>
                <json var="body.utterance"><star/></json>
```

(continues on next page)

(continued from previous page)

```
                <json var="body.userId"><get var="__USER_USERID__"/></json>
                <set var="__SYSTEM_METADATA__"><json var="body"/></set>
                <sraix>
                    <host>https://HOSTNAME/bots/BOT_ID/ask</host>

                    <method>POST</method>
                    <header>"Content-Type":"application/json;charset=UTF-8"</
↪header>
                    <body><json var="body"/></body>
                </sraix>

            </think>
            <json var="__SUBAGENT_BODY__.response"/>
        </template>
    </category>

</aiml>
```

1.6 Intent Recognition Engine Linkage

When using the model created by the intent recognition engine, the inference endpoint is set at the time of bot creation.

Use NLU elements to create a pattern branching scenario with intent of the intent recognition engine. In this case, the intent and slot for the intent recognition engine can be obtained by using the nluintent and nluslot element.

In addition, the dialog platform performs a rule-based intent recognition according to the description of the scenario and returns a response according to the results evaluated by pattern matching, but if there is no matching pattern, the dialog control is performed using the intent results of the intent recognition. This is to give priority to what the scenario author describes over the consequences of the intent recognition. As an exception, if there is a category with only a wildcard as a pattern, the match is processed after both the scenario description match and the intent recognition match fail to match. Even if you define a child element nlu, the contents of the pattern element will still result in the usual pattern evaluation; see *nlu* for the attributes of nlu elements.

In the following example, if the result of the intent recognition engine is âĂIJRestaurant SearchâĂİ, it matches the pattern and returns the intent list and slot list of the intent recognition engine.

```
<aiml version="2.0">
    <category>
        <pattern>
            <nlu intent="Restaurant Search"/>
        </pattern>
        <template>
            <think>
                <set var="count">0</set>
                <set var="slotCount"><nluslot name="*" item="count" /></set>
            </think>
            <condition>
                <li var="count"><value><get var="slotCount" /></value></li>
                <li>
                    slot:<nluslot name="*" item="slot"><index><get var="count" /></
↪index></nluslot>
                    entity:<nluslot name="*" item="entity"><index><get var="count"␣
↪/></index></nluslot>
                    <!-- score:<nluslot name="*" item="score"><index><get var=
↪"count" /></index></nluslot> -->
                    <think>
                        <set var="count"><map name="upcount"><get var="count" /></
↪map></set>
```

(continues on next page)

(continued from previous page)

```
                    </think>
                    <loop/>
                </li>
            </condition>
        </template>
    </category>
</aiml>
```

CHAPTER 2

Glossary

This chapter defines the terms used in the documentation.

2.1 Terms related to dialog processing

This section defines the dialog terms used in the documentation.

Term	Definition
Dialog Scenario	A group of files describing processing programs and data related to the execution of dialog processing. Dialog programs are written in AIML(Artificial Intelligence Markup Language).
developer	A person who describes dialog programs/data and develops dialog applications.
Dialog Engine	An execution environment that registers dialog scenarios and performs dialog operations.
bot	A dialog process in which the dialog scenario is registered with the dialog engine and dialog processing can be performed.
user	A person who requests dialog processing from a bot.
System	Another name for a bot. It is sometimes used in the context of comparison with the user.
Request	The dialog request the user makes to the bot, or the contents thereof.
Response	The dialog response that the bot returns to the user it interacts with, or its contents.
Utterance	A string that is included in the request and sent to the bot by the user.
Reply	A string that is included in the response and returned by the bot to the user.
Intent Recognition	A function that infers the intent of a utterance using a machine learning model and extracts keywords (slots) associated with the intent from the utterance, or a module that executes the function.
Dialog log	A log that records the operation process of dialog processing executed by the dialog engine (bot).
SubAgent	It is also called SA (SubAgent).
MainAgent	The bot calling the SubAgent. It is also called MA (MainAgent).

2.2 cAIML

This section defines terms related to cAIML (COTOBA AIML) Used in the documentation.

Term	Definition
AIML	Abbreviation of Artificial Intelligence Markup Language. A language for writing dialog programs. AIML is written in XML format.
cAIML	Abbreviation of COTOBA AIML. AIML with proprietary extensions by COTOBA DESIGN.
tag	A tag (Start-tag âĂŸ<âĂę>âĂŹ and End-tag âĂŸ< /âĂę>âĂŹ) defined in cAIML that specifies the basic structure of XML.
Element	The main component of a tag. It is described in the notation of Start-tag <Element> and End-tag < /Element>.
Attribute	The sub-components that make up the tag. It is described in the notation of the Start-tag <Element Attribute = âĂIJValueâĂİ>. There may be more than one attribute, but the same attribute cannot be specified more than one. The attributes that can be used are specified for each element. You can think of an element as a function and an attributed as an argument of the function.
Content	A string written between the Start-tag and End-tag. It is written in the notation âĂŸ<Element attribute = âĂIJValueâĂİ> contents </ Element>âĂŹ. The content can describe a plurality of normal strings and other elements. As a result, cAIML is a nested description format similar to XML.
Element Name	The name of the element for which you want to specify a particular element.
Attribute Name	The attribute element to specify a specific attribute.
Attribute Value	The value to set for the attribute. When it is described as an attribute value in cAIML, it is stipulated by double quotation.
block	Refers to an entire element (Description between the corresponding Start-tag and End-tag) that contains content.
node	An element name of a tree structure corresponding to an element when an XML data structure of cAIML is viewed as a tree structure.
NLU	Refers to the intent recognition function called from cAIML.

CHAPTER 3

COTOBA Agent dialog engine API

3.1 Dialog API

This API is used when the dialog engine is launched using the REST API class (programy.clients.restful.yadlan.sanic.client). Send a request to the bot that includes the userâĂŹs spoken text and get a response from the bot that includes the response text.

Item	Content
Protocol	HTTP
Method	POST

3.1.1 Request

The contents of the dialog API request are as follows.

- Request Header

Field Name	Value	Description
Content-Type	application/json;charset=UTF-8	Specifies JSON as the content type and UTF8 as the character code.

- Request Body

Item Name	Key	Type	Required	Description
Locale	locale	string	No	Language specification: Specifies a hyphenated combination of country code ISO-3166 and language code ISO-639 . Examples: ja-JP, en-US, zh-CN. If not specified, it will operate in the language specified on the bot side.
Time Information	time	string	No	Requestor time. It is specified in ISO8601(RFC3339) format. Example: 2018 - 07 - 01T 12:18:45 + 09: 00. If not set, it operates at server time.
User ID	userId	string	Yes	Specifies a unique ID for each user.
Topic ID	topic	string	No	Specifies the scenario ID from the dialog scenario. If not specified, it works as topic owned by dialog engine.
User Utterance	utterance	string	Yes	The userâĂŹs utterance.
Task Variables Deleting	deleteVariable	boolean	No	When set to true, the variables set by `data` attribute of the set/get in the AIML description will be deleted all at once. The variables set by name and var are not deleted.
Metadata	metadata	string	No	You can set either JSON format metadata or strings. If the data you set up is to be used in a dialog scenario, the dialog scenario must be written to use this metadata. See Using Variables in *Dialog API Data for more information*.
Config	config		No	Settings for dialog engine operating conditions
Log Level	logLevel	string	No	Sets the dialog log output level for the dialog engine. The setting values are as follows. none: Do not output the dialog log, error: Output a dialog log that is greater than or equal to the error level, warning: Output a dialog log that is greater than or equal to the warning level, info: Output a dialog log that is greater than or equal to the operation status, debug: Output a dialog log that is greater than or equal to the debug dialog log. The large/small relation of the dialog log output is none<error<warning<info<debug. If unspecified, the dialog log output level remains unchanged.

- Example of Request

```
POST /v1.0/ask HTTP/1.1
Host: www.***.com
Accept: */*
Content-Type: application/json;charset=UTF-8

{
    "locale": "en-US",
    "time": "2018-07-01T12:18:45+09:00",
    "userId": "E8BDF659B007ADA2C4841EA364E8A70308E03A71",
    "topic": "greeting",
    "utterance": "Hello.",
```

(continues on next page)

(continued from previous page)

```
    "deleteVariable": false,
    "metadata": {"arg1":"value1","arg2":"value2"},
    "config": {"logLevel":"debug"}
}
```

3.1.2 Response

The body of the response to the dialog API request is in JSON format. The response codes for dialog API requests are listed below. It may also return response codes (HTTP status codes other than those listed below) that are not part of the dialog engine. In this case, the contents of the response body are undefined.

- Response Code

Code	Description
200	Request successful.
400	Parameter error. The content of the request needs to be reviewed.
403	Permission error. The API key needs to be reviewed.
404	Specified bot-id does not exist.

- Response Header

Field Name	Value	Description
Content-Type	application/json;charset=UTF-8	Specify JSON as the content type and UTF8 as the character code.

- Response Body

Item Name	Key	Type	Required	Description
User Utterance	utterance	string	Yes	A user utterance that is processed inside the dialog engine. Returns the result of internal processing such as the character normalized from Full-width alphanumeric to Half-width alphanumeric and normalizing Half-width kana to Full-width Kana.
User ID	userId	string	Yes	Specifies a unique ID for each user. Same as the userId of the request.
Response	response	string	Yes	The response from the dialog engine. Returns a UTF8 string.
Topic Name	topic	string	Yes	The name of the current topic.
latency	latency	number	Yes	In-engine processing time. The processing time from the receipt of the request to the return of the response, in seconds. It is the processing time including pattern match processing, intent recognition processing, and SubAgent processing registered in the scenario.
Metadata	metadata	string	No	Either JSON format metadata or a string is set. The content of the metadata is specified by the description in the dialog scenario.

- Example of Response

```
HTTP/1.1 200 Ok
Content-Type: application/json;charset=UTF-8

{
    "response": "Hello, the weather is nice today, too.",
    "userId": "E8BDF659B007ADA2C4841EA364E8A70308E03A71",
    "topic": "greeting",
    "utterance": "Hello."
}
```

An example in which the user states âĂIJPlay next songâĂİ in the dialog scenario corresponding to music playback, and the dialog scenario is written to set the playback instruction information in metadata.

```
HTTP/1.1 200 Ok
Content-Type: application/json;charset=UTF-8

{
    "response": "I will play the next song.",
    "userId": "E8BDF659B007ADA2C4841EA364E8A70308E03A71",
    "topic": "music_play"
    "utterance": "Hello."
    "metadata": {"play":"next"},
}
```

3.2 Debug API

The debug API is an API for retrieving error information and dialog history information that occurred when registering an uploaded zipped archive dialog scenario file with the dialog engine. You can get the dialog state including the past dialog. You can also set (Change) the value of a global variable for use during dialog.

Item	Content
Protocol	HTTP
Method	POST

3.2.1 Request

This is the content set in the debug API request. Only pre-registered users can access the debug API endpoint.

- Request Header

Field Name	Value	Description
x-dev-key	yyyyyyyyyyyyyyyy	Specify the API key obtained by x-dev-key in *user-information* .
Content-Type	application/json;charset=UTF-8	Specify JSON as the content type and UTF8 as the character code.

- Request Body

Item Name	Key	Type	Required	Description
User ID	userId	string	No	Specifies a unique ID for each user. If the user is unspecified or does not exist, the conversation and logs are not retrieved, only duplicates and errors are.
Variable List	variables		No	Specifies information about a variable set a value in a list format. If the user ID is not specified, the variable list specification is disabled. If the user does not exist, the conversation information including the updated variable information can be obtained, but there is no dialog history.
Variable Type	type	string	No	Specifies the variable type. The type that can be specified is âĂŸnameâĂŹ or âĂŸdataâĂŹ. (Specify with key, value.)
Variable Name	key	string	No	Specify the variable name to set the value. (Specify with type, value.)
Value	value	string	No	Describe the value to be changed. (Specify with type, value.)

- Example of Request

```
POST / HTTP/1.1
Host: www.***.com
Accept: */*
x-dev-key: yyyyyyyyyyyyyyyy

Content-Type: application/json;charset=UTF-8

{
    "userId": "E8BDF659B007ADA2C4841EA364E8A70308000000",
    "variables": [
        {
            "type": "name",
            "key": "name_variable",
            "value": "0"
        },
        {
            "type": "data",
            "key": "data_variable",
            "value": "1"
        },
        ïijŽ
        }
    ]
}
```

3.2.2 Response

The body of the response to the debug API request is in JSON format. The response codes for debug API requests are listed below. It may also return response codes (HTTP status codes other than those listed below) that are not part of the dialog engine. In this case, the contents of the response body are undefined.

If variable list: variables is specified at the time of sending, information reflecting the variable setting is returned in the received data.

- Response Code

Code	Description
200	Request successful.
400	Parameter error. The request needs to be reviewed.
403	Permission error. The API key needs to be reviewed.
404	Specified bot-id does not exist.

- Response Header

Field Name	Value	Description
Content-Type	application/json;charset=UTF-8	Specify JSON as the content type and UTF8 as the character code.

- Response body

Item Name	Key	Type	Required	Description
Speech Content	conversations	json	Yes	Acquires the dialog history of the specified user.
Scenario error information	errors	json	Yes	Acquires the error details when registering the dialog scenario.
Scenario duplicate information	duplicates	json	Yes	Acquires pattern duplication when registering a dialog scenario.
Log Information	logs	json	Yes	Acquires the dialog log contents output by the log tag in the template tag during the most recent dialog processing.

- Example of Response

```
HTTP/1.1 200 Ok
Content-Type: application/json;charset=UTF-8

{
    "conversations": {
        "categories": 1251
        "client_context": {
            "botid": "bot",
            "brainid": "brain",
            "clientid": "yadlan",
            "depth": 0,
            "userid": "E8BDF659B007ADA2C4841EA364E8A70308E03A71"
        },
        "data_properties": {
            "data_variable": "1"
        },
        "exception": null,
        "max_histories": 100,
        "properties": {
            "topic": "daytime",
            "name_variable": "0"
        },
        "questions": [
            {
                "data_properties": {},
                "exception": null,
                "name_properties": {
                    "topic": "daytime"
                },
                "sentences": [
                    {
```

(continues on next page)

(continued from previous page)


```
                        "matched_node": {
                            "end_line": "92",
                            "file_name": "../storage/categories/basic.aiml",
                            "start_line": "78"
                :
            :
        ]
    },
    "duplicates": [
        {
            "category": {
                "end": "35",
                "start": "21"
            },
            "description": "Dupicate grammar tree found [Hello]",
            "file": "../storage/categories/basic.aiml",
            "node": {
                "column": "9",
                "raw": "22"
            }
        }
    ],
    "errors": [
        {
            "category": {
                "end": "None",
                "start": "None"
            },
            "description": "Failed to load contents of AIML file : XML-Parser␣
↪Exception [mismatched tag: line 238, column 25]",
            "file": "../storage/categories/ng.aiml",
            "node": {
                "column": "0",
                "raw": "0"
            },
            "node_name": null
        }
    ]
    "logs": [
        {
            "info": "(templete log-node) log message"
        }
    ]
}
```

CHAPTER 4

COTOBA AIML: Basic Elements

This chapter describes the basic AIML elements supported by COTOBA AIML (cAIML).
The [âĂę] at the top of the section refers to the AIML version in which the element was originally defined.
[custom] is an element of the cAIML proprietary extension.

For more information about the pattern matching precedence rules for cAIML, see *Pattern Matching*.

- *aiml*
- *category*
- *pattern*
- *template*
- *topic*

4.1 aiml

[1.0]

The aiml element is the root element for cAIML. All other cAIML elements must be written as the contents of the aiml element.

- Attribute

Patameter	Type	Requ	Description
version	String	Yes	Specifies the AIML version being described.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <!-- cAIML element should be described here -->
</aiml>
```

4.2 category

[1.0]

The category element corresponds to the base unit of dialog rules in cAIML.
The content of the category element consists of a *pattern* element that specifies a matching pattern for the userâĂŹs utterances and a *template* element that specifies the systemâĂŹs response statements to form a single dialog rule.
The contents of the aiml element can contain blocks of category elements.
All cAIML elements, except the *aiml* and *topic* elements, must be contained within a block of category elements.

- Attribute

None

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Hello.</pattern>
        <template>The weather is nice today, too.</template>
    </category>

    <category>
        <pattern>Goodbye.</pattern>
        <template>See you tomorrow.</template>
    </category>
</aiml>
```

Input: Hello.
Output: The weather is nice today, too.
Input: Goodbye.
Output: See you tomorrow.

See also: *aiml*, *topic* , *pattern*, *template*

4.3 pattern

[1.0]

The pattern element describes the contents of the category element and the content of the pattern element describes a pattern for pattern matching with the userâĂŹs utterance.
If the character string described within a pattern element and the character string of the user utterance are matched, a dialog rule (processing within a block of category elements) containing that pattern element is executed.

- Attribute

  None

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Hello</pattern>
        <template>The weather is nice today, too.</template>
    </category>
</aiml>
```

A description including cAIML elements other than character strings can be done in the content of the pattern element.
This allows complex pattern matching operations.
For more information on cAIML elements that can be described as the contents of a pattern element, see *pattern element* .

4.4 template

[1.0]

The template element is described as the content of the category element, and the content of the template element is the system response statement.
When a dialog rule (Block category element) is executed, the string described in the content of the template element for the category element block is replied as the system response statement.

- Attribute

None

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Hello</pattern>
        <template>The weather is nice today, too.</template>
    </category>
</aiml>
```

A description including cAIML elements other than character strings can be done in the content of the template element.
This allows for complex answer sentence generation processing.
For more information on cAIML elements that can be described in the contents of template elements, see *template elements*.

4.5 topic

[1.0]

The dialog rules can be contextualized by describing multiple dialog rule *category* elements in the topic elements block.

When a dialog rule is contextual, the dialog rule is evaluated only if the value of the topic variable held by the Dialog Engine matches the attribute value specified in the name attribute of the topic element.
The dialog rules (non-contextual dialog rules) *category* elements that are not in the block for the topic element are treated in the same way as the attribute value of the name attribute were specified as a wildcard âĂIJ*âĂİ, and their dialog rules are evaluated regardless of the value of the topic variable, which is maintained by the Dialog Engine.
However, the dialog rule that is not contextual is evaluated only if the dialog rule that is contextual in the topic element is evaluated first and the contextual dialog rule is not executed.

âĂIJtopicâĂİ is a reserved word and cannot be used as a user-defined variable name.

- Attribute

Parameter	Type	Required	Description
name	String	Yes	Specify the topic name.

The topic element allows the matching behavior of the same *pattern* to be used differently depending on the context (topic value), as shown in the example below.
In this use case, the response to the userâĂŹs âĂIJI donâĂŹt put anything in.âĂİ utterance is changed by switching the dialog rule to be evaluated for the userâĂŹs âĂIJ~âĂİ utterance is decided according to the topic value set before the utterance and the response corresponding to the rule is returned.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>I like * . </pattern>
        <template>
            I also like <set name = "topic"><star /></set>.
        </template>
    </category>

    <topic name = "coffee">
        <category>
            <pattern>I like it black.</pattern>
            <template>I like it with cream and sugar.</template>
        </category>
    </topic>

    <topic name = "tea">
        <category>
            <pattern>I like it black.  </pattern>
            <template>I like it with lemon. </template>
        </category>
    </topic>
</aiml>
```

Input: I like coffee.
Output: I also like coffee.
Input: I like it black.
Output: I like it with cream and sugar.
Input: I like tea.
Output: I also like tea.
Input: I like it black.

Output: I like it with lemon.

See also: *that*, *set*, *think*

CHAPTER 5

pattern element

This chapter describes child elements that can be described as the content of the pattern element.

The list of pattern matching elements supported by cAIML is in below.
It adds its own set of elements to the official AIML 2.x specification.

- *bot*
- *iset*
- *nlu*
- *oneormore*
- *priority*
- *regex*
- *set*
- *topic*
- *that*
- *word*
- *zeroormore*

Most of the elements described here can use the variety of data that the Dialog Engine can access by specifying attributes and describing child elements as content.
See *Pattern Matching* for details on how to match patterns by writing pattern elements.
The [âĂę] at the beginning of each element indicates the AIML version in which the element was originally defined.

5.1 bot

[1.0]

The bot element is used to call a custom bot property. These variables are accessible to all users accessing the bot. You can set custom bot properties in the properties.txt file.

- Attribute

Parameter	Type	Required	Description
name	String	Yes	Specifies the bot property name.

- Use Case

The following use case returns the name of the bot. (Assuming properties.txt is set to âĂŸname: BotâĂŹ)

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Are you <bot name="name" />? </pattern>
        <template>My name is <bot name="name" />. </template>
    </category>
</aiml>
```

Input: Are you Bot?
Output: My name is Bot.

See also: *File ManagementïijŽproperties*

5.2 iset

[1.0]

The iset element is used to describe a small number of target words without using a sets file.

- Attribute

Parameter	Type	Required	Description
words	String	Yes	Enter the words to be matched by separating them with commas.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>I live in <iset words="Tokyo, Kanagawa, Chiba, Gunma, Saitama,␣
↪Tochigi" />. </pattern>
        <template>
            I live in Kanto, too.
        </template>
    </category>
</aiml>
```

Input: I live in Tokyo.
Output: I live in Kanto, too.

See also: *set*

5.3 nlu

[custom]

The nlu element is used to interact with the intent recognition results of user utterances by the intent recognition engine.
See *NLU* for more information on attributes, how to set child elements, and how to use the intent recognition engine.

- Attribute

Parameter	Type	Required	Description
intent	String	Yes	Specifies the intent name to match.
scoreGt	String	No	Specifies the confidence level to match. Matches if the confidence of the target intent is greater than the specified value.
scoreGe	String	No	Specifies the confidence level to match. Matches if the subject intentâĂŹs confidence is greater than or equal to the specified value.
score	String	No	Specifies the confidence level to match. Matches the confidence of the target intent to the specified value.
scoreLe	String	No	Specifies the confidence level to match. Matches if the subject intentâĂŹs confidence is less than or equal to the specified value.
scoreLt	String	No	Specifies the confidence level to match. Matches if the confidence of the target intent is less than the specified value.
maxLikelihood	String	No	Specifies `true` or `false`. Specifies whether the confidence of the subject must be the maximum likelihood. If `true` , the target intent will match only at maximum likelihood. If `false` , matches even if the subject intent is not one of a maximum likelihood. If not specified, it is `true` .

The available intent names are limited to the range of intent names described in the learning data that creates the intent recognition model used by the intent recognition engine.
Only one scoreXx can be specified. If there is more than one entry, use scoreGt, scoreGe, score, scoreLe, and scoreLt in that order. (If scoreGt and score are listed, scoreGt is adopted.)

- Use Case

Example of around search intent name set to âĂŸaroundsearchâĂŹ in intent recognition model.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>
            <nlu intent="aroundsearch" />
        </pattern>
        <template>
            Around search.
        </template>
    </category>
</aiml>
```

Input: Look around this area.

Output: Around search.

See also: *NLU*

5.4 oneormore

[1.0]

This element is a wildcard that matches at least one arbitrary word. If the wildcard is at the end of the description in the pattern element, it matches up to the end of the userâĂŹs utterance. Also, if the wildcard is between other AIML pattern matching elements in the pattern element description, match processing continues until the next pattern matching element in the wildcard is matched.

There is a priority that matches are applied between this wildcard matching and the matching of other AIML pattern matching elements.
Wildcards âĂIJ_âĂİ are matched before the AIML pattern matching elements `set` , `iset` , `regex` , and `bot`.
Wildcards âĂIJ*âĂİ are matched after these AIML pattern matching elements.
For more information on the pattern matching process, see *Pattern Matching* .

The following two use cases evaluate the match of 1 word âĂIJHelloâĂİ with one or more words that follow.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Hello _</pattern>
        <template>
            Hello
        </template>
   </category>
</aiml>
```

Input: Hello, the weather is nice.
Output: Hello

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
   <category>
       <pattern>Hello *</pattern>
       <template>
           How are you?
       </template>
   </category>
</aiml>
```

Input: Hello, the weather is nice.
Output: How are you?

See also: *zeroormore* , *Pattern Matching*

5.5 priority

[1.0]

Describe patterns to match mapping words to `wildcard`, `set`, `iset`, `regex` , `bot` and so on.
In AIML, âĂIJ$âĂİ is written at the beginning of a word to be matched in a pattern element to give priority to matching for that word over matching for other AIML pattern matching elements.
In the following use case, even if a dialog rule contains a pattern element with wildcards âĂIJHello *âĂİ, âĂIJHello * friendâĂİ, and so on, it will match the word âĂIJfantasticâĂİ with âĂIJ$âĂİ in preference.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

    <category>
        <pattern>Hello $fantastic day, today. </pattern>
        <template>
            That's right.
        </template>
    </category>

    <category>
            <pattern>Hello * </pattern>
            <template>
                Hello.
            </template>
        </category>

        <category>
            <pattern>Hello * friend.</pattern>
            <template>
                Hi.
            </template>
        </category>

</aiml>
```

Input: Hello fantastic day, today.
Output: ThatâĂŹs right.
Input: Hello fantastic.
Output: Hello.
Input: Hello fantastic friend.
Output: Hi.

See also: *word*, *Pattern Matching*

5.6 regex

[custom]

The use of regex elements allows pattern matching by regular expressions to user utterances. It supports word-by-word regular expressions and regular expressions for strings.

- Attribute

Parameter	Type	Required	Description
pattern	String	No	Describing Words with Regular Expressions
template	String	No	Uses word-by-word regular expressions defined in the regex.txt file
form	String	No	Write regular expressions for strings containing multiple words

When describing a regex element, one of the following attributes is required: pattern, template, or form.

