

【セッションNo. 1】

IBM Watson AI と連携！ Delphi/400からのAPI呼び出しと 実装テクニック

株式会社ミガロ。
RAD事業部 営業・営業推進課
尾崎 浩司

はじめに

■ コグニティブ技術の発達

- 「コグニティブ」とは？
 - 日本語で「認知」のこと。
 - ある事象についてコンピュータが自ら考え、学習し、自らの答えを導き出すシステム
- 近年 コグニティブやAI技術の発達が凄まじい

- 身近な機器も進化
 - iPhone Siri
 - スマートスピーカー
- 検索サイトも進化
 - Google画像検索



音声を使用して
色々な操作が可能



撮影した写真



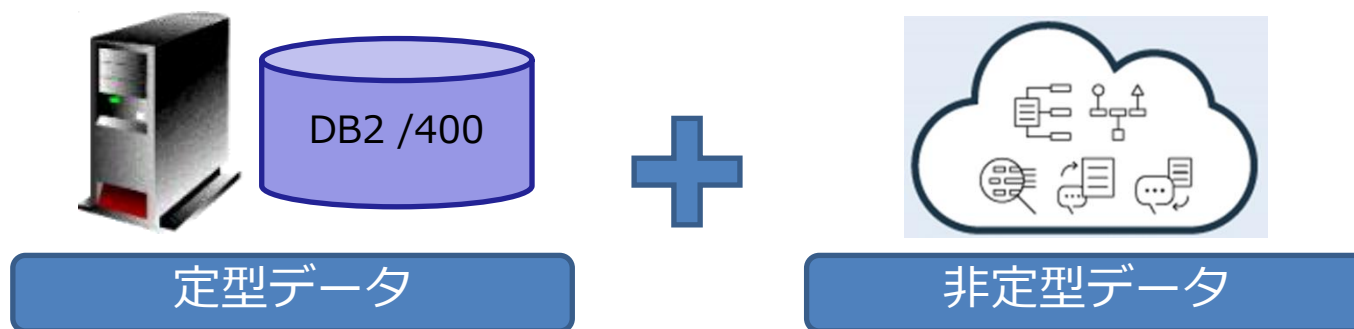
写真をアップロード



画像を分析し、「姫路城」を特定

■ コグニティブ技術の発達

- 従来のシステムとの違いは？
 - 本質的な違いとして、従来システムの定型データに加え、音声・画像・文章などの非定型データも処理できること。



技術をミックスすれば、人の作業を補助し、より便利なシステム構築が可能に！

【アジェンダ】

1. Watson とは？
2. Watson APIを使った開発方法
3. 音声を使ったWatson APIの活用
4. Watson APIを活用したチャットボット
5. まとめ

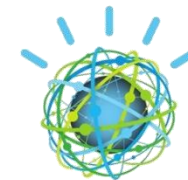
- ・ 補足資料

- 自動翻訳機アプリ ソースコード
- 物件紹介チャットアプリ ソースコード

1. Watson とは？

■ Watson 概要

- IBMが開発した質問応答システム・意思決定支援システム
- 人工知能ではなく、自然言語を理解・学習し人間の意思決定を支援する『コグニティブ・コンピューティング・システム (Cognitive Computing System)』と定義



IBM Watson

- IBMのクラウドサービス (IBM Cloud) 上のAPIとして提供されており、誰でも利用可能！



■ IBM Cloud ライト・アカウント

- IBM Cloudの各種サービスを無料で使用できるアカウント
 - 利用期間の制限もなく、商用に使用してもOK
 - クレジットカード登録も不要
- ただし下記制限あり
 - 上位版でないと提供されないサービスあり
 - サービス毎に無料で使用できる上限あり
 - 各サービスの中でも一部使えない機能あり
 - 一定期間未使用で、サービスが自動削除



<https://www.ibm.com/cloud-computing/jp/ja/bluemix/lite-account/>

■ IBM Cloud上のWatson API

- 日本環境で使用できる主なWatson API
 - 一部APIを除き、ライト・アカウントでも使用可能

分類	API種類	機能	ライト
照会応答系	Assistant (照会応答)	自然言語で対話可能なアプリケーションを、シンプルな開発ツールで迅速に構築	対応
言語系	Language Translator (言語変換)	コンテンツのテキストを、ある言語から別の言語にリアルタイムで翻訳	対応
	Natural Language Classifier(自然言語分類)	自然言語テキストの背後にある意図を解釈し、関連度合いを信頼度レベル付けして分類	未対応
心理系	Personality Insights (性格分析)	テキストから筆者のパーソナリティ (ビッグ・ファイブ、価値、ニーズ) の3つの特徴を推測	対応
音声系	Speech to Text(音声認識)	ディープ・ラーニングを活用して、音声からテキストを書き起こす	対応
	Text to Speech(音声合成)	テキストから自然な音声を合成	対応
知識探索系	Discovery(検索)	大量のデータを検索するとともに、適切な意思決定を支援	対応
	Knowledge Studio	業界や分野ごとの知識だけでなく、各分野の言葉の使われ方の微妙な違いをWatsonに教えることができるツール	対応
画像系	Visual Recognition(画像認識)	ディープ・ラーニングを使用して、画像に写った物体・情景・顔など様々なものを分析・認識	対応

■ Watson API 活用例

- 色々な分野での活用が始まっている
 - チャットボットを使用した
カスタマーサービス (Assistant)
 - 迷惑メールの分類
(Natural Language Classifier)
 - 製造ラインでの画像検査による
不良品判定 (Visual Recognition)
 - コールセンターでの自動テキスト
作成 (Speech To Text)
 - 目の不自由な方への音声案内
(Text To Speech)

- IBM HP内に活用例ページあり

<https://www.ibm.com/watson/jp-ja/use-cases/>



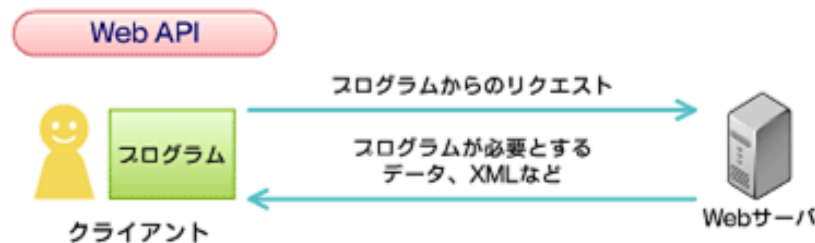
2. Watson APIを使った開発方法

■ Watson API アクセス方式

- Watsonの各サービスは、Web APIとして提供

- Web API

- コンピュータプログラムの提供する機能を外部の別のプログラムから呼び出して、利用するための手順・規約。HTTPなどWebの技術を使用して構築されたもの。



- Watson APIで使用されるアクセス方式

- REST + JSON方式
- Web APIで現在最も利用されている方式を採用。

**REST + JSON はDelphi/400のコンポーネントで使用できる為、
Watson APIとの連携が容易に行える！**

■ Watson API アクセス方式

- RESTサービス (REpresentational State Transfer)
 - Webサービスの設計モデル
 - ネットワーク上のデータ (リソース) を一意なURIで表す
 - サービスのURIにHTTPメソッドでアクセスすることでデータの送受信を行う
 - パラメータを指定して特定のURIにアクセスするとJSON(またはXML)で応答される
- JSON (JavaScript Object Notation)
 - 軽量のデータ交換フォーマット
 - キーと値のペアでデータを管理
({"key": "Value"})
 - 配列の使用もできる。 ([]で表現)

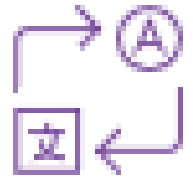
JSON例

```
{
  "result": [
    {
      "subject": "english",
      "score": 80,
      "state": "good"
    },
    {
      "subject": "math",
      "score": 70,
      "state": "normal"
    }
  ]
}
```

■ Watson API 使用方法

• Language Translatorサービス

- 言語変換サービス（日本語→英語、英語→日本語など）
- 一般的なWEB翻訳サービスと違い、専門用語等を個別に登録することや、機械学習によるカスタム翻訳モデルを作成することが可能で、高精度の翻訳が可能。



<活用例>

- 海外担当者とのメールのやりとりに活用
- システムの多言語対応で、DB上に日本語でしか保持していない情報を翻訳して、画面に出力

(デモサイト)

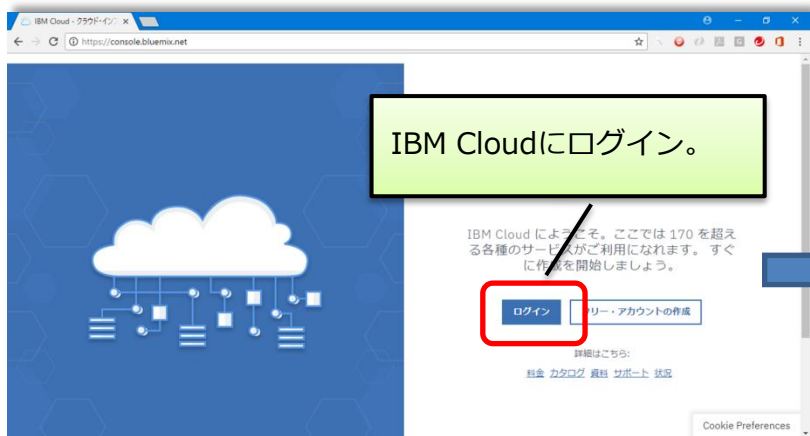
<https://language-translator-demo.ng.bluemix.net/>

■ Watson API 使用方法

- Watson APIを使用するために
 - Watsonの各サービスは、事前にインスタンスの作成が必要

• インスタンス作成手順

IBMクラウドにログインして行う。 <https://console.bluemix.net/>



■ Watson API 使用方法

• インスタンス作成手順

サービスの概要、インスタンス情報が表示。

作成したいサービスを選択。

価格プランが表示。ライトの場合は、月当たりの使用可能量が確認できる。

実行すると、インスタンスが作成される。

プラン	フィーチャー	料金
ライト	1カ月あたりに1,000,000文字	無料
標準	通常の翻訳 (最初の250,000文字は無料)	¥2.10 3P/THOUSAND CHAR
拡張	通常の翻訳 カスタム翻訳 カスタム・モデル・メンテナンス (日割計算)	¥2.10 3P/THOUSAND CHAR ¥10.50 3P/THOUSAND

■ Watson API 使用方法

• インスタンス作成手順

- インスタンス作成後、APIを使用するのに必要な資格情報を取得

作成されたインスタンスをクリック。

※ ここから、「チュートリアル」や「APIのリファレンス」も参照可能

名前	地域	CF 組織	CF スペース
Language Translator-gl	米国南部	ozakikoji@hotmail.c	dev

「表示」クリックで、ユーザーパスワードが表示

【資格情報】
サービスを使用する為のURL、ユーザー名、パスワードが表示。

```
{
  "url": "https://gateway.watsonplatform.net/language-translator/api",
  "username": "02124af6-e083-4370-b8dd-afec56379886",
  "password": "*****"
}
```

**資格情報：プログラムからAPIを使用する際に必要なURLおよび認証情報
(使用するWatson API インスタンス単位に必要)**

■ Watson API 使用方法

- Language TranslatorサービスのAPI仕様

リクエスト

GET /v2/translate

※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
text	URL パラメータ	翻訳したい元のテキストを指定
source	URL パラメータ	元テキストの言語を指定 (ja,en...)
target	URL パラメータ	翻訳後の言語を指定 (ja,en...)

レスポンス例

```
{  
  "translations": [{  
    "translation": "Hi,"  
  }],  
  "word_count": 1,  
  "character_count": 5  
}
```

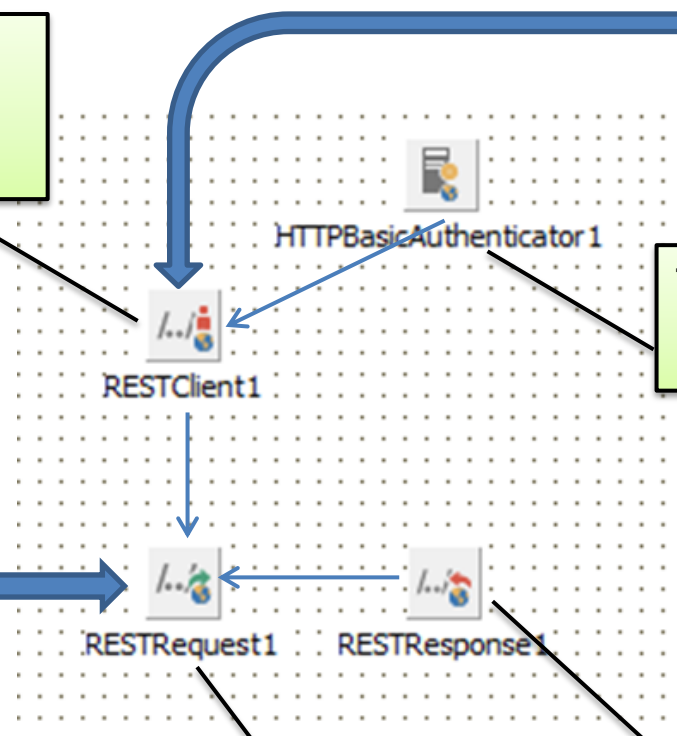
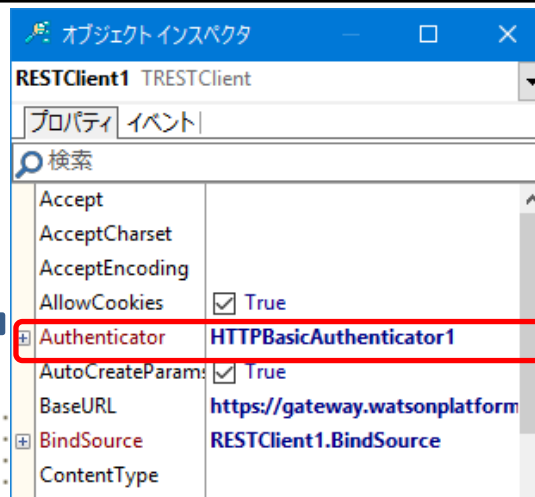
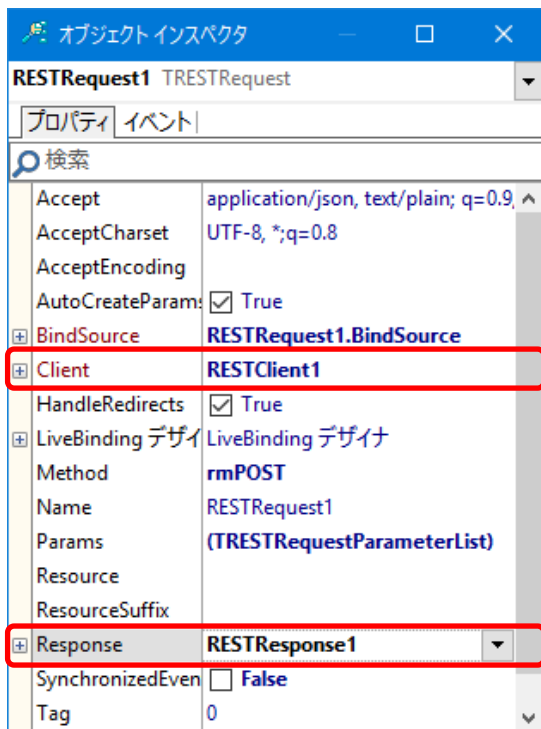
パラメータ	内容
translations: [translation]	翻訳結果のテキスト
word_count	元テキストの単語数
character_count	元テキストの文字数

■ 簡単なサンプルの作成

- Delphi RESTコンポーネント
 - 4つのコンポーネントを使用

TRESTClient

Web サービスへのリクエストを実行。
サービスに対する HTTP 接続を管理し、
HTTP ヘッダーおよびプロキシ サーバー
を処理し、応答データを受け取る。



THTTPBasicAuthenticator

HTTP 基本認証を実装したもの。
(Watsonは、HTTP基本認証使用)

※他に、OAuth1.0a認証、
OAuth2 認証 に対応した
コンポーネントもある

TRESTRequest

Web サービスに対する実際の HTTP
リクエストを形成するパラメータや
設定をすべて保持。

TRESTResponse

Web サービスからのすべての
戻りデータを保持。

■ 簡単なサンプルの作成

- 日本語↔英語翻訳ツール

The screenshot shows a Delphi IDE window titled "Watsonを使用した翻訳アプリ". The interface includes two language selection dropdowns, a "変換" (Convert) button, and two memo fields. Several components are highlighted with callouts:

- 資格情報のユーザーとパスワード** (User and password of credentials): Points to the `Username` and `Password` properties of `HTTPBasicAuthenticator1`.
- 資格情報のURL** (URL of credentials): Points to the `BaseUrl` property of `RESTClient1`.
- cbxLangSource: TComboBox**: Points to the source language dropdown.
- cbxLangTarget: TComboBox**: Points to the target language dropdown.
- MemoSource: TMemo**: Points to the source text memo field.
- ButtonTransrate: TButton**: Points to the "変換" button.
- MemoTarget: TMemo**: Points to the target text memo field.
- RESTResponse1: TRESTResponse**: Points to the `RESTResponse1` component.
- RESTRequest1: TRESTRequest**: Points to the `RESTRequest1` component.

