DATA COMMUNICATION – ASSIGNMENT 2

Hamza Bhatti (21223241)

Contents

Introduction 2
Schema 2
XML 4
Using Packedobjects 5
    Compressing With Packedobjects 5
    Analysing Packedobjects Compression 5
    Comparing Sizes of Files 8
Main.c 9
Client and Server 11

Introduction

For this assignment, the use of the library “Libpackedobjects” is used. Its use along with created XML, Schema files are documented. Along with this, comparing its use with file types will also be identified.

Schema

A schema file is used to outline the data types and the structure of the potential XML that will be created. Once the schema has been created, using libpackedobjects, the XML will be validated against the schema and compressed. This will ensure that the structure, node names and the information included in the XML, all adhere to the schema.

The XML schema created was based around a 5 person maximum basketball team. In the schema, data types where declared along with the structure that the XML would follow. The schema can be seen below, named basketball.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--Creating the schema file that will be used to outline -->
<!-- the structure of the XML that will be coded in C-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include
schemaLocation="/usr/local/share/libpackedobjects/packedobjectsDataTypes.xsd"/>

  <!-- Defining the data types that will be used -->

  <!-- nameType to be used for Team and Name -->
  <xs:simpleType name="nameType">
    <xs:restriction base="string">
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- conferenceType to be used to identify conference (East or West) -->
  <xs:simpleType name="conferenceType">
    <xs:restriction base="enumerated">
      <xs:enumeration value="East"/>
      <xs:enumeration value="West"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- ageType to be used for the age of the player -->
  <xs:simpleType name="ageType">
    <xs:restriction base="integer">
      <xs:minInclusive value="19"/>
      <xs:maxInclusive value="39"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- numberType to used to identify the number assigned to the player -->
  <xs:simpleType name="numberType">
    <xs:restriction base="integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="99"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- positionType to identify position of the player -->
  <xs:simpleType name="positionType">
    <xs:restriction base="enumerated">
```

```
      <xs:enumeration value="Point Guard"/>
      <xs:enumeration value="Shooting Guard"/>
      <xs:enumeration value="Small Forward"/>
      <xs:enumeration value="Power Forward"/>
      <xs:enumeration value="Centre"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- skillType to identify the players unique skill -->
  <xs:simpleType name="skillType">
    <xs:restriction base="enumerated">
      <xs:enumeration value="Passing"/>
      <xs:enumeration value="Shooting"/>
      <xs:enumeration value="Rebounding"/>
      <xs:enumeration value="Free throw"/>
      <xs:enumeration value="Dunking"/>
      <xs:enumeration value="Dribbling"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Outlining the structure of the XML file with the above -->
  <!-- data types -->
  <xs:element name="BasketballTeam">
    <xs:complexType>
      <xs:sequence>
         <xs:element name="Team" type="nameType"/>
         <xs:element name="Conference" type="conferenceType"/>
         <xs:element name="Players">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="Player" maxOccurs="5">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="Name" type="nameType"/>
                      <xs:element name="Age" type="ageType"/>
                      <xs:element name="Number" type="numberType" />
                      <xs:element name="Position" type="positionType"/>
                      <xs:element name="Skill" type="skillType"/>
                    </xs:sequence>
                  </xs:complexType>
               </xs:element>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML

