

SUP OData Client SDK – OData Parser Simplified

This document aims at simplifying the use of SDM Parser by using the c-style macro calls that reduce the number of code lines to perform the same operation.

Author: Pramod.M
Field of work: Lead iOS Developer
Company: SAP Labs India Pvt. Ltd.

TABLE OF CONTENTS

INTRODUCTION	3
SETTING UP THE PROJECT	3
Linker Flags.....	3
Compiler	3
PARSING SERVICE DOCUMENT.....	4
Objective-C Style	4
C-Style.....	4
PARSING META DOCUMENT	4
Objective-C Style	4
C-Style.....	5
PARSING A COLLECTION OF ENTRIES (DATA)	5
Objective-C style.....	5
C-Style.....	5
READING THE PROPERTIES OF AN ENTRY	6
RELATED CONTENT	7

INTRODUCTION

The target audience for this document are the iOS application developers who are using the SUP OData client library.

The OData SDK client on iOS consists of 3 major libraries apart from the **SUPProxyClient** library, which will be used for connectivity via the SUP. They are:-

- SDMConnectivity
- SDMParser
- SDMSupportability

SDMConnectivity is used for direct gateway connectivity bypassing the SUP infrastructure, which requires the enterprise to setup its own reverse proxy to filter the incoming requests.

SDMSupportability is used to log and trace the application requests for easier support during erroneous behavior.

SDMParser is used to parse the OData payload in case the client is connecting to an OData provider back-end. The SDMParser has a basic set of API (Objective-C styled), which we normally use to parse the service document, meta-data and the data document. This document highlights another API set which are written in C-style to simplify the application by reducing the code lines.

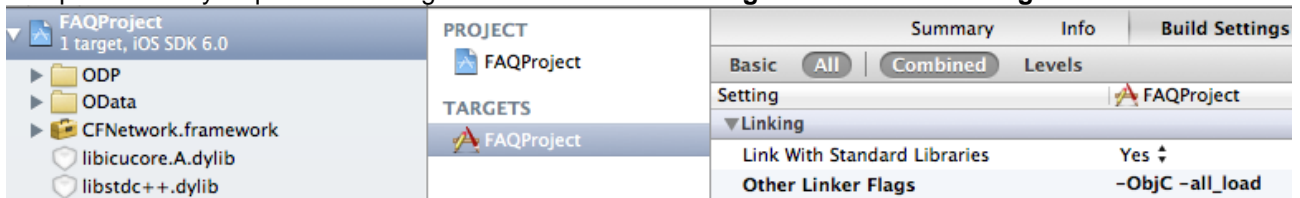
This document gives a comparison of the normal Objective-C method calls an application developer makes to parse the respective documents and the corresponding C-style macros that simplify the application coding. This document explains the process of parsing the data and accessing its properties.

SETTING UP THE PROJECT

Before starting with the application code, this section explains a couple of details that are not to be missed while setting up the project.

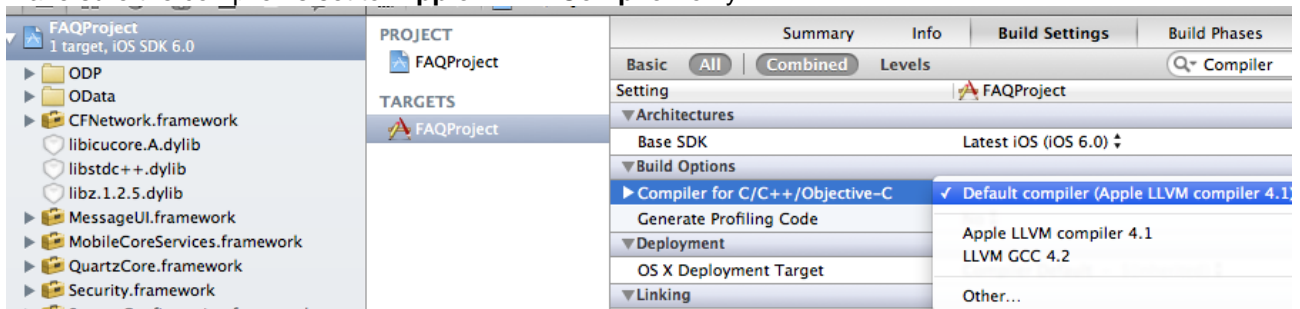
Linker Flags

The parser library requires the 2 flags to be set in **Build Settings -> Other Linker Flags**



Compiler

Make sure the compiler is set to 'Apple LLVM Compiler' only.



PARSING SERVICE DOCUMENT

The parsing begins with the service document without which we will not be able to proceed further. Let us first look at the Objective-C API and then the simplified C-Style macro.