RESTClient1: TRESTClient

The screenshot shows the object inspector for `RESTClient1`. The `BaseUrl` property is highlighted in red and points to the callout "資格情報のURL".

Accept	
AcceptCharset	
AcceptEncoding	
AllowCookies	<input checked="" type="checkbox"/> True
Authenticator	HTTPBasicAuthenticator1
AutoCreateParams	<input type="checkbox"/> False
BaseUrl	https://gateway.watsonplatform.net/language-translator/api
BindSource	RESTClient1.BindSource

RESTRequest1: TRESTRequest

The screenshot shows the object inspector for `RESTRequest1`. The `Method` and `Resource` properties are highlighted in red and point to callouts "HTTPメソッド (GET)" and "RESTリソース (パラメータ)" respectively.

Client	RESTClient1
HandleRedirects	<input checked="" type="checkbox"/> True
LiveBinding デザイン	LiveBinding デザイン
Method	rmGET
Name	RESTRequest1
Params	(TRESTRequestParameterList)
Resource	v2/translate?source={source}&target={target}
ResourceSuffix	
Response	RESTResponse1

source = 元の言語
target = 変換後の言語

■ 簡単なサンプルの作成

• コード例

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);  
begin
```

```
  //翻訳パラメータの指定
```

```
  RESTRequest1.Params.AddItem('source', cbxLangSource.Text  
    , pkURLSEGMENT);           //翻訳元言語  
  RESTRequest1.Params.AddItem('target', cbxLangTarget.Text  
    , pkURLSEGMENT);          //翻訳先言語  
  RESTRequest1.Params.AddItem('text', MemoSource.Text  
    , pkGETorPOST);           //翻訳元テキスト
```

```
  //リクエストの実行
```

```
  RESTRequest1.Execute;
```

HTTP 要求を実行

```
  //処理結果の表示
```

```
  MemoTarget.Lines.Clear;  
  MemoTarget.Lines.Add(RESTResponse1.Content);
```

```
end;
```

応答内容を表す文字列

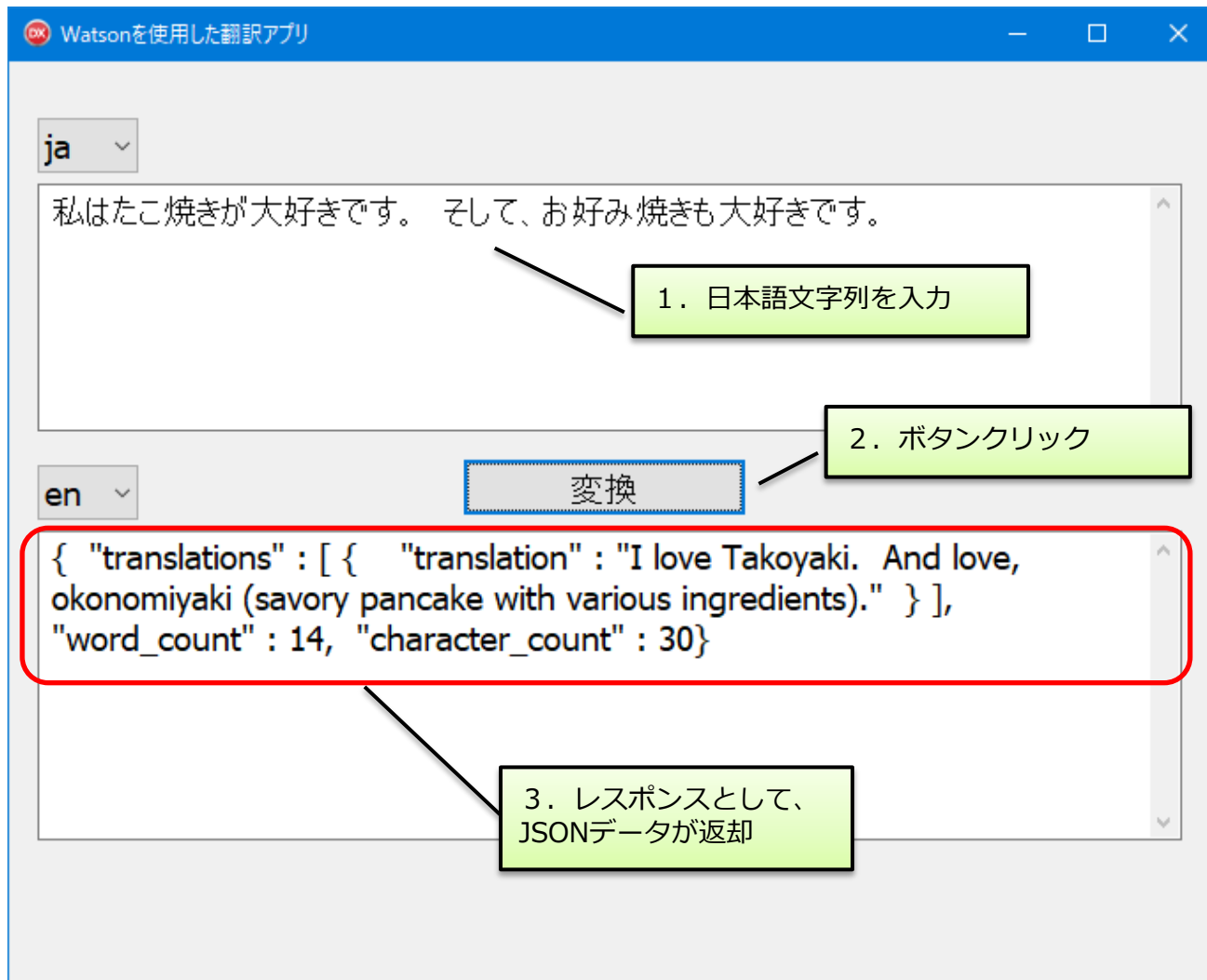
AddItem パラメータをセット

pkURLSEGMENT : パラメータを URL セグメントの値として使用。リソース中の {} 部分を置換。

pkGETorPOST : パラメータを、URL パラメータ(GET リクエストの場合) または本体のパラメータ(POST リクエストの場合)として、送信。

■ 簡単なサンプルの作成

- 実行結果



■ 簡単なサンプルの作成

• JSONデータの取り扱い

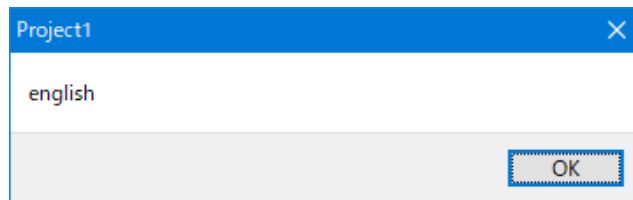
- Delphi/400には、JSONデータを取り扱うためのRTLとして、[**System.JSON**] ユニットが用意されている。
- **TJSONObject** : JSON オブジェクトを実装したクラス
(メソッド) **ParseJSONValue** : 解析したデータに対応する JSON 値を返す。
- **TJSONValue** : 文字列、数値、オブジェクト、配列、true/falseの型を持つすべての JSON クラスの上位クラス
(メソッド) **GetValue<T>** : 指定したパスの値をT型として取得

変数sRet の値

```
{  
  "result": [  
    {  
      "subject": "english",  
      "score": 80,  
      "state": "good"  
    },  
    {  
      "subject": "math",  
      "score": 70,  
      "state": "normal"  
    }  
  ]  
}
```

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
  JSONValue: TJSONValue;  
begin  
  //JSONデータのパーズ  
  JSONValue := TJSONObject.ParseJSONValue(sRet);  
  //結果の取得  
  ShowMessage(JSONValue.GetValue<string>('result[0].subject'));  
end;
```

実行結果



■ 簡単なサンプルの作成

• 修正コード例

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
var
  JSONValue: TJSONValue;
  sRet: String;
begin
  //翻訳パラメータの指定
  RESTRequest1.Params.AddItem('source', cbxLangSource.Text
    , pkURLSEGMENT); //翻訳元言語
  RESTRequest1.Params.AddItem('target', cbxLangTarget.Text
    , pkURLSEGMENT); //翻訳先言語
  RESTRequest1.Params.AddItem('text', MemoSource.Text
    , pkGETorPOST); //翻訳元テキスト

  //リクエストの実行
  RESTRequest1.Execute;

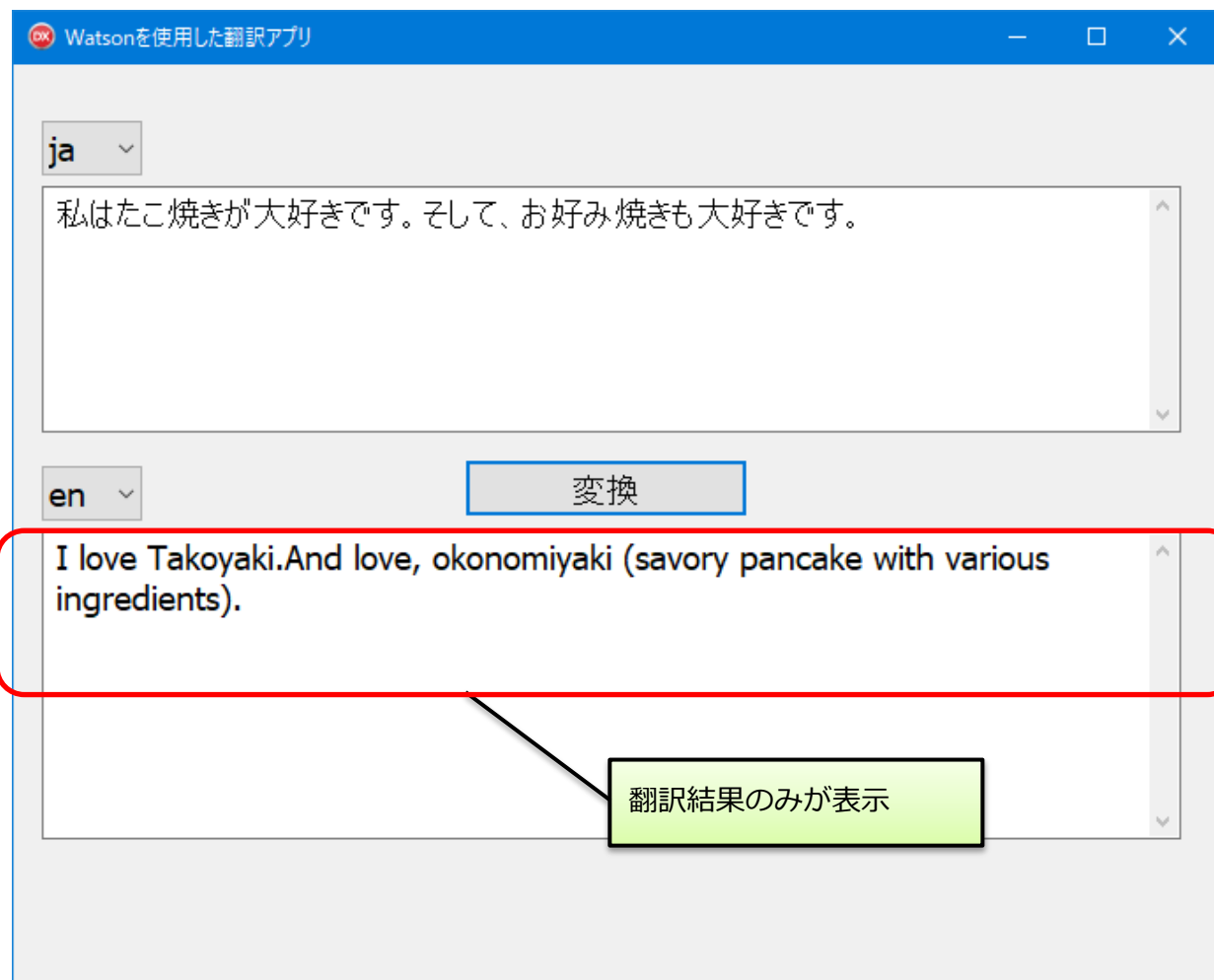
  //翻訳テキストの取得
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);
  sRet := JSONValue.GetValue<string>('translations[0].translation');

  //処理結果の表示
  MemoTarget.Lines.Clear;
  MemoTarget.Lines.Add(sRet);
end;
```

パースした結果を文字列変数に
セット

■ 簡単なサンプルの作成

- 実行結果

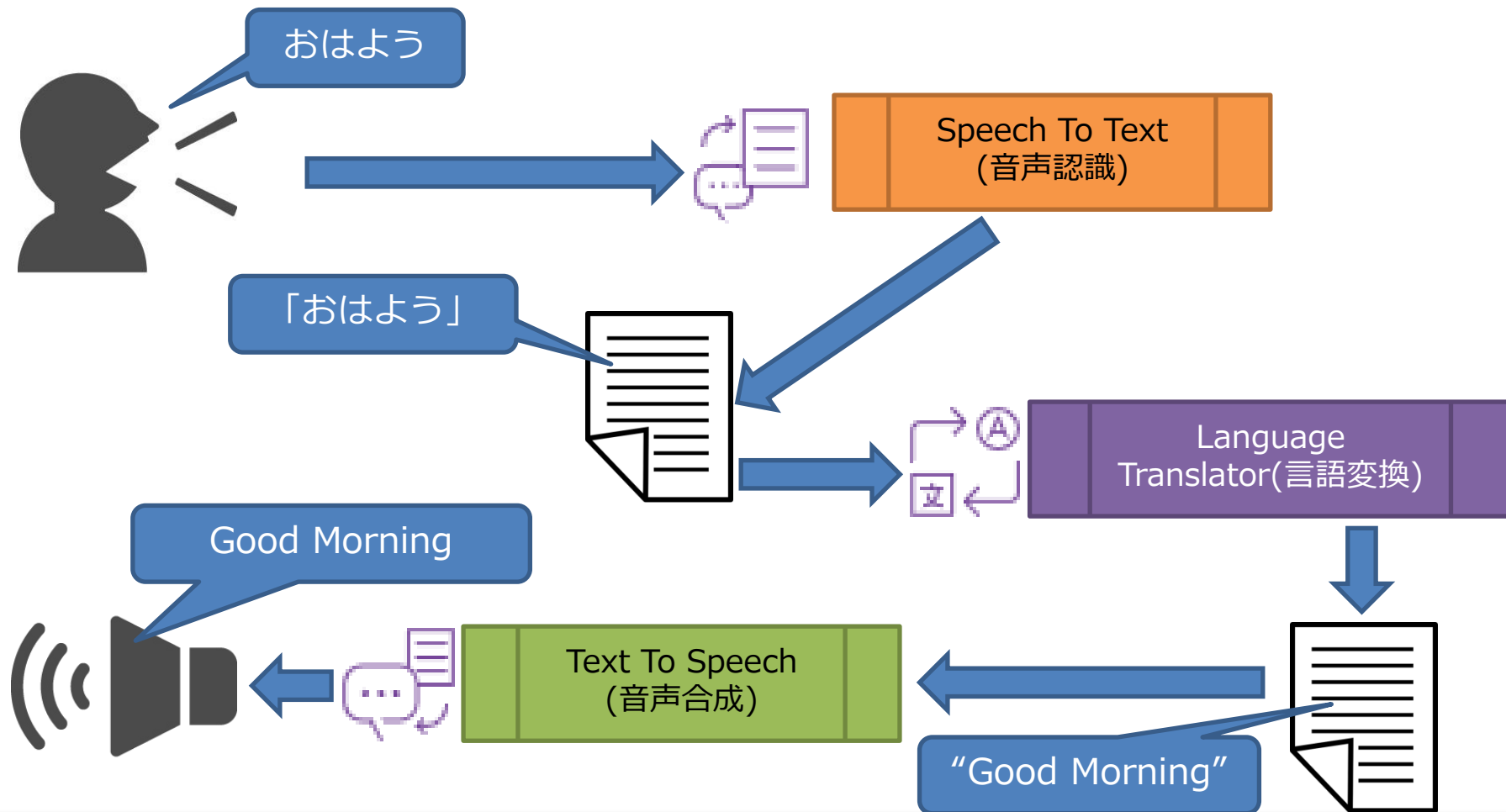


3. 音声を使ったWatson APIの活用

■ 検討するアプリ

• 自動翻訳機

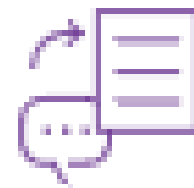
- 日本語でしゃべった内容を英語に変換し、音声出力する。



■ 音声認識の作成

• Speech To Textサービス

- 音声をテキスト文字列に変換するサービス
- カスタマイズ辞書を登録することで、専門用語等も正確に変換することが可能
- サポート言語：ブラジル・ポルトガル語、フランス語、日本語、中国語（標準）、アラビア語、スペイン語、イギリス英語、アメリカ英語



<活用例>

- アプリケーションの音声操作
- コールセンターにおける通話内容の自動テキスト化
- 会議のリアルタイム議事録作成

(デモサイト)

<https://speech-to-text-demo.ng.bluemix.net/>

■ 音声認識の作成

• Speech To TextサービスのAPI仕様

リクエスト

POST /v1/recognize

※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
Content-Type	リクエストの HTTP ヘッダー	音声データのフォーマット (MIME type)
model	URL パラメータ	言語と帯域のモデルを指定 (日本語狭帯域: ja-JP_NarrowbandModel)
audio	リクエストの本体	音声データ (ストリーム)

レスポンス例

