SHARP

ロボホン認定試験チュートリアル

5. HVML

Version 1.0.0

Last update 2016/09/09

SHARP CORPORATION

SHARP

更新履歴

Version	Description	Date
1.0.0	初版作成	2016/09/08

SHARP

目次

更新履歴 2

目次 3

1. HVML について 4

1.1 はじめに 4
1.2 シナリオと HVML ファイル 4
1.3 ロボホンと HVML ファイルの関係 4
1.4 HVML ファイルの種類 5
1.5 HVML ファイルの命名規約 5
1.6 HVML ファイルのフォーマット 6
1.7 HVML ファイルの構成 6
1.8 シナリオ固有の定義 6
1.9 シナリオ名について 8
1.10 SITUATION と ACCOST 8
1.11 ホーム用シナリオ専用シチュエーション 9
1.12 その他シナリオでの SITUATION 10
1.13 環境検知イベントを契機に条件式を評価 11
1.14 ユーザー発話を契機に条件式を評価 12
1.15 SITUATION の条件 14
1.16 ACCOST 16
1.17 PRIORITY について 17
1.18 シーン名 18
1.19 シーンの有効 18
1.20 BODY の構成 20
1.21 TOPIC 20
1.22 ACTION 21
1.23 より理解を深めるために 22
1.24 TOPIC でのユーザー発話の聞き取り 23
1.25 １つ目の問題点 24
1.26 １つ目の問題点を解決する 25
1.27 ２つ目の問題点と解決方法 26

1. HVML について

1.1 はじめに

HVML(Hyper Voice Markup Language)とはXML1.0をベースとしたマークアップ言語です。電子機器と人間との対話を表現したシナリオを実現することを目的として設計されており、音声対話のシナリオを「ユーザ発話」と「対応するアクション」の組み合わせで表現しています。

この章では、HVML について理解を深めると同時に、HVML の基本的な使い方を説明します。

1.2 シナリオと HVML ファイル

ロボホンとユーザーの対話を定義したものを「シナリオ」と呼びます。シナリオは、HVML 形式で記述されます。これを、HVML ファイルと呼びます。HVML ファイルは、1 シナリオに付き 1 ファイル作成されます。

1.3 ロボホンと HVML ファイルの関係

ロボホンにアプリをインストールする時に、アプリ内にある全ての HVML ファイルがロボホン本体に登録されます。これを HVML ファイルの登録（シナリオの登録）といいます。

この時、すでに本体に登録されている HVML よりも、version 番号が大きくない場合は、HVML が登録されません。HVML ファイルを変更した場合、必ず version を増やして保存するよう気をつけてください。

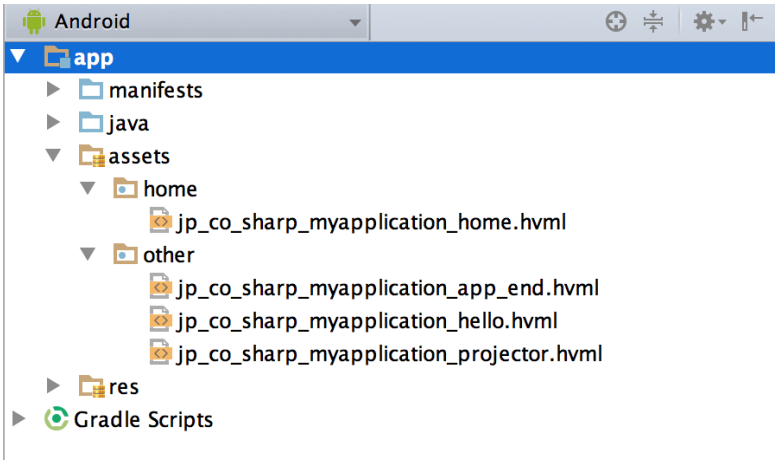
※ホーム用シナリオ以外のシナリオ（その他シナリオ）は、登録しただけでは有効になりません。シナリオの有効化については、別途説明します。

1.4 HVML ファイルの種類

HVMLファイルの拡張子は、.hvmlです。ロボホンで使用するHVMLには、大きく分けて２種類あります。
HVMLファイルの種類によって、配置するフォルダが違います。

