Hands-on Guide

Calling a Bluemix Watson Service from SAP using ABAP

Author: Joachim Rese
IBM Germany Research and Development
rese@de.ibm.com

09-October-2017

Introduction

IBM Bluemix is IBM's cloud platform as a service (PaaS). In addition to providing the infrastructure to deploy and host your cloud applications, it offers a large amount of services for data analytics, big data, machine learning, cognitive (Watson), internet-of-things and others. Most services have a REST interface and therefore can be leveraged by external programs.

This is a step-by-step guide on how to call an IBM Watson service in Bluemix from an SAP system using ABAP code. Although only the Watson Text-to-Speech service is considered here, you can use this documentation as a reference for implementing calls from ABAP to any service in IBM Bluemix that supports REST requests.

The examples in this documentation have been implemented with SAP NetWeaver 7.5. However, the used ABAP classes exist with NetWeaver 7.3x onwards. Thus, the code can easily be adopted to suit for older SAP releases.

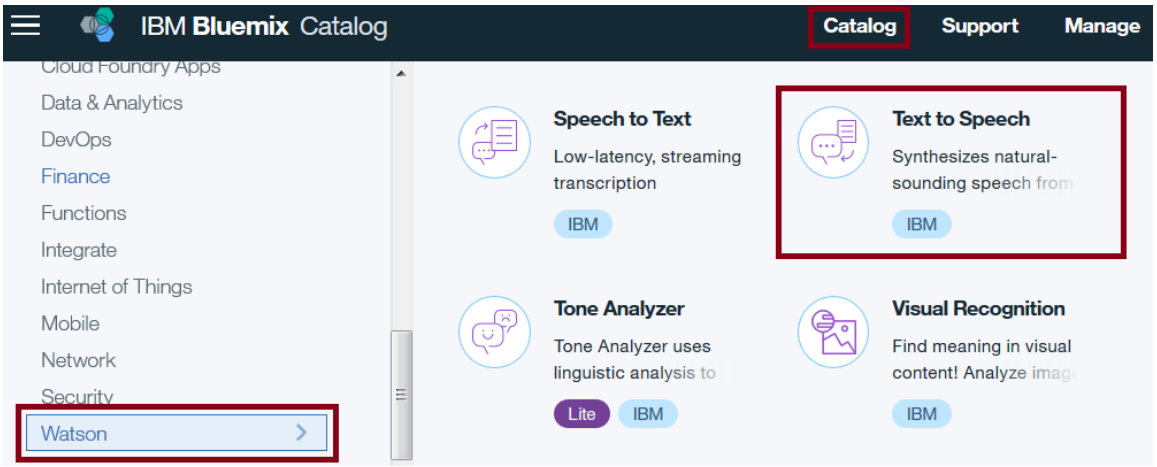
Preparations

Before start coding in ABAP, you must execute the following preparation steps.

Create Watson Text to Speech Service Instance

To implement and run the example code of this documentation, you need to create your own Watson Text-to-Speech service instance. To do so, execute the following steps.

1. Sign in to IBM Bluemix: **https://console.bluemix.net/**
   If you do not have an account in Bluemix, you can create a 30-day trial account for free.
2. In the Bluemix dashboard, click ***Catalog***. In navigation pane on left-hand side click ***Watson***. Then click the ***Text to Speech*** service.