- Use Case

There are three ways to specify a regular expression word for an attribute of a regex element.
The first is to write a regular expression directly into pattern (word by word).
One area where this is useful is in handling the difference between UK-English and American-English spelling.
The first is by specifying the regular expression as the âĂŸpatternâĂŹ attribute as below

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>I did not <regex pattern="reali[z|s]e" />it was that.</pattern>
        <template>
            Did you realize it was that?
        </template>
    </category>
</aiml>
```

The second way is to use template. Specify the template name corresponding to the specified template file.
Define with âĂŸtemplate_name.txtâĂŹ.
Regular expression template files allow you to use a regular expression in a template file that can be referenced by multiple dialog rules.
The contents of the description are the same as the words specified in pattern.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern><regex template="color" /></pattern>
        <template>
            color
        </template>
    </category>
</aiml>
```

Third, you can use form as an attribute to specify a regular expression for the string.

The regular expression you specify for form is a combination of the following:

Notation	Meaning
âĂŸ[âĂę]âĂŹ	Matches any character in âĂŸ[âĂę]âĂŹ.
âĂŸA|BâĂŹ	Matches either of the left or right strings of âĂŸ|âĂŹ.
âĂŸ(X)âĂŹ	A subpattern of the regular expression X. It is OK if it matches X.
âĂŸ()?âĂŹ	Matches or does not match the subpattern just before âĂŸ?âĂŹ.

The following example matches any of the following statements.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>
            <regex form="I like analy[z|s]ing (football|soccer) matches " />
        </pattern>
        <template>
            That's nice.
        </template>
    </category>
</aiml>
```

âĂIJI like analyzing football matchesâĂİ âĂIJI like analysing football matchesâĂİ âĂIJI like analyzing soccer matchesâĂİ âĂIJI like analysing soccer matchesâĂİ

See also: *File ManagementïijŽregex_templates*

5.7 set

[1.0]

Specify the match processing with a word set.
The words to be matched are listed in the sets file in a list format and stored under the directory specified by config.
In the name attribute of the set, set a character string excluding the extension from the file name of the sets file.

- Attribute

Parameter	Type	Required	Description
name	String	Yes	Character string excluding the extension from the sets file name

- Use Case

The example below assumes that prefecture.txt contains the names of prefectures in Japan.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <aiml version="2.0">
     <category>
         <pattern>I live in <set name = "preference" />. </pattern>
         <template>
             I live in Tokyo.
         </template>
     </category>
</aiml>
```

Input: I live in Chiba.
Output: I live in Tokyo.

See also: *iset* , *File ManagementïijŽsets*

5.8 topic

[1.0]

Using the topic element, you can add a condition that the value of the system reserved variable topic matches the value specified for name. âĂIJtopicâĂİ can specify a value using the set of template elements as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern><!-- pattern description goes here --></pattern>
        <template>
            <think><set name="topic">FISHING</set></think>
            <!-- response sentence goes here-->
        </template>
    </category>
</aiml>
```

The condition specified in the topic element is evaluated before pattern matching. For example, you can vary the response statement by subdividing the pattern matching process into conditional branches based on the currently set topic value.

In the example below, for the user utterance âĂIJWhy do you know that?âĂİ, the response sentence changes depending on whether the value of topic is set to âĂIJFISHINGâĂİ or âĂIJCOOKINGâĂİ in the previous dialog.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Why do you know that? </pattern>
        <topic>FISHING</topic>
        <template>
            My father taught me when I was a child.
        </template>
    </category>

    <category>
        <pattern>Why do you know that? </pattern>
        <topic>COOKING</topic>
        <template>
            My mother taught me when I was a child.
        </template>
    </category>
</aiml>
```

See also: *that*, *set(template element)*, *think*

5.9 that

[1.0]

By using the that element, you can add to the condition that the response of the system in the previous dialog matches the specified character string. If the pattern element and that element are contained within the category element, the match process of the pattern element is done only if the last response in the previous response of the system matched the specified character string by the that element. This function of the that element makes possible to write dialog rules considering the flow of dialog contents. For example, A system response sentence

to a general user utterance sentence such as âĂIJYesâĂİ or âĂIJNoâĂİ can be classified according to the content of the previous dialog.

- Use Case

In the example of specification below, depending on whether the system response sentence of the previous conversation was âĂIJWould you like sugar and milk in your coffee?âĂİ or âĂIJWould you like some lemon slices with your tea?âĂİ, the dialog rules that match the âĂIJYesâĂİ and âĂIJNoâĂİ of user utterances are classified so that system response sentences that match the contents of the previous dialog are returned.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>I like coffee.</pattern>
        <template>Would you like sugar and milk in your coffee?</template>
    </category>

    <category>
        <pattern>I like tea.</pattern>
        <template>Would you like some lemon slices with your tea?</template>
    </category>

    <category>
        <pattern>Yes.</pattern>
        <that>Would you like sugar and milk in your coffee?</that>
        <template>Okay.</template>
    </category>

    <category>
        <pattern>No.</pattern>
        <that>Would you like sugar and milk in your coffee?</that>
        <template>I like black coffee.</template>
    </category>

    <category>
        <pattern>Yes.</pattern>
        <that>Would you like some lemon slices with your tea?</that>
        <template>Okay. </template>
    </category>

    <category>
        <pattern>No. </pattern>
        <that>Would you like some lemon slices with your tea ?</that>
        <template>I like black tea.</template>
    </category>
</aiml>
```

éŰćéĂčéăĚçŻő: *topic*

5.10 word

[1.0]

AIMLâĂŹs the most basic pattern matching element. The word element represents a word and is used inside the dialog engine and cannot be described in a scenario. For English words, match processing is not case-sensitive. For double-byte and single-byte characters, matches are performed in single-byte for alphanumeric characters and in double-byte for kana characters.

In the example below, matches any of HELLO, hello, Hello, or HeLlO.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>HELLO</pattern>
        <template>
            Hello
        </template>
    </category>
</aiml>
```

See also: *priority*

5.11 zeroormore

[1.0]

This element is a wildcard that matches at least zero arbitrary words. If the wildcard is at the end in the pattern element, it matches up to the end of the userâĂŹs utterance. Also, if the wildcard is between other AIML pattern matching elements in the pattern element description, match processing continues until the next pattern matching element.

There is a priority in which the matching processes are applied between this wildcard matching process and the matching process of other AIML pattern matching elements.
The wildcard âĂIJ^âĂİ is matched before the AIML pattern matching elements `set` , `iset` , `regex` and `bot`, but the wildcard âĂIJ#âĂİ is matched after these AIML pattern matching elements.

In the example below, the grammar will match any sentence that is either âĂŸHELLOâĂŹ or a sentence that starts with âĂŸHELLOâĂŹ and one or more additional words.

For more information, see *Pattern Matching*.

- Use Case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Hello ^ </pattern>
        <template>
            Hello
        </template>
    </category>

    <category>
        <pattern>Hello #</pattern>
        <template>
            How are you?
        </template>
    </category>
</aiml>
```

See also: *oneormore* , *Pattern Matching*

CHAPTER 6

template element

This chapter describes the AIML elements that can be described within the template element.

The following is a list of AIML elements that can be described within the template elements supported by the Dialog Platform.

- *id*
- *image*
- *implode*
- *input*
- *interval*
- *json*
- *learn*
- *learnf*
- *li*
- *link*
- *list*
- *log*
- *lowercase*
- *map*
- *nluintent*
- *nluslot*
- *normalize*
- *oob*
- *olist*
- *person*
- *person2*
- *program*
- *random*
- *reply*
- *request*
- *resetlearn*
- *resetlearnf*
- *response*
- *rest*
- *set*
- *select*
- *sentence*
- *size*
- *space*
- *split*
- *sr*
- *srai*
- *sraix*
- *star*

- *system*
- *that*
- *thatstar*
- *think*
- *topicstar*
- *uniq*
- *uppercase*
- *video*
- *vocabulary*
- *word*
- *xml*

6.1 Details

This section provides detailed descriptions within the AIML template element.
Most elements take additional data either as xml attributes or child elements.
The [âĂę] at the top of each element indicates the version of AIML in which the element was originally defined.

6.1.1 addtriple

[2.0]

The addtriple element adds the element (knowledge) to the RDF knowledge base. The element has 3 components: subject, predicate, and object. For more information about addtriple elements, see *RDF support*.

In the example below, for the userâĂŹs utterance âĂIJMy favorite food is fishâĂİ, an element (knowledge) consisting of items subject = âĂŸMy favorite foodâĂŹ, pred = âĂŸisâĂŹ, object = âĂŸfishâĂŹ is registered in the RDF knowledge base.

- Use case

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>* favorite food is * </pattern>
        <template>
            <addtriple>
                <subj><star /> favorite food</subj>
                <pred>is</pred>
                <obj><star index="2"/></obj>
            </addtriple>
            Registered the preferences
        </template>
    </category>
</aiml>
```

Input: My favorite food is fish
Output: Registered the preferences

See *uniq*, *select* to check the results of the registration.

See also: *deletetriple*, *select*, *uniq*, *RDF support*

6.1.2 authorise

[1.0]

The authorise element allows the userâĂŹs role to toggle the execution of AIML elements described within the template element. If the userâĂŹs role differs from the role specified in the root attribute of the authorise element, the AIML element described in the authorise element is not executed. See *Security* for more information.

- Attribute

Parameter	Type	Required	Description
role	String	Yes	Role Name
denied_srai	String	No	Srai destination on authentication failure

- Use case

This use case can return the contents of a vocabulary only if the userâĂŹs role is âĂIJrootâĂİ.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Number of vocabulary lists</pattern>
        <template>
            <authorise role="root">
                <vocabulary />
            </authorise>
        </template>
    </category>
</aiml>
```

In addition, you can specify the denied_srai attribute to determine the default behavior when the userâĂŹs role differs from the specified role.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
    <category>
        <pattern>Number of vocabulary lists </pattern>
            <template>
                <authorise role="root" denied_srai="ACCESS_DENIED">
                    <vocabulary />
                </authorise>
            </template>
    </category>
</aiml>
```

See also: *Security*

6.1.3 bot

[1.0]

Gets the properties specific to the bot. This element is read-only. These properties can be specified in properties.txt and read at startup to get as bot-specific information.

- Attribute

Parameter	Type	Required	Description
name	String	Yes	Basically, any of name, birthdate, app_version, grammar_version is described (possible to change in properties.txt).

- Use case

```
 <category>
    <pattern>Who are you? </pattern>
    <template>
        My name is <bot name = "name" />.
        I was born on <bot name = "birthdate" />.
        The application version is <bot name="app_version" />.
        The grammar version is <bot name="grammar_version" />.
    </template>
</category>
```

You can use name as a child element of a bot to describe the same thing as the name attribute.

```
<category>
    <pattern>Who are you? </pattern>
    <template>
        My name is <bot><name>name</name></bot>.
        I was born on <bot><name>birthdate</name></bot>.
        The application version is <bot><name>app_version</name></bot>.
        The grammar version is <bot><name>grammar_version</name></bot>.
    </template>
</category>
```

See also: *File ManagementïijŽproperties*

6.1.4 button

[2.1]

The button element is a rich media element used to prompt the user to tap during a conversation. Text used for the notation of button, postback for Bot, URL when button is pressed can be described as child elements.

- Child element

Parameter	Type	Required	Description
text	String	Yes	Describes the display text for the button.
postback	String	No	Describes the operation when the button is pressed. This message is not shown to the user and is used to respond to a Bot or to process in an application.
url	String	No	Describes the URL when the button is pressed.

- Use case

```
<category>
    <pattern>Transfer</pattern>
    <template>
         <button>
             <text>Do you want to search for transfer? </text>
             <postback>Transfer Guide </postback>
         </button>
    </template>
 </category>

<category>
    <pattern>Search</pattern>
    <template>
         <button>
             <text>Do you want to search?</text>
             <url>https://searchsite.com</url>
         </button>
    </template>
 </category>
```

6.1.5 card

[2.1]

A card is a card that uses several other elements, such as images, buttons, titles, and subtitles. A menu containing all of these rich media elements appears.

- Child element

Parameter	Type	Required	Description
title	String	Yes	Describes the title of the card.
subtitle	String	No	Provide additional information for the card.
image	String	Yes	Describes the image URL etc. for the card.
button	String	Yes	Describes the button information for the card.

- Use case

```
<category>
    <pattern>Search</pattern>
    <template>
        <card>
            <title>Card Menu</title>
            <subtitle>Card Menu Details</subtitle>
            <image>https://searchsite.com/image.png</image>
            <button>
                <text>Do you want to search?</text>
                <url>https://searchsite.com</url>
            </button>
        </card>
    </template>
</category>
```

See also: *button*, *image*

6.1.6 carousel

[2.1]

The carousel element uses several card elements to display a tap through menu. A menu containing all of these rich media elements appears.

- Child element

Parameter	Type	Required	Description
card	String	Yes	Specifies multiple cards. Display one card at a time and tap through to display another card.

- Use case

```
<category>
    <pattern>Restaurant Search</pattern>
    <template>
        <carousel>
            <card>
                <title>Italian</title>
                <subtitle>Searching for Italian restaurants </subtitle>
                <image>https://searchsite.com?q=italian</image>
                <button>Italian Search </button>
            </card>
            <card>
                <title>French</title>
```

(continues on next page)

(continued from previous page)

```
                <subtitle>Searching for French restaurants</subtitle>
                <image>https://searchsite.com?q=french</image>
                <button>French Search</button>
            </card>
        </carousel>
    </template>
</category>
```

See also: *card*, *button*, *image*

6.1.7 condition

[1.0]

This would be used to describe conditional decisions in a template, and possible to describe the processing like switch-case.
The branch is described by determining the variable specified in the condition attribute by the li attribute.
Use the variables defined by get/set and Bot specific information as condition names.
The variable type var is a local variable, name is a global variable, data is a global variable, and acts as a variable to hold until deleteVariable from the API specifies true.
The following describes how conditions are described.

- Attribute

Parameter	Type	Required	Description
name	variable name	No	Specifies the variable to be the branch condition.
var	variable name	No	Specifies the variable to be the branch condition.
data	variable name	No	Specifies the variable to be the branch condition.
bot	property name	No	Specifies the Bot specific information for the branch condition.
value	judgment value	No	Specifies the value for the branch condition.

- Child element

Parameter	Type	Required	Description
li	String	No	Describes the branch condition for the specified variable.

note : Each parameter of the attribute can also be specified as a child element.

Block Condition

The first type of condition is a boolean statement that executes the included template tags if the value is true and does nothing if the value is false.
There are a number of ways the tag can be defined in XML, all 4 of the following statements are syntactically equal returning the value âĂŸXâĂŹ if the value of the predicate property is âĂŸvâĂŹ.

- Use case

```
<condition name="property" value="v">X</condition>
<condition name="property"><value>v</value>X</condition>
<condition value="v"><name>property</name>X</condition>
<condition><name>property</name><value>v</value>X</condition>
```

Single-predicate Condition

The second type of condition acts like a case or a switch statement in some programming languages.
Whereby a single predicate is checked in turn for specific values. As soon as one of the conditions is true, the contained value in then returned.
If no value matches, then the parser checks if there is a default value and if so returns that, otherwise it returns no content.

In the example below, both conditions are syntactically equivalent and will return X if predicate called property has value a, and b if it has value Y, otherwise the statement will return the default value Z.

- Use case

```
<condition name="property">
    <li value="a">X</li>
    <li value="b">Y</li>
    <li>Z</li>
</condition>

<condition>
    <name>property</name>
    <li value="a">X</li>
    <li value="b">Y</li>
    <li>Z</li>
</condition>
```

Multi-predicate Condition

The third form of a condition statement is very much like a nested set of if statements in most programming languages.
Each condition to be checked defined as a <li> tag is checked in turn. Each statement may have different predicate names and values.
As soon as one of the conditions is true further processing of the <li> items stops.

- Use case

```
<condition>
    <li name="1" value="a">X</li>
    <li value="b"><name>1</name>Y</li>
    <li name="1"><value>b</value>Z</li>
    <li><name>1</name><value>b</value>Z</li>
    <li>Z<l/i>
</condition>
```

Looping

<loop>should be listed as one of the child elements of <li>.
If the branch is made to the <li> with <loop>, after the processing of <li> is finished, the content of <condition>is reevaluated.

In the following example, the variable âĂIJtopicâĂİ is evaluated to determine what to return. If the branch condition is not match, âĂIJchatâĂİ is set to âĂIJtopicâĂİ, <condition>is re-evaluated, and âĂIJchatâĂİ exits the loop.

- Use case

```
<condition var="topic">
    <li value="flower">What flowers do you like ? </li>
    <li value="drink">Do you like coffee ? </li>
    <li value="chat">Did something good happen? </li>
    <li><think><set var="topic">chat </set></think><loop/></li>
</condition>
```

See also: *li*, *get*, *set*

6.1.8 date

[1.0]

Gets the date and time string. The locale/time specification in the API changes what is returned.
The format attribute supports Python datetime string formatting. See Python documentation (datetime) for more information.

- Attribute

Parameter	Type	Required	Description
format	String	No	Output format specification. If unspecified, %c.

- Child element

Parameter	Type	Required	Description
format	String	No	Output format specification. If unspecified, %c.

- Use case

```
<category>
    <pattern>What's the date today? </pattern>
    <template>
        Today is <date format="%d/%m/%Y" />.
    </template>
</category>

<category>
    <pattern>What's the date today ?</pattern>
    <template>
        Today is <date><format>%d/%m/%Y</format></date>.
    </template>
</category>
```

See also: *interval*

6.1.9 delay

[2.1]

The delay element is the delay factor. It is used to define the wait time during text-to-speech playback and to specify the delay of the BotâĂŹs response to the user.

- Child element

Parameter	Type	Required	Description
seconds	String	Yes	Specifies the delay in seconds.

- Use case

```
<category>
    <pattern>Wait * seconds</pattern>
    <template>
         <delay>
             <seconds><star/></seconds>
         </delay>
     </template>
 </category>
```

6.1.10 deletetriple

[2.0]

Removes an element of knowledge from the RDF data store.
The element will either have been loaded at startup, or added via the *addtriple* element.
See *RDFSupport* for more information.

- Use case

```
<category>
    <pattern>Remove * is a * </pattern>
    <template>
        <deletetriple>
            <subj><star /></subj>
            <pred>isA</pred>
            <obj><star index="2"/></obj>
        </deletetriple>
    </template>
</category>
```

See also: *addtriple*, *select*, *uniq*, *RDF Support*

6.1.11 denormalize

[1.0]

During pre processing of the input text, the parser converts various characters and combinations of characters into distinct words.
For example âĂŸwww.***.comâĂŹ will be converted to âĂŸwww dot _ _ _ dot comâĂŹ by looking for matches in the mappings contained in dernormal.txt.
If denormalize specifies that âĂŸdotâĂŹ should be transformed to âĂŸ.âĂŹ and âĂŸ_âĂŹ should be transformed to âĂŸ*âĂŹ, normalize/denormalize restores to âĂŸwww.***.comâĂŹ.

- Use case

```
<category>
    <pattern>The URL is *.</pattern>
    <template>
        <think>
            <set var="url"><normalize><star /></normalize></set>
        </think>
        Restore <denormalize><get var = "url" /></denormalize>.
    </template>
</category>
```

<denormalize/>is equivalent to <denormalize><star/></denormalize>.

- Use case

```
<category>
    <pattern>URL is * </pattern>
    <template>
         Restore <denormalize />.
    </template>
</category>
```

Input: The URL is www.***.com.
Output: Restore www.***.com.

See also: *File Management : denormal*, *normalize*

6.1.12 eval

[1.0]

Typically used as part of *learn* or *learnf* elements. Eval evaluates the contained elements returning the textualized content.

In the example below, if a variable with name âĂŸnameâĂŹ was set to TIMMY, and a variable âĂŸanimalâĂŹ was set to dog, then evaluation of this learn node would then see a new category âĂŸlearntâĂŹ to match âĂŸWho is TIMMYâĂŹ with a response, âĂŸYour DOGâĂŹ.

- Use case

```
<category>
    <pattern>Remember * is my pet * .</pattern>
    <template>
        Your pet is <star index = "2" /> <star />.
        <think>
            <set name="animal"><star /></set>
            <set name="name"><star index="2" /></set>
        </think>
        <learnf>
            <category>
                <pattern>
                    <eval>
                        Who is
                        <get name="name"/>?
                    </eval>
                </pattern>
                <template>
                    Your
                    <eval>
```

(continues on next page)

(continued from previous page)

```
                        <get name="animal"/>
                    </eval>
                    .
                </template>
            </category>
        </learnf>
    </template>
</category>
```

Input: Remember TIMMY is my pet DOG.
Output: Your pet is DOG TIMMY.
Input: Who is TIMMY?
Output: Your DOG.

See also: *learn*, *learnf*

6.1.13 explode

[1.0]

Turns the contents of each word contained within the talk into a series single character words separated by spaces. So for example âĂŸFREDâĂŹ would explode to âĂŸF R E DâĂŹ, and âĂŸHello ThereâĂŹ would explode to âĂŸH e l l o T h e r eâĂŹ.

- Use case

```
<category>
    <pattern>EXPLODE *</pattern>
    <template>
        <explode><star /></explode>
    </template>
</category>
```

<explode />is equivalent to <explode><star /></explode>.

```
<category>
    <pattern>EXPLODE *</pattern>
    <template>
        <explode />
    </template>
</category>
```

Input: EXPLODE coffee
Output: c o f f e e

See also: *implode*

6.1.14 image

[2.1]

Use the image element to return image information. You can specify the image URL and file name.

```
<category>
    <pattern>Image display </pattern>
    <template>
        <image>https://url.for.image</image>
    </template>
</category>
```

6.1.15 first

[1.0]

Give a list of words returns the first word. The opposite of rest which returns all but the first word. Usefully for iterating through a series of words in conjunction with the rest tag. Return NIL if there are no more words to return.
If the obtainment fails, the value of âĂIJdefault-getâĂİ set in Config, etc. is returned just like *get*.
Retrieves the first data in the result list when applied to RDF knowledge base search results. See *RDF Support* for more information.

- Use case

```
<category>
    <pattern>My name is * </pattern>
    <template>
        Your first name is <first><star /></first>.
    </template>
</category>
```

<first />is equivalent to <first><star /></first>.

- Use case

```
<category>
    <pattern>My name is * </pattern>
    <template>
       Your first name is <first />.
    </template>
</category>
```

Input: My name is Taro Yamada.
Output: Your first name is Taro.

See also: *rest*

6.1.16 extension

[custom]

It is an element that requires engine customization. An extension provides the mechanism to call out to the underlying Python classes. An extension is essentially made up of a full Python module path to a Python class which implements the `programy.extensions.Extension` interface. See *Extensions* for more information.

- Attribute

Parameter	Type	Required	Description
path	String	Yes	extension name used.

- Use case

```
<category>
    <pattern>
        GEOCODE *
    </pattern>
    <template>
         <extension path="programy.extensions.goecode.geocode.GeoCodeExtension">
             <star />
         </extension>
    </template>
</category>
```

See also: *Extensions*

6.1.17 formal

[1.0]

The formal element will capitalise every distinct word contained between its elements.

- Use case

```
<category>
    <pattern>My name is * *</pattern>
    <template>
     Hello Mr <formal><star /></formal> <formal><star index="2"/></formal>.
    </template>
</category>
```

<formal />is equivalent to <formal><star /></formal>.

- Use case

```
<category>
    <pattern>My name is * *</pattern>
    <template>
        Hello Mr <formal /><formal><star index="2"/></formal>
    </template>
</category>
```

Input: My name is george washington.
Output: Hello Mr George Washington.

6.1.18 gender

[1.0]

The gender element changes to a word of the opposite gender of a personal pronoun that represents the gender contained in the utterance sentence. The content of gender.txt is used for transformation.
The transformation method is described in the before and after sets and is only transformed if there is match within the gender set.

- Use case

```
<category>
    <pattern>Does it belong to *?</pattern>
    <template>
        No, it belongs to <gender><star/></gender>.
    </template>
</category>
```

Input: Does it belong to him?
Output: No, it belongs to her.

See also: *File ManagementïijŽgender*

6.1.19 get

[1.0]

The get element is used to retrieve the value of a variable. If the retrieval fails, the value set by âĂIJdefault-getâĂİ in Config is returned.
(If the definition of âĂIJdefault-getâĂİ was done in the property information of bots: properties.txt , it takes precedences over Config definitions.)
The values that can be retrieved by get are *set* during dialog.
If you want the variable to be set to a value at startup, write it to defaults.txt and the variable can be used as a global one (name).
There are three types of variable types: local and global variables have different retention periods.
You can also retrieve RDF knowledge base elements by specifying child elements <tuples>. See *RDF Support* for more information.

- Local Variables (var)

By specifying the var attribute, it is treated as a local variable.
Local variables are kept only in the category range where set/get is listed. Therefore, it is treated as a separate variable in the reference to srai.

- Persistent Global Variables (name)

It is treated as a global variable by specifying the name attribute. Global variables can also refer to settings in different categories.
In addition, the contents of global variables are maintained continuously, even when dialog processing is repeated.

- Retain Range Global Variable (data)

By specifying the data attribute, it is treated as a global variable. The difference from name is that the variable defined in data is cleared when deleteVariable in the dialog API is set to true.

- Attribute

Parameter	Type	Required	Description
name	Variable Name	Yes	var, name, or data must be set.
var	Variable Name	Yes	var, name, or data must be set.
data	Variable Name	Yes	var, name, or data must be set.

If an AIML variable is specified as a value, it cannot be specified in the attribute, so it can also be specified as a child element.
The behavior is the same as the attribute. If you specify the same attribute name and child element name, the child element setting takes precedence.

- Child element

Parameter	Type	Required	Description
name	Variable Name	Yes	var, name, or data must be set.
var	Variable Name	Yes	var, name, or data must be set.
data	Variable Name	Yes	var, name, or data must be set.