Objective-C Style

First few lines show the necessary header file imports. The code follows afterwards.

```
#import "SDMODataServiceDocumentParser.h"
#import "SDMODataServiceDocument.h"

- (void)parseServiceDocument
{
    NSData* parseableData = [@"response data" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the actual data to be
    parsed in NSData* format
    SDMODataServiceDocumentParser* parser = [[SDMODataServiceDocumentParser alloc] init];
    [parser parse:parseableData];
    SDMODataServiceDocument* serviceDocument = [parser.serviceDocument retain];
    [parser release];
}
```

C-Style

Now, instead of the **SDMODataServiceDocumentParser.h** header file, we will import a new header file **SDMParser.h** which encloses the C-style API. The code for the C-style API follows.

```
#import "SDMParser.h"
#import "SDMODataServiceDocument.h"

- (void)parseServiceDocument
{
    NSData* parseableData = [@"response data" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the actual data to be
    parsed in NSData* format
    SDMODataServiceDocument* serviceDocument = [sdmParseODataServiceDocumentXML(parseableData) retain];
}
```

One line of code replaces 4-5 lines of code and memory management.

PARSING META DOCUMENT

The next document that needs to be parsed is the metadata document that is obtained by appending \$metadata to the service document URL. Like the previous section, we will first look at the normal Objective-C style API first and then the shortened C-Style macro later.

Objective-C Style

We need to import 2 more header files on top of the previously imported header file (refer the previous sections). The code to parse the document follows.

```
#import "SDMODataMetaDocumentParser.h"
#import "SDMODataSchema.h"

- (void)parseMetaData
{
    NSData* parseableData = [@"metadata" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the meta data to be parsed in
    NSData* format
    SDMODataMetaDocumentParser* metaParser = [[SDMODataMetaDocumentParser alloc] initWithServiceDocument:serviceDocument]; //
    Initialize with the previously obtained service document
    [metaParser parse:parseableData];
    SDMODataSchema* metaSchema = [metaParser.serviceDocument getSchema];
}
```

C-Style

Since we have already imported the **SDMParser.h** header file, all you need to do is import the **SDMODataSchema.h** header file. The simple version of the code follows.

```
#import "SDMODataSchema.h"

- (void)parseMetaData
{
    NSData* parseableData = [@"metadata" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the actual data to be parsed in
    NSData* format
    SDMODataSchema* metaSchema = sdmParseODataSchemaXML(parseableData, serviceDocument); // Pass the service document obtained in
    the previous section
}
```

PARSING A COLLECTION OF ENTRIES (DATA)

It is to be noted here that before parsing data, it is mandatory that the developer fetches the service document and meta-data, parses them and stores it locally for it to be used by the data parser.

Objective-C style

We need import 2 more header files as shown below. The code with the Objective-C interface follows.

```
#import "SDMODataCollection.h"
#import "SDMODataDataParser.h"

- (void)parseData
{
    NSData* parseableData = [@"data" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the actual data to be parsed in
    NSData* format
    SDMODataCollection* collection = [metaSchema getCollectionByName:@"CollectionName"]; // Schema obtained from the previous section.
    Collection name of your choice
    SDMODataDataParser* dataParser = [[SDMODataDataParser alloc] initWithEntitySchema:collection.entitySchema
    andServiceDocument:serviceDocument]; // Service document obtained from the first section
    [dataParser parse:parseableData];
    NSMutableArray* entries = dataParser.entries;
}
```

C-Style

```
#import "SDMODataCollection.h"

- (void)parseData
{
    NSData* parseableData = [@"data" dataUsingEncoding:NSUTF8StringEncoding]; // DUMMY : This will hold the actual data to be parsed in
    NSData* format
    SDMODataCollection* collection = [metaSchema getCollectionByName:@"CollectionName"]; // Schema obtained from the previous section
    NSMutableArray* entries = sdmParseODataEntriesXML(parseableData, collection.entitySchema, serviceDocument); // Service document
    obtained from the first section
}
```

READING THE PROPERTIES OF AN ENTRY

Once we obtain the array of entries, we need to read the desired property(ies) of the entry for further operations. It is to be understood that the array so obtained in the previous section will hold objects of type **SDMODataEntry**. So the application can either choose to extract all the entries by looping on the array OR extract the desired entry if the application is already aware of the position of the entry in the array.

Once an entry is extracted, the code snippet below shows how to read the property. Here let us assume the property name is "NAME". We need to import a header file **SDMODataEntry.h** in order to work with its methods.

```
#import "SDMODataEntry.h"

for (SDMODataEntry* entry in entries) // Entries obtained in the previous section
{
    SDMODataPropertyValueObject* propValueObj = [entry getPropertyValueByPath:@"NAME"];
    SDMODataPropertyValueString* propValueStr = (SDMODataPropertyValueString *)propValueObj;
    NSString* propValue = [propValueStr getValue];
}
```

RELATED CONTENT

[SCN Mobile Home Page](#)

© 2012 SAP AG. All rights reserved.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, SAP HANA, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. Business Objects is an SAP company.

Sybase and Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase Inc. Sybase is an SAP company.

Crossgate, m@gic EDDY, B2B 360°, and B2B 360° Services are registered trademarks of Crossgate AG in Germany and other countries. Crossgate is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