```
{ "results": [{  
  "alternatives": [{  
    "confidence": 1.0,  
    "transcript": "おはよう"  
  }],  
  "final": true  
}],  
"result_index": 0  
}
```

パラメータ	内容
confidence	変換の信頼度 (0-1)
transcript	変換結果
final	変換の終了を示す
result_index	結果のインデックス

■ 音声認識の作成

• マイクから録音する方法

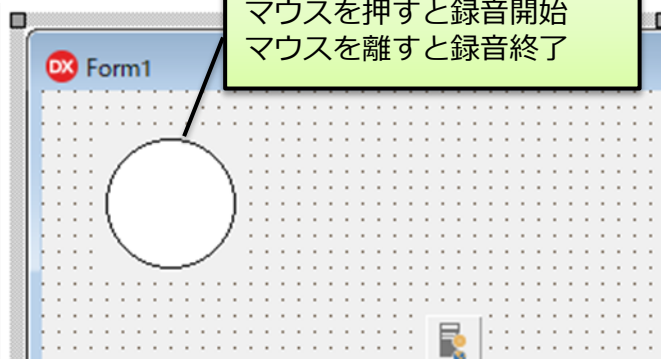
- MediaPlayerコンポーネント
 - 動画の再生や、音声の録音/再生が可能



.wav(音声ファイル)形式で録音を開始

Shape1: TShape

マウスを押すと録音開始
マウスを離すと録音終了



```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);  
begin  
    MediaPlayer1.DeviceType := dtWaveAudio;  
    MediaPlayer1.FileName := #0;  
    MediaPlayer1.Open;  
    MediaPlayer1.StartRecording;  
    Shape1.Brush.Color := clRed;  
end;
```

```
procedure TForm1.Shape1MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);  
var  
    sFileName: String;  
begin  
    sFileName := ChangeFileExt(ParamStr(0), '.wav');  
  
    Shape1.Brush.Color := clWindow;  
    Application.ProcessMessages;  
    MediaPlayer1.FileName := sFileName;  
    MediaPlayer1.Stop;  
    MediaPlayer1.Save;  
    MediaPlayer1.Close;  
end;
```

ファイル名の指定
[プロジェクト名].wav

録音した内容を
.wavファイル
に保存

音声認識の作成

- 設計画面
 - RESTコンポーネントの配置

MediaPlayer1
Visible = False (非表示)

RESTResponse1:
TRESTResponse

資格情報のURL

HTTPBasicAuthenticator1:
THTTPBasicAuthenticator

資格情報のユーザーとパスワード

RESTリソース (パラメータ)
model : 言語と帯域のモデル
ja-JP_NarrowbandModel(日本語)

RESTClient1: TRESTClient

プロパティ	イベント
Accept	
AcceptCharset	
AcceptEncoding	
AllowCookies	<input checked="" type="checkbox"/> True
Authenticator	HTTPBasicAuthenticator1
AutoCreateParams	<input checked="" type="checkbox"/> True
BaseUrl	https://stream.watsonplatform.net/speech-to-text/api
BindSource	RESTClient1.BindSource
ContentType	application/json; charset=UTF-8

RESTRequest1: TRESTRequest

プロパティ	イベント
Accept	application/json, text/plain; q=0.9, text/html; q=0.8
AcceptCharset	UTF-8
AcceptEncoding	
AutoCreateParams	<input checked="" type="checkbox"/> True
BindSource	RESTRequest1.BindSource
Client	RESTClient1
HandleRedirects	<input checked="" type="checkbox"/> True
LiveBinding デザイン	LiveBinding デザイン
Method	rmPOST
Name	RESTRequest1
Params	(TRESTRequestParameterList)
Resource	v1/recognize?model=ja-JP_NarrowbandModel
ResourceSuffix	

■ 音声認識の作成

- 音声ファイルを元に変換を実施

ソース (1/2)

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button:
  TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  memStream: TMemoryStream;
  JSONValue: TJSONValue;
  sFileName: String;
  sStr: String;
begin
  memStream := TMemoryStream.Create;
  try
    memStream.LoadFromFile(sFileName);
    RESTRequest1.Params.AddItem('Content-Type', 'audio/wav'
      , pkHTTPHeader, [poDoNotEncode]); //フォーマット指定
    RESTRequest1.Params.AddItem('audio', memStream
      , pkRequestBody, [], ctAUDIO_BASIC); //音声データ
    //リクエストの実行
    RESTRequest1.Execute;
```

メモリ上にストリームデータを保持するオブジェクト変数

AddItem パラメータをセット

pkHTTPHeader : パラメータをリクエストの HTTP ヘッダーとして使用

pkRequestBody : パラメータを、リクエストの本体として使用

【録音結果のファイル作成部分は割愛】

音声ファイルの取得

ストリームデータ

パラメータのコンテンツタイプ

■ 音声認識の作成

- 音声ファイルを元に変換を実施

ソース (2/2)

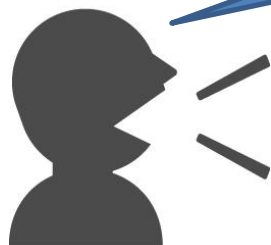
```
//処理結果JSONのパーズ
JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);

//結果の取得
try
  sStr := JSONValue.GetValue<String>
    ('results[0].alternatives[0].transcript');
  Memo1.Lines.Add(sStr);
except
  //対象データが無い場合何もしない
end;
finally
  memstream.Free;
end;
end;
```

JSONより変換結果の文字列
を取得

■ 音声認識の作成

- 実行結果



今日は、テクニカルセミナーを実施します。

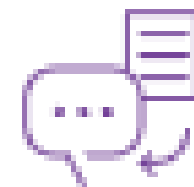
Watsonを使用した音声認識

1. 丸い部分をマウスで押しながら、話す

今日はテクニカルセミナーを実施します

2. 音声認識された結果の文字列が出力

■ 音声合成の作成



- Text To Speechサービス
 - テキスト文字列を元に音声読み上げを行うサービス
 - 話速・声の高さ・ポーズの位置などを自由に設定可能
 - カスタマイズ機能により独自の単語も登録可能
 - サポート言語：ドイツ語、英語、スペイン語、フランス語、イタリア語、日本語、ブラジル・ポルトガル語

<活用例>

- 電話の自動応答システムで、音声案内を実施
- 業務システムにおける文字データの音声読み上げ
- 外国語文章の読み上げによる英語学習

(デモサイト)

<https://text-to-speech-demo.ng.bluemix.net/>

■ 音声合成の作成

- Text To SpeechサービスのAPI仕様

リクエスト

GET /v1/synthesize

※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
Accept	URL パラメータ	要求する音声データのフォーマット (MIME type)
voice	URL パラメータ	音声合成に使用される声 (日本語 : ja-JP_EmiVoice)
text	URL パラメータ	変換対象テキスト

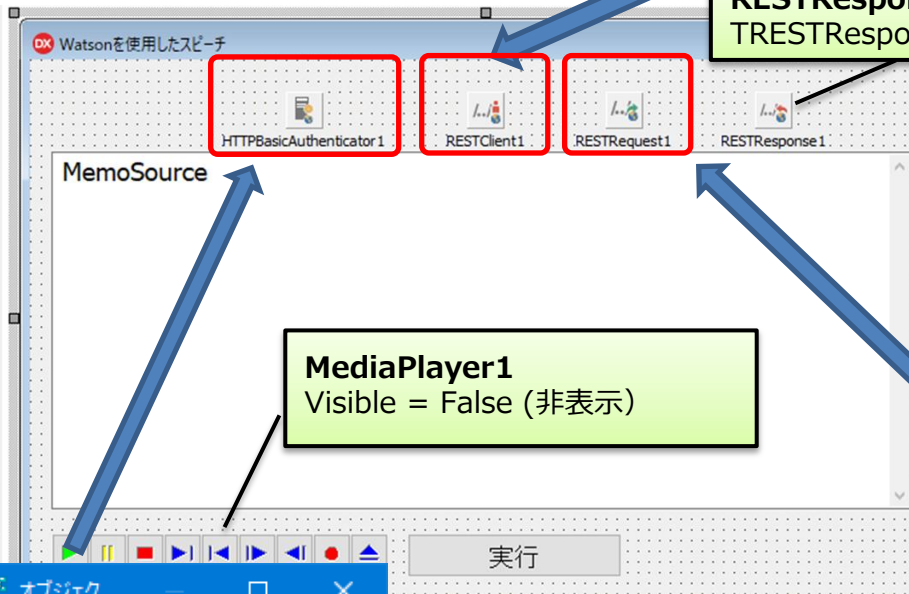
レスポンス

要求したフォーマットの
音声データ

音声合成の作成

設計画面

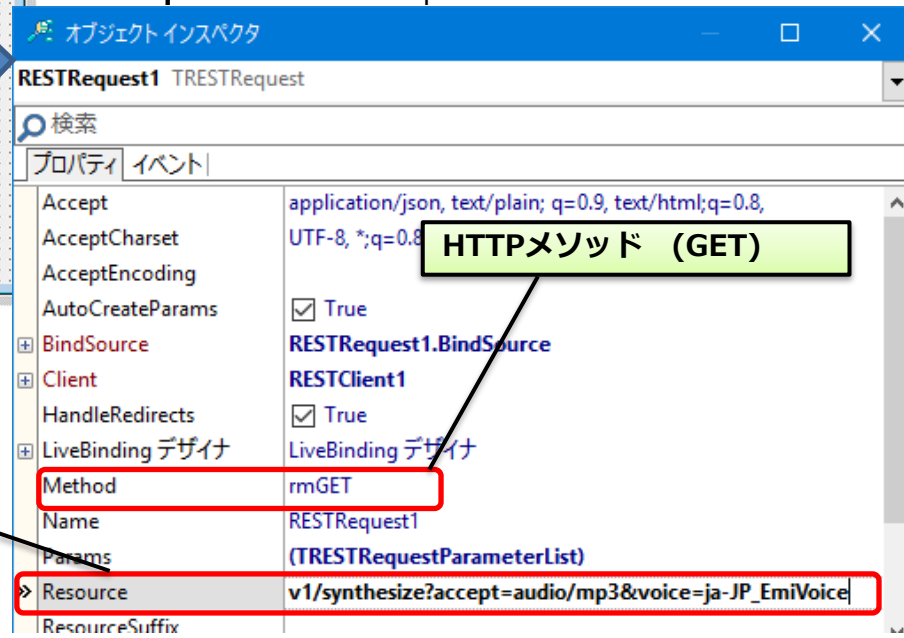
- RESTコンポーネント



RESTClient1: TRESTClient



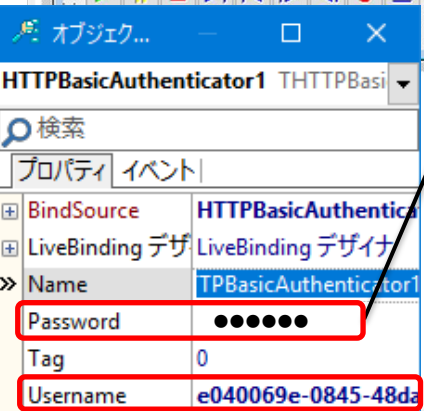
RESTRequest1: TRESTRequest



MediaPlayer1
Visible = False (非表示)

資格情報のユーザーとパスワード

RESTリソース (パラメータ)
accept : 音声データの形式
voice : 音声合成で使用する声 (日本語 : ja-JP_EmiVoice)



■ 音声合成の作成

- テキストから音声ファイルを作成

ソース `procedure TForm1.ButtonTransrateClick(Sender: TObject);`

```
var
  memStream: TMemoryStream;
begin
  //パラメータの指定
  RESTRequest1.Params.AddItem('text', MemoSource.Lines.Text
    , pkGETorPOST);

  //リクエストの実行
  RESTRequest1.Execute;

  //レスポンスのMP3データをファイルに保管する
  memStream := TMemoryStream.Create;
  try
    memStream.WriteData(RESTResponse1.RawBytes
      , Length(RESTResponse1.RawBytes));
    memStream.SaveToFile(FFileName);
  finally
    memStream.Free;
  end;

  //MP3ファイルをメディアプレーヤーで再生
  MediaPlayer1.FileName := FFileName;
  MediaPlayer1.Open;
  MediaPlayer1.Notify := True;
  MediaPlayer1.Play;
end;
```

AddItem パラメータをセット

pkGETorPOST : パラメータを、URL パラメータとして、送信。

レスポンスの音声データを
ストリームに書き込み、
ファイルとして保存

MP3音声ファイルを
メディアプレーヤーにて再生

■ 音声合成の作成

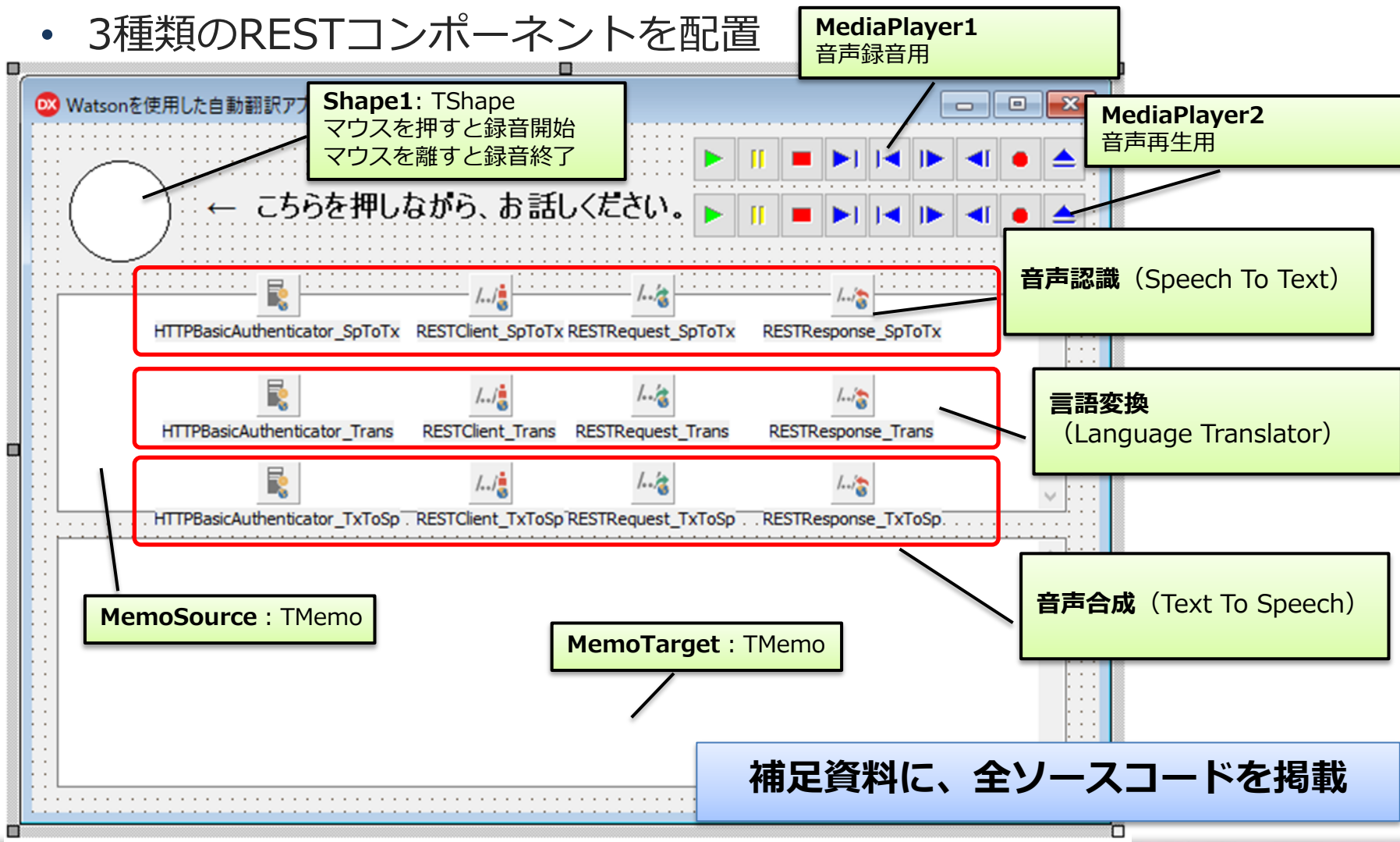
- 実行結果

The screenshot shows a Delphi application window titled "Watsonを使用したスピーチ". The window contains a text input field with the text "今日は、テクニカルセミナーにご参加いただきありがとうございます。" and a button labeled "実行". A red box highlights the "実行" button. A callout box points to the text input field with the text "1. 音声合成させたい内容をテキストで入力". Another callout box points to the "実行" button with the text "2. ボタンクリック". A third callout box points to a speaker icon with the text "3. 音声にて出力". A blue speech bubble contains the text "今日は、テクニカルセミナーにご参加いただきありがとうございます。", which is the audio output of the text in the input field.