- Use case

```
<!-- Access Global Variable -->
<category>
    <pattern>Today is * .</pattern>
    <template>
        <think><set name="weather"><star/></set></think>
         Today's weather is <get name = "weather" />.
    </template>
</category>

<!-- Access Local Variable -->
<category>
    <pattern>Tomorrow is * .</pattern>
    <template>
        <think><set var="weather"><star/></set></think>
         Today's weather is <get name="weather" />, tomorrow's weather is <get var␣
↪= "weather" />.
    </template>
</category>
<category>
    <pattern>What's the weather? </pattern>
    <template>
         Today's weather is <get name = "weather" />, tomorrow's weather is <get␣
↪var = "weather" />.
    </template>
</category>
```

Input: Today is sunny.
Output: TodayâĂŹs weather is sunny.
Input: Tomorrow is rainny.
Output: Output: TodayâĂŹs weather is sunny, tomorrowâĂŹs weather is rainny.

Input: WhatâĂŹs the weather?
Output: TodayâĂŹs weather is sunny, tomorrowâĂŹs weather is unknown.

See also: *set*, *File ManagementïijŽproperties*

6.1.20 id

[1.0]

Returns the client name. The client name is specified by the client developer in Config.

- Use case

```
<category>
    <pattern>What's your name? </pattern>
    <template>
        <id />
    </template>
</category>
```

Input: WhatâĂŹs your name?
Output: console

6.1.21 implode

[custom]

Given a string of words, concatenates them all into a single string with no spaces. If enable the implode, âĂŸc o f f e eâĂŹ will be transformed to âĂŸcoffeeâĂŹ.

- Use case

```
<category>
    <pattern>Implode *</pattern>
    <template>
        <implode><star /></implode>
    </template>
</category>
```

<implode />is equivalent to <implode><star /></implode>.

- Use case

```
<category>
    <pattern>Implode *</pattern>
    <template>
        <implode />
    </template>
</category>
```

Input: Implode c o f f e e
Output: coffee

See also: *explode*

6.1.22 input

[1.0]

Returns the entire pattern sentences. This is different to the wildcard <star/> tag which only returns those values that match one of the wildcards in the pattern.

- Use case

```
<category>
    <pattern>What was my question ?</pattern>
    <template>
        Your question was "<input />".
    </template>
</category>
```

Input: What was my question?
Output: Your question was âĂIJWhat was my question?âĂİ.

6.1.23 interval

[1.0]

Calculates the difference between 2 time entities.
The format attribute supports Python formatting of date and time strings. See Python documentation for details.

- Child element

Parameter	Type	Required	Description
from	String	Yes	Describes the starting time of the calculation.
to	String	Yes	Describes the final starting time of the calculation.
style	String	Yes	The unit to return in interval. One of years, months, days, or seconds.

- Use case

```
<category>
    <pattern>How old are you? </pattern>
    <template>
         <interval format="%B %d, %Y">
             <style>years</style>
             <from><bot name="birthdate"/></from>
             <to><date format="%B %d, %Y" /></to>
         </interval>
         years old.
    </template>
</category>
```

Input: How old are you?
Output: 5 years old.

See also: *date*

6.1.24 json

[custom]

This is a function for using JSON in AIML.
Use to leverage JSON data for use with *SubAgent*, *metadata*, *NLU* (intent recognition),etc.
See *JSON* for more information.

The variable name specified in name/var/data of the attribute/child element is the variable name defined in get/set.
The variable type var is a local variable, name is a global variable, data is a global variable, and acts as a variable to hold until deleteVariable from the API specifies true.
You can also use system fixed variable names, such as metadata variables and subagent return values.

- Attribute

Parameter	Set value	Type	Required	Description
name		JSON Name	Yes	Specify the JSON to parse. var, name, or data must be set.
var		JSON Name	Yes	Specify the JSON to parse. var, name, or data must be set.
data		JSON Name	Yes	Specify the JSON to parse. var, name, or data must be set.
function		Function Name	No	Describes the processing for JSON.
	len	Function Name	No	If the JSON property is an array, gets array length. For JSON objects, get the number of elements in the JSON object.
	delete	Function Name	No	Deletes the target property. If index is specified in the array, the target element is deleted.
	insert	Function Name	No	Specifies the addition of a value to a JSON array. It is specified with an array number (index).
index		Index	No	Specifies the index when obtaining JSON data. If the target is an array, it is specified the array number. In a JSON object, it specifies the key count from the head. When setting or modifying JSON data, it can be specified only arrays.
item		Get key name	No	Used to get the key from JSON data. Specify this attribute to get keys instead of values.
key		Key specification	No	Specifies the key to manipulate JSON data.

- Child element

If an AIML variable is specified as a value, it cannot be specified in the attribute, so it can also be specified as a child element.
The behavior is the same as the attribute. If you specify the same attribute name and child element name, the child element setting takes precedence.

Parameter	Type	Required	Description
function	Function name	No	Describes the processing for JSON. See the attributeâĂŹs function for its contents.
index	Index	No	You can specify it for a JSON objects and an array. For an array, it specifies the array index and for a JSON objects, it specifies the key count from the beginning. When setting or modifying JSON data, you can only specify arrays.
item	Get key name	No	Used to get the key from JSON data. Specify this attribute to get keys instead of values.
key	Key specification	No	Specifies the key to manipulate the JSON data.

- Use case

This section describes how to get JSON data when a response is returned from a SubAgent called âĂIJtransitâĂİ and use it as a response.
When the following json data is returned from the SubAgent, âĂIJ__SUBAGENT__.transitâĂİ becomes the storage variable name of the response data from the SubAgent.
When obtaining JSON data, specifies the target JSON name in the attribute. In this case, âĂIJ__SUBAGENT__.transitâĂİ is the target JSON name.
When obtaining child elements of JSON data, the json name should be a property of the per-element key name concatenated with âĂIJ.âĂİ.

```
{
    "transportation":{
        "station":{
            "departure":"Tokyo",
            "arrival":"Kyoto"
        },
        "time":{
            "departure":"11/1/2018 11:00",
            "arrival":"11/1/2018 13:30"
        }
    }
}
```

As in the example above, if you want to return the transportation.station.separate, you can use the following:

```
<category>
    <pattern>From Tokyo to Kyoto.</pattern>
    <template>
        Departure from <json var = "__SUBAGENT__.transit.transportation.station.
↪departure" />.
    </template>
</category>
```

Input: From Tokyo to Kyoto.
Output: Departure from Tokyo.

See also: *JSON*, *SubAgent*

6.1.25 learn

[2.0]

The learn element enables the new category depending on the dialog condition.
This new category is held in memory and only takes effect when accessed by the same client for the duration of context.

Because learnf is file preserving, it retains its state when the bot is restarted, but learn is initialized when the bot is restarted.

- Use case

```
<category>
     <pattern>Remember * is my pet *.</pattern>
     <template>
          <think>
              <set name="name"><star /></set>
              <set name="animal"><star index="2" /></set>
          </think>
          <learn>
              <category>
                  <pattern>
                      Who is
                      <eval>
                          <get name="name"/>
                      </eval>
                      ?
                  </pattern>
                  <template>
                      Your
                      <eval>
                          <get name="animal"/>
                      </eval>.
                  </template>
              </category>
          </learn>
     </template>
 </category>
```

Input: Remember TIMMY is my pet DOG.
Input: Who is TIMMY?
Output: Your DOG.

See also: *eval*, *learnf*

6.1.26 learnf

[2.0]

The learnf element enables the new category depending on the dialog condition.
This new category is kept in the file, and when enabled, its contents are kept. It is only enabled when accessed by the same client.

Since learnf is a file retention, it is reloaded when the bot is restarted.

- Use case

```
<category>
     <pattern>Remember * is my pet *.</pattern>
     <template>
         <think>
             <set name="name"><star /></set>
             <set name="animal"><star index="2" /></set>
         </think>
         <learnf>
             <category>
                 <pattern>
                     Who is
                     <eval>
                         <get name="name"/>
                     </eval>
                     ?
                 </pattern>
                 <template>
                     Your
                     <eval>
                         <get name="animal"/>
                     </eval>
                     .
                 </template>
             </category>
         </learnf>
     </template>
 </category>
```

Input: Remeber TIMMY is my pet DOG.
Input: Who is TIMMY?
Output: Your DOG.

See also: *eval*, *learn*

6.1.27 li

[1.0]

The li element describes the branch condition specified in <condition>. For detailed usage, see *condition* for more information.

- Child element

Parameter	Type	Required	Description
think	String	No	The definition which does not affect the action is described.
set	String	No	Set variables.
get	String	No	Gets the value of a variable.
loop	String	No	Specifies a loop for <condition>.
star	String	No	Reuse the input wildcards.

See also : *condition*, *loop*, *think*, *set*, *get*, *star*

6.1.28 link

[2.1]

The link element is a rich media element used for purposes such as URLs to display to the user during a dialog. Child elements can describe the text used to display or read, and the destination url.

- Child element

Parameter	Type	Required	Description
text	String	Yes	Describes the display text to the button.
url	String	No	Describes the URL when the button is pressed.

```
<category>
    <pattern>Search</pattern>
    <template>
        <link>
            <text>Search site</text>
            <url>searchsite.com</url>
        </link>
    </template>
</category>
```

6.1.29 list

[2.1]

The list element is a rich media element that returns the elements described in item in list format. It can describe the contents of the list to the item of the child element. Also it is possible to nest the item with list.

Parameter	Type	Required	Description
item	String	Yes	Describes the contents of the list.

```
<category>
    <pattern>list</pattern>
    <template>
        <list>
            <item>
                <list>
                    <item>list item 1.1 </item>
                    <item>list item 1.2 </item>
                </list>
            </item>
            <item>list item 2.1 </item>
            <item>list item 3.1 </item>
        </list>
    </template>
</category>
```

6.1.30 log

[custom]

Allows a developer to embed logging elements into the itself. These elements are then output into the bot log file. Logging levels are as follows, and are equivalent to Python Logging .

- Attribute

Parameter	Type	Required	Description
level	Variable Name	No	Specify error, warning, debug, info. The default output is info.

See *Log Settings* for more information.

- Use case

```
<category>
    <pattern>Hello</pattern>
    <template>
        Hello.
        <log>Greetings</log>
    </template>
</category>

<category>
    <pattern>Goodbye</pattern>
    <template>
        Goodbye
        <log level="error">Greetings</log>
    </template>
</category>
```

Input: Hello
Output: Hello NOTE: The log would be output âĂIJGreetingsâĂİ at the info level.
Input: Goodbye
Output: Goodbye NOTE: The log would be output âĂIJGreetingsâĂİ at the error level.

See also: *Log Settings*

6.1.31 lowercase

[1.0]

Changes half-width alphabetic characters to lowercase.

- Use case

```
<category>
    <pattern>HELLO * </pattern>
    <template>
        HELLO <lowercase><star /></lowercase>
    </template>
</category>
```

<lowercase />is equivalent to <lowercase><star /></lowercase>.

- Use case

```
<category>
    <pattern>HELLO *</pattern>
    <template>
        HELLO <lowercase />
    </template>
</category>
```

Input: HELLO GEORGE WASHINGTON
Output: HELLO george washington

See also: *uppercase*

6.1.32 map

[1.0]

At startup, reference a map file listing âĂŸkeys: valueâĂŹ and return the value matching the key. If the key does not match, the value set in Config âĂIJdefault-getâĂİ is returned.
As a map file, reference the file stored in the directory specified by config.

- Attribute

Parameter	Type	Required	Description
name	Variable Name	Yes	Specifies the map file name.

- Use case

```
<category>
    <pattern>Where is the prefectural capital of  *? </pattern>
    <template>
       <map name="prefectural_office"><star/></map>ãĄğãĄźãĂĆ
    </template>
</category>
```

Input: Where is the prefectural capital of Kanagawa Prefecture?
Output: Yokohama.

See also: *File ManagementïijŽmaps*

6.1.33 nluintent

[custom]

This function is used to obtain intent information for NLU results.
Values are returned only if there is an NLU result. Therefore, it is basically used in template when the category that specified *nlu tag* in pattern matches.
See *NLU* for more information.

- Attribute

Parameter	Type	Required	Description
name	Intent Name	Yes	Specify the intent name to get. ⋆ is treated as a wildcard. When a wildcard is specified, the index specifies what to get.
item	Get Item Name	Yes	Gets information about the specified intent. The `intent` , `score` and `count` can be specified. If intent is specified, the intent name can be obtained. When score is specified, the confidence level (0.0 ~ 1.0) is obtained. The count returns the number of intent names.
index	Index	No	Specifies the index number of the intent to get. The index is for the list that only includes the intents matching with intent name specified by name.

If an AIML variable is specified as a value, it cannot be specified in the attribute, so it can also be specified as a child element.
The behavior is the same as the attribute. If the same attribute name and child element name would be specified, the child element setting takes precedence.

- Child element

Parameter	Type	Required	Description
name	Intent Name	Yes	Specifies the name of the intent to get. See the attributeâĂŹs name for its contents.
item	Get Item Name	Yes	Gets information about the specified intent. See the attribute item for its contents.
index	Index	No	Specifies the index number of the intent to get. See the index of the attribute for its contents.

- Use case

Get information from NLU processing results. In the following example, the intent is obtained from the NLU processing result.

```
{
    "intents": [
        {"intent": "restaurantsearch", "score": 0.9 },
        {"intent": "aroundsearch", "score": 0.4 }
    ],
    "slots": [
        {"slot": "genre", "entity": "Italian", "score": 0.95, "startOffset": 0,
↪"endOffset": 5 },
        {"slot": "genre", "entity": "French", "score": 0.86, "startOffset": 7,
↪"endOffset": 10 },
        {"slot": "genre", "entity": "Chinese", "score": 0.75, "startOffset": 12,
↪"endOffset": 14 }
    ]
}
```

To get the intent processed by the NLU. Describe as follows.

```
<category>
    <pattern>
        <nlu intent="restaurantsearch"/>
    </pattern>
    <template>
        <nluintent name="restaurantsearch" item="score"/>
    </template>
</category>
```

Input: Look for Italian, French or Chinese.
Output: 0.9

See also: *NLU* ãĂĄ *get the NLU Intent*

6.1.34 nluslot

[custom]

This function is used to obtain slot information of NLU results.
Values are returned only if there is an NLU result. Therefore, it is basically used in template when a category with nlu tag specified in pattern matches.
See *NLU* for more information.

- Attribute

Parameter	Type	Required	Description
name	Slot Name	Yes	Specifies the slot name to get. ⋆ is a wildcard. When a wildcard is specified, index specifies what to get.
item	Get Item Name	Yes	Gets information about the specified slot. It can be specified `slot` , `entity` , `score` ,‘‘startOffset‘‘ , `endOffset` and `count` . When slot is specified, gets the slot name. If entity is specified, gets the extracted string of the slot, if score is specified, gets the confidence level (0.0 ~ 1.0), if startOffset is specified, gets the start character position of the extracted string, and if endOffset is specified, gets the end character position of the extracted string. The count returns the number of identical slot names.
index	Index	No	Specifies the index number of the slot to gets. Specifies the index number in the list that matches the slot name specified by name.

If an AIML variable is specified as a value, it cannot be specified in the attribute, so it can also be specified as a child element. The behavior is the same as the attribute. If it specifies the same attribute name and child element name, the child element setting takes precedence.

- Child element

Parameter	Type	Required	Description
name	Slot Name	Yes	Specifies the slot name to get. See the attributeâĂŹs name for its contents.
item	Get Item Name	Yes	Gets information about the specified slot. See the attribute item for its contents.
index	Index	No	Specifies the index number of the slot to get. See the index of the attribute for its contents.

- Use case

Get slot information from NLU processing result. This section explains how to obtain a slot from the NLU processing result in the following example.

```
{
    "intents": [
        {"intent": "restaurantsearch", "score": 0.9 },
```

(continues on next page)

(continued from previous page)

```
        {"intent": "aroundsearch", "score": 0.4 }
    ],
    "slots": [
        {"slot": "genre", "entity": "Italian", "score": 0.95, "startOffset": 0,
↪"endOffset": 5 },
        {"slot": "genre", "entity": "French", "score": 0.86, "startOffset": 7,
↪"endOffset": 10 },
        {"slot": "genre", "entity": "Chinese", "score": 0.75, "startOffset": 12,
↪"endOffset": 14 }
    ]
}
```

If you want to get the slots from NLU processing result, describe as follows.

```
<category>
    <pattern>
        <nlu intent="restaurantsearch"/>
    </pattern>
    <template>
        <nluslot name="genre" item="count" />
        <nluslot name="genre" item="entity" index="0"/>
        <nluslot name="genre" item="entity" index="1"/>
        <nluslot name="genre" item="entity" index="2"/>
    </template>
</category>
```

Input: Search for Italian, French or Chinese.
Output: 3 Italian French Chinese

See also: *NLU* ãĂĄ *Get NLU Slot*

6.1.35 normalize

[1.0]

Transforms symbols in the target string or abbreviated string to the specified word. The transformation contents are specified in normal.txt. For example, if âĂŸ.âĂŹ is transformed toâĂŹ dot âĂŸandâĂŹ * âĂŸis transformed toâĂŹ _ âĂŸ, then âĂŸ www.***.comâĂŹ will be transformed to âĂŸwww dot _ _ _ dot comâĂŹ.

- Use case

```
<category>
    <pattern>URL is * </pattern>
    <template>
        Displays <normalize><star /></normalize>.
    </template>
</category>
```

<normalize />is equivalent to <normalize><star /></normalize>.

- Use case

```
<category>
    <pattern>URL is * </pattern>
    <template>
         Displays <normalize />.
    </template>
</category>
```

Input: URL is www.***.com
Output: Displays www dot _ _ _ dot com.

See also: *File ManagementïijŽnormal* , *denormalize*

6.1.36 olist

[2.1]

The olist (ordered list) element is a rich media element that returns the elements listed in item. The item of the child element can contain the contents of the list. You can also nest item with list.

Parameter	Type	Required	Description
item	String	Yes	Describe the contents of the list.

```
<category>
    <pattern>Display the list</pattern>
    <template>
         <olist>
            <item>
                 <card>
                     <image>https://searchsite.com/image0.png</image>
                     <title>Image No.1</title>
                     <subtitle>Tag olist No.1</subtitle>
                     <button>
                         <text>Yes</text>
                         <url>https://searchsite.com:?q=yes</url>
                     </button>
                 </card>
             </item>
             <item>
                 <card>
                     <image>https://searchsite.com/image1.png</image>
                     <title>Image No.2</title>
                     <subtitle>Tag olist No.2</subtitle>
                     <button>
                         <text>No</text>
                         <url>https://searchsite.com:?q=no</url>
                     </button>
                 </card>
             </item>
         </olist>
    </template>
 </category>
```

See also: *card*

6.1.37 oob

[1.0]

OOB stands for âĂIJOut of BandâĂİ and when the oob element is evaluated, the corresponding internal module performs processing and returns the processing result to the client. The processing in the internal module is actually intended for equipment operation and is intended for use in embedded devices. The internal modules that handle OOB are designed and implemented by system developers. See *OOB* for more information.

- Use case

```
<category>
    <pattern>DIAL *</pattern>
    <template>
         <oob><dial><star /></dial></oob>
    </template>
</category>
```

Input: DIAL 0123-456-7890
Output: (DIAL) (The returned contents depend on the implementation of the internal module.)

See also: *xml* ãĂĄ *OOB*

6.1.38 person

[1.0]

The person element converts between the first-person pronouns and second-person pronouns in the utterance sentence. Use the contents of person.txt for transformation.
The transformation method is given in the before and after sets and is only transformed if there is a match in the person set.

- Use case

```
<category>
    <pattern>I am waiting for * .</pattern>
    <template>
        You are waiting for <person><star /></person>.
    </template>
</category>
```

<person />is equivalent to <person><star /></person>.

- Use case

```
<category>
    <pattern>I am waiting for * .</pattern>
    <template>
        You are waiting for <person />.
    </template>
</category>
```

Input: I am waiting for you.
Output: You are waiting for me.

See also: *File ManagementïijŽperson* , *person2*

6.1.39 person2

[1.0]

The person2 element converts between the pronouns of the first person and the pronouns of the third person in the utterance sentence. Use the contents of person2.txt for transformation.
The transformation method is specified in the before and after sets and is only transformed if there is a match in the set person2.

- Use case

```
<category>
    <pattern>Please tell * *. </pattern>
    <template>
        I tell <person2><star/></person2> <star index="2" />.
    </template>
</category>
```

<person2 />is equivalent to <person 2><star /></person 2>.

- Use case

```
<category>
    <pattern>Please tell *  *. </pattern>
    <template>
        I tell <person2 /> <star index="2" />.
    </template>
</category>
```

Input: Please tell me how to get there.
Output: I tell you how to get there.

See also: *File ManagementïijŽperson2* , *person*

6.1.40 program

[1.0]

Returns the program version of the Bot specified in Config.

- Use case

```
<category>
    <pattern>version</pattern>
    <template>
        <program />
    </template>
</category>
```

Input: version
Output: AIML bot version X

6.1.41 random

[1.0]

Randomly selects the <li> elements used by <condition>.

- Use case

```
<category>
    <pattern>Hello</pattern>
    <template>
        <random>
             <li>Hello</li>
             <li>How are you today?</li>
             <li>Shall I check today's schedule?</li>
        </random>
    </template>
</category>
```

Input: Hello
Output: Shall I check todayâĂŹs schedule?
Input: Hello
Output: How are you today?

See also: *li*ãĂĄ*condition*

6.1.42 reply

[2.1]

The reply element is a rich media element similar to the button element. As a child element, you can write the text you want to use for reading and the postback to the Bot. The difference between reply and button is that reply is intended to be used for voice interaction instead of GUI.

- Child element

Parameter	Type	Required	Description
text	String	Yes	Describe text-to-speech text.
postback	String	No	Describe the operation. This message is not shown to the user and is used to respond to a Bot or to make some process in an application.

- Use case

```
<category>
    <pattern>Transfer </pattern>
    <template>
         <reply>
             <text>Do you want to do a transfer search?</text>
             <postback>Transfer Guide </postback>
         </reply>
    </template>
 </category>
```

6.1.43 request

[1.0]

Returns the input history. The index attribute specifies the history number. 0 is the current input, and the higher the number, the more past history. This returns all sentences in an input if the input has multiple sentences.

- Attribute

Parameter	Type	Required	Description
index	String	No	Index number. 0 is the current index number, an alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
<category>
    <pattern>What did I say?</pattern>
    <template>
          You said <request index="1" /> and before that
          you said <request index="2" /> and you just said
          <request index="0" />.
    </template>
</category>
```

Input: Hello
Output: Hello
Input: ItâĂŹs already night.
Output: Good evening.
Input: What did I say?
Output: You said âĂIJItâĂŹs already night.âĂİ and before that you said âĂIJHelloâĂİ and you just said âĂIJWhat did I say?âĂİ.

<request />is equivalent to <request index=âĂİ1âĂİ />.

- Use case

```
<category>
    <pattern>What did I say?</pattern>
    <template>
          You said <request />.
    </template>
</category>
```

Input: Hello
Output: Hello
Input: What did I say?
Output: You said Hello.

See also: *response*

6.1.44 resetlearn

[2.x]

Clears all categories enabled in the `<learn>` `<learnf>` elements.

- Use case

```
<category>
    <pattern>Forget what I said.</pattern>
    <template>
         <think><resetlearn /></think>
```

(continues on next page)

(continued from previous page)

```
        OK, I have forgotten what you taught me.
    </template>
</category>
```

6.1.45 resetlearnf

[2.x]

Clears all categories enabled in the <learn> <learnf> elements. This differs from resetlearn in that it deletes all the files that the learnf element created.

- Use case

```
<category>
    <pattern>Forget what I said. </pattern>
    <template>
        <think><resetlearnf /></think>
        OK, I have forgotten what you taught me.
    </template>
</category>
```

6.1.46 response

[1.0]

Returns the output history. The index attribute specifies the history number. The higher the number, the more past history. This returns all sentences if a multi sentence question is asked.

- Attribute

Parameter	Type	Required	Description
index	String	No	Index number. An alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
<category>
    <pattern>What did you just say?</pattern>
    <template>
          I said <response index="1" />and before that
          I said <response index="2" />.
    </template>
</category>
```

Input: Hello
Output: Hello
Input: ItâĂŹs already night.
Output: Good evening.
Input: What did you just say?
Output: I said âĂIJGood eveningâĂİ and before that I said âĂIJHelloâĂİ.

<response />is equivalent to <response index = âĂIJ1âĂİ />.

- Use case

```
<category>
    <pattern>What did you just say? </pattern>
    <template>
          I said <response/>.
    </template>
</category>
```

Input: Hello
Output: Hello
Input: What did you just say?
Output: I said Hello.

See also: *request*

6.1.47 rest

[2.0]

Given a series of words returns all but the first word. Provides the opposite functionality to the first. If the retrieval fails, the value of âĂIJdefault-getâĂİ set in Config, etc. is returned like the *get* .
For example, âĂIJTaro YamadaâĂİ returns âĂIJYamadaâĂİ.
When applied to RDF Knowledgebase search results, it gets non-head data in the results list. See *RDF Support* for more information.

- Use case

```
<category>
    <pattern>My name is * .</pattern>
    <template>
        Your name is <rest><star /></rest>.
    </template>
</category>
```

Input: My name is Taro Yamada.
Output: Your name is Yamada.

See also: *first*

6.1.48 set

[1.0]

The set elements in the template can set global and local variables. For differences in the variable type: name/var/data, see *get*.

- Use case

```
<!-- global variables -->
<category>
    <pattern>MY NAME IS *</pattern>
```

(continues on next page)

(continued from previous page)

```
    <template>
        <set name="myname"><star /></set>
    </template>
</category>

<!-- local variables -->
<category>
    <pattern>MY NAME IS *</pattern>
    <template>
        <set var="myname"><star /></set>
    </template>
</category>
```

See also: *get*

6.1.49 select

[2.0]

The contents of the RDF file referenced at startup and the RDF knowledgebase added by addtriple would search, and to get the appropriate information.
As an RDF file, it refers to the file stored in the directory specified by config.
See *RDF Support* for more information.