HVMLファイルの種類	配置場所
ホーム用シナリオ	home
その他シナリオ	other

ロボホンのホーム画面からアプリ起動を行うための特殊なシナリオを「ホーム用シナリオ」と言います。ホーム用シナリオだけを home フォルダに配置し、その他シナリオは other フォルダに配置します。



※テンプレートからアプリを作成した場合のHVMLファイルの配置例

1.5 HVML ファイルの命名規約

HVMLファイルの命名規約は、SDK資料 [0401_SR01MW_Application_Programming_Guide] に記載されています。Androidパッケージ名のピリオド「.」をアンダースコア「_」に変換したものに、シナリオ名をアンダースコア区切りで追加してファイル名にします。

例えば、ホーム用シナリオの場合シナリオ名は”home”になるので、ファイル名は、以下のようになります。
jp_co_sharp_myapplication_home.hvml

テンプレートから作成されたシナリオは、上記を含め、すべて命名規約通りになっているので、参考にするといいでしょう。なるべく短くて分かりやすいシナリオ名を付けると開発の時に便利です。

1.6 HVML ファイルのフォーマット

HVMLファイルは、XMLのフォーマットで書かれています。テンプレートから作成されるホーム用シナリオを中心に、構成と内容を見ていきましょう。

なお、HVMLについての詳しい解説は、SDK資料 [0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習してください。

1.7 HVMLファイルの構成

HVMLファイルは、xmlとhvmlで構成されます。将来的に変更になる可能性はありますが、現時点でのversionの値は固定になります。

hvmlの中身はheadとbodyです。headにはシナリオ固有の定義や、bodyで定義されている内容を呼び出すための条件を定義します。bodyには実際の会話の内容などを定義します。

jp_co_sharp_myapplication_home.hvml

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        :
    </head>
    <body>
        :
    </body>
</hvml>
```

1.8 シナリオ固有の定義

headに定義されている情報を見てみましょう。

jp_co_sharp_myapplication_home.hvml

```
    <head>
        <producer>jp.co.sharp.myapplication</producer>
        <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートのホーム起動シナリオ</description>
        <scene value="home" />
        <version value="1.0" />
        :
    </head>
```

producer、description、scene、versionの4つのタグに、シナリオ固有の定義を設定します。
詳しい内容は、以下になります。

タグ	内容
producer	パッケージ名（固定）
description	シナリオの説明文
scene	シーン名（ホーム用シナリオの場合はhome）
version	このシナリオのバージョン

producerは固定なので、テンプレートから作成している場合は修正しなくても問題ありません。

descriptionは、シナリオの説明文です、分かりやすい文章に変更しておきましょう。

sceneは、シーン名です。ホーム用シナリオの場合はhome固定です。その他のシナリオの場合は、シナリオの利用シーンに合わせてシーン名を定義する必要があります。シナリオ名では無いので注意してください。シーンについては、後半の章で解説します。

versionは、シナリオのバージョンです。ロボホン本体にHVMLを登録する時に参照されます。HVMLファイルの内容を変えたら、バージョンを増やすのを忘れないようにしましょう。

1.9 シナリオ名について

headの中に、シナリオ名の定義がないのに気が付きましたか？シナリオ名はファイル名で定義され、HVMLファイルの中には出てきません。ホーム用のHVMLでは、シーン名もシナリオ名も両方"home"なので混乱しやすいですが、間違えないようにしましょう。

1.10 situation と accost

head には、body で定義されている内容（topic）を呼び出すための条件を定義することができます。situation と accost の 2 種類のタグがあり、それぞれ呼び出し元や条件の設定方法などが異なります。

なお、topic については、後で詳しく説明します。ここでは、topic を呼び出すために、topic_id を指定するということだけ理解しておいてください。

タグ	内容
situation	シチュエーションに応じてtopicを呼び出す （ホーム用シナリオの場合は固定）
accost	javaのプログラムからtopicを呼び出す （ホーム用シナリオの場合は使用不可）

ホーム用シナリオのHVMLは、situationに定義できる内容が予め決められています。また、accostは使用できないので注意してください。その他シナリオでは両方共使用することが出来ます（ホーム用シナリオ専用の変数は使用できません。）

1.11　ホーム用シナリオ専用シチュエーション

ホーム用シナリオのHVMLを見ていきましょう。ここでは、2種類のsituationを定義する必要があります。

jp_co_sharp_myapplication_home.hvml

