packedobjectsd tutorial

packedobjectsd - a light weight XML messaging framework

Table of Contents

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 packedobjectsd

To install from the latest source:

```
git clone https://github.com/jnbagale/packedobjectsd.git
cd packedobjectsd
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 avaliable for data sharing which are made available by adding `#include <packedobjectsd/packedobjectsd.h>` to your code. Both API implementations share same initilisation 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:

```
/************ POD PUBLISHER **************/


#include <stdio.h>
#include <stdlib.h>
#include <packedobjectsd/packedobjectsd.h>

int main (int argc, char *argv [])
{
  ///////// Declarations //////////////
  packedobjectsdObject *pod_obj = NULL;
  const char *xml_file = "helloworld.xml";
  const char *schema_file = "helloworld.xsd";

  xmlDocPtr doc_sent = NULL;
  int loop = 1;

  //////// Initialise packedobjectsd with schema file and specify node type  //////////
  if((pod_obj = init_packedobjectsd(schema_file, PUBLISHER, 0)) == NULL) {
    printf("failed to init packedobjectsd");
    exit(EXIT_FAILURE);
  }

  ///////// SENDING SIMPLE XML OVER SIMPLE PUB SUB CONNECTION ///////
  if((doc_sent = xml_new_doc(xml_file)) == NULL) {
    printf("failed to init xml document");
    exit(EXIT_FAILURE);
  }

  /* send a normal pub message */
  printf("Broadcasting message on a pub socket\n");
  if(packedobjectsd_send(pod_obj, doc_sent) == -1){
    printf("%s", pod_strerror(pod_obj->error_code));
```

```
  }
  else {
    printf("message sent successfully\n");
    xml_dump_doc(doc_sent);
  }

  //////// freeing memory ///////////
  free_packedobjectsd(pod_obj);
  xmlFreeDoc(doc_sent);

  return EXIT_SUCCESS;
}



/************ POD SUBSCRIBER **************/


#include <stdio.h>
#include <stdlib.h>
#include <packedobjectsd/packedobjectsd.h>

int main (int argc, char *argv [])
{
  ///////// Declarations /////////////
  packedobjectsdObject *pod_obj = NULL;
  const char *schema_file = "helloworld.xsd";

  xmlDocPtr doc_received = NULL;
  int loop = 100;

  //////// Initialise packedobjectsd with schema file and specify node type  //////////
  if((pod_obj = init_packedobjectsd(schema_file, SUBSCRIBER, 0)) == NULL) {
    printf("failed to init packedobjectsd");
    exit(EXIT_FAILURE);
  }

  ///////// RECEIVING SIMPLE XML OVER SIMPLE PUB SUB CONNECTION ///////

  printf("Receiving broadcast messages on a sub socket\n");

  while(loop) {
    if((doc_received = packedobjectsd_receive(pod_obj)) == NULL){
      printf("%s", pod_strerror(pod_obj->error_code));
    }
    else {
      printf("message received successfully\n");
      xml_dump_doc(doc_received);
    }
    loop--;
  }

  //////// freeing memory ///////////
  free_packedobjectsd(pod_obj);
  xmlFreeDoc(doc_received);

  return EXIT_SUCCESS;
}
```

A very simple program demonstrating the SEARCHER/RESPONDER API is as follows:

```
/************ POD SEARCHER **************/

#include <stdio.h>
#include <stdlib.h>
#include <packedobjectsd/packedobjectsd.h>

int main (int argc, char *argv [])
{
  ///////// Declarations /////////////
  int ret;
  packedobjectsdObject *pod_obj = NULL;
  const char *xml_file = "helloworld.xml";
  const char *schema_file = "helloworld.xsd";

  xmlDocPtr doc_search = NULL;
  xmlDocPtr doc_response_received = NULL;
  int loop = 1;

  //////// Initialise packedobjectsd with schema file and specify node type  //////////
  if((pod_obj = init_packedobjectsd(schema_file, SEARCHER, 0)) == NULL) {
    printf("failed to init packedobjectsd");
    exit(EXIT_FAILURE);
  }

  ///////// SENDING SIMPLE XML OVER SIMPLE SEARCHER CONNECTION ///////
  printf("Sending search messages on a searcher socket\n");

  if((doc_search = xml_new_doc(xml_file)) == NULL) {
    printf("did not find .xml file");
  }

  /* send a search message */
  if((ret = packedobjectsd_send_search(pod_obj, doc_search)) == -1){
    printf("%s", pod_strerror(pod_obj->error_code));
  }
  else {
    printf("search message sent successfully\n");
  }

  /////// Wait for response messages from Respnders ////////
  printf("Waiting for response messages on a searcher socket\n");

  /* USE LOOP HERE TO RECEIVE MESSAGES CONTINUOUSLY */

  /* receive a response message */
  if((doc_response_received = packedobjectsd_receive_response(pod_obj)) == NULL) {
    printf("%s", pod_strerror(pod_obj->error_code));
  }
  else {
    printf("response message received\n");
    xml_dump_doc(doc_response_received);
  }

  //////// freeing memory ////////////
  free_packedobjectsd(pod_obj);
```

```
  xmlFreeDoc(doc_search);

  return EXIT_SUCCESS;
}

/************ POD RESPONDER **************/

#include <stdio.h>
#include <stdlib.h>
#include <packedobjectsd/packedobjectsd.h>

int main (int argc, char *argv [])
{
  ///////// Declarations //////////////
  int ret;
  packedobjectsdObject *pod_obj = NULL;
  const char *schema_file = "helloworld.xsd";
  const char *xml_file = "helloworld.xml";

  xmlDocPtr doc_response = NULL;
  xmlDocPtr doc_search_received = NULL;
  int loop = 100;

  //////// Initialise packedobjectsd with schema file and specify node type  //////////
  if((pod_obj = init_packedobjectsd(schema_file, RESPONDER, 0)) == NULL) {
    printf("failed to init packedobjectsd");
    exit(EXIT_FAILURE);
  }

  ///////// RECEIVING SIMPLE XML OVER SIMPLE RESPONDER CONNECTION ///////

  printf("Receiving search messages on a responder socket\n");

  while(loop) {
     /* receive a search message */
    if((doc_search_received = packedobjectsd_receive_search(pod_obj)) == NULL) {
      printf("%s", pod_strerror(pod_obj->error_code));
    }
    else {
      printf("search message received successfully\n");
      xml_dump_doc(doc_search_received);
    }

    /* PERFORM QUERY PROCESSING HERE! */

    /* send a response message */
    if((doc_response = xml_new_doc(xml_file)) == NULL) {
      printf("did not find .xml file");
    }

    if((ret = packedobjectsd_send_response(pod_obj, doc_response)) == -1){
      printf("%s", pod_strerror(pod_obj->error_code));
    }
    else {
      printf("response message sent successfully\n");
    }

    usleep(1000); // sleep for 1ms
```

```
    loop--;
  }

  //////// freeing memory ///////////
  free_packedobjectsd(pod_obj);
  xmlFreeDoc(doc_search_received);
  xmlFreeDoc(doc_response);

  return EXIT_SUCCESS;
}
```

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>
```

An advanced searcher/responder example is available in the examples folder of the project repository(link below). This example also demonstrates XML schema and some XML database and processing.

`https://github.com/jnbagale/packedobjectsd/tree/master/examples/video-search`

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.