Debugging notes for JSON

In C++ REST SDK, they used non-standard C++ packages and APIs, which makes it a bit troubling to debug, since there are many `strange' ways of using the API function, instead of the STL.

In this document, I will write some debugging problems and their solutions for using the C++ REST SDK.

Pre-reading

The background reading is very important before the start of debugging, in this section, I will discuss about how to use the API and example code for debugging and developing.

References:

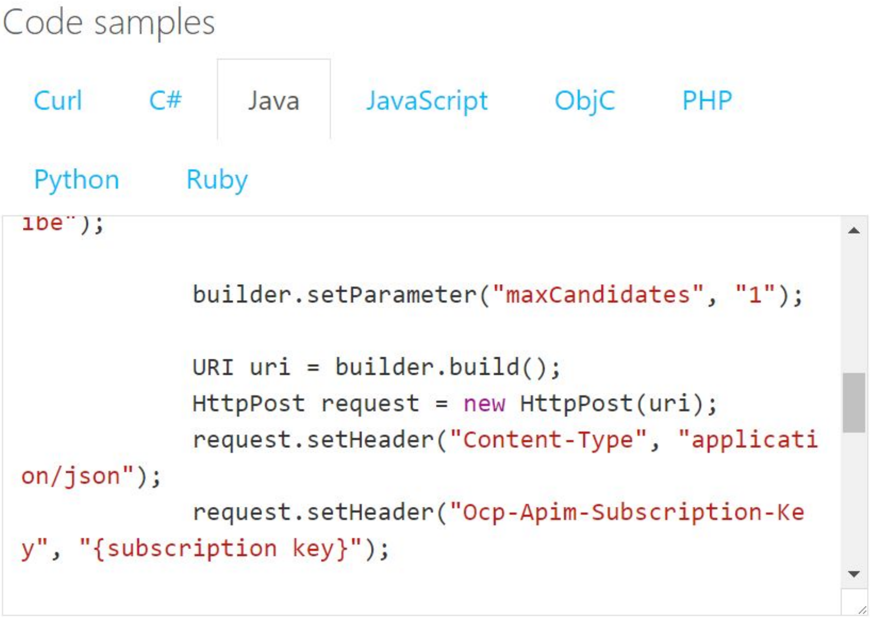
1. API examples:https://westus.dev.cognitive.microsoft.com/docs/services/56f91f2d778daf23d8ec6739/operations/56f91f2e778daf14a499e1fe
2. SDK Documentation:https://microsoft.github.io/cpprestsdk/classweb_1_1http_1_1http__response.html#ae2ee403657d29cc52abb94a7a9b0571c

The above two links are very important for developing. Actually some tutorial of Microsoft Cognitive Service is good for beginners, but here I will assume that you have got known how it works and what's the function of them. This section will only focus on the ***details*** of the develop, not the overview and functions of each API/subscriptions.

Send a REST request:
1)construct a URL(this url may contain many parameters), for example:
`https://westus.api.cognitive.microsoft.com/face/v1.0/detect?returnFaceId=true&returnFaceLandmarks=false`
2)construct request body, this body should contain the http header, subscription keys(otherwise they will not authorize)
3)get the response and parse it.

To finish this request, we have to know how to use their SDK to do that, according to our reading in the Reference 2 in the above, we should use *class http_request.*

Then how to construct such a class?
In the link above, there is code example using C#, Javascript, Python, PHP, etc… but their is no for C++, so the best way is refer to JAVA code, as they are similar to C++ and clear to audience.



```
ibe");

            builder.setParameter("maxCandidates", "1");

            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);
            request.setHeader("Content-Type", "applicati
on/json");
            request.setHeader("Ocp-Apim-Subscription-Ke
y", "{subscription key}");
```

We should use that to setHeader, setbody etc… And then go back to check the SDK of C++ version. Some functions don't have the C++ version, so google the answers. **They're lacking of the examples in C++, lol...**

After that, for request body, there are two ways, use a remote url that points to the actual image, or use a local image. For these two different ways, use different parameters, if remote url, just pass some string to it; but for local images, I have to read from the file, put them into the streams, and then set the stream data in the request body, ALSO, we need to set the correct encode/decode, like `text', or `octex-stream' or so. If not, the server will return a lot of ERROR code(400, 401, 403, 415). Debug according to the code!

The last step is as the following data parsing, we have *http_response class*, then we need to parse from this string to get each useful keyword and its value.

Some problems I faced:
1)string error, like use raw string as subscription key, but **need 'L', or 'U' to constrain** them. Previously I want to store it in variables, but then it failed to compile, they have conversion in the sdk, so add these constraints.
2)network issue, or url problem. As mentioned, the url can have different parameters, if not setting, they will ***have default `false' value, then we lose some information(age, gender,***

etc..), after check the source code of Android APP and C# example, I can get the reasons behind it.
3) network or API calling issue, http_request() and http_response() have documents, but some function is not as well though, google some code(including some error code to learn)
4) some are related with network programming(like REST), first I don't add the parts of HEADer, because their document say it's by default and we don't need it. But server refuse to analyze image if not setting it. I think the document is not complete and misleading.

Data Parsing

After we get the http_response(), then next step is to parse it! We can save directly into file and check its content to see whether it's OK, but for other modules to use(like Qilei's module), then parsing and providing API interface is necessary.

Previous code:

```
        response.extract_json().then([&json](pplx::task<json::value> previousTask) {
                        try
                        {
                                const json::value& v = previousTask.get();
                                printf("a, %d", v.size());
                                //std::string encoding = v.get("UTF-8").asString();
                                // Perform actions here to process the JSON value...
                                //std::string fromStringT(conversions::to_utf8string(v.as_string()));
/// there maybe a problem....
                                printf("cccc\n");  /// cannot output
                        }
                        catch (const http_exception& e)
                        {
                                // Print error.
                                std::wostringstream ss;
                                ss << e.what() << std::endl;
                                std::wcout << ss.str();
                                printf("ddddd\n");
                        }

                });
```

Now code:

```
        printf("Received response status code:%u\n", response.status_code());
                utility::string_t tt = response.extract_string().get();
                std::string from(conversions::to_utf8string(tt));
```

```
std::cout << from << std::endl;
```

Some tricks:

1) The returned string is not directly a JSON string, it contains the `[]’ characters.[**before we use the extract_json, but actually it’s not a JSON string, it contains `[]’**]
2) The function is not directly using std::string, it uses its own string_t, we need to find correct function to convert
3) The extract_json() function is a task, don’t call directly. We have to use other threads or sub-task!