- Use case

```
<category>
    <pattern>* legs animals?</pattern>
    <template>
        <select>
             <vars>?name</vars>
             <q><subj>?name</subj><pred>legs</pred><obj><star/></obj></q>
        </select>
    </template>
</category>
```

Input: 4 legs animals?
Output: [[[âĂIJ?nameâĂİ, âĂIJZEBRAâĂİ]], [[âĂIJ?nameâĂİ, âĂIJLIONâĂİ]], [[âĂIJ?nameâĂİ, âĂIJELEPHANTâĂİ]]]

```
<category>
    <pattern>How many legs animals have?</pattern>
    <template>
     <select>
         <vars>?name ?number</vars>
         <q><subj>?name</subj><pred>legs</pred><obj>?number</obj></q>
     </select>
    </template>
</category>
```

Input: How many legs animals have?

Output: [[[âĂIJ?nameâĂİ, âĂIJANTâĂİ], [âĂIJ?numberâĂİ, âĂIJ6âĂİ]], [[âĂIJ?nameâĂİ, âĂIJBATâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJLIONâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJPIGâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJELEPHANTâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJPERSONâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJBEEâĂİ], [âĂIJ?numberâĂİ, âĂIJ6âĂİ]], [[âĂIJ?nameâĂİ, âĂIJBUFFALOâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJANIMALâĂİ], [âĂIJ?numberâĂİ, âĂIJLegsâĂİ]], [[âĂIJ?nameâĂİ, âĂIJFROGâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJPENGUINâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJDUCKâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJBIRDâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJMONKEYâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJGOOSEâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJFOXâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJKANGAROOâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJDOGâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJCOWâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJSHEEPâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJFISHâĂİ], [âĂIJ?numberâĂİ, âĂIJ0âĂİ]], [[âĂIJ?nameâĂİ, âĂIJOXâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJDOLPHINâĂİ], [âĂIJ?numberâĂİ, âĂIJ0âĂİ]], [[âĂIJ?nameâĂİ, âĂIJBEARâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJWOLFâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJZEBRAâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJCATâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJWHALEâĂİ], [âĂIJ?numberâĂİ, âĂIJ0âĂİ]], [[âĂIJ?nameâĂİ, âĂIJCHICKENâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJTIGERâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJHORSEâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJOWLâĂİ], [âĂIJ?numberâĂİ, âĂIJ2âĂİ]], [[âĂIJ?nameâĂİ, âĂIJGOATâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]], [[âĂIJ?nameâĂİ, âĂIJRABBITâĂİ], [âĂIJ?numberâĂİ, âĂIJ4âĂİ]]]

See also: *addtriple*, *deletetriple*, *uniq*, *RDF Support*, *File ManagementïijŽrdfs*

6.1.50 sentence

[1.0]

Capitalises the first word of the sentence and sets all other words to lowercase.

- Use case

```
<category>
    <pattern>Create a sentence with the word *</pattern>
    <template>
        <sentence>HAVE you Heard ABouT <star/></sentence>
    </template>
</category>
```

Input: Create a sentence with the word AnImAl
Output: Have you heard about animal

<sentence />is equivalent to <sentence><star /></sentence>.

- Use case

```
<category>
    <pattern>CORRECT THIS *</pattern>
    <template>
        <sentence />
    </template>
</category>
```

Input: CORRECT THIS PleAse tEll Us The WeAthEr ToDay.
Output: Please tell us the weather today.

6.1.51 size

[1.0]

Returns the number of categories in the Bot.

- Use case

```
<category>
    <pattern>How many categories do you understand? </pattern>
    <template>
        <size />.
    </template>
</category>
```

Input: How many categories do you understand?
Output: 5000.

6.1.52 space

[custom]

The space element inserts a halfwidth space when the sentence is created.

```
<category>
    <pattern>Good morning.</pattern>
    <template>
         <think>
             <set var="french">French</set>
             <set var="italian">Italian</set>
             <set var="chinese">Chinese</set>
         </think>
         Search <get var="french"/>,<get var="italian"/>,<get var="chinese"/>.
         Search <get var="french"/><space/><get var="italian"/><space/><get var=
↪"chinese"/>.
    </template>
</category>
```

Input: Good morning
Output: Search French,Italian,Chinese. Search French Italian Chinese.

6.1.53 split

[2.1]

The split element is used to split the BotâĂŹs response into multiple parts. Split messages are treated as separate messages.

```
<category>
    <pattern>Good morning.</pattern>
    <template>
         The weather is nice today.
```

(continues on next page)

(continued from previous page)

```
        <split/>
        I hope it will be fine again tomorrow.
    </template>
</category>
```

Input: Good morning.
Output: The weather is nice today.
Output: I hope it will be fine again tomorrow.

6.1.54 sr

[1.0]

Shorthand for `<srai><star/></srai>`.

- Use case

```
<category>
    <pattern>My question is * </pattern>
    <template>
        <sr />
    </template>
</category>
```

<sr />is equivalent to <srai><star /></srai>.

```
<category>
    <pattern>My question is * </pattern>
    <template>
        <srai><star/></srai>
    </template>
</category>
```

See also: *star*, *srai*

6.1.55 srai

[1.0]

The srai element allows your bot to recursively call categories after transforming the userâĂŹs input. So you can define a template that calls another category. The acronym âĂIJsraiâĂİ has no official meaning, but is sometimes defined as symbolic reduction or symbolic recursion.

- Use case

```
<category>
    <pattern>Hello</pattern>
    <template><srai>HI</srai></template>
</category>

<category>
    <pattern>Ciao</pattern>
    <template><srai>HI</srai></template>
</category>

<category>
    <pattern>Hola</pattern>
```

(continues on next page)

(continued from previous page)

```
    <template><srai>HI</srai></template>
</category>

<category>
    <pattern>HI</pattern>
    <template>Hello</template>
</category>
```

Input: Hola
Output: Hello

See also: *star*, *sr*, *sraix*

6.1.56 sraix

[2.0]

Call any external REST API. Used for SubAgent calls. See *SubAgent* for more information on using sraix.

- Attribute

Parameter	Type	Required	Description
service	String	No	The service name of the custom external service.
botid	String	No	The bot ID published on the Dialog Platform.

- Use case

```
<category>
    <pattern>Transfer information from * to *.</pattern>
    <template>
        <sraix service="myService">
            <star/>
            <star index="2"/>
        </sraix>
    </template>
</category>
```

Input: Transfer information from Tokyo to Osaka.
Output: The Nozomi departing at 10:00 a.m. is the first to arrive.

See also: *star*, *sr*, *SubAgent*

6.1.57 star

[1.0]

The star element is a description that uses user input from wildcards.
The index attribute provides the ability to access each individual match in the sentence. Wildcards include the one or more characters * and _ the zero or more characters ^ and # , and index(1~) is displayed in order from the beginning.

The index also includes strings that correspond to `set` , `iset` , `regex` , `bot` and bot elements of pattern elements.
If no such information exists, an empty string is returned.

- Attribute

Parameter	Type	Required	Description
index	String	No	Input number. An alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
<category>
    <pattern>I like * and *ãĂĂ.</pattern>
    <template>
        You like <star /> and <star index = "2" />.
    </template>
</category>
```

Input: I like flowers and cats.
Output: You like flowers and cats.

See also: *sr*, *srai*

6.1.58 system

[1.0]

The system element allows you to make underlying system calls. This is obviously a security concern allowing users unfettered access to the underlying system if they know the operating system and can inject shell scripts.

> By default this is disabled, but can be turned on by setting the configuration option `allow_system_aiml` to True.

- Use case

```
<category>
    <pattern>LIST ALL AIML FILES</pattern>
    <template>
        <system>ls -l *.aiml</system>
    </template>
</category>
```

6.1.59 that

[1.0]

The that element is also defined as a child element of `category` , and is used to match the botâĂŹs previous response, but if specified as a template element, that acts as an element `that` retrieves past response statements from the bot.

- Attribute

Parameter	Type	Required	Description
index	String	No	Input number. An alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
<category>
    <pattern>Hello.</pattern>
    <template>
         Hello.
    </template>
</category>

<category>
    <pattern>Pardon?</pattern>
    <template>
         I said <that/>
    </template>
</category>
```

Input: Hello.
Output: Hello.
Input: Pardon?
Output: I said Hello.

See also: *that(pattern)*, *thatstar*, *topicstar*

6.1.60 thatstar

[1.0]

Use thatstar as a wildcard specification for the that.
The that element can use the full set of wildcard matching that is available in the `pattern`ãĂĂelement. These matches are accessed in the same way as using `<star />` , but for that element, we use `<thatstar />` .
If retrieval fails, an empty string is returned.

- Attribute

Parameter	Type	Required	Description
index	String	No	Input number. An alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
<category>
    <pattern>...</pattern>
    <template>
         Do you like coffee?
    </template>
</category>
<category>
    <pattern>Yes.</pattern>
    <that>Do you like * ? </that>
    <template>
        I like <thatstar />, too.
    </template>
</category>
```

Input: âĂę
Output: Do you like coffee?
Input: Yes.
Output: I like coffee, too.

See also: *that(pattern)*, *that*, *topicstar*

6.1.61 think

[1.0]

The think element allows your bot to set predicates without actually displaying the contents of a set element to the user. This is sometimes referred to as âĂIJsilentlyâĂİ setting a predicate.

- Use case

```
<category>
    <pattern>My name is * </pattern>
    <template>
       <think><set name="name"><star /></set></think>
       I remembered your name.
    </template>
</category>
```

6.1.62 topicstar

[1.0]

Use the topicstar element as a wildcard for the `topic`.
The topic can use the full set of wildcard matching that is available in the pattern. These matches are accessed in the same way as using `<star />`, but for the topic, we use `<topicstar />`.
It is also possible to specify âĂIJindexâĂİ for the attribute. If retrieval fails, an empty string is returned.

- Attribute

Parameter	Type	Required	Description
index	String	No	Input number. An alternative form with no attributes implies the use of index=âĂŸ1âĂŹ.

- Use case

```
 <category>
     <pattern>I like coffee. </pattern>
     <template>
         <think><set name="topic">beverages coffee </set></think>
         OK.
     </template>
 </category>

<topic name="beverages *">
<category>
    <pattern>What's my favorite drink?</pattern>
        <template><topicstar/>. </template>
    </category>
</topic>
```

Input: I like coffee.
Output: OK.
Input: WhatâĂŹs my favorite drink?
Output: Coffee.

See also: *topic(pattern)*, *thatstar*

6.1.63 uniq

[2.0]

Uniq is used in conjunction with *select*. Uniq searches the contents of the RDF file it references at startup and the RDF knowledge base added with addtriple to get the appropriate information.
As an RDF file, it refers to the file stored in the directory specified by the config.
The difference between select and uniq is that select returns the exact result of multiple matches, while uniq returns the result where duplicate matches are excluded.
See *RDF Support* for more information.

- Use case

```
<category>
    <pattern>* are * </pattern>
    <template>
        <addtriple>
            <subj><star /></subj>
            <pred>are </pred>
            <obj><star index="2"/></obj>
        </addtriple>
        Registered
    </template>
</category>
<category>
    <pattern>Search * * * </pattern>
    <template>
        <uniq>
            <subj><star /></subj>
            <pred><star index="2"/></pred>
            <obj><star index="3"/></obj>
        </uniq>
    </template>
</category>
```

Input: Cherry blossoms are Rosaceae
Output: Registered
Input: Strawberries are Rosaceae
Output: Registered
Input: Search Cerry blossoms?
Output: Rosaceae
Input: Search Rosaceae?
Output: Cherry blossom Strawberry

See also: *addtriple*, *deletetriple*, *select*, *RDF Support*, *File ManagementïijŽrdfs*

6.1.64 uppercase

[1.0]

Capitalises half-width alphabetic characters.

- Use case

```
<category>
    <pattern>Hello * </pattern>
    <template>
        Hello <uppercase><star /></uppercase>
    </template>
</category>
```

<uppercase />is equivalent to <uppercase><star /></uppercase>.

- Use case

```
<category>
    <pattern>Hello * </pattern>
    <template>
        Hello <uppercase />
    </template>
</category>
```

Input: Hello george washington
Output: Hello GEORGE WASHINGTON

See also: *lowercase*

6.1.65 vocabulary

[1.0]

Returns the number of distinct words known by the Bot.

- Use case

```
<category>
    <pattern>How many words do you know? </pattern>
    <template>
        <vocabulary />.
    </template>
</category>
```

Input: How many words do you know?
Output: 10000.

See also: *size*

6.1.66 video

[2.1]

If this uses the video element, returns the video information. You can specify a URL or a file name for the video.

- Use case

```
<category>
    <pattern>Video Display </pattern>
    <template>
        <video>https://url.for.video</video>
    </template>
</category>
```

6.1.67 word

[1.0]

A <word> tag as such does not exist, but each individual text word in a sentence is converted into a word node.

- Use case

```
<category>
    <pattern>HELLO</pattern>
    <template>Hi there!</template>
</category>
```

In this use case, HELLO is expanded to the word node.

6.1.68 xml

[1.0]

Like a word node, there is not <XML> tag as such however given that AIML is an XML dialect, and if the parser comes across an xml element it does not undertand, it stores it as pure XML.

- Use case

```
<category>
    <pattern>HIGHLIGHT * IN BOLD</pattern>
    <template>
        <bold><star /></bold>
    </template>
</category>
```

CHAPTER 7

Pattern Matching

AIML pattern matching provides wildcard matching for user input. There are ⋆, ^ , _ and # wildcards, each of which is used differently.

7.1 ⋆ Wildcard

⋆ can be used to extract user input words, and determines one or more matches.

```
<pattern>Hello ⋆</pattern>
```

In the example below, the grammar will match any sentence that starts with âĂŸHelloâĂŹ and 1 or more additional words.

```
Hello!
Hello, Mr. Yamada.
Hello, who is this ?
```

7.2 ^ Wildcard

^ can be used to specify zero or more word matches. That is, even if there is no word corresponding to the wildcard, the evaluation result is obtained even if only the expression of the specified word matches.

```
<pattern>Hello ^</pattern>
```

In the example below, the grammar will match any sentence that is either âĂŸHelloâĂŹ or a sentence that starts with âĂŸHelloâĂŹ and 1 or more additional words.

```
Hello
Hello!
Hello, Mr. Yamada.
Hello, who is this?
```

7.3 _ and # Wildcards

* and ^ evaluate with a lower priority than word matching, i.e.

```
<pattern> Hello, it's a nice day. </pattern>
<pattern> Hello ^</pattern>
<pattern> Hello * </pattern>
```

is defined, the input âĂIJHello, itâĂŹs a nice day.âĂİ matches the pattern at the top.

In contrast, _ and # are evaluated with higher precedence than word matching. _ matches one or more matches, as in * . # matches zero or more as in ^ .

```
<pattern> Hello, it's a nice day. </pattern>
<pattern> Hello _ </pattern>
<pattern> Hello # </pattern>
```

is defined, if the input is âĂIJHello, itâĂŹs a nice day. âĂIJ, matches # .

7.4 Preferred Word

A word specified by $ is evaluated before _ , # .

```
<pattern> Hello, $it's a nice day. </pattern>
<pattern> Hello _ </pattern>
<pattern> Hello # </pattern>
```

In case of above, the input âĂIJHello, itâĂŹs a nice day.âĂİ matches the pattern at the top.

7.5 Decision priority

The priority when specifying a wildcard in the pattern is as follows.

- $(Priority Word)
- # (0 or greater)
- _ (1 or more)
- (word)
- ^ (0 or greater)
- * (1 or more)

CHAPTER 8

File Management

Describes files used on the Dialog Platform.
In the following descriptions, the primary method of file management used by the Dialog Platform is local files.
In addition to local files, file management methods can be also managed in the database, but the configuration is the same.

The definition file extension is aiml for the scenario file and txt for the non-scenario configuration file extension.

8.1 Directory Configuration

The directory structure of the scenario files is as follows.

```
storage Various File Directories
âĬIJâĬĂâĬĂ braintree AIML Extraction File Directory
âĬIJâĬĂâĬĂ categories Scenario File Directory
âĬĆ   âĬĬâĬĂâĬĂ *.aiml
âĬIJâĬĂâĬĂ conversations Dialog History Information Directory
âĬĆ   âĬĬâĬĂâĬĂ *.conv
âĬIJâĬĂâĬĂ debug Dialog History Information Directory
âĬĆ   âĬIJâĬĂâĬĂ duplicates.txt
âĬĆ   âĬIJâĬĂâĬĂ errors.txt
âĬĆ   âĬĬâĬĂâĬĂ *.log
âĬIJâĬĂâĬĂ lookups File Directory for Substitution Process
âĬĆ   âĬIJâĬĂâĬĂ regex.txt
âĬĆ   âĬIJâĬĂâĬĂ denormal.txt
âĬĆ   âĬIJâĬĂâĬĂ gender.txt
âĬĆ   âĬIJâĬĂâĬĂ normal.txt
âĬĆ   âĬIJâĬĂâĬĂ person.txt
âĬĆ   âĬĬâĬĂâĬĂ person2.txt
âĬIJâĬĂâĬĂ maps map Element Usage File Directory
âĬĆ   âĬĬâĬĂâĬĂ *.txt
âĬIJâĬĂâĬĂ nodes map Element Usage File Directory
âĬĆ   âĬIJâĬĂâĬĂ pattern_nodes.conf
âĬĆ   âĬĬâĬĂâĬĂ template_nodes.conf
âĬIJâĬĂâĬĂ properties File Directory for Property lists
âĬĆ   âĬIJâĬĂâĬĂ nlu_servers.yaml
```

(continues on next page)

(continued from previous page)

```
âŤĆ   âŤIJâŤĂâŤĂ defaults.txt
âŤĆ   âŤŤâŤĂâŤĂ properties.txt
âŤIJâŤĂâŤĂ security File Directory for Security
âŤĆ   âŤŤâŤĂâŤĂ usergroups.yaml
âŤŤâŤĂâŤĂ sets Set Usage File Directory
    âŤŤâŤĂâŤĂ *.txt
```

8.2 Entity

On the Dialog Platform, entities are defined by the type of data to be stored, and each entity manages its own files.
The configuration of the file to be used is defined in the config file (config.yaml), and the data type to be used is specified by whether or not the following entities are set.
There are two types of entities, one for common use and one for specific nodes, but they are defined and managed in the same way.
Of the following entities, the data with âĂIJautomatic generationâĂİ of âĂIJNoâĂİ can be edited and used to expanded at startup.
The data with âĂIJsingle fileâĂİ of âĂIJYesâĂİ can specify only one target file, and the file with âĂIJNoâĂİ can specify multiple files by directory specification.

- Common Entities

Entity	Content	Single file	Automatic generation	Description
binaries	Tree data	No	Yes	Stores binary data for the search tree expanding AIML.
braintree	Tree data	No	Yes	Stores XML formatted data for the search tree expanding AIML.
categories	dialog AIML	No	No	Stores AIML files used to control dialog.
conversations	dialog history information	No	Yes	The history information file of dialog processing contents (Contain the values of variables) including the past of each user is stored.
defaults	Property List	Yes	No	Stores definition files for global variables (name) that are expanded at initial startup.
duplicates	Duplicate specification information	Yes	Yes	(For debugging) Stores an error information file when expanding AIML files.
errors	Error Information	Yes	Yes	(For debugging) Stores an error information file when expanding AIML files.
license_keys	License Key	Yes	No	Stores license key files required for external connections.
pattern_nodes	Class definition list	Yes	No	Stores the processing class definition file for each node of pattern.
postprocessors	Class definition list	Yes	No	Stores the processing class definition file for editing response statements.
preprocessors	Class definition list	Yes	No	Stores the processing class definition file for editing a user utterance in a request.
properties	Property List	Yes	No	Stores the property definition files used by the *bot* of pattern and the *bot* node of the template.
spelling_corpus	Spelling Information	No	No	Stores the corpus file used for spelling checker.
template_nodes	Class definition list	Yes	No	Stores the processing class definition file for each node in template.

- Entity for the Pattern node

Entity	Stored Content	Single file	Automatic generation	Description
regex_template	Regular expression list	Yes	No	Stores the regular expression list file used in the template specification of the *regex* node.
sets	Target word list	No	No	Stores the word list file used by the *set* node.

- Entity for the Template node

Entity	Stored Content	Single file	Automatic generation	Description
denormal	Transformation dictionary	Yes	No	Stores the dictionary file used for transformations in the *denormalize* node.
gender	Transformation dictionary	Yes	No	Stores the dictionary file used for transformations in the *gender* node.
learnf	Category list	No	YES	Stores the categories information created by process of the *learnf* node on a per-user basis.
logs	Log Information	No	YES	*log* information created by the processing of the node is stored for each user.
maps	Property List	No	No	Stores the dictionary file used for translation on the *map* node.
normal	Transformation dictionary	Yes	No	Stores the dictionary file used for transformations on the *normalize* node.
person	Transformation dictionary	Yes	No	Stores the dictionary file used for transformations on the *person* node.
person2	Transformation dictionary	Yes	No	Stores the dictionary file used for transformations on the *person2* node.
rdfs	RDF Data List	No	No	Stores the definition file of the *RDF* data to be processed by the RDF related nodes.
rdf_updates	RDF Update Information	Yes	Yes	Stores change history information for RDF data.
usergroups	Security Information	Yes	No	Stores the role definition file used by the *authorise* node.

8.3 Example of definition for using local files

8.3.1 Entity definition

The following is example of defining an entity when using a client named `console` .
In the client configuration, a section called `storage` has entities as a subsection.
Specifies the name of the I/O control method (Store) for each entity as the file management method.
Here, the store method name: `file` is specified.

```
console:
  storage:
      entities:
          binaries: file
```

(continues on next page)

(continued from previous page)

```
          braintree: file
          categories: file
          conversations: file
          defaults: file
          duplicates: file
          errors: file
          license_keys: file
          pattern_nodes: file
          postprocessors: file
          preprocessors: file
          properties: file
          spelling_corpus: file
          template_nodes: file
          regex_templates: file
          sets: file
          denormal: file
          gender: file
          learnf: file
          logs:   file
          maps: file
          normal: file
          person: file
          person2: file
          rdf: file
          rdf_updates: file
          usergroups: file
```

8.3.2 file storage engine definition

Subsection stores in the same storage section specifies the storage engine that performs the actual processing within the store.
Here, for the store name: file, âĂIJtype: file âĂİ specifies to use the storage engine to I/O the local file.
For local file I/O (file specification), the name of the actual storage engine is the `entity name + '_storage'`.
The configuration of the storage engine for each entity is done in the âĂŸconfigâĂŹ subsection as follows:

```
stores:
    file:
        type: file
        config:
          binaries_storage:
            file: ./storage/braintree/braintree.bin
          braintree_storage:
            file: ./storage/braintree/braintree.xml
          categories_storage:
            dirs: ./storage/categories
            subdirs: true
            extension: aiml
          conversations_storage:
            dirs: ./storage/conversations
          defaults_storage:
            file: ./storage/properties/defaults.txt
          duplicates_storage:
            file: ./storage/debug/duplicates.txt
          errors_storage:
            file: ./storage/debug/errors.txt
          license_keys_storage:
```

(continues on next page)

(continued from previous page)

```
    file: ./storage/licenses/license.keys
  pattern_nodes_storage:
    file: ./storage/nodes/pattern_nodes.conf
  postprocessors_storage:
    file: ./storage/processing/postprocessors.conf
  preprocessors_storage:
    file: ./storage/processing/preprocessors.conf
  properties_storage:
    file: ./storage/properties/properties.txt
  spelling_corpus_storage:
    file: ./storage/spelling/corpus.txt
  template_nodes_storage:
    file: ./storage/nodes/template_nodes.conf
  regex_templates_storage:
    file: ./storage/lookups/regex.txt
  sets_storage:
    dirs: ./storage/sets
    extension: txt
  denormal_storage:
    file: ./storage/lookups/denormal.txt
  gender_storage:
    file: ./storage/lookups/gender.txt
  learnf_storage:
    dirs: ./storage/learnf
  logs_storage:
    dirs: ./storage/debug
  maps_storage:
    dirs: ./storage/maps
    extension: txt
  normal_storage:
    file: ./storage/lookups/normal.txt
  person_storage:
    file: ./storage/lookups/person.txt
  person2_storage:
    file: ./storage/lookups/person2.txt
  rdfs_storage:
    dirs: ./storage/rdfs
    subdirs: true
    extension: txt
  rdf_updates_storage:
    dirs: ./storage/rdf_updates
  usergroups_storage:
    file: ./storage/security/usergroups.yaml
```

The definition in the âĂŸconfigâĂŹ subsection is a description method indicating the use of a single file only or multiple files depending on the entity involved.

8.3.3 For entities in a single file

For entities that specify a single file, the âĂŸfileâĂŹ attribute specifies the file path.

```
usergroups_storage:
    file: ./storage/security/usergroups.yaml
```

8.3.4 For entities that can use multiple files

In the case of an entity that can use multiple files, specify the following three attributes. However, for automatically generated entities, only the directory path is specified.

- dirs: Specifies the target file directory path.
- subdirs: true/false set whether to search subdirectories under the target file directory.
- extension: The extension of the file type to load.

```
categories_storage:
  dirs: ./storage/categories
  subdirs: true
  extension: aiml

conversations_storage:
  dirs: ./storage/conversations
```

There are following four storage engines that are not automatically generated and can specify multiple files.

- categories_storage
- sets_storage
- maps_storage
- rdfs_storage

8.4 Example of definition for using a database

An example of using Redis for database management is shown below.

8.4.1 Entity Definition

For the entity managed by Redis, store method name: `redis` is specified in the entities subsection of storage.