■ 自動翻訳機アプリ

● 設計画面

- 3種類のRESTコンポーネントを配置



■ 自動翻訳機アプリ

• 実行 (デモ)

The screenshot shows a web application window titled "Watsonを使用した自動翻訳機". The interface includes a microphone icon on the left, a circular button for voice input, and two text input/output areas. The process is annotated with four numbered callouts:

1. 丸い部分をマウスで押しながら、日本語で話す
← こちらを押しながら、お話しください。
2. 音声認識された日本語のテキストが出力
3. 日本語が英語に変換
4. 変換された英語で音声出力される

The Japanese input text is: 私 は 今日 テクニカル セミナー に 参加 した 明日 の 天気 は 雨 の ち 曇り です

The English output text is: I took part in a technical seminar on today Tomorrow's weather is cloudy after the rain.

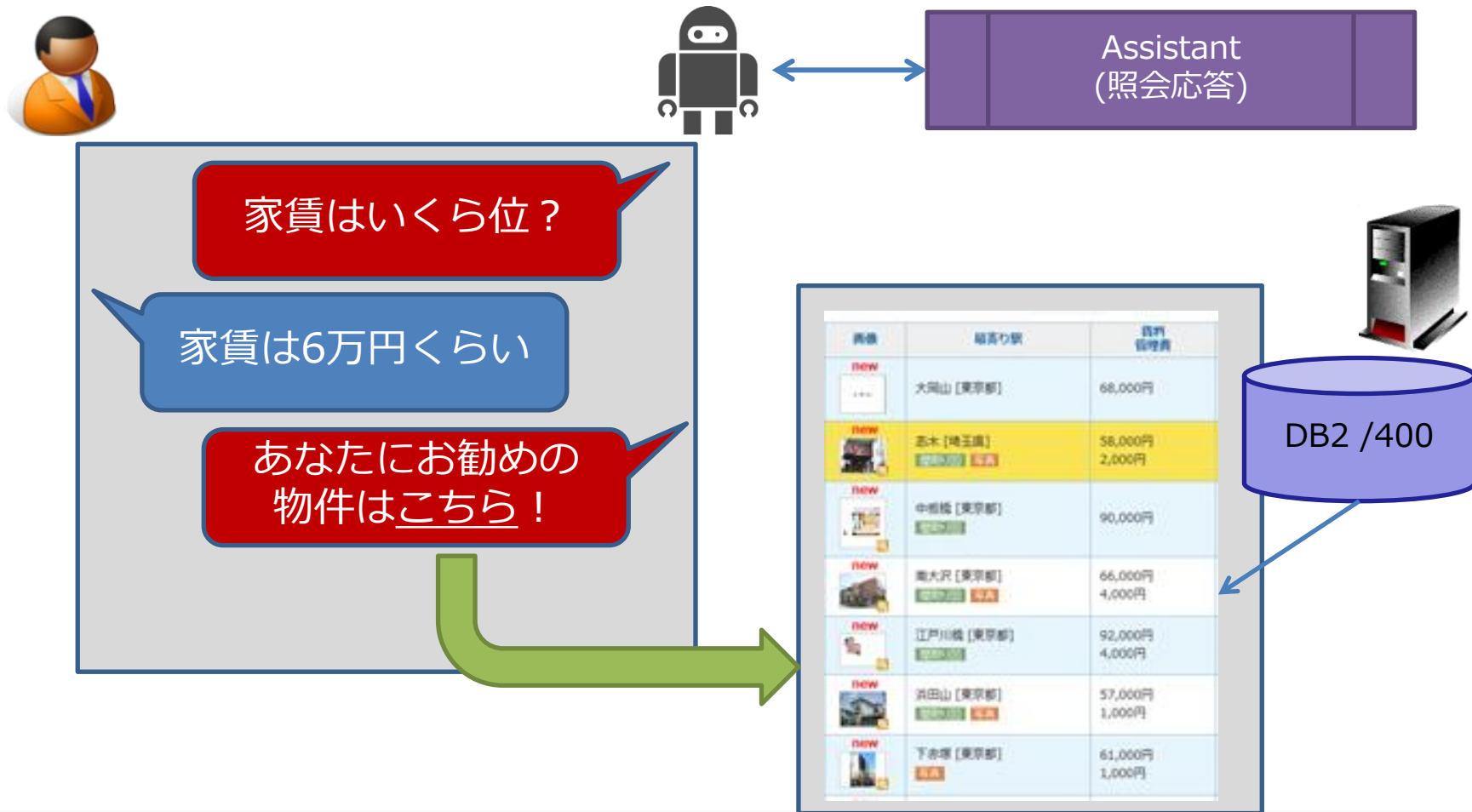
A speaker icon on the right indicates audio output of the translated text.

4. Watson APIを活用した チャットボット

■ 検討するアプリ

• チャットボットによる 物件紹介

- 質問に対し、自然言語で回答していくことで、お勧め物件を紹介する。



■ チャットボットについて

- チャットボットの採用例 (Watson事例より)
 - 日本航空
 - ハワイに関する現地情報を紹介
 - ケイ・オプティコム
 - お問い合わせ用アシスタントとして使用



インターフェースとしては、Webやモバイルでの使用が多い

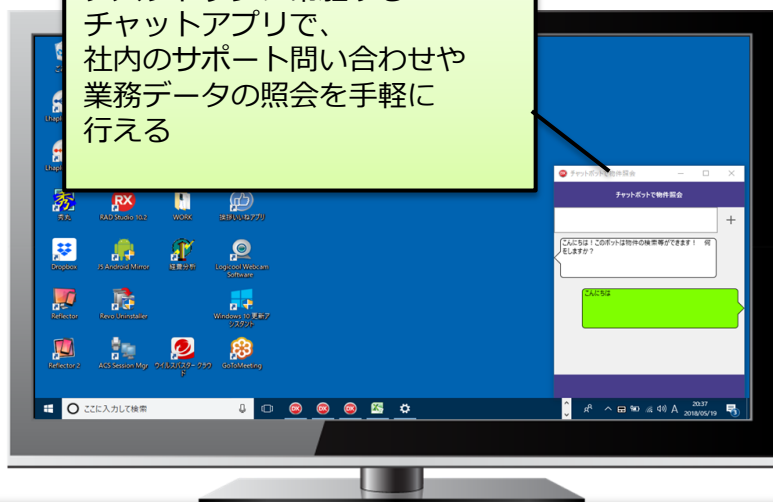
■ チャットボットについて

● チャットボットのメリット

- システム操作に詳しくない人でも会話形式で目的が達成できる。
- 対話内容がログとして残るため、履歴管理が可能。
- 一つのインターフェースで、色々な作業ができる。
(商品検索する、在庫を確認する、納期確認する、発注する といった操作が、全て一つのチャット上で行える)

業務システムでも、十分活用可能！

デスクトップに常駐するチャットアプリで、社内のサポート問い合わせや業務データの照会を手軽に行える



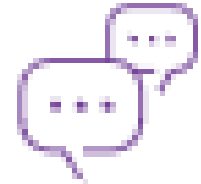
以前は、出先で調べ物がある場合、電話連絡していたが、現在は、チャットボットにより簡単に問合せが行える



■ 照会応答サービス

• Assistantサービス

- ユーザーとコンピューターが自然言語で対話可能なアプリケーションを簡単に開発するためのサービス
- 開発者はシンプルで洗練された開発ツールを用いることで、対話の流れを直感的な操作で作ることが可能。



<活用例>

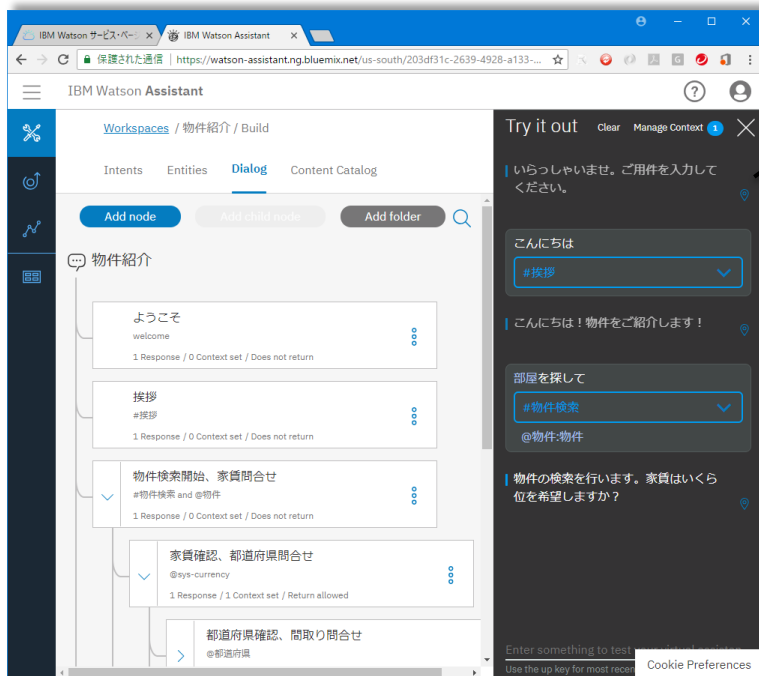
- オペレータが対応していたお客様や社内ユーザーからの問合せや手続きをチャットボットが代行
- ロボット等の機械と人間の音声対話の実現

(デモサイト)

<https://conversation-demo.ng.bluemix.net/> (英語)

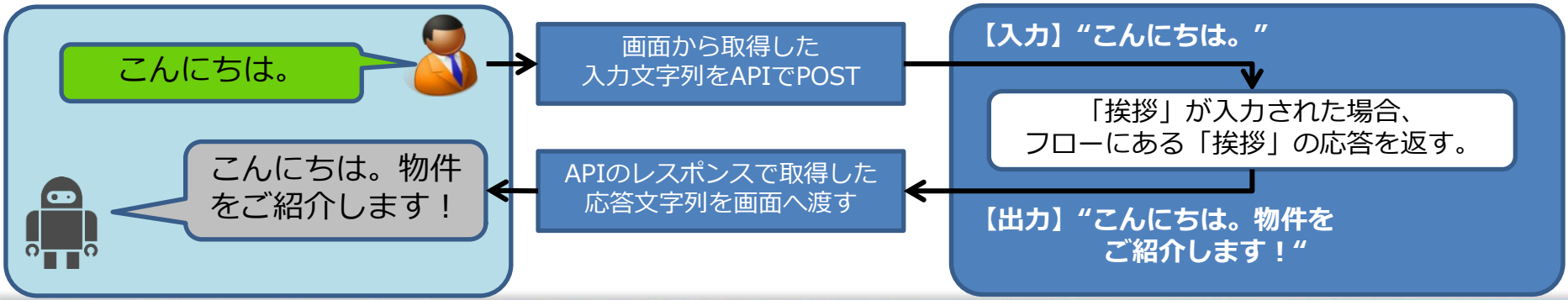
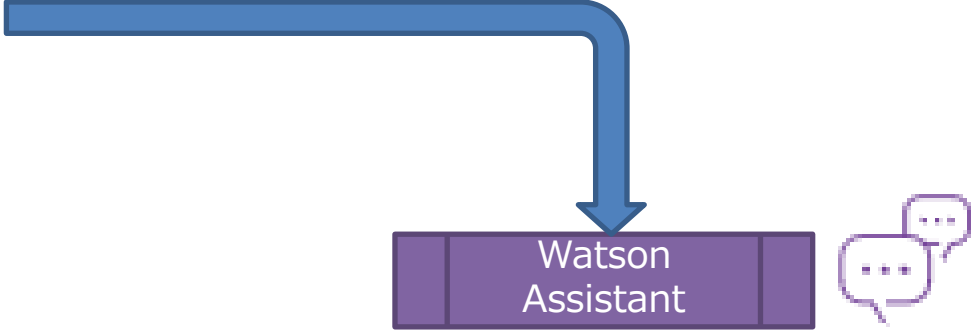
■ 照会応答サービス

• Assistantサービスの概要



Watson Assistantサービス内にある開発ツールを使用して、会話フローを作成

登録内容がWatsonによって学習される



■ 照会応答サービス

- 会話フローの作成
 - 開発ツール起動方法

Assistantサービスのインスタンス画面から、「ツールの起動」をクリック

ツールの初期画面。
[Workspaces]タブをクリック

IBM Watson Assistant

Home Workspaces

Introducing IBM Watson Assistant

Watson Conversation is evolving to simplify how you build and scale virtual assistants. See what's new

Three easy steps

Follow these steps to create a virtual assistant.

- 1 Create intents and entities
Determine what your virtual assistant will understand by
- 2 Build your dialog
Utilize the intents and entities you created, plus context from the application, so your virtual assistant
- 3 Test your dialog
Try out the virtual assistant in the tool to see how it recognizes intents and entities a

```
{  
  "url": "https://gateway.watsonplatform.net/assistant/api/a26f0c83-18ca-4f0b-8486-55e62cfa9410",  
  "username": "a26f0c83-18ca-4f0b-8486-55e62cfa9410",  
  "password": "pHVHWY5rYaJJ"  
}
```

