Internet Engineering Task Force (IETF)
INTERNET-DRAFT
Intended Status:
Expires: January 7, 2016

Phillip Hallam-Baker
Comodo Group Inc.

July 6, 2015

Protocol Specification Tool
draft-hallambaker-protogen-01

Abstract

The syntax for the PROTOGEN protocol specification tool is described and the use of the tool to generate protocol specifications, prototype and production implementations. While the primary focus of PROTOGEN is to develop protocols using JSON message syntax, the PROTOGEN framework has been successfully applied to generate prototypes using ASN.1, TLS, XML and RFC822 style syntax.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Table of Contents

1. Introduction

The use of schemas to describe communication protocols is well established and plays a central role in the development of ASN.1 and XML based protocols. No such tools are currently widely used for writing JSON based protocols.

It is the view of the author that the first, last and only purpose of a protocol schema language is to enable the use of tools to support the development effort. A schema language that delays rather than advances the development of correct code and consistent documentation has become a liabilty, not an enabler.

1.1. Previous work

One of the main reasons for the lack of such a tool has been the widespread concern as to the complexity of traditional schema tools and in particular the tendency of such tools to impose a complex data model on simple problems.

One major difference in the design of the Protogen schema language to its predecessors is that it does not attempt to support every feature of the JSON data model. Protogen is designed to allow programmers to design and implement network service protocols quickly using widely used programming languages such as C, C# and Java. JSON features that do not map conveniently to the majority of widely used languages are best ignored.

The XML Schema language is particularly obtuse presenting a two level type system in which element definitions provide typing for data and element types provide a type system for elements. At least three different inheritance mechanisms are supported.

The ASN.1 schema language introduces a distinction between lists and sets that is entirely frivolous in a serialization format and gratuitous distinctions between implicit and explict tagging.

The lesson to be drawn from these abominations is clear: The primary purpose of a schema language should be to allow the programmer to forget and ignore the wireline representation of protocol messages. Features that allow fine tuning of the wireline representation should be avoided.

While the notion of validating input data against a schema prior to passing data to an application is superficially attractive, schema constraints are rarely sufficient for this purpose. Thus applied to protocol design, schema validation rarely provides a meaningful benefit over checking that an encoding is well formed.

1.2. Schema driven documentation

1.3. Schema driven code generation

1.4. Application Examples

The following is based on an example from [RFC4627].

```
[
      {
         "precision": "zip",
         "Latitude":  37.7668,
         "Longitude": -122.3959,
         "Address":   "",
         "City":      "SAN FRANCISCO",
         "State":     "CA",
         "Zip":       "94107",
         "Country":   "US"
      },
      {
         "precision": "zip",
         "Latitude":  37.371991,
         "Longitude": -122.026020,
         "Address":   "",
         "City":      "SUNNYVALE",
         "State":     "CA",
         "Zip":       "94085",
         "Country":   "US"
      }
   ]
```

The corresponding Protogen schema is:

```
     Structure SiteList
             Description
                     |A list of sites
             Struct Site Sites
                     Multiple

     Structure Site
             Description
                     |A site location
             String          Country
                     Description
                             |ISO ALPHA-2 Country Code.
             String          precision
             Decimal         Latitude
             Decimal         Longitude
             String          Address
             String          City
             String          State
```

For the sake of example, the description of the site structure entries is elided. While Protogen does not require description elements to be provided to produce code, descriptions are of course essential if useful documentation is to be generated.

Protogen is built using the Goedel code metasynthesizer which attempts to eliminate all unnecessary clutter from the code specification to minimize error. By default, indentation and the off-side rule are used to denote block structure following the approach used in occam and Python. Punctuation characters are only used to delimit strings ("), text blocks (|) and comments (!).

Note that the Latitude and Longitude are specified using the type Decimal rather than Float. This allows an implementation to avoid the loss of precision that inevitably occurs converting between a binary floating point representation such as IEEE 754 binary 64 and the decimal encoding used in JSON.

The example fragment is sufficient to describe a data structure and generate methods for JSON serialization and deserializtion. It is not however sufficient to generate a useful implementation of a Web service or client access library. to do this we must define a protocol with services, transactions and messages defined as follows:

Protocol
: A collection of related services.

Service
: A set of transactions with a distinct DNS SRV prefix and HTTP well known service label.

Transaction
: A defined sequence of protocol messages supported by a service. Currently only request-response design pattern is supported.

Message
: A JSON document that corresponds to a request or response.

To build a service using the Site structure, we prepend add following declaration:

```
Protocol Sitefinder STFND

     Service Finder "_siteFinder._wks" "SiteFinder" Request Response
             Description
                     |Find sites for new donut stores.

     Message Request
             Struct Site WhereIAm
     Message Response
             Struct Site WhereAreDonuts
                     Multiple
```

We can now run Protogen to generate any of the following:

* Documentation in HTML

* Documentation in RFC2XML schema

* A C# client access library.

* A C# stub service library.

* A C header file describing corresponding C structures and data tables to enable serialization/deserialization.

Support for partial classes makes C# a particularly attractive target language for code generation as it allows classes produced by generated code to be conveniently extended. Support for other modern languages aligned with the Java/.NET data model requires only straightforward modification of the code generator.