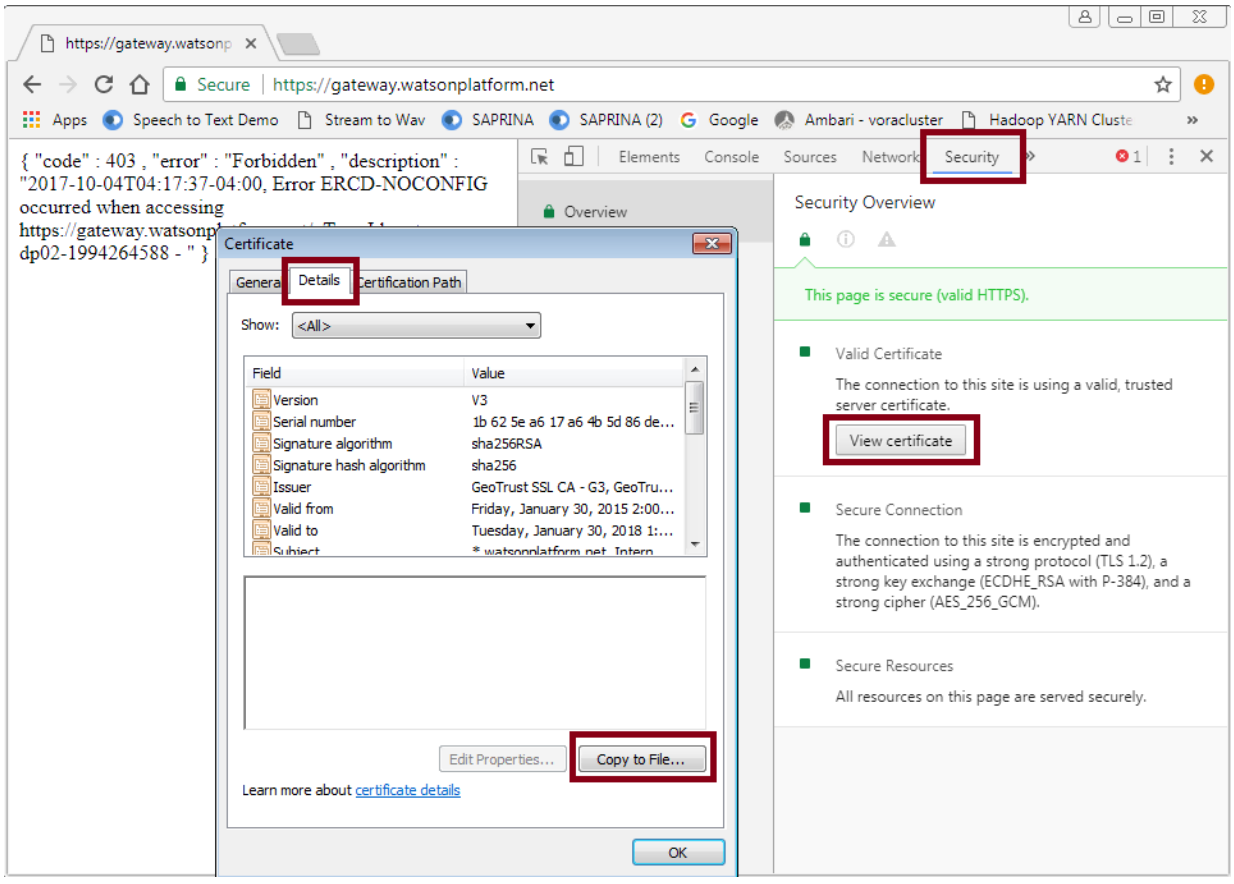
3. Click ***Create***.
4. Click ***Service credentials*** and open ***View credentials***. Url, username and password for your Text-to-Speech service instance are shown. This information needed to call the service from ABAP.

Use the credentials of your individual Watson Text-to-Speech service instance when implementing the example in this documentation. Certainly, your credentials differ from those shown in the figure above!

Get Certificate for Bluemix Server

You need an SSL certificate for the Bluemix server to be installed in SAP. The following steps demonstrate how you can export the appropriate certificate using Google Chrome.

1. Start Google Chrome and go to url **https://gateway.watsonplatform.net/**.
   You will receive http status *403 – Forbidden* unless your browser already has installed an appropriate certificate. Ignore the error.
2. Press **F12** to start the developer tools.
3. Click on tab ***Security***.
4. Click button ***View certificate***.
5. In popup dialog click on tab ***Details***.
6. Click button ***Copy to file …*** .