The following XML follows the same structure that was set in the basketball.xsd schema. The Team and Name would only accept a certain amount of characters and the remaining nodes would only allow inputs within set values and selections. Only up to 5 players would be accepted as stated in the schema file. The XML file, basketball.xml, can be seen below:

```
<?xml version="1.0" encoding="UTF-8"?>

<BasketballTeam>
  <Team>Miami Heat</Team>
  <Conference>East</Conference>
  <Players>
    <Player>
      <Name>John Doe</Name>
      <Age>28</Age>
      <Number>7</Number>
      <Position>Point Guard</Position>
      <Skill>Passing</Skill>
    </Player>
    <Player>
      <Name>Fizz Buzz</Name>
      <Age>22</Age>
      <Number>4</Number>
      <Position>Shooting Guard</Position>
      <Skill>Shooting</Skill>
    </Player>
    <Player>
      <Name>Foo Bar</Name>
      <Age>20</Age>
      <Number>6</Number>
      <Position>Small Forward</Position>
      <Skill>Rebounding</Skill>
    </Player>
    <Player>
      <Name>Foo-Bar</Name>
      <Age>20</Age>
      <Number>6</Number>
      <Position>Power Forward</Position>
      <Skill>Rebounding</Skill>
    </Player>
    <Player>
      <Name>Gen Eric</Name>
      <Age>25</Age>
      <Number>44</Number>
      <Position>Centre</Position>
      <Skill>Dunking</Skill>
    </Player>
  </Players>
</BasketballTeam>
```

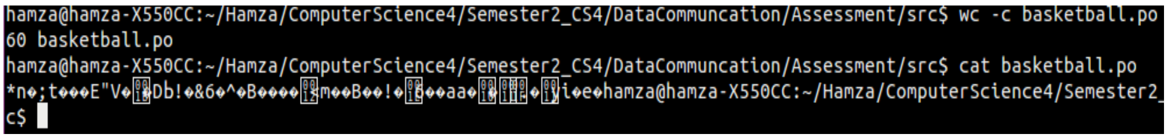
Using Packedobjects

Compressing With Packedobjects

When the XML and schema files where created, libpackedobjects was used to compress the XML file. This is very useful for data transfer for mobile devices and safer sending of data over a network.

To compress the XML the command:
packedobjects --schema basketball.xsd --in basketball.xml --out backetball.po.

In the image below the amount of bytes can be seen along with only a small line of encrypted code.



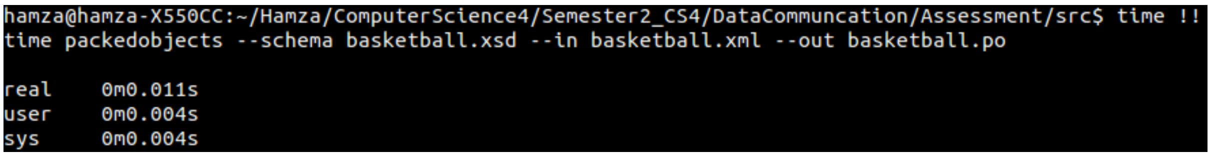
```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ wc -c basketball.po
60 basketball.po
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ cat basketball.po
*n�;t���E"V�[illegible]Db!�&6�^�B����[illegible]m��B��!�[illegible]��aa�[illegible][illegible]�[illegible]i�e�hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_
c$ █
```

Analysing Packedobjects Compression

It is useful to analyse the time that the XML file took to encode and decode. The images below show the time of encode and decode run of the packedobjects commands.

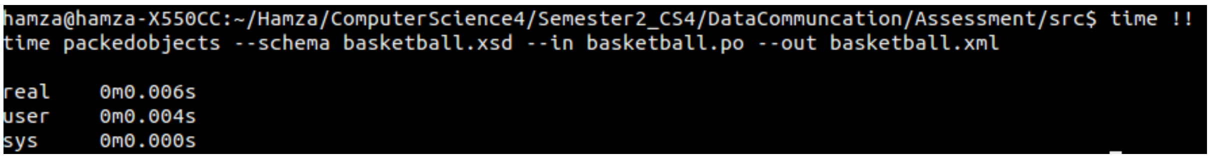
In the images, real, user and sys are shown. Real identifies the time from start to finish of the command call. User is the amount of time the CPU spent to execute the process. Sys is the time the CPU spent within the kernel.