While the C# generator is optimized for development of protocols and production code, the generator for C is intended for developing production code after the protocol architecture is largely static. The generator is intentionally biased towards flexibility rather than functionality since a modern programer using C is most likely to be doing so to build on a legacy code base. The ability to easily adapt the output of the generator to the existing coding style(s) is likely to be more highly valued than minimizing implementation effort.

2. Protocol Specification

2.1. Protocol

Top level specification of a protocol. The Protocol element contains two attributes and a list of entries as follows:

Namespace
: Namespace identifier for use in .NET and Java style programming environments

Prefix
Prefix for use in C style programming environments.

Entries
A list of [Service Transaction Message Structure Description Using] elements

2.2. Description

Describes the parent element. Multiple description elements may be specified in which case the first SHOULD be a standalone short description. The description element has one attribute:

Text
Text field data identified by use of the | prefix.

2.3. Service

A service is a named set of transactions within a protocol namespace.

At present, due to an implementation limitation, all request and response messages used in a service MUST inherit from a single message type. This is bogus and should be fixed.

The service element has the following attributes:

ID
The code identifier of the service

Discovery
The DNS service prefix of the service for use in SRV, NAPTR style discovery

WellKnown
The HTTP well known service prefix.

Request
The parent class for all request messages supported by the service.

Response
The parent class for all response messages supported by the service.

Entries
A list of [Description Status] entries

2.4. Transaction

Specifies a Request-Response transaction supported by a specified service.

At present transactions are specific to a service which is kind of bogus if multiple services were defined.

The Transaction element has the following attributes:

Service
: The identifier of the service

ID
: The identifier of the transaction

Request
: The request message which must not be an abstract type.

Response
: The response message returned for normal completion. An abstract type may be specified.

Entries
: A list of [Description Status] entries

2.5. Message

Specifies a protocol message. This is almost the same as a structure except that the name of a request message is a command to a server and the name of a response message identifies a response.

Id
: The message identifier

Entries
: A list of [Description Abstract Inherits Boolean Integer Binary Float Label Name String URI DateTime Struct Enum Status Authentication Format Decimal] entries

2.6. Structure

2.7. Status

This feature is not yet implemented, the idea being that status codes should be represented at both the HTTP layer and JSON layer so that appropriate handling can be specified at either.

2.8. Using

Specifies a message or structure defined in another schema.

3. Data Types

Protogen recognizes ten intrinsic data types. While this is considerably larger than the three intrinsic types supported in JSON, the additional expressive power allows the tools to do more work for the programmer. For example, distinguishing strings that represent date-time values from other strings allows the tool to perform the work of encoding/decoding these values.

The following table sumarizes the Protogen schema types and their (default) corresponding C#/C equivalents.

Schema	JSON	C#	C
Boolean	true | false	bool	bool
Float	number	double	double
Decimal	number	Int64	long long
Integer	number	Int64	int
Binary	string (base64 encoded)	byte[] Data	BinaryType
Label	string	string	StringType
Name	string	string	StringType
String	string	string	StringType
URI	string	string	StringType
DateTime	string	DateTime	struct tm

Every data type supports the following options:

Required
: The minimum number of occurrences is 1.

Multiple
: Multiple values may be specified.

Description
: Description of the element for use in code generation.

Deaful
Default value for the element if unspecified.

While the Protogen schema definition does include additional options for some data types (e.g. LengthBits, LengthFixed) these are only used in the TLS encoding generator and are ignored when JSON encoding is being used.

3.1. Abstract

Messages and structures may be marked Abstract which means that they may be used as base classes for inheritance from other messages or structures but cannot appear on the wire.

3.2. Inherits

Specifies that a message or structure inherits from another message or structure.

Note that inheritance relationships are represented in the generated code for languages that support inheritance (e.g. C#) and flattened out in languages that do not (e.g. C).

3.3. Null Values

No distinction is made between a value that is not present and a value that is present with the value null. Thus the following JSON documents are considered to specify the same object.

```
{ "Value": 1 }

{ "Value": 1,
  "Optional": null }
```

An entry that has the 'Required' option set MUST always be specified even if the value is null.

3.4. Lists

No distinction is made between a list that is not present, a list with the null value and an empty list. Thus the following encodings desribe the same object:

```
{ "Value": 1 }

{ "Value": 1,
  "List": null }

{ "Value": 1,
  "List": [] }
```

To simplify scripting language implementation an entry that has the 'Multiple' option MUST be encoded as a list.

3.5. Decimal

The decimal encoding provides an alternative to use of floating point to represent decimal fractions.

Since 10 is not a power of 2, conversion between decimal and binary fractions is inexact and using Real32 or Real64 values for this purpose introduces an unnecessary loss of precision.

Since modern programming languages lack support for a Decimal intrinsic type, this is implemented by mapping the datum to a 64 bit integer with an offset of 1,000,000,000. This approach allows for numbers up to 9,223,372 to be represented with nine digit precision.

3.6. DateTime

Date Time Values are encoded as strings in IETF format.

3.7. Binary

Binary values are encoded using BASE64URL encoding.

4. Further Work

5. Acnowledgements

6. References

6.1. Normative References

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

philliph@comodo.com