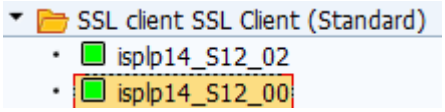


7. Execute wizard to save ***Base-64 encoded X.509*** certificate as file `watsonplatformBase64X509.cer`.

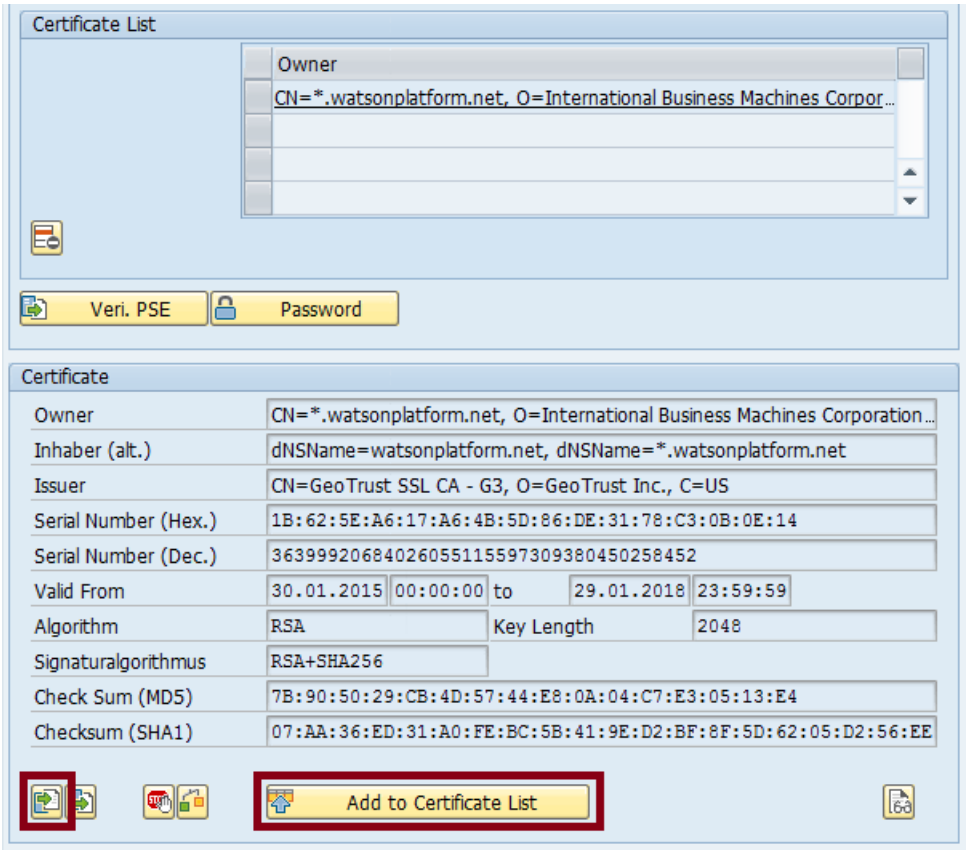
Install Certificate in the SAP System

Proceed as follows to install the exported SSL certificate in your SAP system.

1. In SAP, call transaction **STRUST**.
2. Switch to edit mode (press according tool bar icon).
3. If a local PSE file does not exist already, create it by right-clicking on ***SSL client SSL Client (Standard)*** and selecting ***Create*** from context menu. Keep all default settings in next popup dialog.



4. In Certificate section, click ***Import*** (alternatively select menu item *Certificate* → *Import*). Choose file `watsonplatformBase64X509.cer` and import the certificate.
5. Click ***Add to Certificate List***.



6. Click ***Save*** (F3).

ABAP Code

In this section, all steps are described that you must implement for calling an IBM Watson service from ABAP.

The IBM Watson Text-to-Speech service is called twice. First a REST GET request is issued to receive data about available voices. The response in json format in transformed into an ABAP internal table. Afterwards a REST POST request is sent to synthesize text. In response binary data is received by the ABAP program.

The following ABAP classes are used.
`cl_http_client`
`cl_rest_http_client`

Throughout this section only code snippets are provided. Find the complete executable ABAP code in the appendix. You might want to implement and run the ABAP program before reading this section to get a better understanding about the objective of the steps that are described here.