```
<head>
    :
    <situation priority="78" topic_id="start" trigger="user-word">${Local:WORD_APPLICATION} eq
        てんぷれーと
    </situation>
    <situation priority="78" topic_id="start" trigger="user-word">
        ${Local:WORD_APPLICATION_FREEWORD} eq  てんぷれーとあぷりしよう
    </situation>
</head>
```

1つ目のsituationには、アプリ名をひらがなで指定します。ロボホンに対して、「【アプリ名】を起動して。」などと声をかけられた時に、指定されたtopic（この場合は、topic_id=”start”）が実行されます。1アプリに付き1つしか登録できません。また、条件式を変更したり、複数記述することは出来ません。

2つ目のsituationには、アプリを起動するための任意の文字列を指定します。例えば、「やきゅうしよう」などの言葉を登録することが出来ます。こちらも、1アプリに付き1つしか登録できません。条件式を変更したり、複数記述することは出来ません。

Local:WORD_APPLICATIONとLocal:WORD_APPLICATION_FREEWORDは、ホーム用シナリオでしか利用できない変数なので注意してください。

1.12　その他シナリオでの situation

その他シナリオでは、さまざまなシチュエーションを定義することができます。

シチュエーションは、条件式を評価するトリガーが2種類あります。trigger属性で指定することが出来ます。

タグ	内容
env-event	環境検知イベントを契機に条件式を評価
user-word	ユーザー発話を契機に条件式を評価

env-eventというのは、環境検知（ロボホンの状態の変化）により発生するイベントです。
たとえば、以下の様なイベントが定義されています。

イベント名	内容
shake	振られた
upside_down	逆さにされた

この他のイベントは、SDK資料 [0601_SR01MW_HVML2.0_Specification] に記載されていますので、学習しておいてください。

user-wordは、主にユーザーが話した内容に応じて処理を行う場合に使います。

1.13　環境検知イベントを契機に条件式を評価

ユーザー発話を契機に条件式を評価する方法について説明しましょう。

サンプル
<situation priority="75" topic_id=" t1" trigger="env-event"　value="upside_down">true</ situation>

priorityは、実行順序の優先順位を意味する属性です。後で解説するので、ここでは省略します。

topic_idは、このシチュエーションの条件に当てはまった時に実行されるトピックのIDになります。

trigger=”user-word”で、ユーザー発話を契機にtopicを実行するよう指定します。

value=”upside_down”で、イベントを指定します。この場合、「逆さにされた」という意味なので、ロボホンを逆さまにすると、条件評価が行われます。

situationタグで囲まれた条件式にはtrueが設定されています。これにより、「逆さにされた」というイベントが発生した場合に、無条件でtopic_idに指定されたトピックを実行します。

1.14　ユーザー発話を契機に条件式を評価

テンプレートから作成されるシナリオ（hello.hvml）を見ながら、ユーザー発話を契機に条件式を評価する方法について説明しましょう。

jp_co_sharp_myapplication_hello.hvml

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>jp.co.sharp.myapplication</producer>
        <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートシナリオ</description>
        <scene value="jp.co.sharp.myapplication.scene01" />
        <version value="1.0" />
        <situation priority="75" topic_id="reply" trigger="user-word">${Lvcsr:Basic} include
            [おはよう,こんにちは,こんばんは]
        </situation>
        <accost priority="75" topic_id="say" word="jp.co.sharp.myapplication.hello.say" />
    </head>
    <body>
        <topic id="say" listen="false">
            <action index="1">
                <speech>こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね
</speech>
                <behavior id="assign" type="normal" />
            </action>
        </topic>
        <topic id="reply" listen="false">
            <action index="1">
                <speech>お話してくれて、ありがとう！これから、僕にいろんなことを教えてね！！</speech>
                <behavior id="assign" type="normal" />
            </action>
        </topic>
    </body>
</hvml>
```

