libpackedobjects tutorial

Table of Contents

1 Introduction **1**
1.1 What is libpackedobjects? 1
1.2 Key features 1
1.3 Limitations 1

2 Installation **2**
2.1 Installing libpackedobjects 2
2.2 Further reading 2

3 Getting started **3**
3.1 Quick start 3
3.2 Writing a schema 3
3.2.1 API basics 3

4 Data types **5**
4.1 Simple types 5
4.2 Complex types 5

Index **6**

1 Introduction

1.1 What is libpackedobjects?

libpackedobjects is a C library which can be used to efficiently compress an XML DOM by using the information provided by a corresponding XML Schema. The level of compression achieved is very similar to EXI but unlike EXI, libpackedobjects is designed to be light-weight and simple to implement. Therefore libpackedobjects is suited to embedded systems and mobile devices. The tool is designed for writing network protocols which strive to minimise the amount of data communicated. In addition to compression all data is validated by the schema during the encode and decode process.

libpackedobjects is based on libxml2 and therefore should run on any system that libxml2 runs on.

1.2 Key features

- Very efficient encoding size
- Light-weight and fast
- Validates XML data on encode and decode
- Good choice of data types including the ability to apply range and size constraints
- Fully dynamic including the ability to change the protocol at runtime
- Simple API with two main function calls
- Highly portable - designed for embedded and mobile devices
- Simple subset of XML Schema required to create protocols

1.3 Limitations

libpackedobjects is not a general purpose document compression tool. It is intended to be used in an application that generates XML that you wish to communicate over a network. As such it provides a simple DOM-based API for encoding and decoding structured data. The compression technique used is based on applying knowledge of the data types specified in a schema to provide better performance over statistical compression techniques. Therefore, you must write a valid schema for your data. The style of schema required is based on a small subset of XML Schema. This schema serves the purpose of formalising the network protocol and provides validation. Thus we think having a schema is a good thing!

2 Installation

2.1 Installing libpackedobjects

To install from the latest source:

```
git clone git://gitorious.org/libpackedobjects/libpackedobjects.git
cd libpackedobjects
autoreconf -i
./configure
make
make check
sudo make install
```

2.2 Further reading

3 Getting started

3.1 Quick start

After compiling and running 'make check' you should find a binary called 'packedobjects' in your src directory. This is command-line tool built with libpackedobjects which you can use to test out encoding and decoding:

```
$ ./packedobjects --help
usage: packedobjects --schema <file> --in <file> --out <file>
```

To encode run:

```
$ ./packedobjects --schema foo.xsd --in foo.xml --out foo.po
```

To decode run:

```
$ ./packedobjects --schema foo.xsd --in foo.po --out foo.new.xml
```

3.2 Writing a schema

libpackedobjects uses a subset of XML Schema. This provides a focus similar to the concept of a Domain Specific Language. You are provided with suitable data types to be able to create functional network protocols. We will use the canonical “Hello World” example. We first write a schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="http://zedstar.org/xml/schema/packedobjectsDataTypes.xsd"/>
  <xs:element name="foo" type="string"/>
</xs:schema>
```

We then create the corresponding data:

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>Hello World!</foo>
```

This simple example only defines one data type which happens to be a string. However it provides a template for the basic structure of all schemas. You first must include the libpackedobjects data types and then specify a root element. In this case the root element is called 'name'.

3.2.1 API basics

There are only 4 main function calls:

```
packedobjectsContext *init_packedobjects(const char *schema_file);

char *packedobjects_encode(packedobjectsContext *pc, xmlDocPtr doc);

xmlDocPtr packedobjects_decode(packedobjectsContext *pc, char *pdu);

void free_packedobjects(packedobjectsContext *poCtxPtr);
```

You first must initalise the library using your XML Schema. Typical use would be one called to init_packedobjects at startup and then multiple calls to encode/decode based on your protocol. If during runtime your schema changed you must called the init function again with the new file. The library is designed to do preposing of the schema during the init function which then allows efficient encoding and decoding plus validation to take place. Therefore, it makes no sense to call init_packedobjects more than once if you do not plan on supporting dynamically changing protocols at runtime. If you do you will suffer a needless performance cost.

4 Data types

4.1 Simple types

4.2 Complex types

Index

A

API basics 3

C

Complex types 5

F

Further reading 2

I

Installing libpackedobjects 2

K

Key features 1

L

Limitations 1

Q

Quick start 3

S

Simple types 5

W

What is libpackedobjects 1
Writing a schema 3