Create REST client object

First you must create a http client object by calling factory method cl_http_client=>create_by_url. This method requires the service host url (including protocol prefix https://), the proxy server (without protocol prefix) and the proxy server port.

```
data:
  lo_http_client type ref to if_http_client.
cl_http_client=>create_by_url(
  exporting
    url                = 'https://stream-s.watsonplatform.net'
    proxy_host         = 'proxy.domain.com'
    proxy_service      = '8080'
  importing
    client             = lo_http_client
  exceptions
    argument_not_found = 1
    plugin_not_active  = 2
    internal_error     = 3
    others             = 4 ).
lo_http_client->request->set_version( if_http_request=>co_protocol_version_1_1 ).
```

Pass service credentials of your Watson service instance in Bluemix to the http client object. Username and password shown in this code snippet are examples and invalid. Do not use them in your code!

```
lo_http_client->authenticate(
  username = 'b2967e87-4946-47c2-ad7b-3e13f235dfef'
  password = 'nNMGwP7TNJ8E' ).
lo_http_client->propertytype_logon_popup = if_http_client=>co_disabled.
```

REST Call – HTTP GET Request

At first you might want to call the Watson Text-to-Speech service to get information about the available voices. To do so, pass the service uri path to the request object which is an attribute to the http client object.

```
cl_http_utility=>set_request_uri(
  exporting
    request = lo_http_client->request
    uri     = '/text-to-speech/api/v1/voices' ).
```

Based on the http client object you can create a REST client object:

```
data:
  lo_rest_client type ref to cl_rest_http_client.
create object lo_rest_client
  exporting
    io_http_client = lo_http_client.
```

Now you are ready to issue the HTTP GET request.

```
try.
    lo_rest_client->if_rest_client~get( ).
  catch cx_rest_client_exception into data(lo_exception).
    data(lv_msg) = `HTTP GET failed: ` && lo_exception->get_text( ).
endtry.
```

Get the response object and check the request status (http status 200 means OK).

```
data(lo_response) = lo_rest_client->if_rest_client~get_response_entity( ).
data(lv_http_status) = lo_response->get_header_field( '~status_code' ).
if lv_http_status ne 200.
  data(lv_reason) = lo_response->get_header_field( '~status_reason' ).
endif.
```

It's time to receive the response data.

```
data(lv_json_data) = lo_response->get_string_data( ).
```

You get a string variable that contains the response data in json format. To process the data in ABAP, it must be converted into an appropriate ABAP structure. It is recommended to use the ABAP built-in transformation for this task. However, the json string as generated by the Watson service contains new line characters and small letter property names that are not accepted by the ABAP json parser. Thus, you must adjust the json string accordingly.

```
replace all occurrences of cl_abap_char_utilities=>newline in lv_json_data with space.
do.
  find regex '("+.?":)' in lv_json_data submatches data(lv_prop_lowercase).
  if sy-subrc <> 0. exit. endif.
```

```
    data(lv_prop_uppercase) = lv_prop_lowercase.
    translate lv_prop_uppercase to upper case.
    replace all occurrences of lv_prop_lowercase in lv_json_data with lv_prop_uppercase.
  enddo.
```

Afterwards convert the json string to an internal ABAP table using the built-in transformations for json functionality. If the Watson service provides more data property values than fits into your internal ABAP structure, like in this example, specify option *accept data loss*.

```
  types:
    begin of ty_s_voice,
      name        type string,
      language    type string,
      gender      type string,
      description type string,
    end of ty_s_voice.
  data:
    lt_voice type table of ty_s_voice.

  call transformation id source xml lv_json_data
                        result     voices = lt_voice
                        options    value_handling = 'accept_data_loss'.
```

REST CALL – HTTP POST Request

Now you want to let the Watson Text-to-Speech service synthesize some text. Thus, specify the service uri path and parameters to the request object and instantiate a REST client object.