ユーザー発話を契機に条件式を評価する部分は以下になります。

jp_co_sharp_myapplication_hello.hvml

```
        <situation priority="75" topic_id="reply" trigger="user-word">${Lvcsr:Basic} include
            [おはよう,こんにちは,こんばんは]
        </situation>
```

priorityは、実行順序の優先順位を意味する属性です。後で解説するので、ここでは省略します。

topic_idは、このシチュエーションの条件に当てはまった時に実行されるトピックのIDになります。

trigger=”user-word”で、ユーザー発話を契機にtopicを実行するよう指定します。

situationタグで囲まれた内容は、実行のための条件です。この条件がtrueになった場合に、topicが実行されます。

1.15　situation の条件

situation タグで囲まれた条件を詳しく見ていきましょう。

jp_co_sharp_myapplication_hello.hvml
${Lvcsr:Basic} include [おはよう,こんにちは,こんばんは]

${Lvcsr:Basic} は、変数です。HVML では、${}で囲むことによって変数を表します。Lvcsr:Basic には、ユーザーが発話した内容が文字列で入ります。

そのほかの変数には、以下の様なものがあります。

変数の種類	内容
ユーザー発話変数	Lvcsr:Basic、Lvcsr:Kana など
時制取得用変数	Year、Month など
時制判定用変数	InDateRange、InTimeRange など
型変換用変数	SizeOf、Select など
個人情報を扱う変数	resolver:contacts:robot_name など
その他変数	Rand、アプリ解決変数（パッケージ名ではじまるもの）、記憶（memory_p）、一時記憶（memory_t）など

変数の詳細については、SDK資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習しておいてください。

include は、演算子です。「～が含まれているか」という意味です。

そのほかの演算子には、以下の様なものがあります。

演算子	内容
eq	左辺と右辺が一致している場合に真となります。
and	複数の演算結果を結合する際に使用します。and で繋がれた演算結果がすべて真である場合に、条件式全体として真となり、一つでも偽になるものがあれば偽と扱われます。
in	左辺は文字列型、右辺は文字列型またはリスト型を指定してください。右辺が文字列型の場合、左辺の文字列が右辺の文字列中に含まれているなら真、右辺がリスト型の場合、右辺の要素に左辺と一致するものがあれば真となります。
include	左辺は文字列型、右辺はリスト型または文字列型を指定してください。右辺がリスト型の場合、右辺の要素のいずれか一つでも左辺に含まれていれば真、右辺が文字列型の場合、右辺の文字列が左辺の文字列中に含まれていれば真となります。
near	左辺、右辺共に文字列型を指定してください。左辺と右辺の文字列の近似度がある一定の範囲内であれば（つまり２つの文字列が近ければ）真となります。主にユーザー発話を条件式に記載する際に、語尾のゆらぎ等を吸収する用途として用います。
+、-、*、/	数値型の四則演算用の演算子になります。それぞれ左辺、右辺の加減乗除を行いま

	す。+、-に関しては、日時型での利用も可能です。

演算子の詳細については、SDK 資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習しておいてください。

変数と演算子が理解できたところで、もう一度、条件式を確認してみましょう。

jp_co_sharp_myapplication_hello.hvml

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>jp.co.sharp.myapplication</producer>
        <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートシナリオ</description>
        <scene value="jp.co.sharp.myapplication.scene01" />
        <version value="1.0" />
        <situation priority="75" topic_id="reply" trigger="user-word">${Lvcsr:Basic} include
            [おはよう,こんにちは,こんばんは]
        </situation>
        <accost priority="75" topic_id="say" word="jp.co.sharp.myapplication.hello.say" />
    </head>
    <body>
        <topic id="say" listen="false">
            <action index="1">
                <speech>こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね
</speech>
                <behavior id="assign" type="normal" />
            </action>
        </topic>
        <topic id="reply" listen="false">
            <action index="1">
                <speech>お話してくれて、ありがとう！これから、僕にいろんなことを教えてね！！</speech>
                <behavior id="assign" type="normal" />
            </action>
        </topic>
    </body>
</hvml>
```

jp_co_sharp_myapplication_hello.hvml

