C++ Wrapper for XML Configuration

OpenDNSSEC Enforcer

René Post, rene@xpt.nl

January 11, 2011

Contents

1 Overview

The enforcer needs a way to read existing kasp.xml and conf.xml configurations and act on them. The preferred way to do this is to create wrapper classes to provide type save access to that information.

A total of 2 days were used to create a proof of concept for this using Google's protocol buffers.

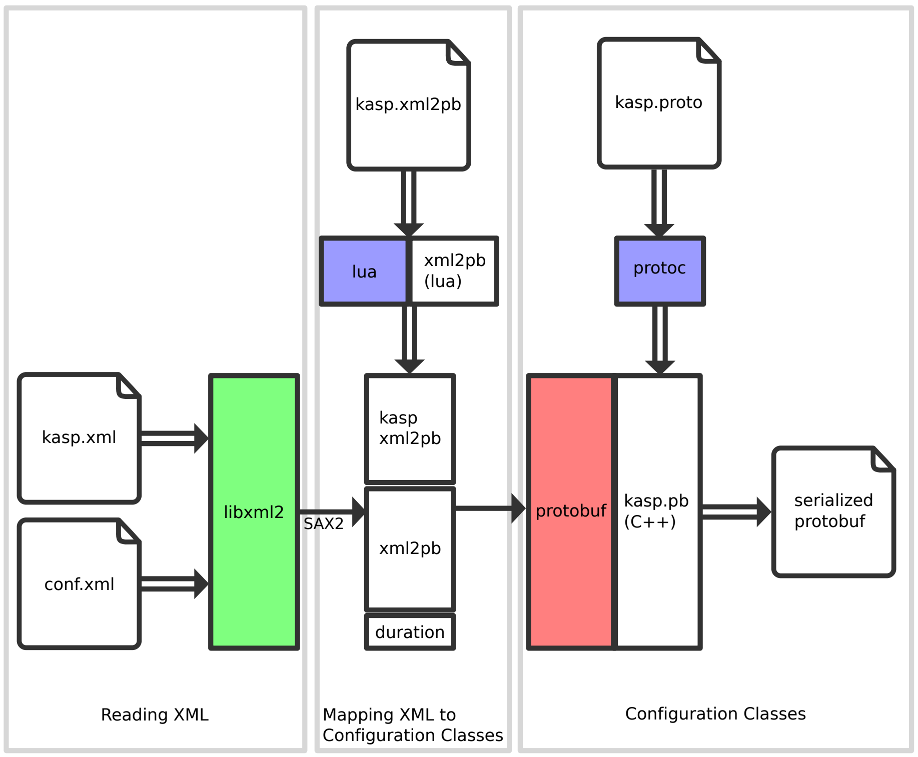

Figure 1: XML to Protocol Buffer

The colours in the diagram have the follwing meaning.

green: This is an existing dependency of OpenDNSSEC.

blue: This is not a current dependency of OpenDNSSEC. The module is needed in order to generate code from a specification for the enforcer.

red: This is not a current dependency of OpenDNSSEC. The module is needed in order to build the enforcer.

2 Reading XML

XML can be processed in two ways. The first way is by reading in a complete XML document and then process the document object model (DOM). The second way is to process the XML document while it is being read and just act on whatever elements and text are being served up by the parser (SAX2).

Because it is faster and uses less memory we use SAX2 for xml processing. Configurations don't profit that much from this because the files are not that big to begin with, but for processing a huge zonelist xml file this can be a big time and space saver.

2.1 libxml2

libxml2 is an existing dependency of OpenDNSSEC. For reading in the xml configurations, the streaming SAX2 API is used.

2.2 kasp.xml

The kasp.xml file as it is currently being used in OpenDNSSEC.

2.3 conf.xml

The conf.xml file as it is currently being used in OpenDNSSEC.

3 Mapping XML to C++ Classes

The configuration data wrapper classes are easier to use if they do not have the same deeply nested structure as the XML configuration data. To allow for this, a mapping layer has been created that can read XML element values into the C++ wrapper classes using reflection.

The mapping layer can be kept very small (a few hunderd LOCs) because it uses reflection to reason about the configuration wrapper C++ classes to which the XML data needs to be stored.

3.1 kasp.xml2pb

This contains a simple mapping from xml elements to configuration wrapper items. It is meant to be human readable and maintainable.

3.2 lua and xml2pb.lua

Lua is scripting language that is only required to generate code. xml2pb.lua is a tiny script that generates the kasp.xml2pb C++ mapping code.

3.3 xml2pb

This C++ code will use reflection to read XML elements it gets via SAX2 to the correct values in the protocol buffer.

3.4 duration

The duration code from the OpenDNSSEC signer utils. Used to map date time values in XML to a numerical value in seconds.

4 Configuration Classes

The configuration wrapper classes have been generated with (BSD licensed) Google protocol buffers. Because the generated wrappers depend on a run-time support library, we need to decide whether we want to allow our code to depend on this library or if we want to replace it with something we write ourselves.

The protocol buffers library has great support for specifying nested structured data in a type safe way. Protocol buffers have other interesting properties that make them very usefull for storing xml configurations.

simple specification: A small but powerful specification language has been defined that allows specfcation of nested structures efficiently. Every field in a structure has a unique numerical tag that identifies that field. Fields can be required, optional or repeated and defaults for fields can be specified.

wrapper generation: Specifications are transformed into C++ wrapper classes allowing type safe access.

structure versioning: By following some rules the protocol buffers support modifications to the structure including deprecation of fields and introduction of new fields.

efficient serialization: By using defaults value and optional fields the serialized protocol buffer can be very small. Also because it is a binary format reading a protocol buffer back is very fast.

reflection: Protocol buffers can be generated with reflection. This will allows C++ code to reason investigate the fields in the strucures. This makes writing generic code that transforms e.g. an XML file to protocol buffers very easy.

4.1 kasp.proto

This file contains the specification of the C++ wrappers.

4.2 protoc

This is the protocol buffer compiler used to generate the C++ classes.

4.3 kasp.pb (C++)

This is the resulting C++ generated from the proto file specification.

4.4 protobuf

The protocol buffer run-time implements the base functionality used by the generated C++ code.

4.5 Serialized Protobuf

This is the binary representation of a protocol buffer. Every policy in the configuration can be stored as a blob value in a database table. This ensures that it will be present in a backup/copy of the database.

5 Implementation Notes

Protocol buffers are a good solution for representing the internal configuration data of the enforcer. But because the number of actual data types being used is limited and the nature of the data itself is not extremely complex, it is possible to rewrite the wrappers in direct C++ code.