```
console:
  storage:
      entities:
          binaries: file
          braintree: file
          categories: redis
          ïijŽ
```

The entities that can input and output in Redis are as follows.

- binaries ïijŽ Stores binary data for the AIML expanded search tree.
- braintree ïijŽ Stores XML format data for the AIML expanded search tree.
- conversations ïijŽ History information of dialog processing contents (Contain the values of variables) including the past of each user is stored.
- duplicates ïijŽ (For debugging) Stores category duplicate information when extracting AIML files.
- errors ïijŽ (For debugging) Stores error information when extracting AIML files.
- learnf ïijŽStores categories information created by processing the *learnf* node for each user.
- logs ïijŽLog information created by processing the *log* node is stored for each user.

8.4.2 Redis Storage Engine Definition Example

In the storage section, the stores subsection specifies that I/O is to be made to Redis by âĂIJtype: redisâĂİ for the store name: redis.

In the âĂŸconfigâĂŹ subsection, specify the common parameters required for using Redis, and the actual I/O are performed by the storage engine for each entity, including the key setting.

```
stores:
   redis:
       type: redis
       config:
           host: localhost
           port: 6379
           password: xxxx
           db: 0
           prefix: programy
           drop_all_first: false
```

8.5 How to describe a definition file

Describes the definition files that are commonly used outside of AIML in editable files.

8.5.1 Property List File

The file specified by the following entities is described in the form of âĂŸName: ValueâĂŹ for the purpose of setting the value of variables, etc.

- defaults ïijŽ Defines the initial values of global variables (name).
- properties : Defines bot properties used in pattern *bot* and template *bot* nodes.
- maps ïijŽ Defines the key/value relationship used by the *map* node.
- nlu_servers : Configure the NLU server.

defaults

The `defaults` entity enables you to set the value of global variables (name) at initial startup for use in scenarios.
However, if there is a dialog information history for each user, this setting is disabled because the latest value on the history is reflected.
That is, it is only valid for new users.
The following example specifies that name variable: initial_variable should be set to value: âĂIJinitial valueâĂİ.

```
initial_variable: initial value
```

properties

The `properties` entity sets parameters that determine the default behavior of the bot, along with information references at the bot node.

Entity	Content	Description	Default Value
name	bot name	The value obtained when the *bot* name attribute is specified as name.	(If unspecified, the value of default-get)
birthdate	bot creation date	The value obtained when the *bot* name attribute is specified as name.	(If unspecified, the value of default-get)
grammar_version	grammar version	The value obtained when specifying grammar_version for the *bot* name attribute.	(If unspecified, the value of default-get)
app_version	app version	The value obtained when app_version is specified for the *bot* name attribute.	(If unspecified, the value of default-get)
default-response	default response	Return if no pattern matches.	unknown
default-get	default get	A string that can be retrieved by a get on an undefined variable.	unknown
joiner_terminator	end-of-sentence character	Specifies the character string to which the ending phrase of the response sentence is added. If not specified, nothing is given.	.
joiner_join_excluded	excluded end-of-sentence character	Specifies a string that is used to combine and exclude the joiner_terminator character when automatically appending the statement terminator with the joiner_terminator specification. If not specified, the character specified by joiner_terminator is appended.	.?!
splitter_split_char	sentence separator	Specifies the character used internally to devide a sentence. If the specified string is included in the statement, it is treated as a multiple statement, and response returns a string created by combining multiple response statements. However, metadata only returns what you set in the final statement. If not specified, the utterance is treated as one sentence.	.
punctuation_delimiter	delimiter	Specifies the character to be treated as a delimiter. Delimiters are excluded from matching, and the matching process is performed to the form created by excluding delimiters from utterance sentences and reponse sentences.	(None)

- Configuration Example

```
name:Basic Response
birthdate:March 01, 2019
```

(continues on next page)

(continued from previous page)

```
grammar_version:0.0.1
app_version: 0.0.1

default-response: Sorry, I didn't understand.
default-get:  I don't know.
version: v0.0.1

joiner_terminator: .
joiner_join_chars: .?!
splitter_split_chars: .
punctuation_chars: ;'",!()[]ïijŽâĂŹâĂİïijż
```

joiner_terminator

Specifies the character string to which the ending phrase of the response sentence is automatically added. If you specify âĂIJHelloâĂİ for template,

- Configuration Example

```
joiner_terminator: .
```

```
<category>
    <pattern> Hello </pattern>
    <template>Let's have a great day. </template>
</category>
```

Input: Hello
Output: LetâĂŹs have a great day.

The response of the dialog API response: To prevent a punctuation mark from being added at the end of the response, define the following. (You can disable it by leaving after âĂIJ:âĂİ blank.)

```
joiner_terminator:
```

If unspecified, no punctuation is added to the response.

Input: Hello
Output: LetâĂŹs have a great day

joiner_join_chars

Specifies a string that is used to combine and exclude the joiner_terminator character when automatically appending the statement terminator with the joiner_terminator specification.
In the case where joiner_join_chars is not specified, if you include a response such as âĂIJLetâĂŹs have a great day.âĂİ or âĂIJI feel great!âĂİ and combine the response with the characters specified by the joiner_terminator, it adds to the response the ending character, e.g., punctuation mark, specified by joiner_terminator even if the response already has the ending character and returns a sentence such as âĂIJLetâĂŹs have a great day..âĂİ or âĂIJI feel great!.âĂİ.

- Configuration Example

```
joiner_terminator: .
joiner_join_chars: .?!
```

```
<category>
    <pattern> Hello </pattern>
    <template>Let's have a great day.</template>
</category>
<category>
    <pattern>It's nice weather today, too.</pattern>
    <template> I feel great! </template>
</category>
```

Input: Hello
Output: LetâĂŹs have a great day.
Input: ItâĂŹs nice weather today, too
Output: I feel great!

If joiner_join_chars is unspecified, the characters specified in joiner_terminator are always combined.

```
joiner_terminator: .
joiner_join_chars:
```

Input: Hello
Output: LetâĂŹs have a great day..
Input: ItâĂŹs nice weather today, too
Output: I feel great!.

splitter_split_chars

Specifies the character used intrernally to divide a sentence. If the specified string is included in the statement, it is treated as a multiple statement, and returns a string created by combining multiple response statements. For the âĂIJHello. ItâĂŹs nice weather today, too.âĂİ utterance, if you specify âĂIJ. âĂİ for splitter_split_chars, it will process 2 statements: âĂIJHello.âĂİ and âĂIJItâĂŹs nice weather today, too.âĂİ.

- Configuration Example

```
joiner_terminator: .
splitter_split_chars:  .
```

```
<category>
    <pattern>Hello. </pattern>
    <template>It's nice weather today, too.</template>
</category>
<category>
    <pattern>Let's have a great day. </pattern>
    <template>I feel great.</template>
</category>
```

Input: Hello. ItâĂŹs nice weather today, too.
Output: LetâĂŹs have a great day. I feel great.

If splitter_split_chars is not specified, the utterance is not split, so matching with âĂIJHello. ItâĂŹs nice weather today, too.âĂİ as a single sentence is performed, and no response is given because there is no matching utterance in AIML described above.

```
splitter_split_chars:
```

Input: Hello. ItâĂŹs nice weather today, too.
Output: Sorry, I didnâĂŹt understand.

punctuation_chars

Specifies the character to be treated as a delimiter.Delimiters are excluded from the matching process and are excluded from utterance sentences. If you have the input âĂIJHello.âĂİ and âĂIJHelloâĂİ, the characters in punctuation_chars are ignored and treated as the same utterance.

- Configuration Example

```
punctuation_chars: ;'",!()[]ïijŽâĂŹâĂİïijż
```

```
<category>
    <pattern>Hello</pattern>
    <template>Let's have a great day</template>
</category>
```

Input: Hello.
Output: LetâĂŹs have a great day.
Input: Hello
Output: LetâĂŹs have a great day.

If punctuation_chars is not specified, âĂIJ.âĂİ is also matched, so âĂIJHello.âĂİ and âĂIJHelloâĂİ are treated as the different utterances.

```
punctuation_chars:
```

Input: Hello
Output: LetâĂŹs have a great day.
Input: Hello.
Output: Sorry, I didnâĂŹt understand.

maps

In the `maps` entity, information references in the map node are done by file names (exclude extension), so you can separate files by type of information.
The following example is from prefectural_office.txt, which lists the relationships between prefectures and prefectural capitals.

```
Tokyo Metropolis: Tokyo
Tokyo: Tokyo
```

(continues on next page)

(continued from previous page)

```
Kanagawa Prefecture: Yokohama City
Kanagawa: Yokohama City
Osaka Prefecture: Osaka City
Osaka: Osaka City
 ïijŽ
```

nlu_servers

`nlu_servers` entity sets URL for access (endpoint) and API key within the nlu node.
Please contact COTOBA DESIGN for endpoints and API keys to use within the nlu node.
(https://www.cotoba.net)
The following example shows how to set two URLs: the first URL has no API key set, and the second URL has an API key set.

```
nlu:
  - url: http://localhost:5201/run
  - url: http://localhost:3000/run
    apikey: test_key
```

8.5.2 Word list file

The file specified by the following entities lists the words and strings to be processed.

- sets ïijŽ Defines a list of words to be matched for the *set* node.

The `sets` entity is able to separate files for each type of information because the set node references the information by file name (exclude extension).
In the case of Japanese, a match may not be made depending on the result of word division performed during the match processing, so the match processing is performed as a character string instead of a word.
The following is an example of prefecture.txt listing the state/province names:.

```
Tokyo Metropolis
Tokyo
Kanagawa Prefecture
Kanagawa
Osaka Prefecture
Osaka
 ïijŽ
```

8.5.3 Regular expression list file

The file specified by the following entity should be of the form âĂŸregular expression name: regular expression stringâĂŹ.

- regex_templates ïijŽDefine regular expression list to be used in template specification of *regex* node.

The `regex_templates` entity is used to commonly define a regular expression string in a file for use in matching operations performed on regex nodes.
On the regex node side, the template attribute specifies the regular expression name. Regular expression descriptions must be specified on a word basis.

The description example is as follows.

```
realize:REALI[Z|S]E
elevator: elevator | lift
  ïijŽ
```

8.5.4 Conversion dictionary file

The file specified by the following entity is of the form âĂIJValue to be convertedâĂİ, âĂIJConverted valueâĂİ for the purpose of creating a table for translation.

- normal ïijŽDefines a list of conversion tables for the *normalize* node.
- denormal ïijŽDefines a list of conversion tables for the *denormalize* node.
- gender ïijŽDefines a list of conversion tables for the *gender* node.
- person ïijŽDefines a list of conversion tables for a *person* node.
- person2 ïijŽDefines a list of conversion tables for the *person2* node.

normal

In `normal` entities, symbols etc. in strings are converted into independent words by defining them as follows.
For alphabetic characters, if the 1st character of âĂIJValue to be convertedâĂİ is not âĂŸâĂŹ (Blank), all matches in the subject string are converted (character substitution), assuming symbol conversion.
On the other hand, if the 1st character is âĂŸ âĂŸ(Blank), the conversion is done by word (word substitution).
The normalize conversion is done in the following order: character conversion, word conversion. In both cases, blank is inserted before and after the converted text.
As an example of combining the 2 transforms, you could convert âĂIJMr.âĂİ to âĂİ mister dot âĂİ by converting âĂIJ.âĂİ to âĂIJdotâĂİ and âĂIJMrâĂİ to âĂIJmisterâĂİ.
Japanese words are converted to units.

```
".","dot"
"/","slash"
":","colon"
"*","_"
" Mr","mister"
" can t","can not"
```

denormal

A `denormal` entity, which is paired with a `normal` entity conversion, is defined as follows and convert words to character strings such as symbols.
For alphabetic characters, âĂIJValue to be convertedâĂİ is a word, and âĂIJConverted valueâĂİ includes the need for âĂŸ âĂŸ(Blank) when concatenating with the surrounding string. If there is no space, it is concatenated with the surrounding string.
As the opposite of normalize, if you want to change âĂIJmister dotâĂİ back to âĂIJMr.âĂİ, you can also specify âĂIJDotâĂİ as âĂIJ.âĂİ and âĂIJmister âĂIJin theâĂİ MrâĂİ as 2 separate specifications.

```
"dot","."
"slash","/"
"colon",":"
"_","*"
"mister dot"," Mr."
"can not"," can't "
```

gender,person,person2

The entities `gender` , `person` and `person2` specify word-by-word conversions; for example, gender defines:.

```
"he","she"
"his","her"
"him","her"
"her","him"
"she","he"
```

8.5.5 Other definition files

See *RDF Support* for the file format specified by the `rdfs` entity.

See *Security* for the file format specified by the `usergroups` entity.

The following entities contain implementation-dependent content (Class definitions, etc.) and therefore do not need to be described:.

- license_keys ïijŽ The license key files required for external connections, etc.
- pattern_nodes ïijŽ Contains the processing class definition files for each node of the pattern.
- postprocessors ïijŽ The processing class definition file for editing response statements.
- preprocessors ïijŽ A processing class definition file for editing user utterances in a request.
- spelling_corpus ïijŽ The corpus file from which the spelling checker is based.
- template_nodes ïijŽ The processing class definition file for each node of template.

CHAPTER 9

JSON

This is a function for using JSON in AIML. Used to leverage JSON data in AIML, including SubAgent, metadata, and intent recognition results.

The variable name specified in name/var/data is the variable name defined in get/set. get/set var is a local variable, get/set name is a global variable, and get/set data is a global variable that will be hold until deleteVariable is set true. In addition, system fixed variable names such as metadata and subagent return values can be used as name.

- Attribute

Parameter	Setting Value	Type	Required	Description
name		JSON name	Yes	Specifies the JSON that performs parse. var, name, or data must be set.
var		JSON name	Yes	Specifies the JSON that performs parse. var, name, or data must be set.
data		JSON name	Yes	Specifies the JSON that performs parse. var, name, or data must be set.
key		Key specification	No	Specifies the key to manipulate the JSON data.
item		Get key name	No	Used to get keys from the JSON data. Specify this attribute to get keys instead of values.
function		Function name	No	Describes the processing for JSON.
	len	Function name	No	If the JSON property is an array, get the array length. For JSON objects, get the number of elements in the JSON object.
	delete	Function name	No	Deletes the target property. If index is specified in the array, the element is deleted.
	insert	Function name	No	Specifies the addition of a value to the JSON array.
index		Index	No	Specifies the index when getting the JSON data. If the target is an array, it is the array number. In a JSON object, the keys are counted from the beginning. When setting or modifying JSON data, only specify the arrays.
type		Configuration Type	No	Sets a numeric value, a boolean character, or null as a string in a data update.

- Child element

If an AIML variable is specified as a value, it cannot be specified in the attribute, so it can also be specified as a child element. The behavior is the same as the attribute. If the same attribute name and child element name can be specified, the child element setting takes precedence. However, only when key is used in a child element, name and var are distinguished and treated separately.

Parameter	Type	Required	Description
function	Function name	No	Describes the processing for the JSON. See the attribute for the function.
index	Index	No	Specifies the index when getting the JSON data. If the target is an array, it is the array number. In a JSON object, the keys are counted from the beginning. When setting or modifying JSON data, only specify arrays.
item	Get Key name	No	Used to get keys from JSON data. Specify this attribute to get keys instead of values.
key	Key specification	No	Specifies the key to manipulate the JSON data.

9.1 Basic usage

The name, data, or var attribute of the JSON element specifies the target json data. When the contents of a JSON element is set, the value of the target JSON data is set and updated. If the contents of the JSON element was not set, gets the value of that key, unless delete it. If the retrieval fails, the value of âĂIJdefault-getâĂİ set in Config, etc. is returned like the *get* . Describes how to get and set up data for JSON as follows. This is an example of using the subagent return value. The subagent return value is assumed to be held in the variable `__SUBAGENT__.profile`.

```
{
    "profile": {
        "name": "Ichiro",
        "birthday": "01011970",
        "age": 48,
        "sex": "male",
        "home": { "address": "Minato-ku, Tokyo","coordinates": "x,y" },
        "family": {
            "father": "Taro",
            "mother": "Hanako",
            "brother": [ "Jiro", "Kikuko" ],
            "children": [ "Saburo", "Momoko" ]
        },
        "relative": {
            "grandfather": [ "Shiro", "Goro" ],
            "grandmother": [ "Kuriko", "Umeko" ],
            "cousin": ["Kotaro", "Sakiko"],
            "grandchildren": [ "Aki" ]
        },
        "friend": [ "Kazuhiro", "Kyoko" ],
        "hobby": [ "golf", "baseball" ],
        "medical history": [ "high blood pressure", "hay fever" ],
        "allergy": [ "milk", "egg" ]
    }
}
```

9.1.1 How to specify attributes/child elements when getting the JSON data

Describes how to specify attributes and child elements. Although they are described differently, the results are the same.

Getting Key Specification Values

If this gets the value of âĂIJfatherâĂİ, the following description is given. For attributes, specify the keys you want to get separated by `.`. For child elements, enter the key you want to get in the contents of `<key>`.

```
<json var="__USER_METADATA__.profile.family.father" />
<!-- <json var="__USER_METADATA__.profile"><key>family.father </key> </json> The␣
↪same as above -->
```

Getting Arrays

When you get the value of âĂIJbrotherâĂİ which is an array, as with the getting value, you get the specified array by describing the key you want to get by separating `.` or the child element `<key>`. Get [âĂIJJiroâĂİ, âĂIJKikukoâĂİ] as the result of the execution.

```
<json var="__USER_METADATA__.profile.family.brother" />
<!-- <json var="__USER_METADATA__.profile.family"> <key> brother</key> </json> The␣
↪same as above -->
```

Getting Array Length

Set function to âĂIJlenâĂİ to get the array length. If it is âĂIJbrotherâĂİ, returns `2`.

```
<json var="__USER_METADATA__.profile.family.brother" function="len"/>
<!-- <json var="__USER_METADATA__.profile.family.brother"><function>len</function>
↪</json>   The same as above -->
```

Getting Array Values

When retrieving the contents of an array, specify the array index. Gets the 0 th value for âĂIJbrotherâĂİ `Jiro`.

```
<json var="__USER_METADATA__.profile.family.brother" index="0"/>
<!-- <json var="__USER_METADATA__.profile.family.brother"><index>0</index></json> ␣
↪The same as above -->
```

Getting JSON Object Element Count

When you want to get the number of elements in a JSON object, specify the `len` for the function. Gets `11` elements if specified the profile.

```
<json var="__USER_METADATA__.profile" function="len"/>
<!-- <json var="__USER_METADATA__.profile"><function>len</function></json>  The␣
↪same as above -->
```

Getting a key for a JSON object

When gets the key of a JSON object, specify a `key` for item. The fifth profile key gets the `family` .

```
<json var="__USER_METADATA__.profile" item="key" index="5"/>
<!-- <json var="__USER_METADATA__.profile"><item>key</item><index>5</index></json>␣
↪ The same as above -->
```

9.1.2 Update JSON Data

When changing the value of a key already in the JSON data, describe the value in the contents of the JSON element. Update âĂIJAddressâĂİ to âĂIJShin-YokohamaâĂİ by describing the contents. If empty, specify "" .

```
<json var="__USER_METADATA__.profile.home.address">Shin-Yokohama</json>
<json var="__USER_METADATA__.profile.home.coordinates">""</json>
<!-- <json var="__USER_METADATA__.profile.home"><key>address</key>Shin-Yokohama</
↪json>
    <json var="__USER_METADATA__.profile.home"><key>coordinates</key>""</json> ␣
↪The same as above -->
```

Before Update

```
"home": {"address" : "Minato-ku, Tokyo", "coordinates" : "x,y"},
```

After Update

```
"home": {"address" : "Shin-Yokohama", "coordinates" : ""},
```

Append to JSON Data

To add a new key, add it by specifying a value for the new key.

```
<json var="__USER_METADATA__.profile.zip-code">222-0033</json>
<!-- <json var="__USER_METADATA__.profile"><key>zip-code</key>222-0033</json>  The␣
↪same as above -->
```

Before Update

```
{
    "profile":{
        "medical history": ["high blood pressure", "hay fever"],
        "allergy" : ["milk", "egg"]
    }
}
```

After Update

```
{
    "profile":{
        "medical history": ["high blood pressure", "hay fever"],
        "allergy" : ["milk", "egg"],
        "zip-code" : "222-0033"
    }
}
```

Changing the contents of an array

To change the contents of an array, specify the array index. If change the 0 value of âĂIJhobbyâĂİ, get as follows.

```
<json var="__USER_METADATA__.profile.hobby" index="0">soccer </json>
<!-- <json var="__USER_METADATA__.profile"><key>hobby</key><index>0</index>soccer</
↪json>  The same as above-->
```

(continues on next page)

(continued from previous page)

Before Update

```
{
    "profile":{
        "hobby": ["golf", "baseball"]
    }
}
```

After Update

```
{
    "profile":{
        "hobby": ["soccer", "baseball"]
    }
}
```

Modifying Arrays

To modify all the elements of âĂIJhobbyâĂİ in the array, enclose individual elements in double quotes and separate them with commas.

```
<json var="__USER_METADATA__.profile.hobby">"soccer","fishing","movie watching"</
↪json>
<!-- <json var="__USER_METADATA__.profile"><key>hobby</key>"soccer","fishing",
↪"movie watching"</json>  The same as above -->
```

is specified,

Before Update

```
"hobby": ["golf","baseball"],
```

After Update

```
"hobby": ["soccer","fishing","movie watching"],
```

Adding Elements to Arrays

To add an element to the array, specifies the function to insert and index to set the insertion point. To add a value at the beginning, specify 0 for index. A minus index represents the trailing index value, e.g., index = -1 appends the value to the end of the array. Enclose individual elements in double quotes and separate them with commas.

In the example below, index = âĂIJ0âĂİ is specified for âĂIJhobbyâĂİ that is an array, and the value is added to the beginning of the array.

```
<json var="__USER_METADATA__.profile.hobby" function="insert" index="0">"soccer",
↪"fishing", "movie watching", "travel (overseas, domestic)" </ json>
<!-- <json var="__USER_METADATA__.profile"><key>hobby</key><function>insert</
↪function><index>0</index>"soccer","fishing", "movie watching", "travel (overseas,
↪ domestic)"</json>   The same as above -->
```

Before Update

```
"hobby": ["golf","baseball"],
```

After Update

```
"hobby": ["soccer", "fishing", "movie watching", "travel (overseas, domestic)",
↪"golf", "baseball"],
```

In the example below, the value is added to the end of the array by specifying index = âĂIJ2âĂİ that is the number of the elements in array âĂIJhobbyâĂİ.

```
<json var="__USER_METADATA__.profile.hobby" function="insert" index="2">"soccer ,
↪"fishing", "movie watching", "travel (overseas,domestic)"</json>
<!-- <json var="__USER_METADATA__.profile"><key>hobby</key><function>insert</
↪function><index>2</index>"soccer , "fishing", "movie watching", "travel␣
↪(overseas,domestic)"</json>   The same operation as above -->
```

Before Update

```
"hobby": ["golf", "baseball"],
```

After Update

```
"hobby": ["golf", "baseball", "soccer", "fishing", "movie watching", "travel␣
↪(overseas, domestic)"],
```

Similarly, index = âĂIJ-1âĂİ adds a value to the end of the array.

```
<json var="__USER_METADATA__.profile.hobby" function="insert" index="-1">"soccer ,
↪"fishing", "movie watching", "travel (overseas,domestic)"</json>
<!-- <json var="__USER_METADATA__.profile"><key>hobby</key><function>insert</
↪function><index>-1</index>"soccer , "fishing", "movie watching", "travel␣
↪(overseas,domestic)"</json>  The same as above -->
```

Before Update

```
"hobby": ["golf", "baseball"],
```

After Update

```
"hobby": ["golf", "baseball", "soccer", "fishing", "movie watching", " travel␣
↪(oversea, domestic)"],
```

Creating Arrays

To create an array, set elements separated by commas or specify the function with insert and set index to 0 or -1. (Not created if index is not 0 or -1)

The following example creates a âĂIJeducationâĂİ as a new array element.

```
<json var="__USER_METADATA__.profile.education"> "A Elementary School", "B Junior␣
↪High School", "C high school", "D University" </json>
<!-- <json var="__USER_METADATA__.profile.education" function="insert" index="0">
↪"A Elementary School", "B Junior High School", "C high school", "D University" </
↪json> The same as above -->
<!-- <json var="__USER_METADATA__.profile.education" function="insert" index="-1">
↪"A Elementary School", "B Junior High School", "C high school", "D University" </
↪json> The same as above -->
<!-- <json var="__USER_METADATA__.profile"><key>education</key><function>insert</
↪function><index>0</index> "A Elementary School", "B Junior High School", "C high␣
↪school", "D University" </json> The same as above -->
```

After creation

```
"education":["A Elementary School","B Junior High School","C high school","D␣
↪University"]
```

In the case of a single element, a JSON object can be created without specifying insert in the function, but it cannot be changed to an array by specifying insert. If you have more than one element, you need to create it as an array element.

```
<!-- <json var="__USER_METADATA__.profile.education" function = "insert" index = "0
↪"> "D University" </json> The same as above -->
<!-- <json var="__USER_METADATA__.profile.education" function = "insert" index = "-
↪1"> "D University" </json> The same as above -->
<!-- <json var="__USER_METADATA__.profile"> <key> education </key> <function>␣
↪insert </function> <index> 0 </index> "D University" </json> The same as above --
↪>
```

After the update, a one-element array is created.

```
"education" : ["D University"]
```

If insert is not specified for function,

```
<json var="__USER_METADATA__.profile.education"> "D University"</json>
```

After update, a JSON object is created.

```
"education" : "D University"
```

9.1.3 Deleting JSON Data

Deleting Array Elements

To delete an array element, set function to delete and index to the index of the target value. Deletes the value at the specified index. A negative index represents a trailing index value, and index = -1 deletes the last value in the array.