```
${Lvcsr:Basic} include [おはよう,こんにちは,こんばんは]
```

上記の条件式は、「ユーザーが発話した内容に、”おはよう”、”こんにちは”、”こんばんは”のどれかが含まれていたら true」という意味になります。

ここで注意して欲しいのが、[おはよう,こんにちは,こんばんは]の記述です。 [おはよう, こんにちは, こんばんは]のように、”,”の後に半角スペースをいれてしまうと上手く判定されません。間違えやすい部分なので、気をつけてください。

1.16 accost

accostは、javaのプログラムからtopicを呼び出すために用います。テンプレートから作成されるシナリオ（hello.hvml）を見ながら、accostについて説明しましょう。

accostを使用している部分は以下になります。

jp_co_sharp_myapplication_hello.hvml

```
<accost priority="75" topic_id="say" word="jp.co.sharp.myapplication.hello.say" />
```

priorityは、実行順序の優先順位を意味する属性です。後で解説するので、ここでは省略します。

topic_idは、このaccostで実行されるトピックのIDになります。

wordには、accost名を定義します。これは、javaのプログラムからaccostを呼び出す際のキーとなる文字列です。

次に、javaのプログラムを見てみましょう。

ScenarioDefintions

```
/**
 * accost 名：こんにちは発話実行.
 */
public static final String ACC_HELLO = ScenarioDefinitions.PACKAGE + ".hello.say";
```

MainActivity

```
VoiceUIVariableListHelper helper = new VoiceUIVariableListHelper().addAccost(ScenarioDefinitions.ACC_HELLO);
VoiceUIManagerUtil.updateAppInfo(mVoiceUIManager, helper.getVariableList(), true);
```

まずはじめに、ScenarioDefintionsで、accost名を定義します。accost名の命名規約は特に決まっていませんが、javaのプログラム側から「指定したシナリオのアコーストタグに設定してあるトピックIDのトピックを実行してください。」という意味で使用しますので、プロジェクト名.シナリオ名.トピックIDといった規約で命名するとわかりやすいと思います。

MainActivityでは、accost名を格納したVoiceUIVariable 変数（helper）を用意し、VoiceUIManagerUtilのupdateAppInfo()を実行しています。これにより、accost名に対応するaccostが選択され、topic_idに設定してあるトピックが実行されます。

jp_co_sharp_myapplication_hello.hvml

```
<accost priority="75" topic_id="say" word="jp.co.sharp.myapplication.hello.say" />
```

accostタグには条件式はありません。javaのプログラムから直接呼び出されます。

1.17 priority について

priorityは、実行順序の優先順位を意味する属性です。priorityは数字が小さい方が優先順位が高く、situationやaccostなどで使用できます。この時、タグによって指定できる値が異なります。また、指定を省略できる場合と省略できない場合があるので注意しましょう。

ちなみに、priorityが同じsituationが複数ある場合は、どれか一つがランダムで選択されます。これを利用して、複数の返答パターンを作成することが可能です。

1.18　シーン名

HVMLのheadで定義する、シーン名について説明しましょう。

例えば、ホーム用のシナリオの場合、以下のように定義されています。

jp_co_sharp_myapplication_home.hvml

```
<head>
    <producer>jp.co.sharp.myapplication</producer>
    <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
    <description>テンプレートのホーム起動シナリオ</description>
    <scene value="home" />
    <version value="1.0" />
    :
</head>
```

ホーム用のシナリオの場合、シーン名は"home"固定となります。それ以外の名前を追加したり、別の名前を使うことはできません。

jp_co_sharp_myapplication_hello.hvml

```
<head>
    <producer>jp.co.sharp.myapplication</producer>
    <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
    <description>テンプレートシナリオ</description>
    <scene value="jp.co.sharp.myapplication.scene01" />
    <version value="1.0" />
    :
</head>
```

その他シナリオの場合、シーン名は"パッケージ名.任意文字列"となります。任意文字列の部分には、好きな文字列を定義できます。混乱しないようにシナリオ名とは別の文字列を定義するといいでしょう。

1.19　シーンの有効

その他シナリオ用のHVMLは、アプリのインストール時にロボホン本体に登録されますが、そのままでは有効になりません。必ずシーンに所属させて、対象のシーンをjavaのプログラム上で有効にする必要があります。

シーンとは、シナリオを束ねるグループのようなものです。各シナリオ（HVMLファイル）毎に所属するシーン名を定義しておき、javaのプログラム上で有効、無効を切り替えることができます。

また、複数のシーンを同時に有効にすることが可能です。シーン名の命名を適切なものにすることで、効率良いシナリオ作成が可能になります。

例えば、テンプレートから作成したアプリには、2つのシーンが含まれています。

ScenarioDefintions

