ISSN 2319 – 1953
International Journal of Scientific Research in Computer Science Applications and Management Studies

Protocol Buffers: an Overview(case study in C++)

Jyoti Lakhani
Department of Computer Science
Maharaja Ganga Singh University, Bikaner
Bikaner, India
jyotilakhanimgsu@gmail.com

***Abstract*— This paper is impeding light on a new data interchange format on the web, the "*Protocol Buffers*". Protocol buffers can be compared with Extensible Mark-up Language (*XML*) for flexibility and platform independence. This communication gives an overview of protocol buffers from their installation to compilation and implementation.**

Keywords- Protocol Buffers, Google, Interchange Language, Intermediate Language, C++

I. INTRODUCTION

Protocol Buffers is an internal open source language of Google for data interchange on the web. It is a business rival of extensible mark-up language (*XML*). *XML* is a famous data interchange format of web1.0 and 2.0. Like *XML*, protocol buffers are flexible and platform independent message format for data interchange on the modern web. Protocol buffers are used to transfer data of web services and web applications on the web using Remote Procedure Calls (RPC). This communication is discussing about how protocol buffers can be used. The C++ programming language is taken as a case to understand the protocol buffers implementation and use. A systematic layout is presented here for installation, compilation, designing and implementation of messages of protocol buffers in C++ programming language.

II. STRUCTURE OF A PROTOCOL BUFFER MESSAGE

A message can be written in protocol buffers in following format (in the .proto file).Maintaining the Integrity of the Specifications

```
message <Message_Name>
{
  <Field_Rule> <Field_Type> <Field_Name> = <Number-Tag>;
}
```

In above message format <*Message-Name*> is a self-explanatory name of the message. A message can contain more than one field (a piece of message information) declarations. Each field declaration consist of <*Field-Rule*>, <*Field-Type*>, <*Field-Name*> and <*Number-Tag*>. The <*Field-Name*> is the name of the field information. The <*Field-Type*> is the indication of type of the data this field can hold. *Field-Type* can be scalar or composite. As per the language guide provided online by the official site of protocol buffers, there are fifteen different scalar field types available in protocol buffers. These are double, float, int32, int64, uint32, uint64, sint32, sint64, fixed32, fixed64, sfixed32, sfixed64, bool, string and bytes. It is mandatory to specify one of the Field-Rule from *required*, *optional* and *repeated*. A well formed message can have exactly one field of *required* type and zero or more values for *optional* type. The repeated fields can be repeated 0 to any number of times. The order of repeated values will be preserved. Each field in the message definition has a unique <*number-tag*>. As this message will be transferred to the destination in the binary format (through the network), this unique <*number-tag*> will be treated as an identification mark for the field. The smallest value of tag can be 0 and the largest value of tag can be 2^{29}-1(536,870,911). Numbers between the ranges 19000 to 19999 are reserved values. These cannot be used to assign the <*number-tag*>. Enumerations (*enum*) can be added in a composite type of a message. Enumeration (*enum*) is a list of constants. A field with enum type can have one of the constant value declared in enum[7].

Another charming feature of protocol buffers is that messages can be nested.

```
message <Message_Name_outer>
{
   <Field_Rule> <Field_Type> <Field_Name> = <Number-Tag>;

        message <Message_Name_inner>
          {
                <Field_Rule> <Field_Type> <Field_Name> = <Number-Tag>;
          }
}
```

Another way to nest the fields of messages is by implementing *groups.* A *group* combines a nested message type into a single message definition. It is possible to extend a pre-written protocol buffers message by the third party. This is possible by the use of *extensions.*

```
message <Message_Name>
{
  <Field_Rule> <Field_Type> <Field_Name> = <Number-Tag>;
   extensions <Number-Tag> to <Number-Tag>
}
```

ISSN 2319 – 1953
International Journal of Scientific Research in Computer Science Applications and Management Studies

At third party end, the user can extend the message as follows:

```
extend <Message_Name>
{
 <Field_Rule> <Field_Type> <Field_Name> = <Number-Tag>;
}
```

If the protocol buffers will be used for Remote Procedure Call (RPC), an "RPC service interface" should be defined in the .proto file. This will direct the protocol buffers compiler to generate service interface code and stubs in the chosen language of the user. This can be done as following:

```
Service <Service_Name>
{
 rpc Search (SearchRequest) returns (SearchResponse);
}
```

III. INSTALL PROTOCOL BUFFERS COMPILER

The compiler for protocol buffers can be downloaded from protocol buffers official site [8]. The site contains links for different compilers of various versions. Download the latest version (protoc-2.5.0-win32.zip) from the compiler list. This is the binary version of protocol buffers compiler. To install the compiler, copy the protoc (protocol buffers compiler) at the desired location. Include the exact path of the protoc file to the system's path. The protocol buffers compiler (protoc) will now be ready to use.