```
<json var="__USER_METADATA__.profile.hobby" index="0" function="delete" />
<!-- <json var="__USER_METADATA__.profile.hobby"><index>0</index><function>delete</
↪function></json>  The same as above -->
<json var="__USER_METADATA__.profile.hobby" index="-1" function="delete" />
<!-- <json var="__USER_METADATA__.profile.hobby"><index>-1</index><function>delete
↪</function></json>  The same as above -->
```

Before Update

```
"hobby": ["golf", "baseball", "reading"],
```

After Update

```
"hobby": ["baseball"],
```

Deleting Keys

To delete keys, set function to delete. Deletes the specified key and value.

Delete the âĂIJhobbyâĂİ key and value by specifying âĂIJdeleteâĂİ for âĂIJfunctionâĂİ.

```
<json var="__USER_METADATA__.profile.hobby" function="delete" />
<!-- <json var="__USER_METADATA__.profile.hobby"><function>delete</function></json>
↪  The same as above -->
```

Before Update

```
"friend": ["Kazuhiro", "Kyoko"],
"hobby": ["golf", "baseball"],
"medical history": ["high blood pressure", "hay fever"],
```

After Update

```
"friend": ["Kazuhiro", "Kyoko"],
"medical history": ["high blood pressure", "hay fever"],
```

9.1.4 Specify JSON format data

To set the element to JSON format data, specify the JSON string format enclosed in curly braces: {}. Although it can be specified by array operation, it is necessary to specify one element at a time because the description in JSON string format cannot be specified in the list.

```
<json var="__USER_METADATA__.profile.family.father">{"name": "Taro", "age": 80}</
↪json>
<!-- <json var="__USER_METADATA__.profile.family"><key>father</key>{"name": "Taro",
↪ "age": 80}</json>  The same as above -->
```

Before Update

```
"father":"Taro",
```

After Update

```
"father": {
       "name": "Taro",
       "age": 80
       },
```

Note that this function sets the contents of the JSON key, and it is not possible to set the data in JSON format directly to the variable as shown below. To *set* a variable to JSON formatted content, use the set element.

```
<!-- Not Configurable --><json var = "__USER_METADATA__"> {"profile" : {"family" :
↪{"father" : {"name": "Taro", "age": 80}}}} </json>

<!-- Configurable --> <set var="__USER_METADATA__">{"profile": {"family": {"father
↪": {"name": "Taro", "age": 80}}}}</set>
```

See: *SubAgent* ,:doc:*metadata<Metadata>* ,:doc:*intent recognition<NLU>*

9.1.5 Handling of Numeric, Boolean, and null

AIML only handles strings, not JSON numbers, truth values, or nulls directly. The operation for setting and getting these contents in the JSON element is explained.

Settings

If only a numerical value is specified at the time of setting, it will be treated internally as a numerical value.
If a Non-numeric string is contained, it will be treated as strings.
When âĂIJtrueâĂİ or âĂIJfalseâĂİ indicating boolean value is set, it will be registered in JSON as boolean value.
If the value is set to null, set null to JSON.
If set a numeric value, true, false, or null as a character string, specify `string` for type.

Example:

```
<json var="__USER_METADATA__.profile.age">30</json>
<json var="__USER_METADATA__.profile.fullage">31 years old</json>
<json var="__USER_METADATA__.profile.birthday" type="string">01011970</json>
<json var="__USER_METADATA__.profile.self-Introduction">null</json>
<json var="__USER_METADATA__.profile.telephone-authentication">true</json>
<json var="__USER_METADATA__.profile.mail-authentication" type="string">false</
↪json>
```

The setting result of the example is the following JSON.

```
{
    "profile": {
        "age": 30,
        "fullage": "31 years old",
        "birthday": "01011970",
        "self-introduction": null,
        "telephone-authentication": true,
        "mail-authentication": "false"
    }
}
```

Getting

When a JSON element gets a numeric value, a boolean, or a null, these are obtained as a string. If it is a numeric value, it will be obtained as a numeric string, if it is true or false, it will be obtained as a string of âĂIJtrueâĂİ or âĂIJfalseâĂİ, if it is null, it will be obtained as a string of âĂIJnullâĂİ.

Example:

```
<json var="__USER_METADATA__.profile.age"/>
<json var="__USER_METADATA__.profile.fullage"/>
<json var="__USER_METADATA__.profile.birthday"/>
<json var="__USER_METADATA__.profile.telephone-authentication"/>
<json var="__USER_METADATA__.profile.mail-authentication"/>
<json var="__USER_METADATA__.profile.self-introduction"/>
```

The scenario designer should be aware of the source data type because each retrieval value is obtained as a string, as shown below.

```
30
31 years old
01011970
true
false
null
```

CHAPTER 10

NLU

The definition of the pattern element to perform intent recognition processing.
If `nlu` is defined as a child element of pattern, it is evaluated for matching the intent of the intent recognition.
In the dialog control, the rule base intention interpretation is carried out according to the description of the scenario, and the response is returned according to the result of the evaluation by pattern matching. If there is no matching pattern, it performs dialog control using the intent result of advanced intent interpretation.
This is to give priority to what the scenario developer describes rather than the result of intent recognition.
As an exception, if there is a category in the pattern that only describes wildcards, the wildcard-only matching process will be performed after both scenario description matching and intent recognition matching have failed to match.
Even if you define nlu for a child element, the contents of the pattern element do the usual pattern evaluation.
For nlu element attributes, see *nlu* .

10.1 Basic usage

The following example shows the return of intent and slot information in the following format from the intent recognition.

```
{
    "intents": [
        {"intent": "transportation", "score": 0.9 },
        {"intent": "aroundsearch", "score": 0.8 }
    ],
    "slots": [
        {"slot": "departure", "entity": "Tokyo", "score": 0.85, "startOffset": 3,
↪"endOffset": 5 },
        {"slot": "arrival", "entity": "Kyoto", "score": 0.86, "startOffset": 8,
↪"endOffset": 10 },
        {"slot": "departure_time", "entity": "2018/11/1 19:00", "score": 0.87,
↪"startOffset": 12, "endOffset": 14 },
        {"slot": "arrival_time", "entity": "2018/11/1 11:00", "score": 0.88,
↪"startOffset": 13, "endOffset": 18 }
    ]
}
```

10.1.1 Intent Matching

Describes nlu as a child element of pattern in AIML.
In the example below, the intent returned from the intent recognition is âĂŸtransportationâĂŹ, it will match and return a response.

```
<category>
    <pattern>
        <nlu intent="transportation" />
    </pattern>
    <template>
        Is it transfer information?
    </template>
</category>
```

Input: I want to go from Tokyo to Kyoto.
Output: Is it transfer information?

10.1.2 Pattern that is a candidate intent but does not match

In the following example, the intent for intent recognition is âĂŸaroundsearchâĂŹ.
âĂŸaroundsearchâĂŹ is a candidate intent, but it is not a maximum likelihood candidate, so it does not match.

```
<category>
    <pattern>
        <nlu intent="aroundsearch" />
    </pattern>
    <template>
        Around search?
    </template>
</category>
```

Input: I want to go from Tokyo to Kyoto.
Output: NO_MATCH

10.1.3 Matching when it is not a maximum likelihood candidate

Set the attribute `maxLikelihood` to `false` if you want to match the intent including âĂŸaroundsearchâĂŹ even if âĂŸaroundsearchâĂŹ is not a maximum likelihood candidate.
If `maxLikelihood` is not specified, the behavior is the same as specifying `true` .

```
<category>
    <pattern>
        <nlu intent="aroundsearch" maxLikelihood="false" />
    </pattern>
    <template>
        Around search?
    </template>
</category>
```

Input: I want to go from Tokyo to Kyoto.
Output: Around search?

10.1.4 Match with score specification

Describes the match condition by the score value of the intent.
Five types of attributes, scoreGt, scoreGe, score, scoreLe, and scoreLt, can now be specified, and the settings are as follows.
Also, if this attribute is specified, `maxLikelihood` will be treated as `false` for comparison matching by confidence.

Parameter Name	Meaning	Description
scoreGt	>	Matches if the confidence level of the target intent is greater than the specified value.
scoreGe	>=	Matches if the confidence level of the target intent is greater than or equal to the specified value.
score	=	Matches if the confidence level of the target intent is equal to the specified value.
scoreLe	<=	Matches if the confidence level of the target intent is less than or equal to the specified value.
scoreLt	<	Matches if the confidence level of the target intent is less than the specified value.

The operation when scoreXx is specified is the following matching.

```
<nlu intent="transportation" scoreGt="0.9"/>  Do not match transportation.
<nlu intent="transportation" scoreGe="0.9"/>  Matching transportation.
<nlu intent="transportation" score="0.9"/>    Matching transportation.
<nlu intent="aroundsearch" scoreLe="0.8"/>  Matching aroundsearch.
<nlu intent="aroundsearch" scoreLt="0.8"/>  Do not match aroundsearch.
```

As shown in the following example, it is possible to describe multiple conditions depending on the result of Intent Recognition, but the order of category in the AIML file is the first to be applied.
When multiple AIML files are used, the AIML expansion process is performed in ascending order by directory name and file name, so the order must be kept in mind.
(If a subdirectory is used, the files in the upper directory are processed before moving to the file processing under the subdirectory.)

```
<category>
    <pattern><nlu intent="transportation" scoreGe="0.8"/></pattern>
    <template> Is it transfer information?</template>
</category>

<category>
    <pattern><nlu intent="aroundsearch" scoreGe="0.8"/></pattern>
    <template> Around search?</template>
</category>
```

10.1.5 Intent matches and wildcards

The following is an example of calling a chat subagent if it does not match the rule base or intent recognition result. If there is a category with only wildcards as pattern, it will match after matching both the scenario description and the intent recognition.

```
<aiml>
    <category>
        <pattern>Hello </pattern>
        <template>Hello </template>
    </category>

    <category>
        <pattern><nlu intent="aroundsearch" /></pattern>
        <template>
            Around search.
        </template>
    </category>

    <category>
        <pattern>
            *
        </pattern>
        <template>
            <sraix service="chatting"><get var="__USER_UTTERANCE__" /></sraix>
        </template>
    </category>
</aiml>
```

Input: Hello
Output: Hello
Input: Convenience stores around here
Output: Around search.
Input: chatting
Output: ItâĂŹs the result of a chatting.

10.2 Getting NLU Data

The NLU data is expanded into the variable __SYSTEM_NLUDATA__ . Here is an example of using a JSON element to get data for *an example of a intent recognition result* .

```
<category>
    <pattern>
        <nlu intent="transportation" />
    </pattern>
    <template>
        <think>
                <set var="slot"><json var="__SYSTEM_NLUDATA__.slots"><index>1</
↪index></json></set>
                <set var="entity"><json var="slot.entity" /></set>
                <set var="score"><json var="slot.score" /></set>
        </think>
        <get var="entity"/> has a score of <get var="score" />.
    </template>
</category>
```

Input: I want to go from Tokyo to Kyoto.
Output: Tokyo has a score of 0.85.

See also: *nlu*ãĂĄ *JSON element*

10.3 Getting NLU Intent

Use *nluintent* to get the contents of an intent with template. The NLU processing result explains how to get intent information on the assumption that the following result is obtained.

```
{
    "intents": [
        {"intent": "restaurantsearch", "score": 0.9 },
        {"intent": "aroundsearch", "score": 0.4 }
    ],
    "slots": [
        {"slot": "genre", "entity": "Italian", "score": 0.95, "startOffset": 0,
↪"endOffset": 5 },
        {"slot": "genre", "entity": "French", "score": 0.86, "startOffset": 7,
↪"endOffset": 10 },
        {"slot": "genre", "entity": "Chinese", "score": 0.75, "startOffset": 12,
↪"endOffset": 14 }
    ]
}
```

This example gets the intent information processed by the NLU. The map is supposed to be defined to count up values. Keep the intent count in the intentCount and get the intent name and score for each slot until variable count equals the intentCount.

```
<category>
    <pattern>
        <nlu intent="restaurantsearch"/>
    </pattern>
    <template>
        <think>
          <set var="count">0</set>
          <set var="intentCount"><nluintent name="*" item="count" /></set>
        </think>
        <condition>
            <li var="count"><value><get var="intentCount" /></value></li>
            <li>
                intent:<nluintent name="*" item="intent"><index><get var="count" />
↪</index></nluintent>
                score:<nluintent name="*" item="score"><index><get var="count" /></
↪index></nluintent>
                <think>
                    <set var="count"><map name="upcount"><get var="count" /></map>
↪</set>
                </think>
                <loop/>
            </li>
        </condition>
    </template>
</category>
```

Input: Look for Italian, French or Chinese.
Output: intent:restaurantsearch score:0.9 intent:aroundsearch score:0.4

See also: *nluintent*

10.4 Getting NLU Slots

Use *nluslot* to get the contents of the slot resulting from NLU processing with template. The NLU processing result explains how to get slot information on the assumption that the following result is obtained.

```
{
    "intents": [
        {"intent": "restaurantsearch", "score": 0.9 },
        {"intent": "aroundsearch", "score": 0.4 }
    ],
    "slots": [
        {"slot": "genre", "entity": "Italian", "score": 0.95, "startOffset": 0,
↪"endOffset": 5 },
        {"slot": "genre", "entity": "French", "score": 0.86, "startOffset": 7,
↪"endOffset": 10 },
        {"slot": "genre", "entity": "Chinese", "score": 0.75, "startOffset": 12,
↪"endOffset": 14 }
    ]
}
```

This example gets the slot information processed by the NLU. The map is supposed to be defined to count up values. Keep the slot count in the slotCount and get the slot name, entity and score for each slot until variable count equals the slotCount.

```
<category>
    <pattern>
        <nlu intent="restaurantsearch" />
    </pattern>
    <template>
        <think>
          <set var="count">0</set>
          <set var="slotCount"><nluslot name="*" item="count" /></set>
        </think>
        <condition>
            <li var="count"><value><get var="slotCount" /></value></li>
            <li>
                slot:<nluslot name="*" item="slot"><index><get var="count" /></
↪index></nluslot>
                entity:<nluslot name="*" item="entity"><index><get var="count" /></
↪index></nluslot>
                score:<nluslot name="*" item="score"><index><get var="count" /></
↪index></nluslot>
                <think>
                    <set var="count"><map name="upcount"><get var="count" /></map>
↪</set>
                </think>
                <loop/>
            </li>
        </condition>
    </template>
</category>
```

Input: Look for Italian, French or Chinese.
Output: slot:genre entity:Italian score:0.95 slot:genre entity:French score:0.86 slot:genre entity:Chinese score:0.75

See also: *nluslot*

CHAPTER 11

metadata

11.1 Summary

The metadata information set in the body of the dialog API request can be used as variable data (Text or JSON) for the scenario.
It can be also set to metadata information in the body of a dialog API response as variable data for a scenario.

11.2 Using metadata set in the dialog API

Retains metadata information specified as JSON elements in the dialog API request expanded into variables available in the dialog scenario.
The `metadata` element in JSON passed from the API expands to the variable name `__USER_METADATA__` .
If the metadata element is in JSON format, you can get the element in __USER_METADATA__ by using the json *json* tag.
The contents of __USER_METADATA__ remain in effect until the dialog API response is returned. To use the contents of __USER_METADATA__ continuously, assign it to a variable separately.

11.2.1 How metadata is expanded into variables

Expands the data passed as metadata from the dialog API into the variable __USER_METADATA__.
You can set up metadata with text data and JSON data. How to handle each scenario is explained in the following.

In the following description, information is obtained using the myService subagent, and the following results are returned as JSON data.
The returned data is expanded into the variable `__SUBAGENT__.myService`. For more information about subagent federation, see *SubAgent* .

```
{
    "transportation": {
```

(continues on next page)

(continued from previous page)

```
        "station": {
            "departure": "Tokyo",
            "arrival": "Kyoto"
        },
        "time": {
            "departure": "11/1/2018 11:00",
            "arrival": "11/1/2018 13:30"
        },
        "facility": [" Rokuon-ji Temple "," Kiyomizu-dera Temple "," Fushimi Inari-
↪taisha Shrine "]
    }
}
```

How to Treat Text Data

For text data, you can get the contents with the *get* tag for the variable __USER_METADATA__.

Description for the case where a string is given as metadata in a dialog API call is shown below.

```
{
    "locale": "en-US",
    "time": "2018-07-01T12:18:45+09:00",
    "topic": "*",
    "utterance": "subagent Hello",
    "metadata": "Metadata Test"
}
```

The scenario is described as follows.
You can pass the contents of __USER_METADATA__ to the myService subagent by getting the contents with the *get* tag and setting it as the contents of the *sraix* tag.
The subagentâĂŹs return data is held in `__SUBAGENT__.myService` (See *SubAgent* for more details), which gets the value in JSON by specifying the elementâĂŹs key.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <think>
                <sraix service="myService">
                    <star />
                    <get var="__USER_METADATA__" />
                </sraix>
                <set name="departure"><json var="__SUBAGENT__.myService.
↪transportation.station.departure" /></set>
                <set name="arrival"><json var="__SUBAGENT__.myService.
↪transportation.station.arrival" /></set>
            </think>
            Searches for the route from<get name="departure" /> to <get name=
↪"arrival" />.
        </template>
    </category>
</aiml>
```

If the user utterance is âĂIJsubagent HelloâĂİ, the data developed below is passed to the subagent.

argument number	contents passed to the subagent
The first argument	Hello
The second argument	Metadata Test

How to treat it as JSON data

The following describes the case where JSON data is provided as metadata in a dialog API call.

```
{
    "locale": "en-US",
    "time": "2018-07-01T12:18:45+09:00",
    "topic": "*",
    "utterance": "subagent Hello",
    "metadata": {"arg1": "value1", "arg2": "value2", "arg3": "value3"}
}
```

If metadata is JSON, it can be handled as JSON format data by using *json* tag.
If you want to get a JSON value and set it individually, you can pass it to the subagent by keying it in the *json* tag and setting it as the content of the *sraix tag*.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <think>
                <sraix service="myService">
                    <star />
                    <json var="__USER_METADATA__.arg1" />
                    <json var="__USER_METADATA__.arg2" />
                    <json var="__USER_METADATA__.arg3" />
                </sraix>
                <set name="departure"><json var="__SUBAGENT__.myService.
↪transportation.station.departure" /></set>
                <set name="arrival"><json var="__SUBAGENT__.myService.
↪transportation.station.arrival" /></set>
            </think>
            Searches for the route from <get name="departure" /> to <get name=
↪"arrival" />.
        </template>
    </category>
</aiml>
```

If the user utterance is âĂIJsubagent HelloâĂİ, the data developed below is passed to the subagent.

Argument number	Contents passed to the subagent
The first argument	Hello
The second argument	value1
The third argument	value2
The fourth argument	value3

11.2.2 How to Pass All Metadata to Subagent

JSON data passed as metadata from the dialog API can be passed directly to the subagent as JSON.
By specifying __USER_METADATA__ for the attribute of the *json* tag, all data set to metadata is got and passed to the subagent.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <think>
                <sraix service="myService">
                    <star />
                    <json var="__USER_METADATA__" />
                </sraix>
                <set name=departure><json var="__SUBAGENT__.myService.
↪transportation.station.departure" /></set>
                <set name=arrival><json var="__SUBAGENT__.myService.transportation.
↪station.arrival" /></set>
            </think>
            Searches for the route from <get name = 'departure'> to <get name =
↪'arrival'>.
        </template>
    </category>
</aiml>
```

If the user utterance is âĂIJsubagent helloâĂİ, the JSON specified in the second argument to the myService subagent is passed as is.

Argument number	Contents passed to the subagent
Argument number	Hello
The second argument	{âĂŸarg1âĂŹ: âĂŸvalue1âĂŹ, âĂŸarg2âĂŹ: âĂŸvalue2âĂŹ, âĂŸarg3âĂŹ: âĂŸvalue3âĂŹ}

11.3 Setting Metadata to Return to the dialog API

Specify the metadata element to be set in the dialog API response by setting the data in the scenario return metadata variable __SYSTEM_METADATA__.
The metadata element of the response can be set either text data orJSON data, and in the following how to handle it in each scenario is shown.

11.3.1 Handling as text data

In the example below, the text âĂIJdepartureâĂİ in the data obtained from the myService subagent is set to the return metadata variable.
From the JSON returned by the subagent, get the element (Text) of Departure: âĂIJstation.departureâĂİ and set it to __SYSTEM_METADATA__.
This returns the text data as a metadata element in the dialog API response.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <think>
                <sraix service="myService">
                    <star />
                </sraix>
                <set var="__SYSTEM_METADATA__"><json var="__SUBAGENT__.myService.
↪transportation.station.departure" /></set>
            </think>
            The departure in the metadata is set.
        </template>
    </category>
</aiml>
```

Input: subagent Tokyo
Output: The departure in the metadata is set.
Metadata content: âĂIJTokyoâĂİ

11.3.2 Handling JSON Data

In the example below, JSON data obtained from the myService subagent is set to the returned metadata variable.
The *json* tag sets the entire JSON data returned from the subagent to __SYSTEM_METADATA__.
This returns the JSON data as a metadata element in the dialog API response.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <think>
                <sraix service="myService">
                    <star />
                </sraix>
                <set var="__SYSTEM_METADATA__"><json var="__SUBAGENT__.myService" /
↪></set>
            </think>
            The sub agent processing result is set to the metadata.
        </template>
    </category>
</aiml>
```

Input: subagent Tokyo
Output: The sub agent processing result is set to the metadata.
metadata content: {âĂIJtransportationâĂİ: {âĂIJstationâĂİ: {âĂIJdepartureâĂİ: âĂIJTokyoâĂİ, âĂIJarrivalâĂİ: âĂIJKyotoâĂİ}, âĂIJtimeâĂİ: {âĂIJdepartureâĂİ: âĂIJ2018/11/1 11:00âĂİ, âĂIJarrivalâĂİ: âĂIJ2018/11/1 13:30âĂİ}, âĂIJfacility: [âĂIJRokuon-ji TempleâĂİ, âĂIJKiyomizu-dera TempleâĂİ, âĂİ Fushimi Inari-taisha ShrineâĂİ]}}

See also: *dialogAPI*, *dialog API data variable usage*, *JSON*, *SubAgent*

CHAPTER 12

Using Variables in dialog API Data

12.1 Summary

Describes how to use the data specified in the *dialog API*

12.2 Using Variables in dialog API Data

The data provided by the dialog API is kept expanded to variables that can be used in the dialog scenario. The retention period is until the response is returned to the client.To keep the content continously, assign a variable separately. If the variable is unset during a dialog API request, âĂŸNoneâĂŹ is returned.

The variable names that hold the dialog API data are as follows.

Item Name	Variable Name	Type	Description
Locale	__USER_LOCALE__	string	The language code specified in the dialog API `locale`.
Time Information	__USER_TIME__	string	The time information specified in the dialog API `time`.
User ID	__USER_USERID__	string	The user ID specified in the dialog API `userId`.
User Utterance	__USER_UTTERANCE__	string	The user utterance specified in the dialog API `utterance`.
Metadata	__USER_METADATA__	string	Metadata specified in the dialog API `metadata`. It is held as a string as a variable. Used as JSON data when handling with *json* tag.

12.3 Usage Examples

If the dialog API data is given in the dialog API from the client,

```
{
    "locale": "en-US",
    "time": "2018-07-01T12:18:45+09:00",
    "userId": "E8BDF659B007ADA2C4841EA364E8A70308E03A71",
    "topic": "*",
    "utterance": "Hello",
    "metadata": {"arg1": "value1", "arg2": "value2"}
}
```

The handling in the scenario is as follows.

```
<aiml>
    <category>
        <pattern> Hello </pattern>
        <template>
            <get var="__USER_LOCALE__" /> ,
            <get var="__USER_TIME__" /> ,
            <get var="__USER_USERID__" /> ,
            <get var="__USER_UTTERANCE__" />
        </template>
    </category>
</aiml>
```

Input: Hello
Output: en-US,2018-07-01T12:18:45+09:00,E8BDF659B007ADA2C4841EA364E8A70308E03A71,Hello

For information about metadata, see *metadata* .

CHAPTER 13

SubAgent

13.1 Summary

The subagent (SubAgent) is an external service invoked using the sraix element. Describes how to invoke the external service specified by sraix.

There are three ways to call external services.

- **Generic REST interface** If âĂIJbotIdâĂİ, âĂIJserviceâĂİ is not specified in the attribute, the external service is invoked as a REST API using the child elements of host, header, query, body, etc.
- **Public bot calls on the dialog platform** If the attribute is âĂIJbotIdâĂİ, it calls the bot exposed on the dialog platform.
- **Custom External Service Implementation** If the attribute is âĂIJserviceâĂİ, the custom implementation process is used to invoke external services.
- Attribute

Parameter	Type	Required	Description
Unspecified		No	Generic REST API calls with child elements.
service	String	No	The service name of the custom external service.
botId	String	No	The bot ID exposed on the dialog platform.
host	String	No	Dialog platform or host name.
default	String	No	Response statement for external call failure.

The body of the response from the SubAgent expands to the local variables (var) available in the scenario (AIML). Because AIML can only process text, it does not support binary data return values.

If the response body contains JSON, the json tag can get the parameters inside JSON.

The content expanded to the local variable (var) is valid only in the range of the AIML category, so it should be assigned to the global variable (name/data) separately when it is used in other categories (contains srai) continuously.

Also, if the subagent is called multiple times within a category, the return value from the subagent may be overwritten, so assign the required response contents to variables individually.

The âĂIJdefaultâĂİ attribute can be specified in any of three ways to call an external service, and in the event where the call fails, returns the set contents as the return value.
The âĂIJhostâĂİ attribute is only available on the dialog platform and is used differently from the âĂIJhostâĂİ of the child element.

13.2 Generic REST interface

If the attribute âĂIJbotIdâĂİ, âĂIJserviceâĂİ of sraix is unspecified, the external service is invoked as a REST API using the child elements of host, header, query, body, etc.

13.2.1 Send

- Child element

Tag	Required	Description
host	Yes	The URL to connect to.
method	No	HTTP method. When not specified, GET is used. POST/PUT/DELETE are also supported.
query	No	Specify parameters for query with an associative array.
header	No	Specify keys and values for header with an associative array.
body	No	Specifies what is set for the body.