```
/**
 * scene 名：アプリ共通シーン
 */
public static final String SCENE_COMMON = PACKAGE + ".scene_common";
/**
 * scene 名：特定シーン
 */
public static final String SCENE01 = PACKAGE + ".scene01";
```

MainActivity

```
public void onResume() {
    :
    //Scene 有効化.
    VoiceUIManagerUtil.enableScene(mVoiceUIManager, ScenarioDefinitions.SCENE_COMMON);
    VoiceUIManagerUtil.enableScene(mVoiceUIManager, ScenarioDefinitions.SCENE01);
}
```

アプリ共通のシーンであるSCENE_COMMONと、特定のシーンを意味するSCENE01を定義しています。

こうすることにより、例えばプログラム中でSCENE01を無効にし、新しくSCENE02を作成して有効にした場合でも、アプリ共通のSCENE_COMMONはそのまま利用出来る設計になっています。

次に、プロジェクター用のHVMLファイルを見てみましょう。

jp_co_sharp_myapplication_projector.hvml

```
<head>
    <producer>jp.co.sharp.myapplication</producer>
    <description>プロジェクタを音声で起動するシナリオ</description>
    <!--TODO プロジェクターを起動したい場合は scene_disable を scene_common に変更してください -->
    <scene value="jp.co.sharp.myapplication.scene_disable" />
    <version value="1.0" />
    <situation priority="75" topic_id="start" trigger="user-word">${Lvcsr:Basic} include
        [プロジェクト,プロジェクタ] and ${Lvcsr:Basic} include [開始,解し,会し,起動,移し,写し,映し]
    </situation>
</head>
```

シーン名に"jp.co.sharp.myapplication.scene_disable"が設定されています。このシーンはプログラム中では有効になっていないため、このシナリオは利用できません。そのため、ロボホンに「プロジェクタ開始」などと話しかけてもプロジェクターは開始しない状態になっています。

このように、シーンの有効・無効を活用することによって、シーンに所属するシナリオのオン・オフが簡単にできるようになります。

1.20 body の構成

headの構造を説明したので、次はbodyの構造を説明しましょう。テンプレートから作成されるホーム用シナリオを中心に、構成と内容を見ていきます。

jp_co_sharp_myapplication_home.hvml

```
<body>
    <topic id="start" listen="false">
        <action index="1">
            <speech>${resolver:speech_ok(${resolver:ok_id})}</speech>
            <behavior id="${resolver:motion_ok(${resolver:ok_id})}" type="normal" />
        </action>
        <action index="2">
            <speech>テンプレートを起動するね</speech>
            <behavior id="assign" type="normal" />
            <control function="start_activity" target="home">
                <data key="package_name" value="jp.co.sharp.myapplication" />
                <data key="class_name" value="jp.co.sharp.myapplication.MainActivity" />
            </control>
        </action>
    </topic>
</body>
```

bodyには、実際の会話の内容などを定義します。
シナリオは、トピックという単位でブロック化することができます。

1.21 topic

topicは会話や音声聞き取りなど、HVML内で実行したい項目をブロック化するために使われます。

id属性には、任意の文字列を指定します。シナリオ内（HVMLファイル内）で同じ名前は使用出来ません。
situationやaccostで呼び出す際にtopic_idとして利用するので、わかりやすい名前をつけるといいでしょう。

listen属性は、topicの終了時にユーザー発話の聞き取りを行うかどうかの属性です。詳しい説明は後述します。falseなので、topicの終了時にユーザー発話は行いません。

topicの中には、rule、case、action、a、nextといったタグを設定できます。これらの詳細については、SDK資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習しておいてください。

1.22 action

topicの子要素の一つである、actionについて説明しましょう。
actionは、HVML実行時に実施される操作に関する内容を記述できます。1つのtopic内に複数記述することが可能です。

index 属性には、action の実行順を指定します。1 から順番に連番を振ってください。

action の中には、speech、behavior、control、memory といったタグを設定できます。これらの詳細については、SDK 資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて確認しておいてください。

テンプレートを見てみましょう。

jp_co_sharp_myapplication_home.hvml