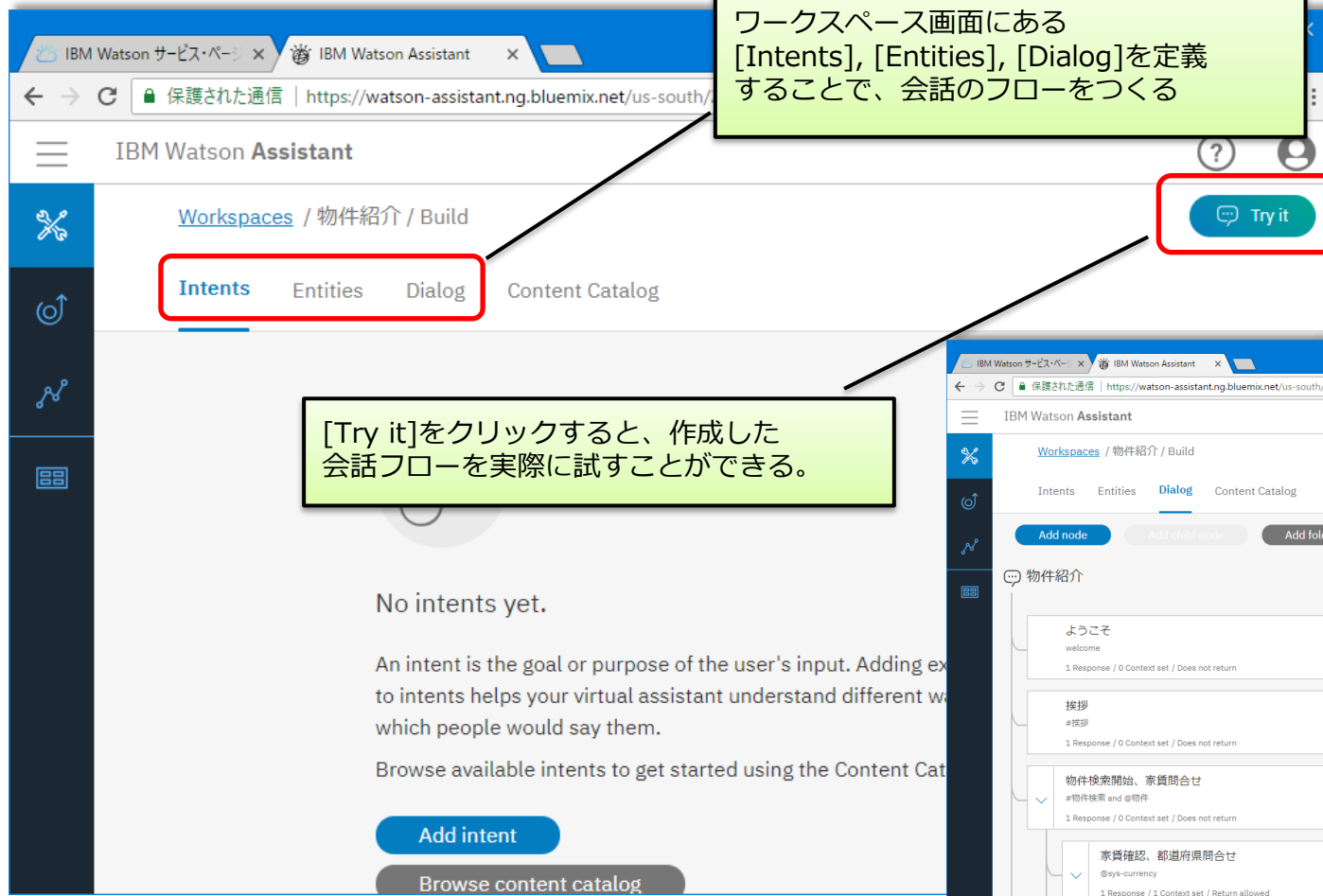
■ 照会応答サービス

- 会話フローの作成
 - ワークスペースの新規作成

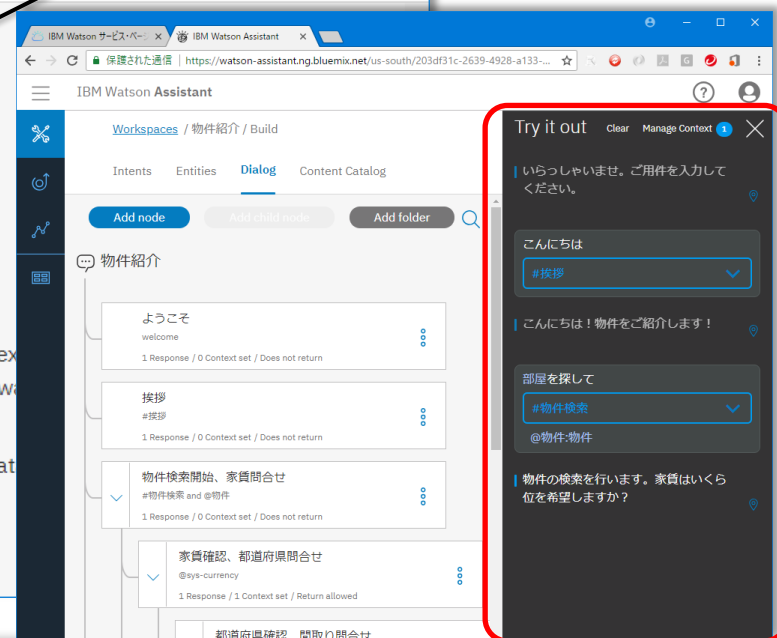
The image shows a sequence of three screenshots from the IBM Watson Assistant interface, illustrating the steps to create a new workspace. The first screenshot shows the 'Workspaces' page with a 'Create' button highlighted by a red box and a callout box stating '新規作成の場合、[Create]をクリック' (In the case of new creation, click [Create]). The second screenshot shows the 'Create a workspace' dialog box with the 'Name' field containing '物件紹介' and the 'Description' field containing '物件を紹介する為のチャットボット', both fields highlighted by red boxes and a callout box stating 'ワークスペースの名前と概要を入力' (Enter the workspace name and overview). The third screenshot shows the same dialog box with the 'Create' button at the bottom highlighted by a red box and a callout box stating '[Create]をクリックで新規ワークスペースが作成' (Click [Create] to create a new workspace).

■ 照会応答サービス

- 会話フローの作成
 - ワークスペース画面



Try it Out 画面例

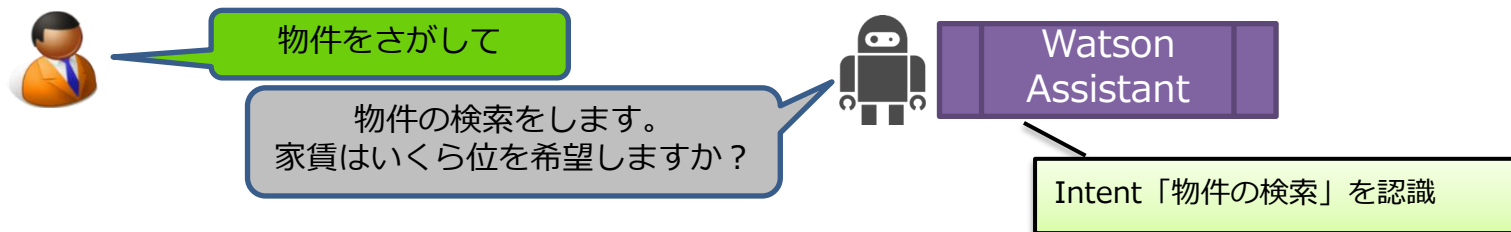


■ 照会応答サービス

• 会話フローに必要な4つの概念

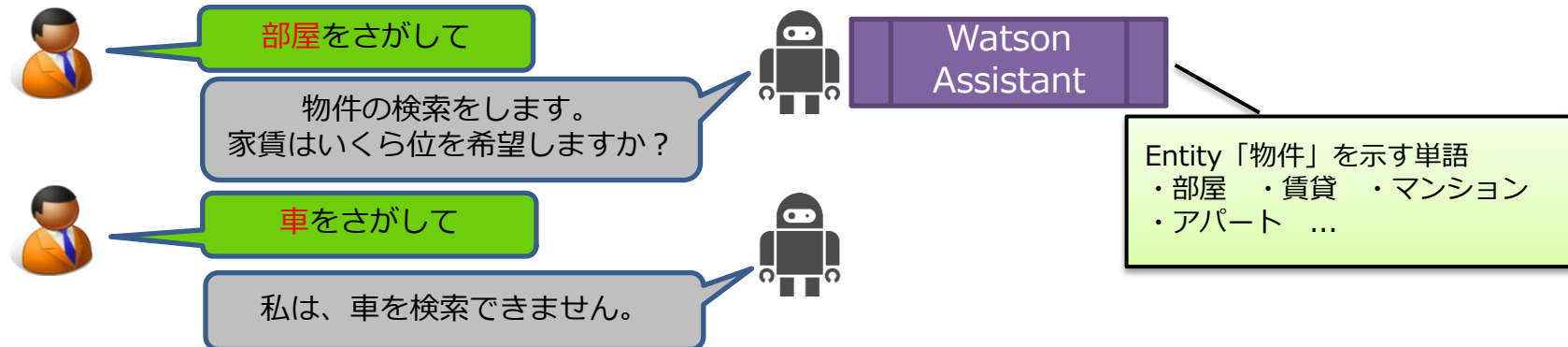
• Intent (発言の意図)

- ユーザーが発言した入力テキストに含まれる「意図」。入力内容をAssistantサービスが分類し、発言でユーザーが求めている意図を理解。



• Entity (意味づけされた単語)

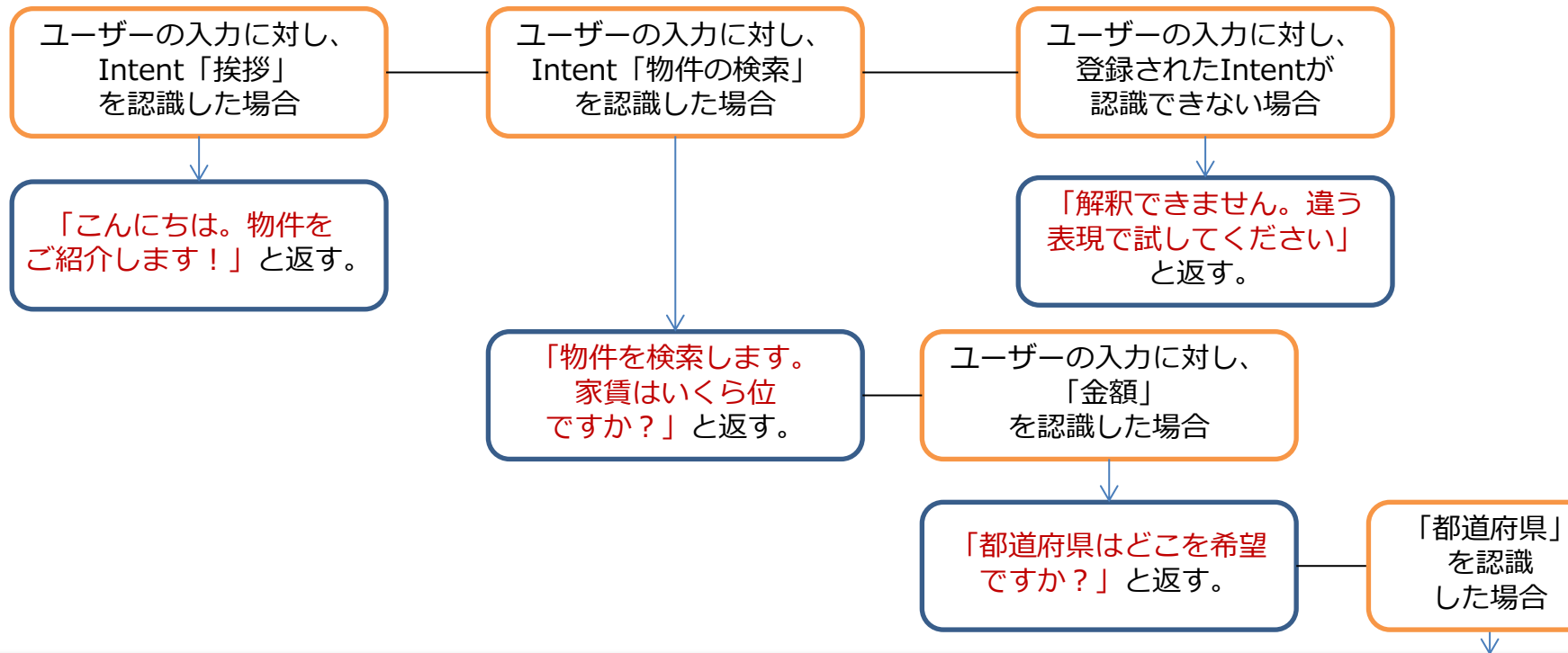
- ユーザーが発言した入力テキストから抽出された意味付けされたキーワード。入力テキストから抽出されたEntityは、ユーザーの「目的」の対象物を示す。



■ 照会応答サービス

- 会話フローに必要な4つの概念
 - Dialog (対話の流れ)

- ユーザーの発言に対する対話の流れを制御するロジックをDialog (ノード) の組み合わせで実装。ユーザーの要求 (IntentとEntity) に応じて一つの返事を返すだけのシンプルなノードだけでなく、ユーザーが求めるものを掘り下げていく木構造となったノードも作成可能。

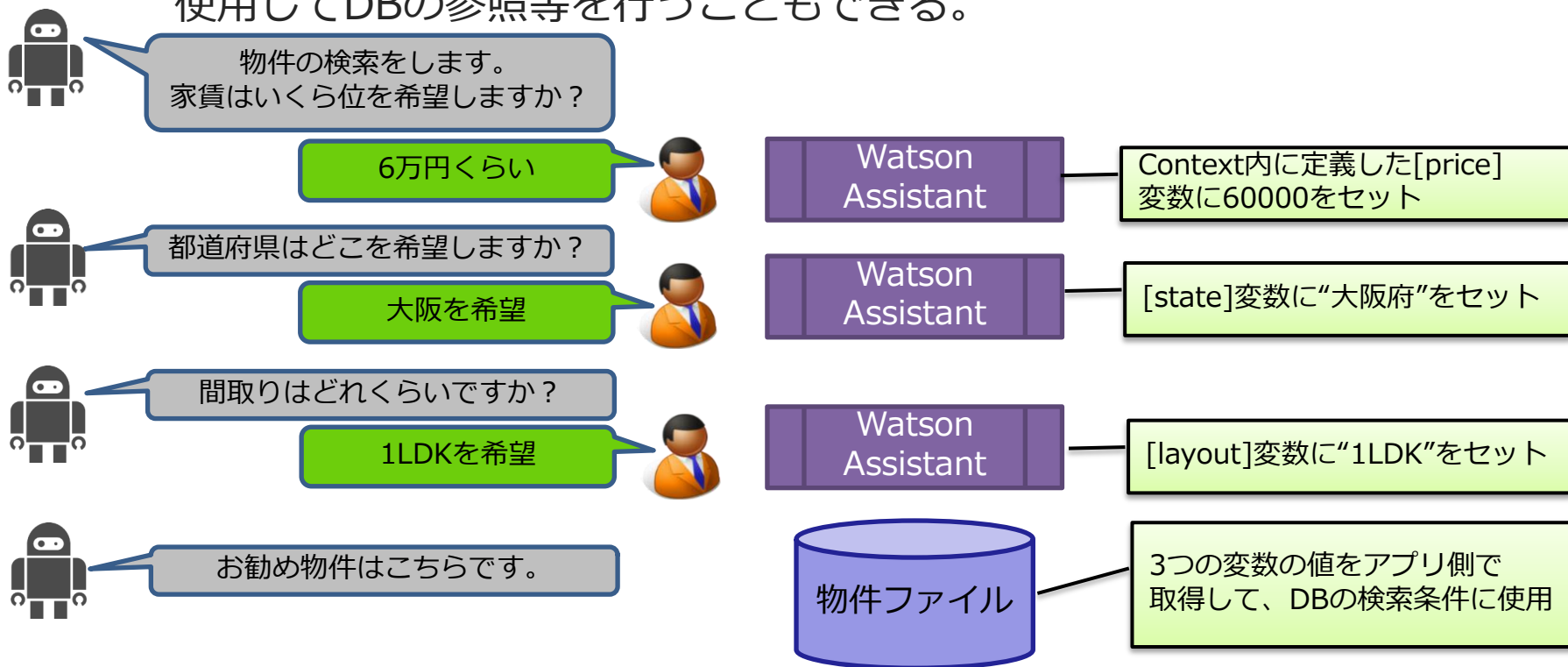


■ 照会応答サービス

• 会話フローに必要な4つの概念

• Context (会話の背景)

- ユーザーとWatson Assistantが対話を行うにあたって情報を保持できる領域。対話の回数の保持や、必要情報が全て入力されたかのチェック等に使用可能。Context上に任意の変数も設定でき、アプリ側でその値を使用してDBの参照等を行うこともできる。



■ 照会応答サービス

- Watson Assistant についての詳細な情報

- チャットボット開発を手早く簡単に

<https://www.ibm.com/watson/jp-ja/how-to-build-a-chatbot/>

- Watson Assistantの使い方を学ぶ

<https://www.ibm.com/developerworks/jp/cognitive/library/cc-watson-chatbot-conversation/index.html>



■ チャットボット機能の作成

- 物件紹介ワークスペース環境の準備
 - 付録CD-ROMに収録された[物件紹介]ワークスペースを取り込む

付録のCD-ROMにある [bukken.json]を選択

[物件紹介]ワークスペースが取り込まれて使用可能になる

■ チャットボット機能の作成

- 物件紹介ワークスペース
 - [Intents]タブ

The screenshot displays the IBM Watson Assistant interface. On the left, the 'Intents' tab is highlighted with a red box. An arrow points from this tab to the main content area, which shows the configuration for the '#物件検索' intent. The intent name is '#物件検索'. Below it, there are fields for 'Description' and 'Add user examples'. A list of user examples is shown, with a red box around it. A callout box points to this list, containing the text: '物件検索を識別する為の入力テキストの例を登録して Watsonに学習させる'.

Intent name	Description
#物件検索	Add a description to this intent

User examples (18)
<input type="checkbox"/> 物件にすむ
<input type="checkbox"/> 物件をかりる
<input type="checkbox"/> 物件をさがす
<input type="checkbox"/> 物件にすみたい
<input type="checkbox"/> 物件をかりたい
<input type="checkbox"/> 物件を見つける
<input type="checkbox"/> 物件をさがしたい
<input type="checkbox"/> 物件をみつけない

■ チャットボット機能の作成

- 物件紹介ワークスペース
 - [Entities]タブ

The screenshot shows the IBM Watson Assistant interface. On the left, the 'Entities' tab is selected, showing a list of entities. The '@物件' entity is highlighted with a red box. A blue arrow points from this entity to the right-hand screenshot, which shows the configuration page for '@物件'. In this configuration page, the 'Entity values' list is highlighted with a red box, showing various property types like 'お部屋', '部屋', 'マンション', etc.

「物件」という目的を表す
単語の登録

■ チャットボット機能の作成

- 物件紹介ワークスペース
 - [Dialog]タブ

The image shows two overlapping screenshots of the IBM Watson Assistant interface. The left screenshot shows the 'Dialog' tab with a list of nodes. The right screenshot shows a detailed view of a dialog node configuration.

