

packedobjectsd tutorial

Table of Contents

1	Introduction	1
1.1	What is packedobjects?	1
1.2	Key features	1
1.3	Limitations	1
2	Installation	2
2.1	Installing packedobjects	2
2.2	Further reading	2
3	Getting started	3
3.1	Quick start	3
3.2	API basics	3

1 Introduction

1.1 What is packedobjectsd?

packedobjectsd is a light-weight XML messaging framework. It is created using multiple publisher and multiple subscriber model and built using packedobjects and ZeroMQ in C language. It is simple to implement and is suited to embedded systems and mobile devices. The library provides simple API to send and receive XML data to and from multiple nodes. The library connects all the nodes sending/receiving XML data using the same XML schema to one group. The data is compressed and validated by the schema during the send and receive process.

packedobjectsd is based on packedobjects, libxml2 and ZeroMQ and therefore should run on any system that libxml2 and ZeroMQ runs on.

1.2 Key features

- Light-weight and simple
- Validates XML data on send and receive
- Simple API with four main function calls
- Highly portable - designed for embedded and mobile devices
- Simple subset of XML Schema required to create protocols

1.3 Limitations

todo

2 Installation

2.1 Installing packedobjects

To install from the latest source:

```
git clone https://github.com/jnbagale/packedobjects.git
cd packedobjects
autoreconf -i
./configure
make
make check
sudo make install
```

2.2 Further reading

Packedobjects:
<http://packedobjects.org/pages/home.html>

ZeroMQ publisher subscriber:
<http://zguide.zeromq.org/page:all#Getting-the-Message-Out>

Libxml2:
<http://www.xmlsoft.org/examples/index.html>

3 Getting started

3.1 Quick start

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

```
$ ./packedobjectsdtest --help
usage: packedobjectsdtest --schema <file> --xml <file>
```

To send and receive run:

```
$ ./packedobjectsdtest --schema foo.xsd --xml foo.xml
```

3.2 API basics

There are 2 types of API implementations available for data sharing which are made available by adding `#include <packedobjectsd/packedobjectsd.h>` to your code. Both API implementations share same initialisation and free function calls.

```
packedobjectsdObject *init_packedobjectsd(const char *schema_file, int node_type, int options)
void free_packedobjectsd(packedobjectsdObject *pod_obj);
```

Options can be:

- NO_COMPRESSION
- NO_HEARTBEAT (* not yet fully implemented)

```
# The Compression and Heartbeat options will be enabled by default if 0 is specified for options
and the node_type depends on the API type
```

1. Simple Pub/Sub API Node types PUBLISHER SUBSCRIBER PUBSUB

```
int packedobjectsd_send(packedobjectsdObject *pod_obj, xmlDocPtr doc);

xmlDocPtr packedobjectsd_receive(packedobjectsdObject *pod_obj);
```

PUBLISHER nodes can only use the send function, SUBSCRIBER nodes can only use the receive function whereas PUBSUB node (Publisher/Subscriber combo) can use both. The receive function is blocking so it will wait until it receives message from the publisher. It can be run in a loop to receive continuous messages and threads can be used to allow the program to perform other tasks in parallel while waiting for the messages.

You first must initialise the library using your XML Schema. Typical use would be one called to `init_packedobjectsd` at startup and then multiple calls to send/receive data. The interface to the `packedobjectsd_send` function requires a `libxml2` doc type and returns the number of bytes sent. The `packedobjectsd_receive` function returns a `libxml2` doc type.

1. Searcher/Responder API Node types SEARCHER RESPONDER SEARES

```

int packedobjectsd_send_search(packedobjectsdObject *pod_obj, xmlDocPtr doc); [SEARCHER]

xmlDocPtr packedobjectsd_receive_search(packedobjectsdObject *pod_obj); [RESPONDER]

int packedobjectsd_send_response(packedobjectsdObject *pod_obj, xmlDocPtr doc); [RESPONDER]■
xmlDocPtr packedobjectsd_receive_response(packedobjectsdObject *pod_obj); [SEARCHER]

```

The functions will be called in specific sequence due to the nature of the communication between the SEARCHER and the RESPONDER nodes. The responder nodes consist of database to be searched by searcher nodes and thus needs to be initialised/run before the searcher nodes. Once the responder node is ready, searcher nodes can send their search query as XML doc using the send_search function. The responder node will retrieve that using the receive_search which should run in a loop if multiple search queries are expected. The receive_search function provides the search query and the searcher's unique id which will be used later to send response. The responder then sends the response XML using send_response function to the specific searcher by using the unique id. The searcher then receives the response by using receive_response function.

A very simple program demonstrating the simple Pub/Sub API is as follows:

```

#include <stdio.h>
#include <unistd.h>      /* for sleep() */
#include <packedobjectsd/packedobjectsd.h>

#define XML_DATA "helloworld.xml"
#define XML_SCHEMA "helloworld.xsd"

int main ()
{
    packedobjectsdObject *pod_obj = NULL;
    xmlDocPtr doc_sent = NULL;
    xmlDocPtr doc_received = NULL;

    /////////////// Initialising ///////////////////

    /* Initialise packedobjectsd */

    if((pod_obj = init_packedobjectsd(XML_SCHEMA, PUBSUB, 0)) == NULL) {
        printf("failed to initialise libpackedobjectsd");
        exit(1);
    }

    sleep(1); /* Allow broker to start if it's not already running */

    /////////////// Sending XML data ///////////////////

    /* create an XML DOM */
    if((doc_sent = xml_new_doc(XML_DATA)) == NULL) {
        printf("did not find .xml file");
        exit(1);
    }

    /* send the XML DOM */

```

```

if(packedobjectsd_send(pod_obj, doc_sent) == -1){
    printf("failed to send with error %s", pod_strerror(pod_obj->error_code));
    exit(1);
}

printf("size of the original xml: %d bytes\n", xml_doc_size(doc_sent));
printf("size after the encoding: %d bytes\n", pod_obj->bytes_sent);
/* free the XML DOM */
xmlFreeDoc(doc_sent);

////////// Receiving XML data //////////

if((doc_received = packedobjectsd_receive(pod_obj)) == NULL) {
    printf("failed to receive with error %s", pod_strerror(pod_obj->error_code));
    exit(1);
}

printf("size before the decoding: %d bytes\n", pod_obj->bytes_received);
printf("size of the decoded xml: %d bytes\n", xml_doc_size(doc_received));

/* output the DOM for checking */
xml_dump_doc(doc_received);
/* free the XML DOM */
xmlFreeDoc(doc_received);

////////// Freeing //////////

/* free memory created by packedobjectsd */
free_packedobjectsd(pod_obj);

return 0;
}

```

helloworld.xsd

```

<?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>

```

helloworld.xml

```

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

```

If during runtime your schema changed you must call the init function again with the new file. The library is designed to communicate to the server during the init function which gives it back the network address and port numbers to send or receive the data. Therefore, do not call `init_packedobjectsd` more than once if you do not plan on supporting dynamically changing protocols at runtime.

To build an application with the software you must link with the library. Using `autoconf` you can add `PKG_CHECK_MODULES([LIBPACKEDOBJECTSD], [libpackedobjectsd])`

to your `configure.ac` file and then use the variables `$(LIBPACKEDOBJECTSD_CFLAGS)` and `$(LIBPACKEDOBJECTSD_LIBS)` in your `Makefile.am` file.

A

API basics 3

F

Further reading 2

I

Installing `packedobjectsd` 2

K

Key features 1

L

Limitations 1

Q

Quick start 3

W

What is `packedobjectsd` 1