IV. COMPILATION OF .PROTO FILE

The .proto file can be compiled using protoc (Protocol buffers compiler) by typing the following line-

(If the current path is different than the source and destination paths of proto files)

```
protoc --proto_path = IMPORT_PATH --cpp_out = DST_DIR path/to/File_name.proto
```

Here, the --proto_path is the path of .proto file. IMPORT_PATH is the path of imported files. If this is omitted, the current path is taken as default. --cpp_out is used by compiler to create c++ code in DST_DIR (destination directory)

(If proto files are situated in the current path, means current path itself is the source as well as destination)

```
protoc -- cpp_out =. File_name.proto
```

Here, the . (dot) is the symbolic notion of current path. [7]

The above compilation code will generate two files-

1. File_name.pb.h
2. File_name.pb.cc

The File_name.pb.h file contains the classes declared in .proto file in the form of messages. The File_name.pb.cc contains the implementation of classes in C++ format. These two files in combination are protocol buffers API. [9]

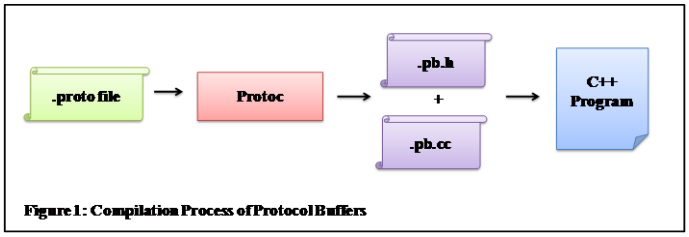

Figure 1: Compilation Process of Protocol Buffers

V. CREATION OF C++ SOURCE CODE FILE

The message written in .proto file (in the form of classes in .pb.h file) can be used in C++ source code. One has to include the File_name.pb.h file created by the protocol buffers compiler in the cpp source file before using the message fields.

```
#include "File_name.pb.h"
```

The message fields can be accessed by using <Message_Name> followed by . (dot) and then <Field_Name> (typical C++ style).

```
<Message-Name>.<Field_Name>
```

This is because protocol buffers compiler generates a class for each message with the respective name. All the message fields are included in classes as members. The C++ style of accessing a member from a class is class_name.member_name.

This way the message fields can be used in protocol buffers.

VI. CONCLUSION

The protocol buffers are modern data exchange format used by the Google. Protocol buffers are platform independent, intermediate language initially direct supported in java, C++ and python. Protocol buffers provide an interface to create and extend messages. This communication gives a step by step procedure for installation, compilation, implement and use of the protocol buffers.

REFERENCES

[1]. K. Varda, Protocol Buffers: Google's Data Interchange Format , Monday, July 7, 2008. The Google Open Source Blog
[2]. Protocol Buffers, http://code.google.com/p/protobuf/
[3]. Interview: Google Open Sources Protocol Buffers, By *Kurt Cagle* July 18, 2008, O'Really Blogs
[4]. W. Schwitzer, V. Popa, Using Protocol Buffers for Resource-Constrained Distributed Embedded Systems2011.
[5]. Developer Guide: Overview, https://developers.google.com/protocol-buffers/docs/overview
[6]. Other Languages, https://developers.google.com/protocol-buffers/docs/reference/other?hl=en
[7]. Language Guide, https://developers.google.com/protocol-buffers/docs/proto#scalar
[8]. Protobuf Download, http://code.google.com/p/protobuf/downloads/list
[9]. https://developers.google.com/protocol-buffers/docs/cpptutorial
[10]. Cpp tutorial, https://developers.google.com/protocol-buffers/docs/cpptutorial