Callout 1 (Green box): ノードを登録して、会話のフローを作成 (Register the node and create the conversation flow)

Callout 2 (Green box): 「もし～を認識したら」と登録 (Register with "If recognized")
物件に関連する単語で、物件検索を認識する場合のノード (Node for recognizing property search related terms)

Callout 3 (Green box): 「～と返答し」を登録 (Register with "Respond with")
メッセージだけでなく、Contextに独自の変数も登録可能 (Not only messages, but also custom variables in Context are registrable)

Callout 4 (Green box): 「～する」を登録 (Register with "Do")
次の入力を待ち受けたり、別のノードに移動したりする (Wait for the next input or move to another node)

Node List (Left Screenshot):

- ようこそ (welcome) - 1 Response / 0 Context set / Does not return
- 挨拶 (#挨拶) - 1 Response / 0 Context set / Does not return
- 物件検索開始、家賃問合せ (#物件検索 and @物件)** - 1 Response / 0 Context set / Does not return
- 家賃確認、都道府県問合せ (@sys-currency) - 1 Response / 1 Context set / Return allowed
- 家賃以外指定 (anything_else)

Node Configuration (Right Screenshot):

- If bot recognizes:** #物件検索 and @物件
- Then respond with:** 1. 物件の検索を行います。家賃はいくら位を希望しますか?
- And finally:** Wait for user input

■ チャットボット機能の作成

- ワークスペースIDの取得
 - プログラムからAPIを使用する場合、ワークスペースを識別する為に Workspace IDを使用する

The screenshot shows the IBM Watson Assistant 'Workspaces' page. On the left, a workspace named '物件紹介' (Property Introduction) is listed. A red box highlights the 'View details' button. A green arrow points from this button to a detailed view of the workspace on the right. In this detailed view, the 'Workspace ID' is highlighted with a red box. A callout box explains that this ID is used to identify the workspace in API calls. Another callout box points to the 'View details' button, explaining that it should be selected from the target workspace.

Workspace ID
ワークスペースを識別するID
APIで呼び出す際、対象となる
Workspace IDを指定する

対象となるワークスペースより
[View details]を選択

■ チャットボット機能の作成

• AssistantサービスのAPI仕様

リクエスト

POST /v1/workspaces/{workspace_id}/message

※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
workspace_id	URL パラメータ	使用するWorkspace ID
version	URL パラメータ	APIのバージョン日付（現在は、2018-02-16を指定）
body	リクエストの本体	送信するメッセージおよび会話の状態（Context）をセット（JSON形式）

レスポンス例

```
{
  "context": { ... },
  "entities": [],
  "intents": [
    { ... },
  ],
  "output": {
    "log_messages": [],
    "text": ["おはようございます！"],
    "nodes_visited": ["node_3_1480025899490"],
    "input": {"text": "おはようございます"}
  }
}
```

パラメータ	内容
output:text	会話の応答
input:text	送信した会話
context	会話の状態

■ チャットボット機能の作成

• 設計画面

- RESTコンポーネント

The image shows a Delphi IDE design view for a chatbot application. The main window is titled "Watsonとの会話" (Conversation with Watson) and contains a text input field (labeled "EditInput: TEdit") and a "会話" (Conversation) button (labeled "ButtonConv: TButton"). Below the input field are three REST components: "HTTPBasicAuthenticator1", "RESTClient1", and "RESTRequest1".

Annotations and component details:

- RESTClient1 (TRESTClient):** Properties include "AllowCookies" (True), "Authenticator" (HTTPBasicAuthenticator1), "AutoCreateParams" (True), "BaseURL" (https://gateway.watsonplatform.net/assistant/api), "BindSource" (RESTClient1.BindSource), "ContentType", and "FallbackCharsetEncoding" (UTF-8). The "BaseURL" property is highlighted with a red box and labeled "資格情報のURL" (URL of credentials).
- HTTPBasicAuthenticator1 (THTTPBasicAuthenticator):** Properties include "Name" (HTTPBasicAuthentic...), "Password" (masked with dots), "Tag" (0), and "Username" (d3565e1c-a290-414). The "Password" and "Username" properties are highlighted with red boxes and labeled "資格情報のユーザーとパスワード" (User and password of credentials).
- RESTRequest1 (TRESTRequest):** Properties include "LiveBinding" (LiveBindingデザイナ), "Method" (rmPOST), "Name" (RESTRequest1), "Params" (TRESTRequestParameterList), "Resource" (v1/workspaces/01503987-eb00-491d-a5b6-7feca88f1bb0/message?version=2018-02-16), and "ResourceSuffix". The "Method" and "Resource" properties are highlighted with red boxes and labeled "HTTPメソッド (POST)" (HTTP Method (POST)).
- RESTResponse1 (TRESTResponse):** This component is also present in the design view.
- MemoConv: TMemo:** A memo component is also present in the design view.
- EditPrice: TEdit, EditState: TEdit, EditLayout: TEdit:** These are text edit components at the bottom of the design view.

Additional annotations include "資格情報のユーザーとパスワード" (User and password of credentials) pointing to the authenticator's password field, and "RESTリソース (パラメータ) Workspace IDを指定 version : APIバージョン日付をセット" (REST Resource (Parameter) Specify Workspace ID, set version : API version date) pointing to the resource property of the RESTRequest1 component.

■ チャットボット機能の作成

• ソース(1/2)

```
procedure TForm1.ButtonConvClick(Sender: TObject);
var
  JSONValue: TJSONValue;
  JSONPair: TJSONPair;
  sMsg: String;
  sRet: String;
begin
  MemoConv.Lines.Add('【あなた】 : ' + EditInput.Text);

  RESTRequest1.Params.Clear;
  //リクエストパラメータのセット
  if FContext = '' then //----- 会話が初回の場合
    sMsg := '{"input": {"text": "' + EditInput.Text + '"}}';
  else //----- 会話が2回目以降の場合
    sMsg := '{"input": {"text": "' + EditInput.Text + '"}, '
      + FContext + '>'; //会話の状態(FContext)をセット

  RESTRequest1.Params.AddItem('body', sMsg, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //処理結果JSONのパーズ
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);
```

会話情報のJSONテキスト作成

Private変数 FContext
: コンテキスト (会話の状態)

会話を継続する為に、2回目以降
前回のコンテキスト値を再セット

AddItem パラメータをセット

会話のメッセージおよび状態をセット
コンテンツタイプには下記を指定
ctAPPLICATION_JSON

■ チャットボット機能の作成

• ソース(2/2)

//結果 (output->text)の取得

```
sRet := JSONValue.GetValue<String>('output.text[0]');  
MemoConv.Lines.Add('【Watson】 : ' + sRet);
```

会話の応答を取得

//会話の状態 (context) を取得

```
JSONPair := TJSONObject(JSONValue).Get('context');  
FContext := JSONPair.ToJSON;
```

会話の状態を取得

※contextの中にある
"conversation_id"が会話を
特定するID

//選択値の取得

```
try  
  EditPrice.Text := JSONValue.GetValue<String>('context.price');  
except  
  EditPrice.Text := '';  
end;
```

```
try  
  EditState.Text := JSONValue.GetValue<String>('context.state');  
except  
  EditState.Text := '';  
end;
```

ワークスペース内で
セットされた変数を
取得

```
try  
  EditLayout.Text := JSONValue.GetValue<String>('context.layout');  
except  
  EditLayout.Text := '';  
end;
```

```
end;
```

■ チャットボット機能の作成

● 実行結果

1. 文章を入力

2. ボタンクリック

3. 入力内容と応答内容が出力

4. 会話が進むにつれ、contextに格納された変数の値が順番に表示
↓
最終的にこの条件を使ってDBを検索すればよい。

Watsonとの会話

はい

会話

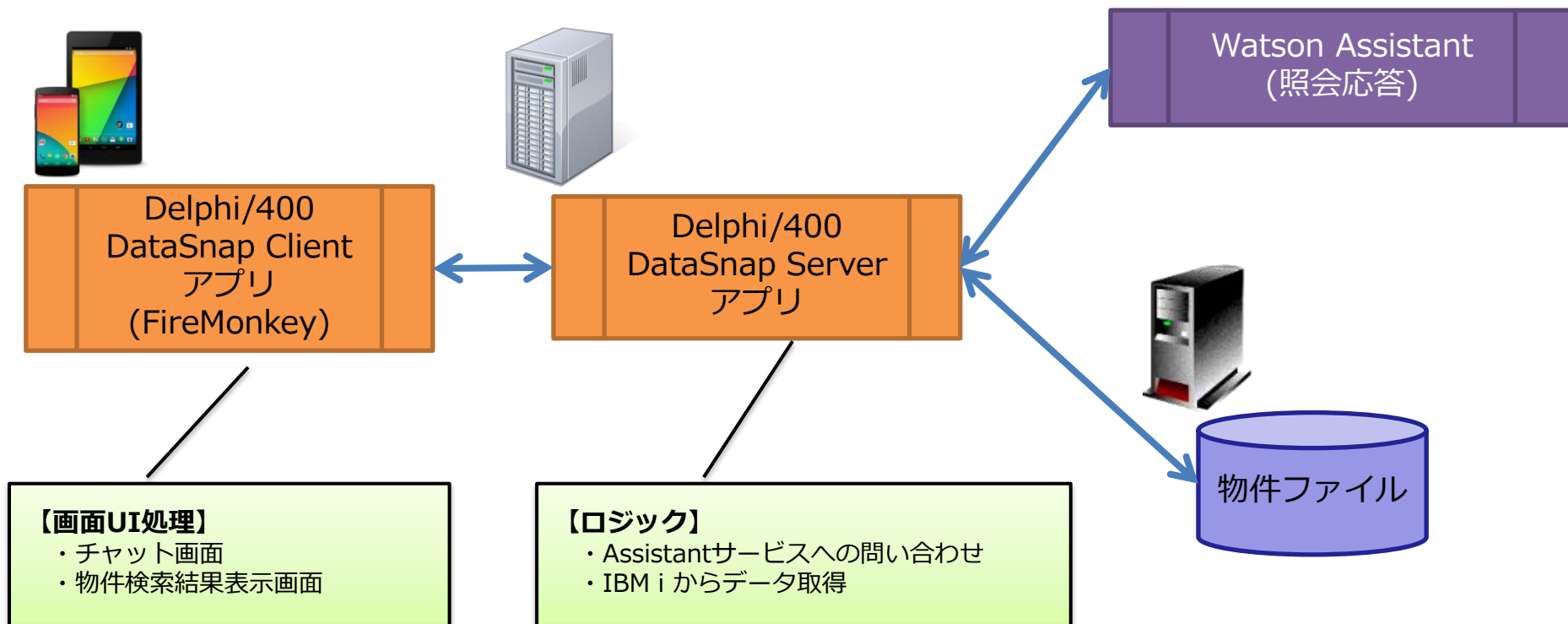
【あなた】:こんにちは
【Watson】:こんにちは！このボットは物件の検索等ができます！何をしますか？
【あなた】:部屋を探して
【Watson】:物件の検索を行います。家賃はいくら位を希望しますか？
【あなた】:6万円くらいで
【Watson】:家賃は60000円位ですね。ご希望の都道府県はどこですか？(埼玉県、千葉県、東京都、神奈川県、愛知県、京都府、大阪府、兵庫県のいずれか)
【あなた】:大阪でお願い
【Watson】:ご希望は、大阪府ですね。次に間取りはどれくらいがよいですか？
【あなた】:1Kで
【Watson】:1Kですね。条件がそろったので物件検索してもよいですか？
【あなた】:はい
【Watson】:検索結果をご覧ください。良い結果はありましたか？

家賃 都道府県 間取り

60000 大阪府 1K

■ 物件紹介チャットアプリ

- モバイルから実行できるチャットボットアプリの作成
 - DataSnapサーバーを使用した3層アプリ

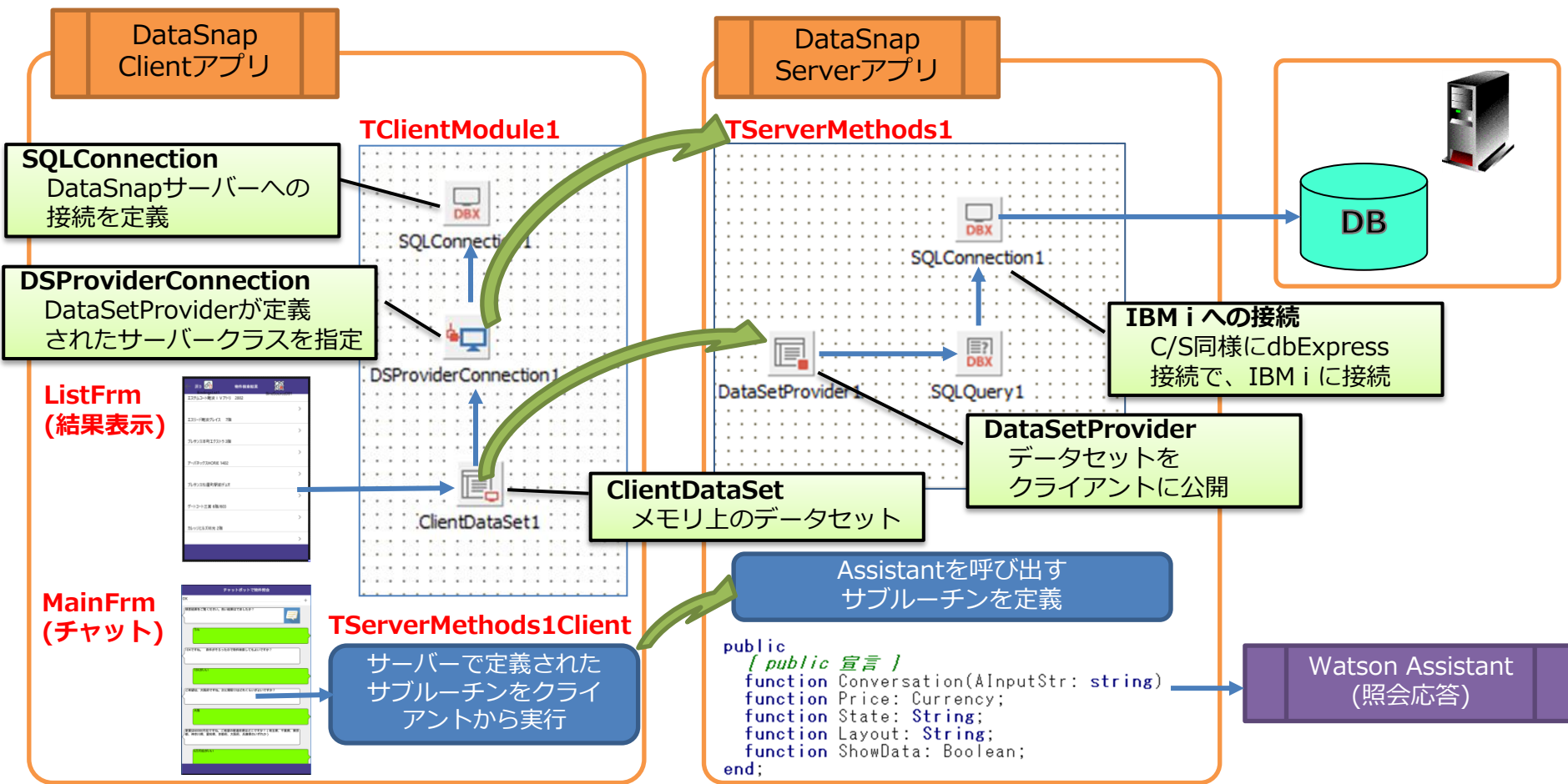


■ 物件紹介チャットアプリ

• DataSnapアプリ

- dbExpress接続を使用したDataSnapアプリの構成

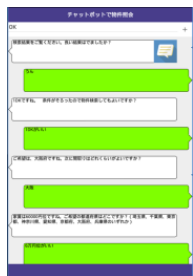
DataSnapアプリ開発の詳細については、下記を参照
 第14回テクニカルセミナー
 『Delphi/400 XE5
 -こんなに簡単！ IBM i スマートデバイスネイティブ開発-』



■ 物件紹介チャットアプリ

• 処理概要

MainFrm (チャット画面)



チャット形式の会話をを行う

条件が決まるごとに、contextの各変数の値を渡す

show_data="1"を受け取ったら、
吹き出し内に画像を表示



画像クリックにより
ListFrmに遷移

ListFrm (検索結果表示画面)



各変数を使用して
SQL文を作成して実行



補足資料に、全ソースコードを掲載



Watson Assistant
(照会応答)

物件の条件を会話で確認

家賃 (context: price)

都道府県 (context: state)

間取り (context: layout)