At the time of request, specify as follows. Set a character string to body.

```
<category>
    <pattern>XXX</pattern>
    <template>
        <think>
            <sraix>
                <host>http://www.***.com/ask</host>
                <method>POST</method>
                <query>"userid":"1234567890","q":"question"</query>
                <header>"Authorization":"yyyyyyyyyyyyyyyyy","Content-Type":
↪"application/json;charset=UTF-8"</header>
                <body>{"question":"Ask this question"}</body>
            </sraix>
        </think>
    </template>
</category>
```

Sent

```
POST /ask?userid=1234567890&q=question HTTP/1.1
Host: www.***.com
Content-Type: application/json;charset=UTF-8
Authorization: yyyyyyyyyyyyyyyyy

{
    "question":"Ask this question"
}
```

When specifying the `metadata` specified by the dialog API in the body, get `__USER_METADATA__` in the json tag and set it in the child element âĂIJbodyâĂİ.

```
<category>
    <pattern>XXX</pattern>
    <template>
```

(continues on next page)

(continued from previous page)

```
        <think>
            <sraix>
                <host>http://somehost.com</host>
                <method>POST</method>
                <query>"userid":"1234567890","q":"question"</query>
                <header>"Authorization":"yyyyyyyyyyyyyyyyy","Content-Type":
↪"application/json;charset=UTF-8"</header>
                <body><json var="__USER_METADATA__" /></body>
            </sraix>
        </think>
    </template>
</category>
```

13.2.2 Receive

Returns the body contents of the receive result as the result of the sraix.
Because AIML can only handle text, it does not support binary bodies.
The receive results are also expanded to the local variable (var): __SUBAGENT_BODY__ . By specifying <get var = âĂIJ__ SUBAGENT_BODY __âĂİ> in get, the string of the body can be obtained.
The contents of local variables (var) are held in category units, so it should be assigned to the global variable (name/data) separately when you use the contents of responses continuously.
Also, if the generic REST interface is called more than once within a category, the __SUBAGENT_BODY__ is overwritten, so assign the required response to a variable.

If the body content is JSON, the *json* tag can get parameters inside JSON. The contents of the body is,

```
{
    "transportation": {
        "station": {
            "departure": "Tokyo",
            "arrival": "Kyoto"
        },
        "time": {
            "departure": "11/1/2018 11:00",
            "arrival": "11/1/2018 13:30"
        },
        "facility": ["Rokuon-ji Temple", "Kiyomizu-dera Temple", "Fushimi Inari␣
↪Taisha Shrine"]
    }
}
```

then

```
<json var="__SUBAGENT_BODY__.transportation.station.departure" />
<json var="__SUBAGENT_BODY__.facility" function="len" />
<json var="__SUBAGENT_BODY__.facility"><index>1</index></json>
```

With the description, the internal information of the body can be obtained by JSON tag.

13.2.3 Response for communication failure (default)

In case of communication failure, the string specified by the âĂIJdefaultâĂİ attribute is returned as the return value of sraix. The same is true for access to both the *custom external service implementations*. The following is an example of using a bot call exposed on a dialog platform.

```
<category>
    <pattern> bot status check * </pattern>
    <template>
        The status of <star /> is <sraix service = "sameBot" default =
↪"communication failed"> <star /> </sraix>.
     </template>
</category>
```

Input: bot status check public bot
Output: The status of public bot is communication failed.

13.3 Public bot calls on dialog platforms

When âĂIJbotIdâĂİ is specified in the attribute of sraix, bot (public bot) published on the dialog platform is called. The âĂIJbotIdâĂİ is the ID of the bot and specified by the dialog platform, and the content of sraix is sent as an input sentence (utterance sentence) to the public bot. The content returned from the public bot is in JSON format specified in the received data of the *dialog API* , and the return value of sraix returns the response element within it.

13.3.1 Send

The following example is for a public bot that returns âĂIJOkâĂİ as a response.

```
<category>
    <pattern> bot status check * </ pattern>
    <template>
        The status of <star /> is <sraix botId="sameBot"> <star /> </sraix>.
     </template>
</category>
```

Input: bot status check public bot
Output: The status of public bot status is OK.

Describe the parameters as chile elements when using the public bot. The content of the child element is sent as the content of the body of the *dialog API* .
See *dialog API* , about the meaning of child elements.
If not specified, some elements inherit what is specified in the dialog API. If the element does not need to inherit anything, child elements must be configured (null string, etc.).
(For sraix, if no user ID is specified, uses another ID generated from the user ID specified in the dialog API.)
When using a public bot, sraix has no child elements to configure user utterances, and is treated as a utterance that informs the public bot of the contents of sraix.

- Child element

Item	Tag Name	Type	Required	Inheritation from the dialog API
Locale	locale	string	No	Yes
Time information	time	string	No	Yes
User ID	userId	string	Yes	No (Generate a different user ID)
Topic ID	topic	string	No	No
Delete Task Variable	deleteVariable	boolean	No	No
Metadata	metadata	string	No	Yes
Configure	config		No	No
LogLevel	logLevel	string	No	No

In the following example, topic, deleteVariable, metadata, and config are specified in a scenario and locale and time are specified by inheriting the contents of the collerâĂŹs request.

```
<category>
    <pattern> bot status check * </pattern>
    <template>
        <think>
            <json var="askSubagent.zip">222-0033</json>
            <json var="config.logLevel">debug</json>
        </think>
        <sraix botId="sameBot">
            <star/>
            <topic>*</topic>
            <deleteVariable>true</deleteVariable>
            <metadata><json var="askSubagent"/></metadata>
            <config><json var="config"/></config>
        </sraix>
    </template>
</category>
```

Input: bot status check zip search
Output: Shin-Yokohama

13.3.2 Receive

The âĂIJresponseâĂİ element in the body (JSON format) received from the public bot will be set as the return value of sraix. In the following example, if the received data from sameBot is

```
HTTP/1.1 200 Ok
Content-Type: application/json;charset=UTF-8

{
    "response": "Hello, it's nice weather today, too.",
    "topic": "greeting"
}
```

then the following AIML results are,

```
<category>
     <pattern>*</pattern>
     <template>
        <sraix botId="sameBot"><star/></sraix>
     </template>
</category>
```

Input: Hello
Output: Hello, itâĂŹs nice weather today, too.

Reception from a public bot is expanded to the local variable (var) `__SUBAGENT_EXTBOT__.botID` and can be obtained with the get. In addition, the variable is held in category units, so it must be assigned to the global variable (name/data) to use it continuously.

```
<json var="__SUBAGENT_EXTBOT__.sameBot" />
```

Because the body content from the public bot is JSON, you can get the parameters inside JSON with *json* tag. If the `metadata` content is JSON, the JSON tag can also retrieve the parameters in `metadata` .

If the metadata content is

```
"metadata":{"broadcaster":"OBS","title":"afternoon news"}
```

then

```
<json var="__SUBAGENT_EXTBOT__.sameBot.response" />
<json var="__SUBAGENT_EXTBOT__.sameBot.utterance" />
<json var="__SUBAGENT_EXTBOT__.sameBot.topic" />
<json var="__SUBAGENT_EXTBOT__.sameBot.metadata" />
<json var="__SUBAGENT_EXTBOT__.sameBot.metadata.broadcaster" />
<json var="__SUBAGENT_EXTBOT__.sameBot.metadata.title" />
```

can get the return value from the public bot and the metadata information.

13.4 Custom External Service Implementation

If the attribute is set to âĂIJserviceâĂİ, then the custom implementation can be used to call external services. Custom external services are implemented individually by inheriting the following base classes, since each service (SubAgent) used has its own call method that needs to be implemented.

```
programy.services.service.Service
```

The implementation of the processing class creates a class that inherits from the base class and implements a process that returns a result string as the ask_question() function, using the âĂIJquestionâĂİ argument corresponding to the utterance data. When linking with external service, REST communication function will be implemented in ask_question().

```
from programy.services.service import Service

class StatusCheck(Service):
   __metaclass__ = ABCMeta

   def __init__(self, config: BrainServiceConfiguration):
       self._config = config

   @property
   def configuration(self):
       return self._config

   def load_additional_config(self, service_config):
       pass

   @abstractmethod
   def def ask_question(self, client_context, question: str):
       return "OK"
```

Then add the entry for the custom external service to the `services` section of the configuration definition: config.yaml so that it can be used as a service name for sraix.

```
myService:
    classname: programy.services.myService.StatusCheck
    url: http://myService.com/api/statuscheck
```

For use with AIML, specify the entry name of the custom external service in the sraix attribute âĂIJserviceâĂİ, as in the following example. As the processing of the custom external service for sraix, load the class defined by the classname of the entry of the custom external service, and call the function: ask_question(). The return value of the function: ask_question() is the result of sraix.

```
<category>
    <pattern>Status Check *</pattern>
    <template>
        The status of <star /> is <sraix service="myService"><star/></sraix>.
    </template>
</category>
```

Input: Status Check custom
Output: The status of custom is OK.

13.4.1 Arguments and Return Values to Custom External Services

Arguments

The sraix service=âĂİmyServiceâĂİ is a custom external service call that treats the inside of a sraix element as an argument. Argument definitions depend on the argument I/F of each custom external service and must be implemented for each service. The following example uses an external service called myService and assumes that four arguments are set.

```
<aiml>
    <!-- sub agent execute -->
    <category>
        <pattern>subagent *</pattern>
        <template>
            <set var="text">
                <sraix service="myService">
                    <star/>
                    <json var="__USER_METADATA__.arg1" />
                    <json var="__USER_METADATA__.arg2" />
                    <json var="__USER_METADATA__.arg3" />
                </sraix>
            </set>
            <think>
                <set name="departure"><json var="__SUBAGENT__.myService.
↪transportation.station.departure" /></set>
                <set name="arrival"><json var="__SUBAGENT__.myService.
↪transportation.station.arrival" /></set>
            </think>
            Searches for <get name="departure"> through <get name="arrival">.
        </template>
    </category>
</aiml>
```

Return Values

The return value of the custom external service, that is, the return value of ask_question() function of the individual implementation, expands to the local variable (var) `__SUBAGENT__.service name`. These variables are held in category units, so they must be assigned to global variables (name/data) when it is used continuously.

The format stored in the variable can be text or JSON, and the custom implementation cannot use binaries.

In the following example, the return value of the operation on myService is expanded to `__SUBAGENT__.myService` , but its contents are in JSON format,

```
{
    "transportation": {
        "station": {
            "departure" :"Tokyo",
            "arrival: "Kyoto"
        },
        "time": {
            "departure": "2018/11/1 11:00",
            "arrival": "2018/11/1 13:30"
        },
        "facility: ["Rokuon-ji Temple", "Kiyomizu-dera Temple", "Fushimi Inari-
↪taisha Shrine"]
    }
}
```

then

```
<json var="__SUBAGENT__.myService.transportation.station.departure" />
<json var="__SUBAGENT__.myService.transportation.station.arrival" />
```

As a, you can use the json tag to get internal information about the body. `__SUBAGENT__.myService` is text, it will be retrieved with the get tag.

See Also: *metadata*, *Dialog API*, *JSON*

CHAPTER 14

Extensions

14.1 Summary

Extensions is a feature that allows you to implement extensions separately. It provides a mechanism to call additional functionality by writing the full Python path in the path of the extension element.

14.2 Implementing Custom Extensions

The extension must inherit from the base class.

```
programy.extensions.Extension
```

The following is an example of implementation. The method that executes the contents of the extension element is execute, and implements the execute method with bot, clientid, and data as arguments. In data, the content of the extension element is set with a space delimiter.

```
from programy.extensions.base import Extension

class GeoCodeExtension(Extension):
    __metaclass__ = ABCMeta

    @abstractmethod
    def execute(self, context, data):
        latlng = getGeoCode(data)
        if latlng is not None:
            return "%s,%s"%(
                latlng.latitude, latlng.longitude
            )

        return None
```

When using a custom extension using extension in AIML, describe the extension element as shown in the example below. As the node processing of the AIML extension tag, the class defined by the attribute path is loaded and execute() is called. The return value of execute() is the result of template. The return value of execute() is the result of template.

```
<category>
    <pattern>
        What are the coordinates of * ?
    </pattern>
    <template>
         Latitude and longitude are,
         <extension path="programy.extensions.geocode.geocode.GeoCodeExtension">
             <star />
         </extension>
         .
    </template>
</category>
```

Input: What are the coordinates of Tokyo Station?
Output: The latitude and the longitude are 35.681236, 139.767125.

CHAPTER 15

OOB

For each `OOB(Out of Band)` node to be used, it is necessary to implement a Python class and associate it with config. The OOB implementation base class is defined as follows.

Parameter Name	Description	
default		
	classname	Defines the python path for the default OOB.
Name of the OOB		The name set in this item is the OOB name specified in AIML. In the example, âĂŸemailâĂŹ is an OOB tag.
	classname	Defines the path to python for the OOB to implement. The child elements to be implemented in OOB are defined in the implementation class and need not be specified in config.

```
import xml.etree.ElementTree as ET

class OutOfBandProcessor(object):

    def __init__(self):
        return

    # Override this method to extract the data for your command
    # See actual implementations for details of how to do this
    def parse_oob_xml(self, oob: ET.Element):
        return

    # Override this method in your own class to do something
    # useful with the command data
    def execute_oob_command(self, bot, clientid):
        return ""

    def process_out_of_bounds(self, bot, clientid, oob):
        if self.parse_oob_xml(oob) is True:
            return self.execute_oob_command(bot, clientid)
        else:
            return ""
```

If there is an OOB function to send e-mail, AIML using OOB is described as follows.

```
<oob>
    <email>
        <to>Destination </to>
        <subject>Subject </subject>
        <body>Articles </body>
    </email>
</oob>
```

The implementation class looks like the following. Get and hold child elements with parse_oob_xml() method, and implement the process to actually send mail by send_oob_command () method.

```
class EmailOutOfBandProcessor(OutOfBandProcessor):

    def __init__(self):
        OutOfBandProcessor.__init__(self)
        self._to = None
        self._subject = None
        self._body = None

    def parse_oob_xml(self, oob: ET.Element):
        for child in oob:
            if child.tag == 'to':
                self._to = child.text
            elif child.tag == 'subject':
                self._subject = child.text
            elif child.tag == 'body':
                self._body = child.text
            else:
                logging.error ("Unknown child element [%s] in email oob"%(child.
↪tag))

            if self._to is not None and \
                self._subject is not None and \
                self.body is not None:
                return True

            logging.error("Invalid email oob command")
            return False

    def execute_oob_command(self, bot, clientid):
        logging.info("EmailOutOfBandProcessor: Emailing=%s", self._to)
        return ""
```

For OOB settings, configure the following settings in config.yaml.

```
oob:
    default:
        classname: programy.oob.default.DefaultOutOfBandProcessor
    email:
        classname: programy.oob.email.EmailOutOfBandProcessor
```

For more information on OOB settings, see *OOB settings*.

CHAPTER 16

RDF support

RDF (Resource Descriptor Framework) is a W3C recommendation for describing meta-content based on XML.
The basic data structure of RDF is a set of 3 pieces of data called âĂIJtripleâĂİ, each of which consists of a subject (subject), a predicate (predicate), and an object (object).
For more information, see RDF Concepts And Abstract .

The related AIML tags are addtriple, deletetriple, select, tuple, uniq.

16.1 AIML triple file

âĂIJtripleâĂİ is a single line of text with the subject, predicate and object separated by a colon âĂŸ:âĂŹ. In this program, this line is referred to as an element.
Explain the RDF definition with an airplane example.
In this definition, âĂIJsubjectâĂİ is listed as âĂIJAIRPLANEâĂİ and a number of âĂIJpredicateâĂİ describe each value (âĂIJobjectâĂİ).

```
AIRPLANE:hasPurpose:to transport us through the air
AIRPLANE:hasSize:9
AIRPLANE:hasSpeed:12
AIRPLANE:hasSyllables:1
AIRPLANE:isa:Aircraft
AIRPLANE:isa:Transportation
AIRPLANE:lifeArea:Physical
```

16.2 AIML RDF Tag

AIML defines creation, deletion, and search tags such as âĂIJaddtripleâĂİ, âĂIJdeletetripleâĂİ, âĂIJselectâĂİ, and âĂIJuniqâĂİ.
You can also use tags like âĂIJsetâĂİ, âĂIJgetâĂİ, âĂIJfirstâĂİ, and âĂIJrestâĂİ to process search results.

16.2.1 Load Element

This program has a function to load pre-prepared files.
Reads the RDF definition file under the conditions set in rdf_storage of config.

```
client_type:
    storage:
        entities:
            rdf: file

        stores:
            file:
                type: file
                config:
                    rdf_storage:
                        dirs: ../storage/rdfs
                        subdirs: true
                        extension: txt
```

This section uses a definition file that describes the following.

```
ant:legs:6
ant:sound:scratch
bat:legs:2
bat:sound:eee
bear:legs:4
bear:sound:grrrrr
buffalo:legs:4
buffalo:sound:moo
cat:legs:4
cat:sound:meow
chicken:legs:2
chicken:sound:cluck cluck
dolphin:legs:0
dolphin:sound:meep meep
fish:legs:0
fish:sound:bubble bubble
```

16.2.2 Element Generation

A new âĂIJtripleâĂİ can be dynamically added using not only loading data but also the addtriple element.

```
<addtriple>
  <subj>Subject</subj><pred>Predicate</pred><obj>Object</obj>
</addtriple>
```

An example of adding animal characteristics.

```
<addtriple>
  <subj>cow</subj><pred>sound</pred><obj>moo</obj>
</addtriple>
<addtriple>
  <subj>dog</subj><pred>sound</pred><obj>woof</obj>
</addtriple>
```

However, data added with addtriple is not persisted.

16.2.3 Delete Element

Any data including data added by the addtriple element or read from the file can be deleted using the deletetriple element. This includes not only elements added by addtriple, but also elements read from the file.

```
<deletetriple>
  <subj>cow</subj><pred>sound</pred><obj>moo</obj>
</deletetriple>
<deletetriple>
  <subj>ant</subj><pred>sound</pred><obj>scratch</obj>
</deletetriple>
```

If you specify three elements (subject, predicate, and object), only the elements that match all of them will be deleted.
If only subject and predicate are specified, matching elements are removed, regardless of the value of object.
If only subject is specified, all elements matching that subject are removed.

16.2.4 Search

The select element is used to search RDF.

Simple Search

In the case of a simple search, if you specify subject, predicate, and object as the contents of the <q> element, the contents registered as matching results will be returned as a list.

```
<select>
    <q><subj>dog</subj><pred>sound</pred><obj>woof</obj></q>
</select>
```

If that information exists, the following results are returned.
[[[[âĂIJsubjâĂİ, âĂIJDOGâĂİ], [âĂIJpredâĂİ, âĂIJSOUNDâĂİ], [âĂIJobjâĂİ, âĂIJwoofâĂİ]]]]

If you want to retrieve only one specific element, the following can be described.

```
<select>
    <q><subj>dog</subj><pred>sound</pred><obj>?</obj></q>
</select>
```

In this case, the following result indicating the contents of the element specified with âĂIJ?âĂİ Is returned.
[[[âĂIJ?âĂİ, âĂIJwoofâĂİ]]]

Searching by Variable

If you want to return multiple elements or receive a list of matching elements, you must use variables.
Variables are defined in the contents of the vars tag and are prefixed with the variable name âĂIJ?âĂİ.
The query <q> allows you to set a variable in the tag of an element of triple.
In the following case, variable:?x is subject, variable:?y is predicate, variable:?z is the object to store.

```
<select>
    <vars>?x ?y ?z</vars>
    <q><subj>?x</subj><pred>?y</pred><obj>?z</obj></q>
</select>
```

You can get the data corresponding to the variable from all the triples that match the specified data.
The following example searches for the number of feet (legs) in an animal.

```
<select>
    <vars>?x ?y</vars>
    <q><subj>?x</subj><pred>legs</pred><obj>?y</obj></q>
</select>
```

If the search results match, you will get the following results.
[[[âĂIJ?xâĂİ, âĂIJANTâĂİ], [âĂIJ?yâĂİ, âĂIJ6âĂİ]], [[âĂIJ?xâĂİ, âĂIJBATâĂİ], [âĂIJ?yâĂİ, âĂIJ2âĂİ]], [[âĂIJ?xâĂİ, âĂIJBEARâĂİ], [âĂIJ?yâĂİ, âĂIJ4âĂİ]], [[âĂIJ?xâĂİ, âĂIJBUFFALOâĂİ], [âĂIJ?yâĂİ, âĂIJ4âĂİ]], [[âĂIJ?xâĂİ, âĂIJCATâĂİ], [âĂIJ?yâĂİ, âĂIJ4âĂİ]], [[âĂIJ?xâĂİ, âĂIJCHICKENâĂİ], [âĂIJ?yâĂİ, âĂIJ2âĂİ]], [[âĂIJ?xâĂİ, âĂIJDOLPHINâĂİ], [âĂIJ?yâĂİ, âĂIJ0âĂİ]], [[âĂIJ?xâĂİ, âĂIJFISHâĂİ], [âĂIJ?yâĂİ, âĂIJ0âĂİ]]]

Complex Condition Search

If you need to perform more complex searches, you can chain multiple queries, each joined as a âĂŸandâĂŹ query.
There are 2 types of queries. <q> tag returns results matching its condition and <notq> tag returns results not matching its condition.

```
<select>
    <vars>?x ?y ?z</vars>
    <q><subj>?x</subj><pred>legs</pred><obj>?y</obj></q>
    <notq><subj>?z</subj><pred>legs</pred><obj>0</obj></notq>
</select>
```

Data Retrieval

The select element is used to create a data set, as in the SQL SELECT statement.
The following example stores the result of a select element in tuples using the âĂŸsetâĂŹ tag.

```
<set var="tuples">
    <select>
        <vars>?x ?y</vars>
        <q><subj>?x</subj><pred>sound</pred><obj>?y</obj></q>
    </select>
</set>
```

In this case, the following contents are set in tuples.

[[[âĂIJ?xâĂİ, âĂIJBATâĂİ], [âĂIJ?yâĂİ, âĂIJeeeâĂİ]], [[âĂIJ?xâĂİ, âĂIJBEARâĂİ], [âĂIJ?yâĂİ, âĂIJgrrrrrâĂİ]], [[âĂIJ?xâĂİ, âĂIJBUFFALOâĂİ], [âĂIJ?yâĂİ, âĂIJmooâĂİ]], [[âĂIJ?xâĂİ, âĂIJCATâĂİ], [âĂIJ?yâĂİ, âĂIJmeowâĂİ]], [[âĂIJ?xâĂİ, âĂIJCHICKENâĂİ], [âĂIJ?yâĂİ, âĂIJcluck cluckâĂİ]], [[âĂIJ?xâĂİ, âĂIJDOLPHINâĂİ], [âĂIJ?yâĂİ, âĂIJmeep meepâĂİ]], [[âĂIJ?xâĂİ, âĂIJFISHâĂİ], [âĂIJ?yâĂİ, âĂIJbubble bubbleâĂİ]], [[âĂIJ?xâĂİ, âĂIJDOGâĂİ], [âĂIJ?yâĂİ, âĂIJwoofâĂİ]]]

To get the data generated from the âĂŸselectâĂŹ element described above, use the âĂŸtupleâĂŹ element to get the âĂŸgetâĂŹ tag as a child element.

```
<get var="?x">
    <tuple>
        <get var="tuples" />
    </tuple>
</get>
<get var="?y">
    <tuple>
        <get var="tuples" />
    </tuple>
</get>
```

In this example, the âĂŸvarâĂŹ attribute of get specifies the variable âĂIJ?xâĂİ specified in the select element. By specifying âĂIJtuplesâĂİ (list object) that stores the result of the select element in the tuple tag, data corresponding to the variable âĂIJ?xâĂİ can be obtained from âĂIJtuplesâĂİ.

As a result, the following contents can be obtained from the variable âĂIJ?xâĂİ.

BAT BEAR BUFFALO CAT CHICKEN DOLPHIN FISH DOG

Similarly, the following contents can be obtained from the variable âĂIJ?yâĂİ.

eee grrrrr moo meow cluck cluck meep meep bubble bubble woof

Also, by using the first tag and rest tag for âĂIJtuplesâĂİ, partial results can be obtained as follows.

```
<get var="?x">
    <tuple>
        <first><get var="tuples" /></first>
    </tuple>
</get>
<get var="?y">
    <tuple>
        <rest><get var="tuples" /></rest>
    </tuple>
</get>
```

As a result, in the first tag (obtaining first data), the value obtained from the variable âĂIJ?xâĂİ is as follows.

BAT

Similarly, the value obtained from the variable âĂIJ?yâĂİ with the rest tag (obtaining data other than the head) is as follows.

grrrrr moo meow cluck cluck meep meep bubble bubble woof

CHAPTER 17

Security

There are two levels of security definitions: authentication and authorisation.