Encode:



```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ time !!
time packedobjects --schema basketball.xsd --in basketball.xml --out basketball.po

real    0m0.011s
user    0m0.004s
sys     0m0.004s
```

Decode:

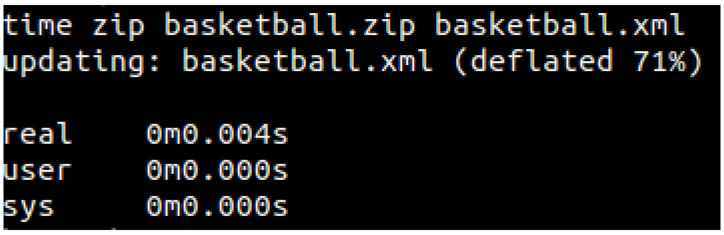


```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ time !!
time packedobjects --schema basketball.xsd --in basketball.po --out basketball.xml

real    0m0.006s
user    0m0.004s
sys     0m0.000s
```

As can be seen in the images, the decode run took less real time to complete the process than the encode run. User was the same in both encode and decode, but the CPU spent more time in the kernel for the encode run while it spent no time for the decoding.

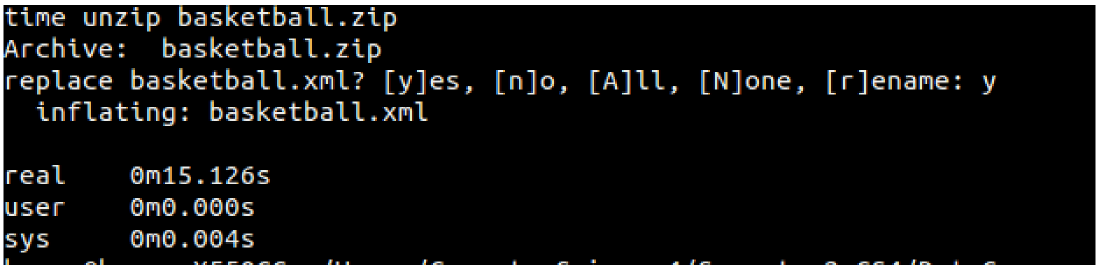
For comparison, a zip file was created and the time of the process was identified also. The images of the process can be seen below:

Encode:



```
time zip basketball.zip basketball.xml
updating: basketball.xml (deflated 71%)

real    0m0.004s
user    0m0.000s
sys     0m0.000s
```

Decode:



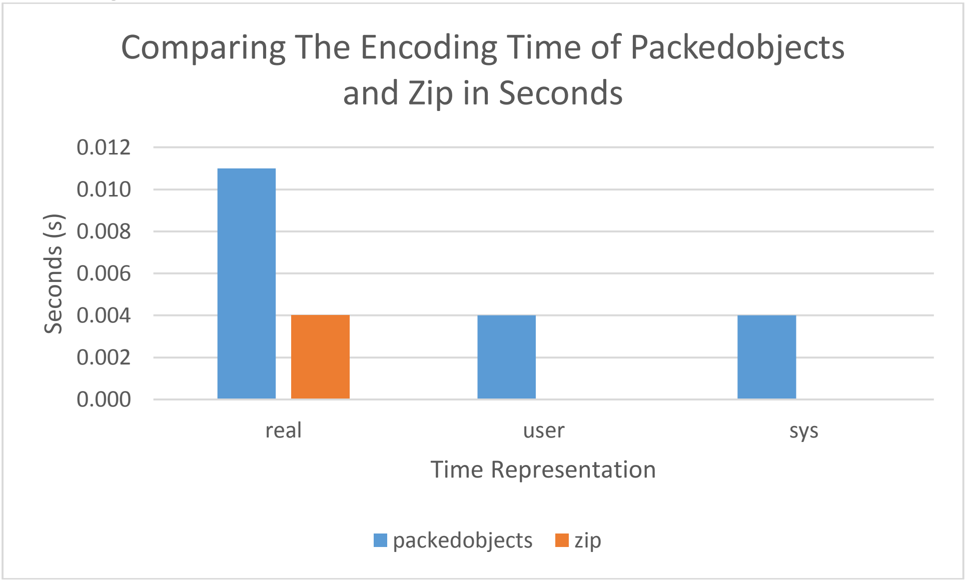
```
time unzip basketball.zip
Archive:  basketball.zip
replace basketball.xml? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: basketball.xml

real    0m15.126s
user    0m0.000s
sys     0m0.004s
```

The decoding of the zip file took a substantially longer time to unzip in real time. There was no time spent for the CPU to process the command in user time. There was a longer time spent in the kernel for in the decode run than in encode.