条件がよいかの
確認

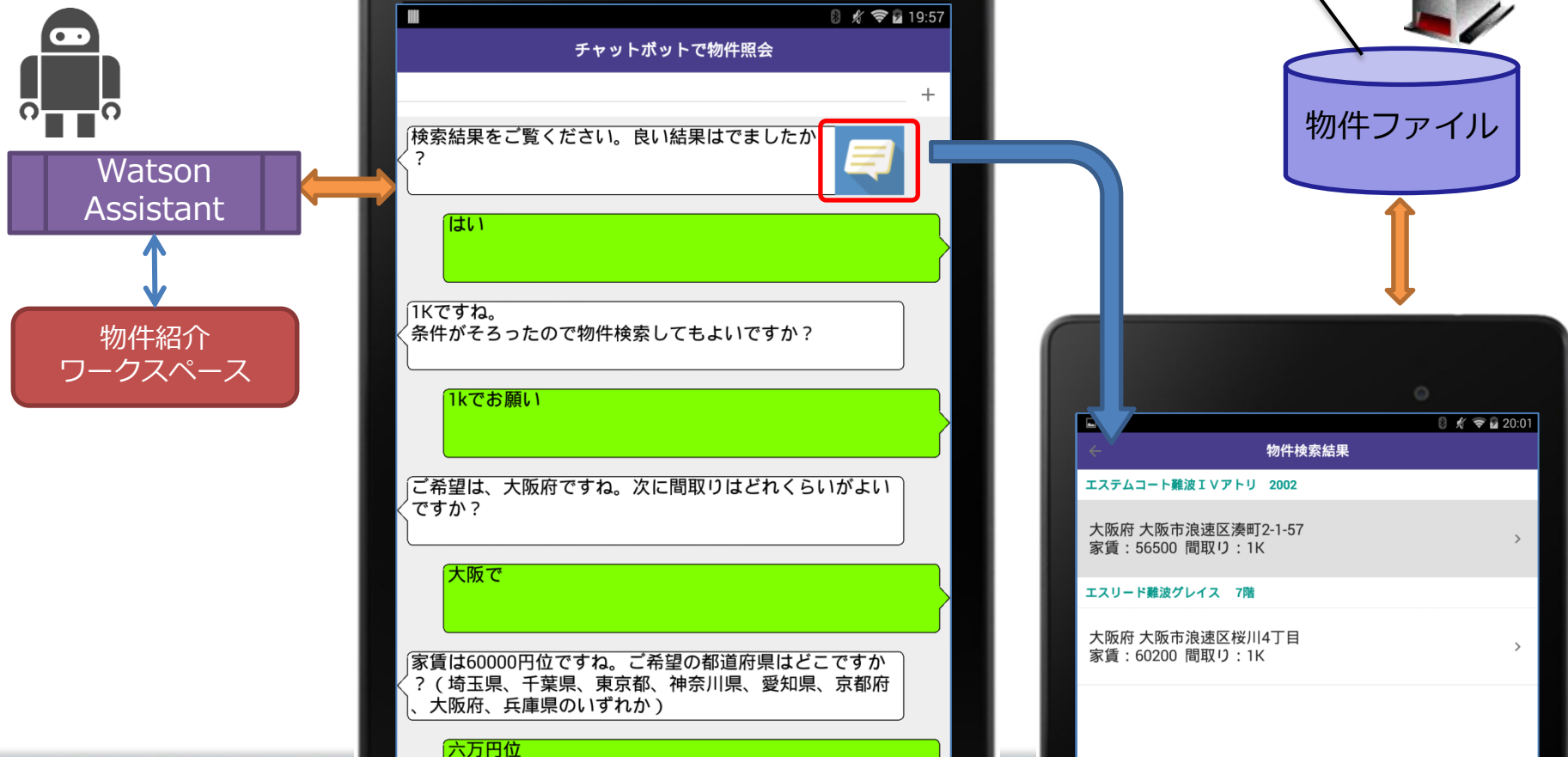
N

Y

条件終了フラグ
(context: show_data="1")

■ 物件紹介チャットアプリ

• 実行 (デモ)



5. まとめ

■ まとめ

• Watson API

- WatsonはIBM Cloudの一機能として使用可能
- ライト・アカウントであれば無償で使用可能
- Watson API は、REST/JSONサービスとして使用
- APIリファレンスに、呼出方法が記載されている

• Delphi/400からのWatson API呼び出し

- RESTコンポーネントにより呼び出しが可能
- JSONをパース（解析）するユニットも使用可能

**Delphi/400アプリに色々なWatson APIと連携させることで、
より便利なアプリ構築が可能に！**

ご清聴ありがとうございました。

- 補足資料： 自動翻訳機アプリ ソースコード

■ 補足：自動翻訳機アプリ ソースコード

フォーム 宣言部

```
private
  [ Private 宣言 ]
  FWavFileName: String;
  Fmp3FileName: String;
  procedure RecStart;
  procedure RecEnd(AWavFileName: String);
  function SpeechToText(AWavFileName: String): String;
  function Transrate(AInputText: String): String;
  procedure TextToSpeech(AText, Amp3FileName: String);
  procedure PlaySpeech(Amp3FileName: String);
public
  [ Public 宣言 ]
end;
```

録音ファイル名を保持

再生ファイル名を保持

フォーム 実装部 (1)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  MemoSource.Lines.Clear;
  MemoTarget.Lines.Clear;
end;

procedure TForm1.MediaPlayer2Notify(Sender: TObject);
begin
  MediaPlayer2.Close;

  //音声ファイルを削除
  if FileExists(Fmp3FileName) then
    DeleteFile(Fmp3FileName);
end;
```

再生終了時イベント
音声ファイルを削除

音声再生処理

```
procedure TForm1.PlaySpeech(Amp3FileName: String);
begin
  //MP3ファイルをメディアプレーヤーで再生
  MediaPlayer2.DeviceType := dtAutoSelect;
  MediaPlayer2.FileName := Amp3FileName;
  MediaPlayer2.Open;
  MediaPlayer2.Notify := True;
  MediaPlayer2.Play;
end;

procedure TForm1.RecEnd(AWavFileName: String);
begin
  //音声wavファイルに保存
  MediaPlayer1.FileName := AWavFileName;
  MediaPlayer1.Stop;
  MediaPlayer1.Save;
  MediaPlayer1.Close;
end;

procedure TForm1.RecStart;
begin
  //録音処理の開始
  MediaPlayer1.DeviceType := dtWaveAudio;
  MediaPlayer1.FileName := #0;
  MediaPlayer1.Open;
  MediaPlayer1.StartRecording;
end;
```

録音終了処理
録音ファイルに保存

録音開始処理

■ 補足：自動翻訳機アプリ ソースコード

フォーム 実装部 (2)

マウス押した時のイベント
録音開始

音声ファイルを文字列に
変換する処理

マウス離れた時のイベント
録音終了から翻訳、音声再生
まで実施 (メイン処理)

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button:
  TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Shape1.Brush.Color := clRed;
  Application.ProcessMessages;
  //録音開始
  RecStart;
end;

procedure TForm1.Shape1MouseUp(Sender: TObject; Button:
  TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  sSourceText: String;
  sTargetText: String;
begin
  Shape1.Brush.Color := clWindow;
  Application.ProcessMessages;

  //ファイル名の取得
  FWavFileName := ChangeFileExt(ParamStr(0), '.wav');
  FMp3FileName := ChangeFileExt(ParamStr(0), '.mp3');

  //録音終了
  RecEnd(FWavFileName);

  //テキスト変換
  sSourceText := SpeechToText(FWavFileName);
  MemoSource.Lines.Add(sSourceText);

  //翻訳処理
  sTargetText := Transrate(sSourceText);
  MemoTarget.Lines.Add(sTargetText);

  //音声変換
  TextToSpeech(sTargetText, FMp3FileName);

  //音声再生
  PlaySpeech(FMp3FileName);
end;
```

```
function TForm1.SpeechToText(AWavFileName: String): String;
var
  memStream: TMemoryStream;
  JSONValue: TJSONValue;
begin
  memStream := TMemoryStream.Create;
  try
    memStream.LoadFromFile(AWavFileName);

    with RESTRequest_SpToTx do
      begin
        //変換パラメータの指定
        Params.AddItem('Content-Type', 'audio/wav',
          , pkHTTPHeader, [poDoNotEncode]);
        Params.AddItem('audio', memStream, pkRequestBody, [], ctAUDIO_BASIC);
        //リクエスト実行
        Execute;
      end;

      //処理結果JSONのパーズ
      JSONValue := TJSONObject.ParseJSONValue(RESTResponse_SpToTx.Content);
      try
        Result := JSONValue.GetValue<String>
          ('results[0].alternatives[0].transcript');
      except
        //エラー
        raise Exception.Create('文字取得できる音声データではありません');
      end;
    finally
      memStream.Free;
    end;

    //変換終了した後ファイルを削除
    if FileExists(AWavFileName) then
      DeleteFile(AWavFileName);
  end;
```


■ 補足：自動翻訳機アプリ ソースコード

フォーム 実装部 (3)

テキスト文字列を音声ファイルに変換する処理

```
procedure TForm1.TextToSpeech(AText, AMp3FileName: String);
var
  memStream: TMemoryStream;
  sFileName: string;
begin
  //エラーチェック
  if Trim(AText) = '' then
    raise Exception.Create('音声変換するテキストがありません');

  with RESTRequest_TxToSp do
  begin
    //変換パラメータの指定
    Params.AddItem('text', AText, pkGETorPOST);
    //リクエスト実行
    Execute;
  end;

  //レスポンスのMP3データをファイルに保管する
  memStream := TMemoryStream.Create;
  try
    memStream.WriteData(RESTResponse_TxToSp.RawBytes
      , Length(RESTResponse_TxToSp.RawBytes));

    memStream.SaveToFile(AMp3FileName);
  finally
    memStream.Free;
  end;
end;
```

入力文字列を翻訳する処理

```
function TForm1.Transrate(AInputText: String): String;
var
  JSONValue: TJSONValue;
  sRet: String;
begin
  //エラーチェック
  if Trim(AInputText) = '' then
    raise Exception.Create('翻訳元テキストがありません');

  with RESTRequest_Trans do
  begin
    //翻訳パラメータの指定
    Params.AddItem('text', AInputText, pkGETorPOST); //翻訳元テキスト
    //リクエスト実行
    Execute;
  end;

  //翻訳テキストの取得
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse_Trans.Content);
  Result := JSONValue.GetValue<string>('translations[0].translation');
end;
```

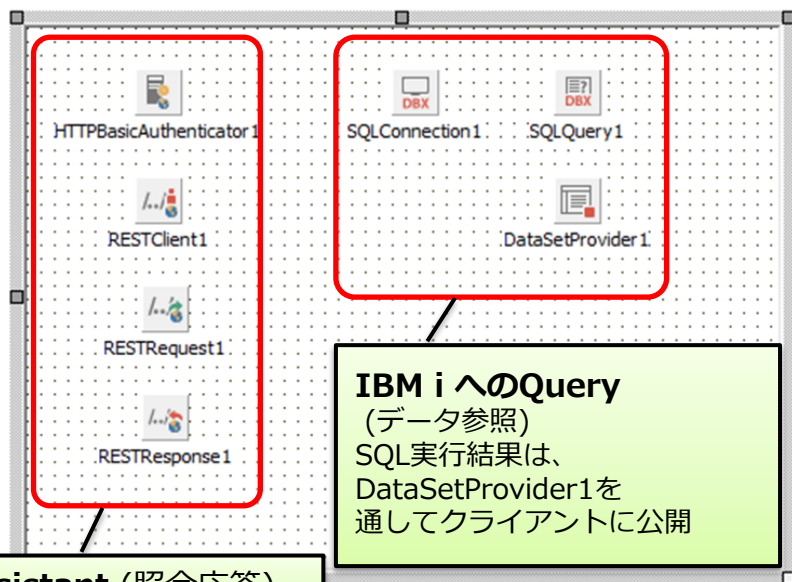
- 補足資料：物件紹介チャットアプリ ソースコード

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Server アプリ

- AssistantサービスとIBM iへの問い合わせ

TServerMethods1 【宣言部】



```
type
TServerMethods1 = class(TDSServerModule)
  RESTResponse1: TRESTResponse;
  RESTRequest1: TRESTRequest;
  RESTClient1: TRESTClient;
  HTTPBasicAuthenticator1: THTTPBasicAuthenticator;
  SQLConnection1: TSQLConnection;
  SQLQuery1: TSQLQuery;
  SQLQuery1BUBKCD: TStringField;
  SQLQuery1BUBKNM: TStringField;
  SQLQuery1BUSTAT: TStringField;
  SQLQuery1BUADRS: TStringField;
  SQLQuery1BUPRIC: TIntegerField;
  SQLQuery1BULAYT: TStringField;
  DataSetProvider1: TDataSetProvider;
  procedure DSServerModuleCreate(Sender: TObject);
private
  [ private 宣言 ]
  FContext: String;
  FPrice: Currency;
  FState: String;
  FLayout: String;
  FShowData: Boolean;
public
  [ public 宣言 ]
  function Conversation(AInputStr: string): string;
  function Price: Currency;
  function State: String;
  function Layout: String;
  function ShowData: Boolean;
end;
```

クライアントに公開するサブルーチン

Conversation：入力文字を入れると応答文字を返却
Price：家賃の現在値を取得
State：都道府県の現在値を取得
Layout：間取りの現在値を取得
ShowData：検索の条件がそろった場合、True

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Server アプリ

フォーム 実装部 (1)

```
function TServerMethods1.Conversation(AInputStr: string): string;
var
  JSONValue: TJSONValue;
  JSONPair: TJSONPair;
  sMsg: String;
begin
  RESTRequest1.Params.Clear;
  RESTRequest1.Params.AddItem;

  RESTRequest1.Params.Clear;
  //リクエストパラメータのセット
  if FContext = '' then //----- 会話が初回の場合
    sMsg := '['input': [{"text": "" + AInputStr + ""}]]'
  else //----- 会話が2回目以降の場合
    sMsg := '['input': [{"text": "" + AInputStr + ""}], ['context': '" + FContext + "']]; //会話の状態 (FContext) をセット

  RESTRequest1.Params.AddItem('body', sMsg, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //処理結果JSONのパーズ
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);

  //結果 (output->text) の取得
  Result := JSONValue.GetValue<String>('output.text[0]');

  //会話の状態 (context) を取得
  JSONPair := TJSONObject(JSONValue).Get('context');
  FContext := JSONPair.ToJSON;

  //会話の状態より、変数値を取得
  try
    FPrice := JSONValue.GetValue<Currency>('context.price'); // 家賃
  except
    FPrice := 0; //キーが存在しない場合
  end;

  try
    FState := JSONValue.GetValue<String>('context.state'); // 都道府県
  except
```

入力文字列をもとに、会話の応答を返却する処理

```
    FState := ''; //キーが存在しない場合
  end;

  try
    FLayout := JSONValue.GetValue<String>('context.layout'); // 間取り
  except
    FLayout := ''; //キーが存在しない場合
  end;

  try
    FShowData := JSONValue.GetValue<String>('context.show_data') <> ''; //検索結果表示
  except
    FShowData := False; //キーが存在しない場合
  end;
end;

procedure TServerMethods1.DSServerModuleCreate(Sender: TObject);
begin
  //変数初期化
  FContext := '';
  FPrice := 0;
  FState := '';
  FLayout := '';
  FShowData := False;

  //IBMi接続の初期設定
  with SQLConnection1 do
  begin
    Params.Values['RoleName'] := 'D4TEC22LIB';
    Params.Values['HostName'] := 'MIGARO15';
    Params.Values['User_Name'] := 'D400';
    Params.Values['Password'] := 'D400';
    Params.Values['Database'] := 'MIGARO15';
  end;

  //IBMiへ接続
  SQLConnection1.Connected := True;
end;
```

OnCreateイベント
初期処理 変数初期化、DB接続

IBMi の接続環境にあわせて要修正

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Server アプリ

フォーム 実装部 (2)

```
function TServerMethods1.Layout: String;  
begin  
    Result := FLayout; // 間取り  
end;
```

間取りの現在値を返却する処理

```
function TServerMethods1.Price: Currency;  
begin  
    Result := FPrice; // 家賃  
end;
```

家賃の現在値を返却する処理

```
function TServerMethods1.ShowData: Boolean;  
begin  
    Result := FShowData; // 検索結果表示  
end;
```

検索結果を表示するフラグの現在値を返す処理

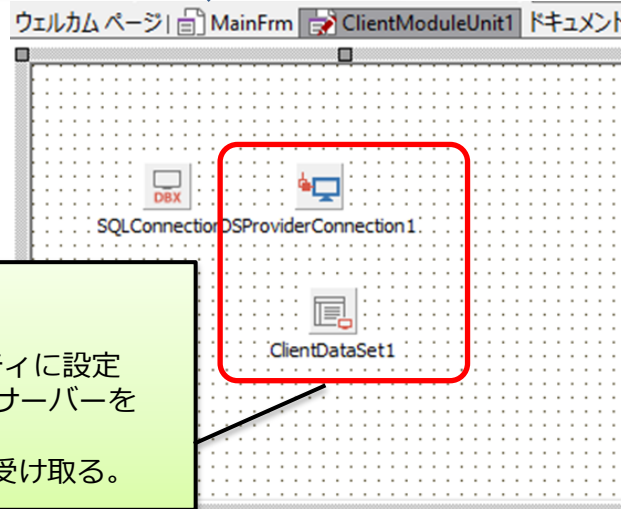
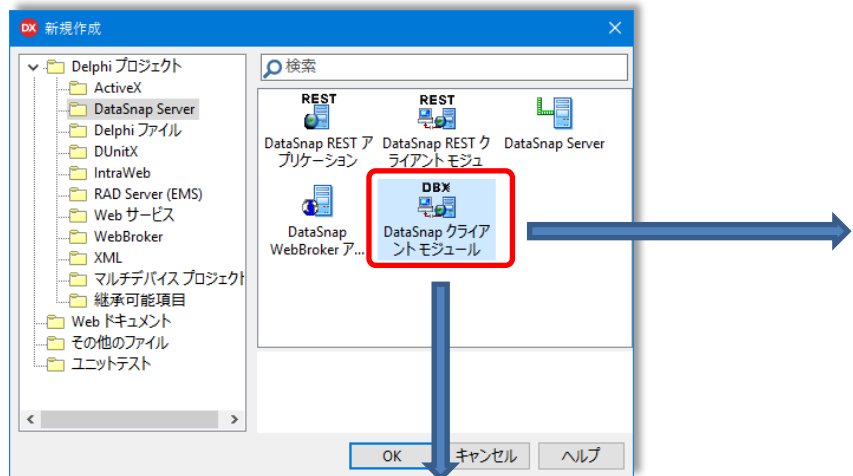
```
function TServerMethods1.State: String;  
begin  
    Result := FState; // 都道府県  
end;
```