Be careful, the parameter string is case sensitive!

```
cl_http_utility=>set_request_uri(
  exporting
    request = lo_http_client->request
    uri     = '/text-to-speech/api/v1/synthesize?voice=en-US_MichaelVoice').

create object lo_rest_client
  exporting
    io_http_client = lo_http_client.
```

Next you must create a header for the REST request. Here the audio data should be provided in wav format.

```
data(lo_request) = lo_rest_client->if_rest_client~create_request_entity( ).
lo_request->set_content_type( iv_media_type = if_rest_media_type=>gc_appl_json ).
lo_request->set_header_field( iv_name = 'Accept' iv_value = 'audio/wav' ).
```

Create the request body. You must provide a simple structure having only one component with name `text`. The value specifies the text that is supposed to be synthesized. This structure must be provided as an anonymous json object.

```
lo_request->set_string_data(
  '{ "text": "Hi there. This is the IBM Watson Text-to-Speech Service." }' ).
```

Finally, you can issue the POST request ...

```
try.
    lo_rest_client->if_rest_resource~post( lo_request ).
  catch cx_rest_client_exception into data(lo_exception).
    data(lv_msg) = `HTTP POST failed: ` && lo_exception->get_text( ).
endtry.
```

… and receive the binary data in a variable of ABAP type xstring. Verification of http status is skipped here.

```
data(lo_response) = lo_rest_client->if_rest_client~get_response_entity( ).
data(ex_audio) = lo_response->get_binary_data( ).
```

Troubleshooting

Calling a Watson service in Bluemix from within an SAP system may fail for various reason. The following describe some symptoms and how to proceed if you are experiencing those.

GET or POST method throws exception Communication Error

- Verify that the specified host name is correct.
- Check proxy server settings.

HTTP status is 400 (Bad Request)

- Check uri path and parameter string. Pay attention to lower and upper cases.

HTTP status is 401 (Unauthorized)

- Check that user name and password as specified in your program matches those of your service instance in Bluemix.

HTTP status is 403 (Forbidden)

- Procced as follows to check if the SSL certificate for the Bluemix server is installed correctly.
  1. Call transaction **SMICM**.
  2. Select menu item Goto → Trace File → Display End
  3. If you find the following messages, re-install the SSL certificate.

```
[Thr nn] Peer not trusted
[Thr nn] Certificate:
[Thr nn]  Certificate:
[Thr nn]   Subject:         CN=*.watsonplatform.net, O=International
Business Machines Corporation, L=Armonk, SP=New York, C=US
[Thr nn]   Issuer:          CN=GeoTrust SSL CA - G3, O=GeoTrust Inc., C=US
[Thr nn]   Serial Number:   1B:62:5E:A6:17:A6:4B:5D:86:DE:31:78:C3:0B:0E:14
[Thr nn]  Verification result:
[Thr nn]  …
```

References

- IBM Bluemix, includes link to free 30-day trial
  https://www.ibm.com/marketplace/cloud-platform
- IBM Watson Products and Services
  https://www.ibm.com/watson/products-services/
- IBM Watson Developer Community
  https://developer.ibm.com/watson/
- SAP Help: ABAP REST Library
  https://help.sap.com/viewer/753088fc00704d0a80e7fbd6803c8adb/7.5.9/en-US/2850217946b54e718e1f4afb35c4c283.html
- SAP Help: Transformations for JSON
  https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenabap_json_trafos.htm

Appendix

Find here the complete code of demo ABAP report Z_WATSON_SERVICE.

This report first calls the IBM Watson Text-to-Speech service to receives information about all supported voices and writes this data to a list box. The end user selects his favorite voice from that list box, enters an arbitrary text and executes the report. Watson Text-to-Speech service is again called to synthesize the text into binary data in wav format. Afterwards the wav data is downloaded onto the user's frontend pc and a media player is called to play the speech.