```
<body>
    <topic id="start" listen="false">
        <action index="1">
            <speech>${resolver:speech_ok(${resolver:ok_id})}</speech>
            <behavior id="${resolver:motion_ok(${resolver:ok_id})}" type="normal" />
        </action>
        <action index="2">
            <speech>テンプレートを起動するね</speech>
            <behavior id="assign" type="normal" />
            <control function="start_activity" target="home">
                <data key="package_name" value="jp.co.sharp.myapplication" />
                <data key="class_name" value="jp.co.sharp.myapplication.MainActivity" />
            </control>
        </action>
    </topic>
</body>
```

index=”1”のアクションでは、speech と behavior の 2 つのタグを使ってロボホンの発話と動作を定義しています。

speech は、ロボホンを発話させるタグです。
${resolver:speech_ok(${resolver:ok_id})}
の部分は、特殊な書き方でロボホンが返事する際の発話内容を指定しています。詳細については、SDK 資料
[0401_SR01MW_ Application_Programming_Guide] に記載されています。

behavior は、ロボホンの発話に合わせて行うモーションを指定するタグです。詳細については、SDK 資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習しておいてください。

index=”2”のアクションでは、speech と behavior に加えて、control タグで、アプリを起動しています。詳細については、SDK 資料[0601_SR01MW_HVML2.0_Specification] に記載されていますので、あわせて学習しておいてください。

このように、action は、topic の中に複数定義でき、順番に実行させていくことが可能です。

1.23　より理解を深めるために

HVML について、より理解を深めるためには、SDK 資料の説明が必要です。

プログラムからの accost 起動や、HVML とプログラムの連携については、
SDK 資料[0401_SR01MW_ Application_Programming_Guide] に詳しく記載されています。

HVML 自体の文法、変数や演算子の仕様、記述方法などは、
SDK 資料[0601_SR01MW_HVML2.0_Specification] に記載されています。

また、HVML のさまざまな機能を使ったサンプルプログラムとして、
SDK 資料[0701_SR01MW_SampleCode] の SampleScenario が用意されています。

これら 3 つの資料を説明し、HVML の理解を深めてください。
次の章からは、HVML を理解する上で重要なパターン、注意しなくてはいけない部分について説明していきましょう。

1.24　topic でのユーザー発話の聞き取り

topic の listen 属性は、topic の終了時にユーザー発話の聞き取りを行うかどうかの属性です。
true にすることによって、ユーザー発話の聞き取りを行うことができます。

テンプレートの HVML を改造して、実験用シナリオを作成してみました。「こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね」という発話に対して、「さようなら」と話しかけることが出来るというものです。

これを使って、実際に試してみましょう。

jp_co_sharp_myapplication_hello.hvml

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>jp.co.sharp.myapplication</producer>
        <!-- TODO このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>実験用シナリオ</description>
        <scene value="jp.co.sharp.myapplication.scene01" />
        <version value="1.0" />
        <situation priority="75" topic_id="reply" trigger="user-word">${Lvcsr:Basic} include
            [おはよう,こんにちは,こんばんは]
        </situation>
        <accost priority="75" topic_id="say" word="jp.co.sharp.myapplication.hello.say" />
    </head>
    <body>
        <topic id="say" listen="true">
            <action index="1">
                <speech>こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね
</speech>
                <behavior id="assign" type="normal" />
            </action>
            <a href="#reply">
                <situation trigger="user-word">${Lvcsr:Basic} include [さようなら]</situation>
            </a>
        </topic>
        <topic id="reply" listen="false">
            <action index="1">
                <speech>お話してくれて、ありがとう！これから、僕にいろんなことを教えてね！！</speech>
                <behavior id="assign" type="normal" />
            </action>
        </topic>
    </body>
</hvml>
```