都道府県の現在値を返却する処理

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

- DataSnapクライアントモジュールの作成方法



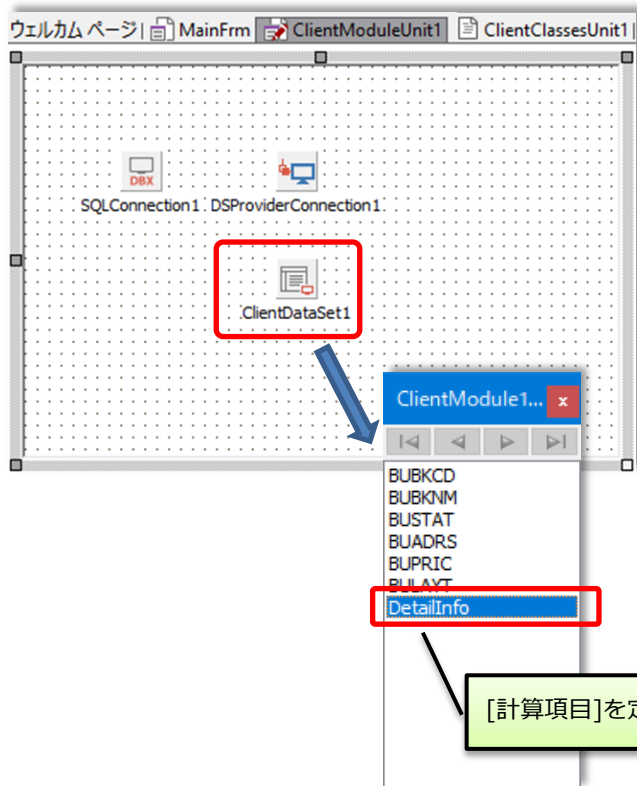
TServerMethods1Client
DataSnapサーバーに用意されたメソッドが自動的に定義。
TServerMethods1Clientクラスのオブジェクトを通して
アクセス可能。

```
// DataSnap プロキシ ジェネレータにより作成。  
// 2018/05/12 20:53:35  
  
unit ClientClassesUnit1;  
  
interface  
  
uses System.JSON, Data.DBXCommon, Data.DBXClient, Data.DBXClient;  
  
type  
  TServerMethods1Client = class(TDSAdminClient)  
  private  
    FDSModuleCreateCommand: TDBXCommand;  
    FConversationCommand: TDBXCommand;  
    FPriceCommand: TDBXCommand;  
    FStateCommand: TDBXCommand;  
    FLayoutCommand: TDBXCommand;  
    FShowDataCommand: TDBXCommand;  
  public  
    constructor Create(ADBXConnection: TDBXConnection);  
    constructor Create(ADBXConnection: TDBXConnection);  
    destructor Destroy; override;  
    procedure DSServerModuleCreate(Sender: TObject);  
    function Conversation(AInputStr: string): string;  
    function Price: Currency;  
    function State: string;  
    function Layout: string;  
    function ShowData: Boolean;  
  end;
```

IBM i へのQuery発行
CommandTextプロパティに設定したSQL文をDataSnapサーバーを通して実行。
結果をClientDataSetが受け取る。

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
 - ClientModuleUnit1



実装部

```
procedure TClientModule1.ClientDataSet1CalcFields(DataSet: TDataSet);
begin
  ClientDataSet1DetailInfo.AsString :=
    ' ' + ClientDataSet1BUSTAT.AsString
  + ' ' + ClientDataSet1BUADRS.AsString
  + #13#10
  + '家賃：' + ClientDataSet1BUPRIC.AsString
  + '間取り：' + ClientDataSet1BULAYT.AsString;
end;
```

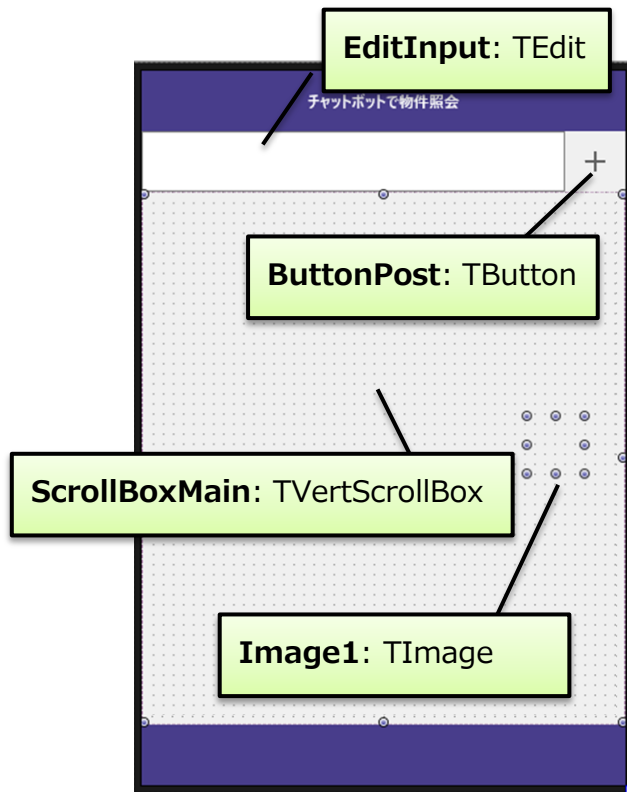
OnCalcFieldsイベント
[計算項目]に値をセット

[計算項目]を定義

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

• MainForm



宣言部

```
type
  THeaderFooterForm = class(TForm)
    Header: TToolBar;
    Footer: TToolBar;
    HeaderLabel: TLabel;
    ToolbarEntry: TToolBar;
    ScrollBoxMain: TVertScrollBar;
    ButtonPost: TButton;
    EditInput: TEdit;
    Rectangle1: TRectangle;
    Rectangle2: TRectangle;
    Image1: TImage;
  procedure ButtonPostClick(Sender: TObject);
  procedure Image1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  private
    [ private 宣言 ]
    FPrice: Currency; // 家賃
    FState: String; // 都道府県
    FLayout: String; // 間取り
    FShowData: Boolean; // 検索画面表示フラグ
  procedure SetRightMsg(sText: String);
  procedure SetLeftMsg(sText: String);
  public
    [ public 宣言 ]
  end;

  //情報保持できるイメージコンポーネント
  TValueImage = class(TImage)
  private
    FPrice: Currency; // 家賃
    FState: String; // 都道府県
    FLayout: String; // 間取り
  public
    constructor Create(AOwner: TComponent; APrice: Currency;
      AState, ALayout: String); // 生成時情報をセット
    property Price: Currency read FPrice; // プロパティ: 家賃
    property State: String read FState; // プロパティ: 都道府県
    property Layout: String read FLayout; // プロパティ: 間取り
  end;
```

TImageクラスに
家賃、都道府県、間取りの情報を
追加

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

• MainFrm

ボタン押下時処理
(メイン処理)

```
procedure THeaderFooterForm.ButtonPostClick(Sender: TObject);
var
  sRequestMsg: String;
  sResponseMsg: String;
begin
  //入力空白の場合、何もしない
  if EditInput.Text = EmptyStr then
    Exit;

  //リクエストメッセージをセット
  sRequestMsg := EditInput.Text;
  EditInput.Text := '';

  //発言メッセージをセット
  SetRightMsg(sRequestMsg);
  Application.ProcessMessages;

  //Watsonに問い合わせを行い、結果を受け取る
  with ClientModule1.ServerMethods1Client do
  begin
    sResponseMsg := Conversation(sRequestMsg);
    //検索情報の取得
    FPrice := Price;
    FState := State;
    FLayout := Layout;
    FShowData := ShowData;
  end;
  //応答メッセージをセット
  SetLeftMsg(sResponseMsg);

  EditInput.SetFocus;
end;
```

画像クリック時処理
検索結果画面へ遷移

```
procedure THeaderFooterForm.Image1Click(Sender: TObject);
begin
  //検索結果画面を表示
  frmList := TfrmList.Create(Self);
  frmList.Price := TValueImage(Sender).FPrice;
  frmList.State := TValueImage(Sender).FState;
  frmList.Layout := TValueImage(Sender).FLayout;
  frmList.Show;
end;
```

```
procedure THeaderFooterForm.SetLeftMsg(sText: String);
var
  crLeftMsg: TCalloutRectangle;
  txtLeftMsgText: TText;
  TmpImg: TValueImage;
begin
  //左側の吹き出しセット
  crLeftMsg := TCalloutRectangle.Create(Self);
  with crLeftMsg do
  begin
    Parent := ScrollBoxMain;
    Align := TAlignLayout.Top;
    CalloutPosition := TCalloutPosition.Left;
    CornerType := TCornerType.Round;
    XRadius := 5; YRadius := 5;
    Margins.Top := 10;
    Margins.Bottom := 10;
    Margins.Right := 50;
    Height := 75;
    Fill.Color := claWhite;
  end;

  //メッセージセット
  txtLeftMsgText := TText.Create(Self);
  with txtLeftMsgText do
  begin
    Parent := crLeftMsg;
    Align := TAlignLayout.Client;
    TextSettings.HorzAlign := TTextAlign.Leading;
    TextSettings.VertAlign := TTextAlign.Leading;
    Text := sText;
    Margins.Left := 15; Margins.Right := 5;
    Width := crLeftMsg.Width - 20;
    WordWrap := True;
    AutoSize := True;
  end;

  //検索結果ありの場合、アイコンを表示する
  if FShowData then
  begin
    TmpImg := TValueImage.Create(Self, FPrice, FState, FLayout);
    TmpImg.Parent := crLeftMsg;
    TmpImg.Align := TAlignLayout.FitRight;
    TmpImg.Bitmap.Assign(Image1.Bitmap);
    TmpImg.Width := 75;
    TmpImg.OnClick := Image1Click;
  end;
end;
```

左側 (ワトソン側)
吹き出し表示処理

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

• MainForm

```
procedure TForm1.SetRightMsg(sText: String);
var
  crRightMsg: TCalloutRectangle;
  txtRightMsgText: TText;
begin
  //右側の吹き出しセット
  crRightMsg := TCalloutRectangle.Create(Self);
  with crRightMsg do
  begin
    Parent := ScrollBoxMain;
    Align := TAlignLayout.Top;
    CalloutPosition := TCalloutPosition.Right;
    CornerType := TCornerType.Round;
    XRadius := 5; YRadius := 5;
    Margins.Top := 10;
    Margins.Bottom := 10;
    Margins.Left := 50;
    Height := 75;
    Fill.Color := claChartreuse;
  end;
  //メッセージセット
  txtRightMsgText := TText.Create(Self);
  with txtRightMsgText do
  begin
    Parent := crRightMsg;
    Align := TAlignLayout.Client;
    TextSettings.HorzAlign := TTextAlign.Leading;
    TextSettings.VertAlign := TTextAlign.Leading;
    Text := sText;
    Margins.Right := 15; Margins.Left := 5;
    Width := crRightMsg.Width - 20;
    WordWrap := True;
    AutoSize := True;
  end;
end;
```

右側（人側）
吹き出し表示処理

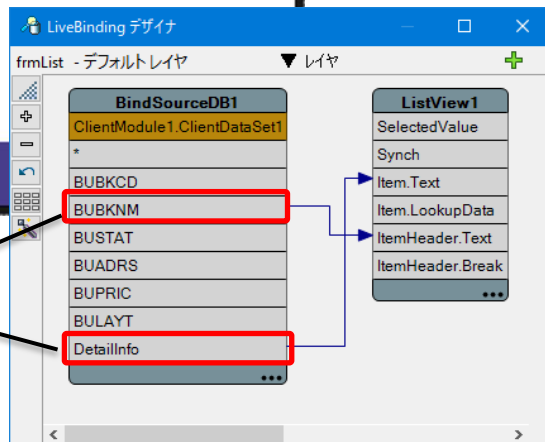
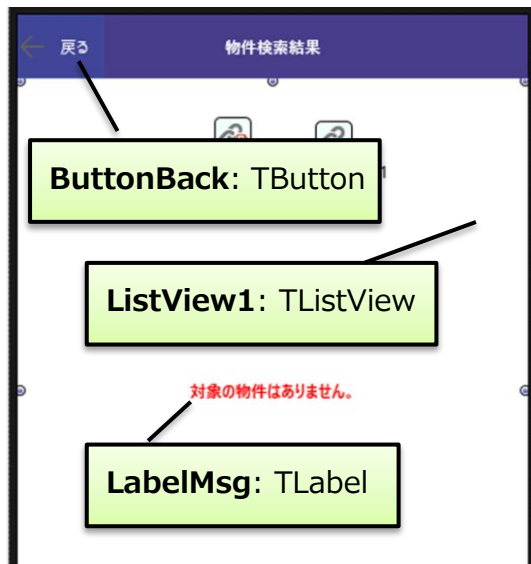
拡張Imageクラスの
生成処理

```
[ TValueImage ]
constructor TValueImage.Create(AOwner: TComponent; APrice:
  Currency; AState, ALayout: String);
begin
  inherited Create(AOwner);
  //データの抽出条件を保持
  FPrice := APrice;
  FState := AState;
  FLayout := ALayout;
end;
```

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

- ListFrm



宣言部

```
type
  TfrmList = class(TForm)
    Footer: TToolBar;
    Rectangle1: TRectangle;
    Header: TToolBar;
    Rectangle2: TRectangle;
    HeaderLabel: TLabel;
    ButtonBack: TButton;
    ListView1: TListView;
    BindSourceDB1: TBindSourceDB;
    BindingsList1: TBindingsList;
    LinkFillControlToField1: TLinkFillControlToField;
    LabelMsg: TLabel;
  procedure ButtonBackClick(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseA);
  procedure FormCreate(Sender: TObject);
  private
    { private 宣言 }
    FLayout: String;
    FPrice: Currency;
    FState: String;
    function SetBukkenData(APrice: Currency; AState,
      ALayout: String): Boolean;
  public
    { public 宣言 }
    property Price: Currency read FPrice write FPrice;
    property State: String read FState write FState;
    property Layout: String read FLayout write FLayout;
  end;
```

■ 補足：物件紹介チャットアプリソースコード

• DataSnap Client アプリ

• ListFrm

戻るボタン押下時処理

```
procedure TfrmList.ButtonBackClick(Sender: TObject);  
begin  
    //画面を終了する  
    Close;  
end;
```

フォーム終了処理

```
procedure TfrmList.FormClose(Sender: TObject; var Action:  
begin  
    //フォームメモリを解放  
    Action := TCloseAction.caFree;  
end;
```

フォーム表示処理
物件データを取得

```
procedure TfrmList.FormShow(Sender: TObject);  
begin  
    LabelMsg.Visible := False;  
  
    //データの取得（データない場合、メッセージ表示）  
    if not SetBukkenData(FPrice, FState, FLayout) then  
        LabelMsg.Visible := True;  
end;
```

物件データ取得処理
SQLを指定して
ClientDataSet取得

```
function TfrmList.SetBukkenData(APrice: Currency;  
    AState, ALayout: String): Boolean;  
var  
    sSQL: String;  
begin  
    //初期処理  
    Result := False;  
  
    //SQL文の作成  
    sSQL := 'SELECT * FROM BUKENP'  
        + ' WHERE BUSTAT = ' + AState + ''  
        + ' AND BUPRIC >= ' + CurrToStr(APrice - 20000)  
        + ' AND BUPRIC <= ' + CurrToStr(APrice + 5000)  
        + ' AND BULAYT = ' + ALayout + ''  
        + ' ORDER BY BUBKCD ';  
  
    //SQLを実行して、結果を取得  
    with ClientModule1.ClientDataSet1 do  
    begin  
        Active := False;  
        CommandText := sSQL;  
        Active := True;  
  
        if Eof and Bof then  
        begin  
            //対象データが無い場合、データセットを閉じる  
            Active := False;  
            Exit;  
        end;  
  
        //データが存在する場合 True  
        Result := True;  
    end;  
end;
```