When implementing the ABAP report you must change the constants in the first lines according to the credentials of your individual Watson Text-to-Speech service instance in Bluemix. Also, adjust settings for the proxy server.

```
*&---------------------------------------------------------------------*
*& Report Z_WATSON_SERVICE
*&---------------------------------------------------------------------*
*& Example for calling IBM Watson services from ABAP.
*&
*& !! This report comes "as is" without any support and warranty.
*& !! Intended for educational purpose only.
*&
*& Adjust constants below according to the credentials of your
*& Watson Text-to-Speech service instance in IBM Bluemix and
*& your proxy server settings
*&---------------------------------------------------------------------*
report z_watson_service no standard page heading.

constants:
  gc_host           type string value 'https://stream-s.watsonplatform.net',   " <- CHANGE
  gc_srv_user       type string value 'b2967e87-4946-47c2-ad7b-3e13f235dfef', " <- CHANGE
  gc_srv_password   type string value 'nNMGwP7TNJ8E',                         " <- CHANGE
  gc_proxy_host     type string value 'proxy.domain.com',                     " <- CHANGE
  gc_proxy_service  type string value '8080',                                 " <- CHANGE
  gc_uri_voices     type string value '/text-to-speech/api/v1/voices',        " <- check version
  gc_uri_synthesize type string value '/text-to-speech/api/v1/synthesize',    " <- check version

  gc_filename_audio type string value 'watson_speech.wav'.


" selection screen for text and voice input parameters
selection-screen begin of block b_input with frame title t_input.
selection-screen begin of line.
selection-screen comment 1(10) t_text for field i_text.
parameters: i_text type string lower case obligatory.
selection-screen end of line.

selection-screen begin of line.
selection-screen comment 1(10) t_voice for field i_voice.
parameters: i_voice type char32 as listbox lower case visible length 64 obligatory.
selection-screen end of line.
selection-screen end of block b_input.

initialization.
  t_input  = 'Input'.                                    "#EC NOTEXT
```

```
  t_text   = 'Text'.                                        "#EC NOTEXT
  t_voice  = 'Voice'.                                       "#EC NOTEXT

at selection-screen output.
  data:
    l_t_value type vrm_values.

  " call Watson text-to-speech service for parameter options
  perform get_voices changing l_t_value.

  " fill listbox for voice parameter
  call function 'VRM_SET_VALUES'
    exporting
      id     = 'I_VOICE'
      values = l_t_value
    exceptions
      others = 0.


*---------------------------------------------------------------------*
*     main program
*---------------------------------------------------------------------*
start-of-selection.

  data:
    lx_audio type xstring.

  " retrieve binary audio data (wav format) from Watson text-to-speech service
  perform get_audio
    using   i_text
            i_voice
    changing lx_audio.

  " play audio on frontend pc
  perform play_audio
    using lx_audio.

  exit.
*---------------------------------------------------------------------*



*&---------------------------------------------------------------------*
*&      Form  create_rest_client
*&---------------------------------------------------------------------*
*       creates a REST client and sets headers appropriately
*---------------------------------------------------------------------*
*      -->IV_URI             uri path of Watson service to be called
*      <--EO_REST_CLIENT     http REST client object
*---------------------------------------------------------------------*
form create_rest_client
     using    iv_uri         type string
     changing eo_rest_client type ref to cl_rest_http_client.

  data:
    lo_http_client type ref to if_http_client.

  " create http client instance
  cl_http_client=>create_by_url(
    exporting
      url                = gc_host             " Watson service host (w/o uri path)
      proxy_host         = gc_proxy_host       " proxy server (w/o protocol prefix)
```

```
      proxy_service      = gc_proxy_service    " proxy port
    importing
      client             = lo_http_client
    exceptions
      argument_not_found = 1
      plugin_not_active  = 2
      internal_error     = 3
      others             = 4 ).
  if sy-subrc <> 0.
    perform raise_message using 'E' 'Cannot create HTTP client.'. "#EC NOTEXT
  endif.

  " set http protocol version
  lo_http_client->request->set_version( if_http_request=>co_protocol_version_1_1 ).