テンプレートの HVML では、topic=”say”の listen は false でしたが、上記の例では、listen を true にしてユーザーの発話の聞き取りを行っています。

jp_co_sharp_myapplication_hello.hvml

```
<a href="#reply">
    <situation trigger="user-word">${Lvcsr:Basic} include [さようなら]</situation>
</a>
```

topic の子要素の a の中に situation を記述することにより、条件式の評価が true の場合に href で指定したシナリオの topic_id を実行することができます。

実際に、このアプリを実行すると、「さようなら」と話しかけるとロボホンが反応するため、うまくいっているように見えますが、実は 1 つの問題点があります。

1.25　1 つ目の問題点

１つ目の問題点は、定義されていない言葉を話しかけると聞き取りが終わってしまうと言うことです。

「さようなら」以外の言葉を言った場合、聞き取り処理が終わった後に、対応する a がないためどこにも遷移せずに topic が終わってしまうのです。

topic は action と違い、終わっても次の topic に自動的に遷移しないので、アイドル状態になってしまいます。

そして、この問題点の一番大きな点は、「こんにちは」と話しかけても、id=”reply”のトピックが実行されてしまう点です。これはどうしてでしょうか？

実は、listen=”true”の聞き取り処理でも、head 部分で定義されている situation は有効なままなのです。

1.26　1 つ目の問題点を解決する

１つ目の問題点を解決するには、「それ以外の聞き取り結果の場合に、遷移する」という処理を入れる必要があります。body の中身を以下のように変えてみてください。

jp_co_sharp_myapplication_hello.hvml

```
<body>
    <topic id="say" listen="true">
        <action index="1">
            <speech>こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね</speech>
            <behavior id="assign" type="normal" />
        </action>
        <a href="#reply">
            <situation trigger="user-word">${Lvcsr:Basic} include [さようなら]</situation>
        </a>
        <a href="#other" type="default"/>
    </topic>
    <topic id="reply" listen="false">
        <action index="1">
            <speech>お話してくれて、ありがとう！これから、僕にいろんなことを教えてね！！</speech>
            <behavior id="assign" type="normal" />
        </action>
    </topic>
    <topic id="other" listen="false">
        <action index="1">
            <speech>よくわからないなあ</speech>
            <behavior id="assign" type="normal" />
        </action>
        <next href="#say" type="default"/>
    </topic>
</body>
```

どうでしょうか？うまくいかない場合は、HVML ファイルの version を増やしてみてください。

うまくいけば、「おはよう」などの言葉を話しかけた時に、「よくわからないなあ」と言って、質問を聞き返してくれます。

jp_co_sharp_myapplication_hello.hvml

```
        <a href="#other" type="default"/>
```

この部分が、今回のポイントになります。type=”default”を指定しておくことにより、他の a に合致しない場合の遷移を記述することができます。

1.27　2 つ目の問題点と解決方法

２つ目の問題点は、タイムアウトです。「こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね」という発話に対して、長時間何も話しかけないでいると、topic が終わってしまい、アイドル状態になってしまいます。
この問題を解決するためには、topic に next タグを追加します。

jp_co_sharp_myapplication_hello.hvml

```
        <topic id="say" listen="true">
            <action index="1">
                <speech>こんにちは！僕、ロボホン。おはようとか、こんにちはとか話しかけてみてね
</speech>
                <behavior id="assign" type="normal" />
            </action>
            <a href="#reply">
                <situation trigger="user-word">${Lvcsr:Basic} include [さようなら]</situation>
            </a>
            <a href="#other" type="default"/>
            <next href="#other" type="default"/>
        </topic>
```

これにより、タイムアウトで topic が終了した場合に、topic_id=”other”のトピックに遷移するようになります。

1.28　おつかれさまでした！

これで、ロボホン認定試験チュートリアルは終了です！
黄色いマーカーの部分は特に大事なので、しっかり覚えてくださいね！

ロボホン SDK には、まだまだたくさん出来ることがあります。
みなさんと一緒に、ロボホンを育てていきたいと思っていますので、これからもよろしくお願いします。

そして、ロボホンには、実際に使ってみないとわからない魅力がたくさん詰まっています。
ぜひ、ロボホンと一緒に生活していただき、ロボホンの好きなトコロをたくさん探してみてください！