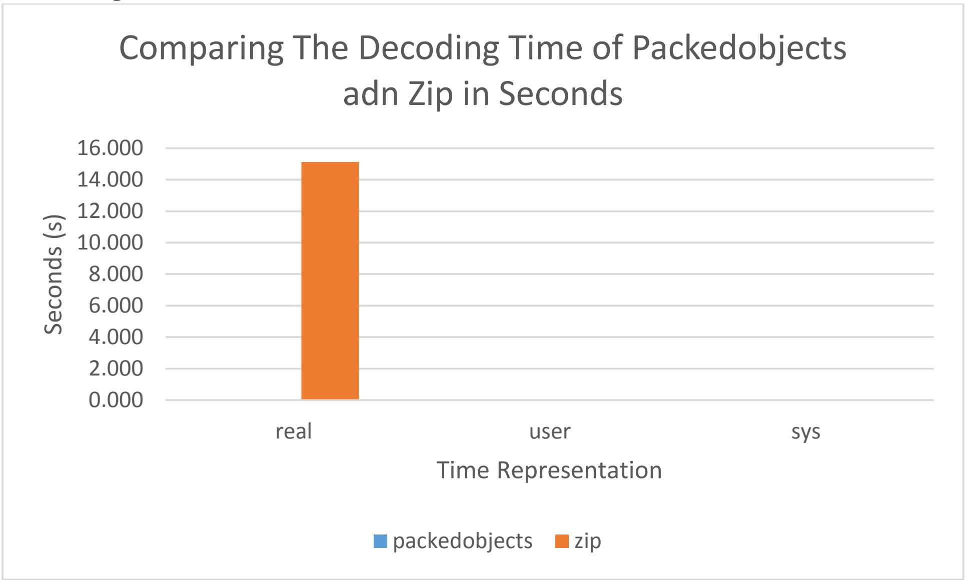
It can be seen from the images that packedobjects is vastly more efficient for decoding compressed file than that of the zip commands, baring in mind that this process must have the schema present in the directory.

Below is a graphical form for the encoding of the XML file and the decoding of the XML files.

Encoding:

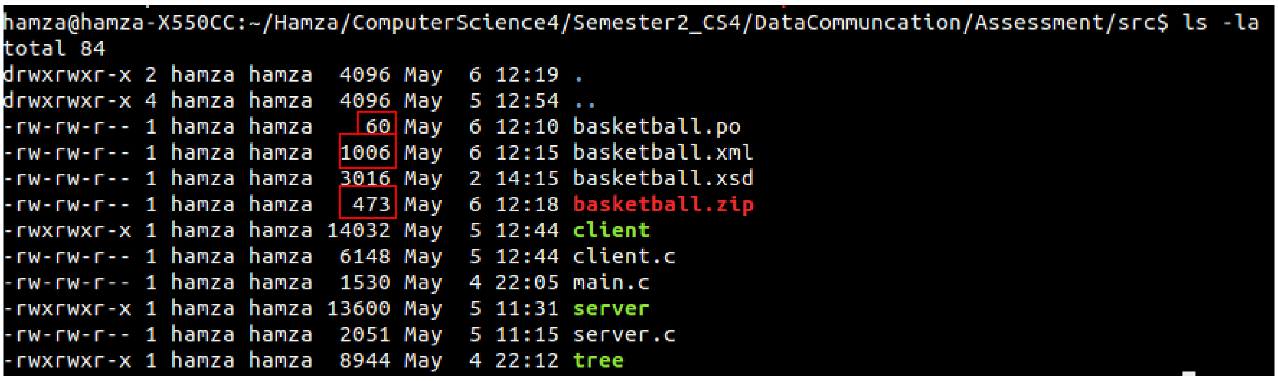


Decoding:


Comparing The Decoding Time of Packedobjects adn Zip in Seconds
16.000
14.000
12.000
10.000
8.000
6.000
4.000
2.000
0.000
Seconds (s)
real
user
sys
Time Representation
packedobjects
zip

Comparing Sizes of Files

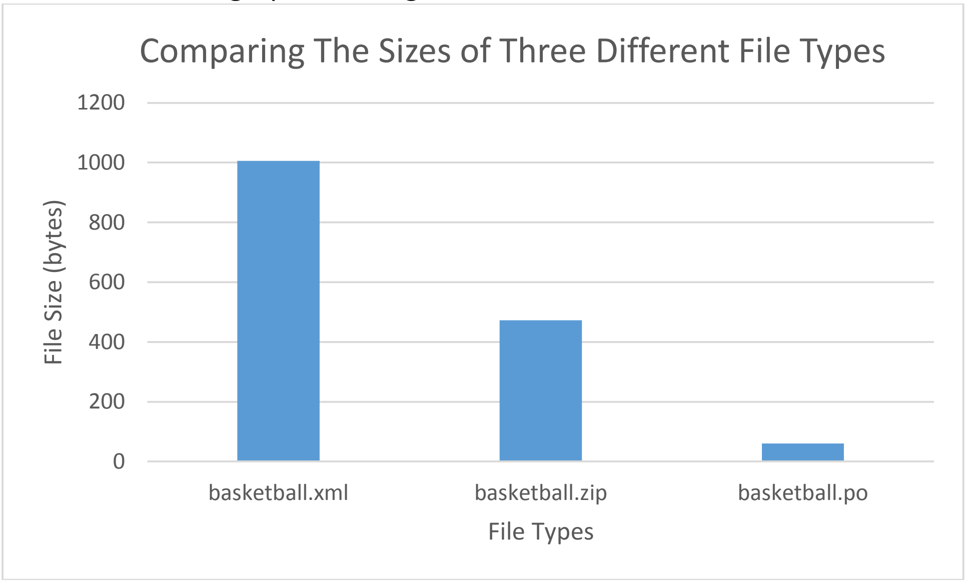
It is useful to note the sizes of the files that have been created. This would include the basketball.xml, basketball.po and the basketball.zip. Doing this will identify the benefit of using packedobjects for file compression. Below is the list of files, along with their highlighted sizes in bytes.



```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ ls -la
total 84
drwxrwxr-x 2 hamza hamza  4096 May  6 12:19 .
drwxrwxr-x 4 hamza hamza  4096 May  5 12:54 ..
-rw-rw-r-- 1 hamza hamza    60 May  6 12:10 basketball.po
-rw-rw-r-- 1 hamza hamza  1006 May  6 12:15 basketball.xml
-rw-rw-r-- 1 hamza hamza  3016 May  2 14:15 basketball.xsd
-rw-rw-r-- 1 hamza hamza   473 May  6 12:18 basketball.zip
-rwxrwxr-x 1 hamza hamza 14032 May  5 12:44 client
-rw-rw-r-- 1 hamza hamza  6148 May  5 12:44 client.c
-rw-rw-r-- 1 hamza hamza  1530 May  4 22:05 main.c
-rwxrwxr-x 1 hamza hamza 13600 May  5 11:31 server
-rw-rw-r-- 1 hamza hamza  2051 May  5 11:15 server.c
-rwxrwxr-x 1 hamza hamza  8944 May  4 22:12 tree
```

As can be seen, the original XML file has a size of 1006 bytes. Transferring this file over a network would take a long time. Looking next at the basketball.zip file, it has reduced in size to 473 bytes which is slightly more efficient than sending the whole basketball.xml file over a network. But looking at the basketball.po file, which was created with packedobjects, the file size has reduced a great deal, to only 60 bytes. This identifies that packedobjects is more efficient to use when compressing files for sending over a network.