  " provide service credentials for authentification
  lo_http_client->authenticate( username = gc_srv_user password = gc_srv_password ).
  lo_http_client->propertytype_logon_popup = if_http_client=>co_disabled.

  " provide uri path (w/o host) in format /../api/v1/service?var=...
  cl_http_utility=>set_request_uri(
    exporting
      request = lo_http_client->request
      uri     = iv_uri ).

  " create REST client instance from http client instance
  create object eo_rest_client
    exporting
      io_http_client = lo_http_client.

endform.                                                    "create_rest_client



*&---------------------------------------------------------------------*
*&      Form  get_voices
*&---------------------------------------------------------------------*
*       calls Watson text-to-speech service via http GET request,
*       receives data in json format and converts it into an internal
*       ABAP structure
*----------------------------------------------------------------------*
*      <--ET_VALUE      abap table with response data
*----------------------------------------------------------------------*
form get_voices changing et_value type vrm_values.

  " abap structure used to be filled by json data
  types:
    begin of ty_s_voice,
      name        type string,
      language    type string,
      gender      type string,
      description type string,
    end of ty_s_voice.

  data:
    lo_rest_client type ref to cl_rest_http_client,
    lt_voice       type table of ty_s_voice,
    ls_value       type line of vrm_values.

  " create http rest client object
  perform create_rest_client
          using    gc_uri_voices
```

```
          changing lo_rest_client.

  " perform HTTP GET request
  try.
      lo_rest_client->if_rest_client~get( ).
    catch cx_rest_client_exception into data(lo_exception).
      data(lv_msg) = `HTTP GET failed: ` && lo_exception->get_text( ).
      perform raise_message using 'E' lv_msg.               "#EC NOTEXT
  endtry.

  " get and check REST call response
  data(lo_response) = lo_rest_client->if_rest_client~get_response_entity( ).
  perform check_response using lo_response.

  " receive response data in json format
  data(lv_json_data) = lo_response->get_string_data( ).

  " tweak json a little bit such that it is understood by abap json parser

  " eliminate 0x000A
  replace all occurrences of cl_abap_char_utilities=>newline in lv_json_data with space.

  " property names must be upper case
  do.
    " the .? in regex makes it non-greedy
    find regex '("+.?":)' in lv_json_data submatches data(lv_prop_lowercase).
    if sy-subrc <> 0. exit. endif.
    data(lv_prop_uppercase) = lv_prop_lowercase.
    translate lv_prop_uppercase to upper case.
    replace all occurrences of lv_prop_lowercase in lv_json_data with lv_prop_uppercase.
  enddo.

  " call json parser (ignore properties that do not exist in abap structure)
  call transformation id source xml lv_json_data
                        result     voices = lt_voice
                        options    value_handling = 'accept_data_loss'.

  " fill returning internal table
  loop at lt_voice into data(ls_voice).
    ls_value-key  = ls_voice-name.
    ls_value-text = ls_voice-description.
    append ls_value to et_value.
  endloop.

endform.                                                    " get_voice



*&---------------------------------------------------------------------*
*&      Form  get_audio
*&---------------------------------------------------------------------*
*       calls Watson text-to-speech service via http POST request and
*       receives binary audio data
*----------------------------------------------------------------------*
*      -->IV_TEXT      text to be synthesized
*      -->IV_VOICE     voice parameter for synthesization
*      <--EX_AUDIO     binary audio data (wav) with sythesized text
*----------------------------------------------------------------------*
form get_audio using    iv_text  type string
                        iv_voice type char32
               changing ex_audio type xstring.
```

```
  data:
    lo_rest_client type ref to cl_rest_http_client,
    lv_uri         type        string,
    lv_body        type        string.

  " create uri; caution: uri is case sensitive!
  " Service does not work when character cases are not accurate.
  lv_uri = gc_uri_synthesize && `?voice=` && iv_voice.

  " create http rest client object
  perform create_rest_client
          using    lv_uri
          changing lo_rest_client.

  " generate http request header
  data(lo_request) = lo_rest_client->if_rest_client~create_request_entity( ).