17.1 Authentication

Authentication is implemented within the core code brain. ask_question is called for each user inquiry. One of its parameters is âĂŸclientidâĂŹ, which is an ID for each user making utterance. It is recommended that you perform the necessary authentication procedures separately. In the following example, the âĂŸclientidâĂŹ is set to âĂŸconsoleâĂŹ for the console client, but if multiple people are accessing the site, such as by REST, it is necessary to assign a unique ID.

```
def ask_question(self, bot, clientid, sentence) -> str:

    if self.authentication is not None:
        if self.authentication.authenticate(clientid) is False:
            logging.error("[%s] failed authentication!")
            return self.authentication.configuration.denied_srai
```

The authentication service is defined by a dynamically loaded class, and its base class is defined as followsãĂĆãĂĆ

```
class Authenticator(object):

    def __init__(self, configuration: BrainSecurityConfiguration):
        self._configuration = configuration

    @property
    def configuration(self):
        return self._configuration

    def get_default_denied_srai(self):
        return self.configuration.denied_srai

    def authenticate(self, clientid: str):
        return False
```

This implementation is a simple one that matches IDs in the âĂŸauthorisedâĂŹ list. For more advanced authentication, an external service can be implemented, but it is recommended that additional authentication procedures be performed when using this program on the server.

```
class ClientIdAuthenticationService(Authenticator):

    def __init__(self, configuration: BrainSecurityConfiguration):
        Authenticator.__init__(self, configuration)
        self.authorised = [
            "console"
        ]

    # Its at this point that we would call a user auth service, and if that passes
    # return True, appending the user to the known authorised list of user
    # This is a very naive approach, and does not cater for users that log out,␣
↪invalidate
    # their credentials, or have a TTL on their credentials
    # #Exercise for the reader......
    def _auth_clientid(self, clientid):
        authorised = False # call user_auth_service()
        if authorised is True:
            self.authorised.append(clientid)
        return authorised

    def authenticate(self, clientid: str):
        try:
            if clientid in self.authorised:
                return True
            else:
                if self._auth_clientid(clientid) is True:
                    return True

                return False
        except Exception as excep:
            logging.error(str(excep))
            return False
```

To use the above features, the following items must be enabled in the Security section of config.yaml.

```
brain:
    security:
        authentication:
            classname: programy.security.authenticate.clientidauth.
↪ClientIdAuthenticationService
            denied_srai: AUTHENTICATION_FAILED
```

Parameter Name	Description
classname	Defines the path to python which implements the base class âĂŸAuthenticatorâĂŹ.
denied_srai	If authentication fails, the interpreter can use the document defined in this configuration as a SRAI. (In the above example, âĂŸAUTHENTICATION_FAILEDâĂŹ is set to SRAI.). The AIML file must contain this as a category pattern with appropriate text to indicate that access is denied.

17.2 Authorisation

Authorisation is defined by users, groups, and roles.

Parameter Name	Description
User	Define authorisation information for a single user. By including users in one or more groups, you can assign both specific and inherited roles.
Group	A group of users assigned one or more roles.
Role	An arbitrary authority string to be assigned to the user group.

The base authorisation class is defined as followsãĂĆ

```
class Authoriser(object):

    def __init__(self, configuration: BrainSecurityConfiguration):
        self._configuration = configuration

    @property
    def configuration(self):
        return self._configuration

    def get_default_denied_srai(self):
        return self.configuration.denied_srai

    def authorise(self, userid, role):
        return False
```

The implementation of this base class for user, group, and role-based authorisation is as followsãĂĆ

```
class BasicUserGroupAuthorisationService(Authoriser):

    def __init__(self, config: BrainSecurityConfiguration):
        Authoriser.__init__(self, config)
        self.load_users_and_groups()

    def load_users_and_groups(self):

        self._users = {}
        self._groups = {}

        if self.configuration.usergroups is not None:
            loader = UserGroupLoader()
            self._users, self._groups = loader.load_users_and_groups_from_
↪file(self.configuration.usergroups)
        else:
            logging.warning("No user groups defined, authorisation tag will not␣
↪work!")

    def authorise(self, clientid, role):
        if clientid not in self._users:
            raise AuthorisationException("User [%s] unknown to system!"%clientid)

        if clientid in self._users:
            user = self._users[clientid]
            return user.has_role(role)
        else:
            return False
```

To use the above features, the following items must be enabled in the Security section of config.yaml.

```
security:
    authorisation:
        classname: programy.security.authorise.usergroupsauthorisor.
↪BasicUserGroupAuthorisationService
        denied_srai: AUTHORISATION_FAILED
```

(continues on next page)

(continued from previous page)

```
        usergroups: ../storage/security/roles.yaml
```

Parameter Name	Description
classname	Define the path of python that implements the base class âĂŸAuthenticatorâĂŹ.
denied_srai	If the authentication fails, the interpreter can use the text defined in the configuration as the SRAI. (In the above example, âĂŸAUTHORISATION_FAILEDâĂŹ is set to SRAI.) The AIML file should be included as a category pattern with appropriate text to indicate that access is denied.
usergroups	Specify users, user groups, and role configuration files.

The format of the role file is as follows.

```
users:
  console:
    roles:
      user
    groups:
      sysadmin

groups:
  sysadmin:
    roles:
      root, admin, system
    groups:
      user

  user:
    roles:
      ask
```

When using the AIML authorisation method, enclose the template in the âĂŸauthoriseâĂŹ tag. If the input is âĂŸâĂŹALLOW ACCESSâĂŹâĂŹ and the user does not have the âĂŸrootâĂŹ privilege associated with them, then SRAI is set to what is defined in denied_srai.

```
<category>
    <pattern>ALLOW ACCESS</pattern>
    <template>
        <authorise role="root">
            Access Allowed
        </authorise>
    </template>
</category>
```

CHAPTER 18

Pre/Post Processors

There are two types of processors in the dialog engine: Pre Processor and Post Processor.
It is used when the utterance sentence needs to be processed before the dialog processing in the dialog engine, and when the response sentence from the dialog engine needs to be processed before the return.

- The *Pre Processors* are processors that pre-process strings before processing them internally in the dialogue engine.
- The *Post Processors* are processors that post-process strings before they are returned by the API for the response sentences that have been interacted with inside the dialog engine.

18.1 Pre Processors

Pre Processors inherit from the following abstract base class.

```
programy.processors.processing.PreProcessor
```

This class has a single method, process, which preprocesses the input string and returns the processed string. The returned string is used within the dialog engine to proceed with the dialog.

```
class PreProcessor(Processor):

    def __init__(self):
        Processor.__init__(self)

    @abstractmethod
    def process(self, bot, clientid, string):
        pass
```

18.2 Post Processors

Post Processors inherit from the following abstract base class.

```
programy.processors.processing.PostProcessor
```

This class has a single method, process, which preprocesses the input string and returns the processed string. Use the returned string as a response to the end user.

```
class PostProcessor(Processor):
    def __init__(self):
        Processor.__init__(self)

    @abstractmethod
    def process(self, bot, clientid, string):
        pass
```

CHAPTER 19

Custom Elements

The dialog engine is designed to replace existing node functionality and extend new elements to further extend AIML functionality.
The addition of new elements requires specific functions in accordance with the requirements of the user and the system, and if an existing pattern element or template element cannot be processed, there is a function to add an element on its own.
It is also designed to easily replace existing elements.

- The definitions of the elements available in *pattern_nodes.conf* pattern.
- The definition of the elements available in *template_nodes.conf* template.

CHAPTER 20

Custom pattern element

The parser supports the AIML tags defined in pattern_nodes.conf.
You can change the implementation of the element or add your own.
`Changing the implementation of an element may cause the parser to stop working or have a significant effect on performance or the behavior of other elements, so use it only after understanding the overall behavior.`

```
#AIML 1.0
root = programy.parser.pattern.nodes.root.PatternRootNode
word = programy.parser.pattern.nodes.word.PatternWordNode
priority = programy.parser.pattern.nodes.priority.PatternPriorityWordNode
oneormore = programy.parser.pattern.nodes.oneormore.PatternOneOrMoreWildCardNode
topic = programy.parser.pattern.nodes.topic.PatternTopicNode
that = programy.parser.pattern.nodes.that.PatternThatNode
template = programy.parser.pattern.nodes.template.PatternTemplateNode

#AIML 2.0
zeroormore = programy.parser.pattern.nodes.zeroormore.PatternZeroOrMoreWildCardNode
set = programy.parser.pattern.nodes.set.PatternSetNode
bot = programy.parser.pattern.nodes.bot.PatternBotNode

#Custom
iset = programy.parser.pattern.nodes.iset.PatternISetNode
regex = programy.parser.pattern.nodes.regex.PatternRegexNode
```

Each pattern element inheritsãĂĂ`programy.parser.pattern.nodes.base.PatternNode` as a base class.
The base class for pattern elements, which encapsulates XML elements in node attributes of the pattern class. It also includes matching words to element types during pattern evaluation.

The main methods to override are the following:

- `def equivalent(self, other)` - This method tests whether a processing element is equivalent to other evaluation elements.

- `def equals(self, bot, client, words, word_no)` - This method determines whether a word matches the rules of an element.
- `def to_string(self, verbose=True)` - This method converts element information to a string representation for debugging or logging.
- `def to_xml(self, bot, clientid)` - This method converts the content of an element to XML format and uses it to output to Braintree in XML format.

CHAPTER 21

Custom template element

The parser supports all AIML tags defined in the system, including template_nodes.conf.
template_nodes.conf also allows you to change the implementation of an element or add your own.
`Changing the implementation of an element may cause the parser to stop working or have a significant effect on performance or the behavior of other elements, so use it only after understanding the overall behavior.`

```
word = programy.parser.template.nodes.word.TemplateWordNode
authorise = programy.parser.template.nodes.authorise.TemplateAuthoriseNode
random = programy.parser.template.nodes.rand.TemplateRandomNode
condition = programy.parser.template.nodes.condition.TemplateConditionNode
srai = programy.parser.template.nodes.srai.TemplateSRAINode
sraix = programy.parser.template.nodes.sraix.TemplateSRAIXNode
get = programy.parser.template.nodes.get.TemplateGetNode
set = programy.parser.template.nodes.set.TemplateSetNode
map = programy.parser.template.nodes.map.TemplateMapNode
bot = programy.parser.template.nodes.bot.TemplateBotNode
think = programy.parser.template.nodes.think.TemplateThinkNode
normalize = programy.parser.template.nodes.normalise.TemplateNormalizeNode
denormalize = programy.parser.template.nodes.denormalise.TemplateDenormalizeNode
person = programy.parser.template.nodes.person.TemplatePersonNode
person2 = programy.parser.template.nodes.person2.TemplatePerson2Node
gender = programy.parser.template.nodes.gender.TemplateGenderNode
sr = programy.parser.template.nodes.sr.TemplateSrNode
id = programy.parser.template.nodes.id.TemplateIdNode
size = programy.parser.template.nodes.size.TemplateSizeNode
vocabulary = programy.parser.template.nodes.vocabulary.TemplateVocabularyNode
eval = programy.parser.template.nodes.eval.TemplateEvalNode
explode = programy.parser.template.nodes.explode.TemplateExplodeNode
implode = programy.parser.template.nodes.implode.TemplateImplodeNode
program = programy.parser.template.nodes.program.TemplateProgramNode
lowercase = programy.parser.template.nodes.lowercase.TemplateLowercaseNode
uppercase = programy.parser.template.nodes.uppercase.TemplateUppercaseNode
sentence = programy.parser.template.nodes.sentence.TemplateSentenceNode
formal = programy.parser.template.nodes.formal.TemplateFormalNode
that = programy.parser.template.nodes.that.TemplateThatNode
thatstar = programy.parser.template.nodes.thatstar.TemplateThatStarNode
```

(continues on next page)

(continued from previous page)

```
topicstar = programy.parser.template.nodes.topicstar.TemplateTopicStarNode
star = programy.parser.template.nodes.star.TemplateStarNode
input = programy.parser.template.nodes.input.TemplateInputNode
request = programy.parser.template.nodes.request.TemplateRequestNode
response = programy.parser.template.nodes.response.TemplateResponseNode
date = programy.parser.template.nodes.date.TemplateDateNode
interval = programy.parser.template.nodes.interval.TemplateIntervalNode
system = programy.parser.template.nodes.system.TemplateSystemNode
extension = programy.parser.template.nodes.extension.TemplateExtensionNode
learn = programy.parser.template.nodes.learn.TemplateLearnNode
learnf = programy.parser.template.nodes.learnf.TemplateLearnfNode
first = programy.parser.template.nodes.first.TemplateFirstNode
rest = programy.parser.template.nodes.rest.TemplateRestNode
log = programy.parser.template.nodes.log.TemplateLogNode
oob = programy.parser.template.nodes.oob.TemplateOOBNode
xml = programy.parser.template.nodes.xml.TemplateXMLNode
addtriple = programy.parser.template.nodes.addtriple.TemplateAddTripleNode
deletetriple = programy.parser.template.nodes.deletetriple.TemplateDeleteTripleNode
select = programy.parser.template.nodes.select.TemplateSelectNode
uniq = programy.parser.template.nodes.uniq.TemplateUniqNode
search = programy.parser.template.nodes.search.TemplateSearchNode
```

Each template element inherits the `programy.parser.template.nodes.base.TemplateNode` as a base class.

programy.parser.template.nodes.base.TemplateNode is the base class of a template element, which encapsulates an XML element into the node attributes of the template class. It is also used to evaluate textification, including child elements, during template evaluation.

The main methods to override are the following.

- `def parse_expression(self, graph, expression)` - Parsing to the elements of XML. If it canâĂŹt be expanded, throw an appropriate exception if necessary.
- `def resolve_to_string(self, bot, clientid)` - Expand an element into a string (response), which is called by resolve().
- `def resolve(self, bot, clientid)` - It is called by brain to expand individual template elements into a string. Iterates through child elements to create a string and concatenates them.
- `def to_string(self)` - A method that converts element information to a string representation for debugging and logging purposes.
- `def to_xml(self, bot, clientid)` - Converts the content of an element to XML format. This is a method that can be used to create an aiml file as part of the *learnf* or to output to *Braintree* in XML format.

CHAPTER 22

Configuration

22.1 Configuration file

The contents of the configuration file are used as parameters.
When the dialog engine is started, specify the configuration file with the –config option.
The format that can be specified in the config file is any of .yaml, .json, .xml.
The format is determined by specifying the format with the -–cformat option or the extension of the configuration file specified with the —config option.

Configuration Example

```
python3 ../../src/programy/clients/console.py --config ./config.yaml --cformat␣
↪yaml --logging ./logging.yaml
```

22.2 Command Line Options

There are command-line options that are used to control startup.
This command line option can specify the same option for all clients.

- -–config [config file path] Specify the path of the config file to be used when executing the client.
- -–cformat [yaml|json|xml] Specify the format of the config file. If not specified, the extension of the configuration file is automatically determined.
- -–logging [logging configuration] Specify the configuration file path for logging.
- -–noloop This option does not execute a dialog loop. Read the config file and AIML, and terminate the dialog engine. Use it to check the start of the dialog engine.
- -âĂŤsubs Specify an alternate argument setting file. Replace the algebra in the config file with the content specified by subs. For more information, see *Command Line Option Substitutions*.

For this tutorial, we will focus on the Yaml version, but the names of values are the same.

An example of a basic configuration file using Yaml file format is below.

22.2.1 Config Section

The configuration file is made upon a number of distinct sections. These sections creation a hierarchy of configuration which reflects how the client, bot and brain elements work together.

The basic configuration is client, which provides the implementation of interfaces such as REST, console, slack, and line. In the client, the following settings are mainly used.

- Client
  - Bots
  - Storage

The main configuration of the bot is as follows.

- *Bot*
  - conversations
  - splitter/joiner
  - spelling
  - translation
  - sentiment

The main structure of the brain is as follows.

- *Brain*
  - Overrides
  - Defaults
  - Binaries
  - Braintree
  - Services
  - Security
  - OOB
  - Dynamic Maps and Sets
  - Tokenizers
  - Debug Files

The log structure is specified in the following file.

- *Logging*

CHAPTER 23

Config Substitutions

23.1 License Key Substitutions

- Function to substitute the setting value of license.keys

  When the configuration item of the config is specified as `LICENSE_KEY:VALUE` , it is a function to substitute it with the value of license.keys.

Configuration Example

Describe `LICENSE_KEY:VALUE` in the config file as shown below.
`LICENSE_KEY:` is a fixed value, and `VALUE` specifies the key name as it describes in license.keys.

```
client:
  email:
    username: LICENSE_KEY:EMAIL_USERNAME
    password: LICENSE_KEY:EMAIL_PASSWORD
```

The following entry in license.keys substitutes the LICENSE_KEY:VALUE above with the contents of license.keys at runtime.

```
EMAIL_USERNAME=sample@somewere.com
EMAIL_PASSWORD=qwerty
```

If you use git, setting license.keys to .gitignore prevents you from unintentionally registering.

23.2 Command Line Option Subsutitutions

When you start the dialog engine with the substitutions.txt file as a command line argument, you can specify parameters to be substituted as startup arguments.
Substitute with the file specified by the command line option –subs.

Configuration Example

Describe $ +VALUE in the config file as shown below.
VALUE specifies the key name as it describes in substitutions.txt.

```
client:
  email:
    host: $EMAIL_HOST
    port: $EMAIL_PORT
```

The following entry in substitutions.txt will replace the above $VALUE with the contents of substitutions.txt and work at runtime.

```
$EMAIL_HOST:prod_server.com
$EMAIL_PORT:9999
```

This allows you to set `--subs substitutions.txt` and the substitution list at startup, without editing the config contents, and change the settings at runtime.

CHAPTER 24

Log setting

Perform logging using the Python logging function. See the Python logging documentation for details on how to specify log configuration options.

This is a log setting example to output to /tmp/y-bot.log.

```
version: 1
disable_existing_loggers: False

formatters:
  simple:
    format: '%(asctime)s  %(name)-10s %(levelname)-7s %(message)s'

handlers:
  file:
    class: logging.handlers.RotatingFileHandler
    formatter: simple
    filename: /tmp/y-bot.log

root:
  level: DEBUG
  handlers:
      - file
```

CHAPTER 25

Bot Configuration

A bot configuration consists of items that control the overall behavior of the dialog engine.

- *initial_question*
- *default_response*
- *empty_string*
- *max_search_depth*
- *override_properties*
- *exit_response*

In addition, it consists of the following subsections such as bot string processing and additional function settings.

- conversations
- splitter/joiner
- spelling
- translation
- sentiment

Configuration Example

```
bot:
    version: v1.0

    brain: brain

    initial_question: Good morning.
    initial_question_srai: Good evening.
    default_response: unknown
    default_response_srai: YEMPTY
    empty_string: YEMPTY
    exit_response: Exit.
    exit_response_srai: YEXITRESPONSE

    override_properties: true

    max_question_recursion: 1000
```

(continues on next page)

(continued from previous page)

```
    max_question_timeout: 60
    max_search_depth: 100
    max_search_timeout: 60
    max_search_condition: 20
    max_search_srai: 50
    max_categories: 20000

    spelling:
        load: false
        classname: programy.spelling.norvig.NorvigSpellingChecker
        corpus: file
        check_before: false
        check_and_retry: false

    conversations:
    save: true
    load: false

    joiner:
        classname: programy.dialog.joiner.joiner.SentenceJoiner
        join_chars: .?!ãĂĆïijŞïijĄ
        terminator: .

    splitter:
        classname: programy.dialog.splitter.splitter.SentenceSplitter
        split_chars: .
```

25.1 initial_question

Table 1: Feature List

Set value	Content
initial_question_srai	A start-up utterance. The utterance to be made at startup of client. The scenario described in AIML works and generates a response.
initial_question	Startup response. Specifies the response to return if the scenario corresponding to initial_question_srai is not listed.

25.2 default_response

Table 2: Feature List

Set value	Content
default_response_srai	An utterance executed if there is no matching pattern. The scenario described in AIML works and generates a response.
default_response	Response to return if there is no matching pattern.Specify the response to return if the scenario corresponding to default_response_srai is not described.The *default-response* in properties has a higher priority than the setting, and the response in properties takes precedence over.

25.3 empty_string

Table 3: Feature List

Set value	Content
empty_string	An utterance to be processed when there is no processing result of pre_processor. If there is no processing result of pre_processor, the scenario operates with this description as an utterance and generates a response.

25.4 max_search_depth

Table 4: Feature List

Set value	Content
max_question_recursion	Maximum number of sentence searches. Specify the maximum number of searches when a long text is input, divided by split characters, and the dialog scenario is executed multiple times. Returns *default-response* if the maximum number of times is reached.
max_question_timeout	Maximum sentence search time. Specify the maximum processing time in units of seconds when a long text is input, divided by split characters, and the dialog scenario is executed multiple times internally. If the maximum processing time is exceeded, *default-response* is returned.
max_search_timeout	Specify the maximum time, in seconds, to search for a word. Specify the maximum search time in seconds for a word search during a sentence search, such as when a long pattern is encountered during a sentence search. Returns *default-response* if maximum processing time is exceeded.
max_search_depth	Maximum number of word search branches. Specify the maximum number of times that a word can be searched for if it becomes too numerous using *wildcards* , *set* and so on. Returns *default-response* if the maximum number of word searches is reached.
max_search_condition	Maximum number of loops of *condition loop* . Specify the maximum number of loops that can occur when a condition loop is spesified. This prevents an infiniteloop when the condition does not match. Returns *default-response* if the maximum number of loops is reached.
max_search_srai	Maximum number of *srai* recursive calls. Specify the maximum number of recursive calls if the description of srai is a recursive call. Returns *default-response* if the maximum number of times is reached.
max_categories	Maximum number of categories to load. Specify the maximum number of AIML categories to load. If the category specified in AIML exceeds the upper limit, loading is not performed. Categories that were not imported will be described in *errors* description as `Max categories [n] exceeded`.

25.5 override_properties

Table 5: Feature List

Set value	Content
override_properties	Override permission flag for the name variable. If false, the name variable of the same name is not overwritten.

25.6 exit_response

Table 6: Feature List

Set value	Content
exit_response_srai	End utterance. An utterance to be executed at the end of the client. The scenario described in AIML operates and generates a response.
exit_response	End response. exit_response_srai specifies a response to return if the corresponding scenario is not described.

CHAPTER 26

Brain Configuration

For brain configuration, brain sets dialog processing and individual settings for each element. In addition, it consists of the following subsections depending on the processing contents in the brain.

The configuration section is as follows:

- Overrides
- Defaults
- Binaries
- Braintree
- Services
- Security
- OOB
- Dynamic Maps and Sets
- Tokenizers
- Debug Files

Configuration Example

```
brain:

   overrides:
     allow_system_aiml: true
     allow_learn_aiml: true
     allow_learnf_aiml: true

   defaults:
     default-get: unknown
     default-property: unknown
     default-map: unknown
     learnf-path: file

   binaries:
     save_binary: true
     load_binary: true
```

(continues on next page)

(continued from previous page)

```
    load_aiml_on_binary_fail: true

  braintree:
    create: true

  services:
      REST:
          classname: programy.services.rest.GenericRESTService
          method: GET
          host: 0.0.0.0
          port: 8080
      Pannous:
          classname: programy.services.pannous.PannousService
          url: http://weannie.pannous.com/api

  security:
      authentication:
          classname: programy.security.authenticate.passthrough.
↪BasicPassThroughAuthenticationService
          denied_srai: AUTHENTICATION_FAILED
      authorisation:
          classname: programy.security.authorise.usergroupsauthorisor.
↪BasicUserGroupAuthorisationService
          denied_srai: AUTHORISATION_FAILED
          usergroups:
            storage: file

  oob:
    default:
      classname: programy.oob.defaults.default.DefaultOutOfBandProcessor
    alarm:
      classname: programy.oob.defaults.alarm.AlarmOutOfBandProcessor
    camera:
      classname: programy.oob.defaults.camera.CameraOutOfBandProcessor
    clear:
      classname: programy.oob.defaults.clear.ClearOutOfBandProcessor
    dial:
      classname: programy.oob.defaults.dial.DialOutOfBandProcessor
    dialog:
      classname: programy.oob.defaults.dialog.DialogOutOfBandProcessor
    email:
      classname: programy.oob.defaults.email.EmailOutOfBandProcessor
    geomap:
      classname: programy.oob.defaults.map.MapOutOfBandProcessor
    schedule:
      classname: programy.oob.defaults.schedule.ScheduleOutOfBandProcessor
    search:
      classname: programy.oob.defaults.search.SearchOutOfBandProcessor
    sms:
      classname: programy.oob.defaults.sms.SMSOutOfBandProcessor
    url:
      classname: programy.oob.defaults.url.URLOutOfBandProcessor
    wifi:
      classname: programy.oob.defaults.wifi.WifiOutOfBandProcessor

  dynamic:
      variables:
          gettime: programy.dynamic.variables.datetime.GetTime
      sets:
          numeric: programy.dynamic.sets.numeric.IsNumeric
          roman:   programy.dynamic.sets.roman.IsRomanNumeral
```

(continues on next page)

(continued from previous page)

```
    maps:
        romantodec: programy.dynamic.maps.roman.MapRomanToDecimal
        dectoroman: programy.dynamic.maps.roman.MapDecimalToRoman
```

CHAPTER 27

OOB Settings

OOB (Out of Band) processing requires a Python class for each OOB child element used. This class processes OOB commands and executes the necessary commands on the client side.

The following is the default implementation class. In the execution function, only argument check is performed, and no specific action is performed. In practice, each system requires its own implementation.

```
oob:
  default:
    classname: programy.oob.defaults.default.DefaultOutOfBandProcessor
  alarm:
    classname: programy.oob.defaults.alarm.AlarmOutOfBandProcessor
  camera:
    classname: programy.oob.defaults.camera.CameraOutOfBandProcessor
  clear:
    classname: programy.oob.defaults.clear.ClearOutOfBandProcessor
  dial:
    classname: programy.oob.defaults.dial.DialOutOfBandProcessor
  dialog:
    classname: programy.oob.defaults.dialog.DialogOutOfBandProcessor
  email:
    classname: programy.oob.defaults.email.EmailOutOfBandProcessor
  geomap:
    classname: programy.oob.defaults.map.MapOutOfBandProcessor
  schedule:
    classname: programy.oob.defaults.schedule.ScheduleOutOfBandProcessor
  search:
    classname: programy.oob.defaults.search.SearchOutOfBandProcessor
  sms:
    classname: programy.oob.defaults.sms.SMSOutOfBandProcessor
  url:
    classname: programy.oob.defaults.url.URLOutOfBandProcessor
  wifi:
    classname: programy.oob.defaults.wifi.WifiOutOfBandProcessor
```

Parameter	Description	Example	Default
classname	Full Python classpath for OOB implementation	programy.oob.defaults.alarm.AlarmOutOfBandProcessor	None

See *OOB* for OOB content.