  lo_request->set_content_type( iv_media_type = if_rest_media_type=>gc_appl_json ).
  lo_request->set_header_field( iv_name = 'Accept' iv_value = 'audio/wav' ). "#EC NOTEXT

  " build json for http request body
  lv_body = `{ "text": "` && iv_text && `" }`.              "#EC NOTEXT
  lo_request->set_string_data( lv_body ).

  " perform HTTP POST request
  try.
      lo_rest_client->if_rest_resource~post( lo_request ).
    catch cx_rest_client_exception into data(lo_exception).
      data(lv_msg) = `HTTP POST failed: ` && lo_exception->get_text( ).
      perform raise_message using 'E' lv_msg.               "#EC NOTEXT
  endtry.

  " get and check REST call response
  data(lo_response) = lo_rest_client->if_rest_client~get_response_entity( ).
  perform check_response using lo_response.

  " receive data
  ex_audio = lo_response->get_binary_data( ).

endform.                                                    " get_audio



*&---------------------------------------------------------------------*
*&      Form  play_audio
*&---------------------------------------------------------------------*
*       saves a binary string as *.wav file on the frontend and
*       starts playing that *.wav file
*----------------------------------------------------------------------*
*      -->IX_AUDIO     audio data in wav format
*----------------------------------------------------------------------*
form play_audio using ix_audio type xstring.
  data:
    lt_bin_data  type standard table of raw255,
    lv_directory type string,
    lv_dummy     type string,                               "#EC NEEDED
    lv_filename  type string.

  " convert xstring to table of raw
  call function 'SCMS_XSTRING_TO_BINARY'
    exporting
      buffer     = ix_audio
    tables
```

```
      binary_tab = lt_bin_data.

  " get default gui directory, usually .../MyDocuments/SAP/SAP GUI/
  call method cl_gui_frontend_services=>get_upload_download_path
    changing
      upload_path   = lv_directory
      download_path = lv_dummy
    exceptions
      others        = 0.  " ignore errors here
  lv_filename = lv_directory && gc_filename_audio.

  " save *.wav file on frontend
  call function 'GUI_DOWNLOAD'
    exporting
      filename = lv_filename
      filetype = 'BIN'
    tables
      data_tab = lt_bin_data
    exceptions
      others   = 99.
  if sy-subrc <> 0.
    perform raise_message using 'E' 'GUI_DOWNLOAD failed.'. "#EC NOTEXT
  endif.

  " execute *.wav file on frontend (use player that the *.wav extention is associated with)
  call method cl_gui_frontend_services=>execute
    exporting
      document  = lv_filename
      minimized = 'X'.

endform.                                                    " play_audio



*&---------------------------------------------------------------------*
*&      Form  check_response
*&---------------------------------------------------------------------*
*       checks the http status of a REST response and issues
*       appropriate message
*----------------------------------------------------------------------*
*      -->IO_RESPONSE      REST response object
*----------------------------------------------------------------------*
form check_response using io_response type ref to if_rest_entity.
  data:
    lv_message type string.
  data(lv_http_status)  = io_response->get_header_field( '~status_code' ).
  data(lv_reason)       = io_response->get_header_field( '~status_reason' ).
  data(lv_content_type) = io_response->get_header_field( 'content-type' ).

  if lv_http_status eq 200.
    " http status 200 = OK
    lv_message = `Data of type ` && lv_content_type && ` received.`. "#EC NOTEXT
    perform raise_message using 'S' lv_message.
  else.
    " http error status
    lv_message = `HTTP status ` && lv_http_status && ` (` && lv_reason && `)`. "#EC NOTEXT
    perform raise_message using 'E' lv_message.
  endif.

endform.                                                    " check_response
```

```
*&---------------------------------------------------------------------*
*&      Form  raise_message
*&---------------------------------------------------------------------*
*       raises a messages
*----------------------------------------------------------------------*
*      -->IV_MTYPE     message type, e.g. 'E' or 'S'
*      -->IV_MESSAGE   message text
*----------------------------------------------------------------------*
form raise_message using iv_mtype   type sy-msgty
                         iv_message type string.
  message iv_message type iv_mtype.
endform.                                                    " raise_message
```