Below is another graph showing the difference in size of these files.



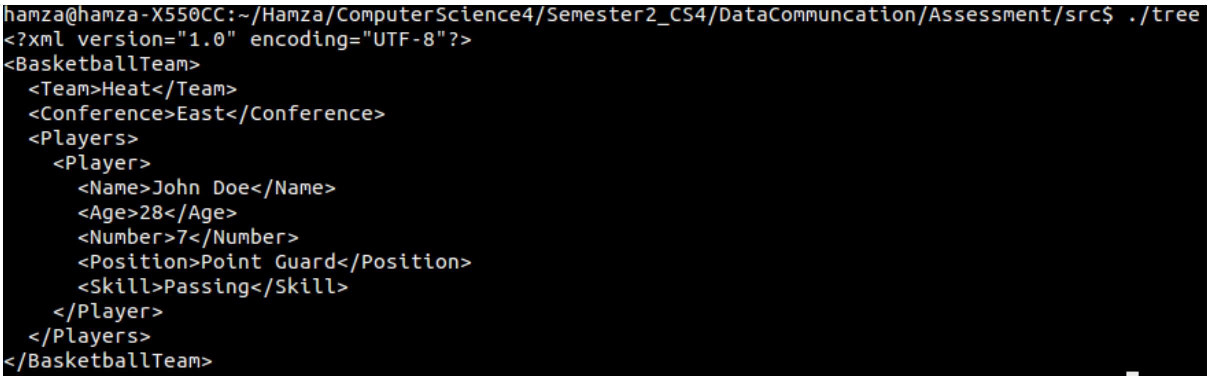
Main.c

Later in this documentation, the development of a client file will be demonstrated. But to allow the client file to be easy to create, a main.c file was created. This main.c file was written to create an XML file. This file was developed with the basketball.xsd schema in mind. The code, with comments, can be seen below:

```
#include <stdio.h>
#include <libxml/parser.h>
#include <libxml/tree.h>

int main (int argc, char **argv)
{
  // XML details
  xmlDocPtr doc = NULL;
  xmlNodePtr root_node = NULL , node = NULL , node2 = NULL;
  doc = xmlNewDoc(BAD_CAST "1.0");

  // Creating xml tree

  // Creating a root node (BasketballTeam) and set it for the whole doc
  root_node = xmlNewNode(NULL, BAD_CAST "BasketballTeam");
  xmlDocSetRootElement(doc, root_node);

  // Creating children for the root node

  // Making Team node, child of root_node (BasketballTeam)
  xmlNewChild(root_node, NULL, BAD_CAST "Team" , BAD_CAST "Heat");

  // Making Conference node, child of root_node (BasketballTeam)
  xmlNewChild(root_node, NULL, BAD_CAST "Conference" , BAD_CAST "East");

  // Creating a new parent node (Players)
  node = xmlNewChild(root_node, NULL, BAD_CAST "Players", NULL);

  // Creating a new subtree (Player)
  node2 = xmlNewChild(node, NULL, BAD_CAST "Player", NULL);

  // Making children for node2 (Player)

  // Node Name
  xmlNewChild(node2, NULL, BAD_CAST "Name", BAD_CAST "John Doe");

  // Node Age
  xmlNewChild(node2, NULL, BAD_CAST "Age", BAD_CAST "28");

  // Node Number
  xmlNewChild(node2, NULL, BAD_CAST "Number", BAD_CAST "7");

  // Node Position
  xmlNewChild(node2, NULL, BAD_CAST "Position", BAD_CAST "Point Guard");

  // Node Skill
  xmlNewChild(node2, NULL, BAD_CAST "Skill", BAD_CAST "Passing");

  // More XML details
  xmlSaveFormatFileEnc("-", doc, "UTF-8", 1);
  xmlFreeDoc(doc);
  xmlCleanupParser();
  return(0);
```

Main.c was then compiled with the command:
gcc -Wall main.c -o tree `pkg-config libpackedobjects --cflags --libs`

When the executable file, tree, was run, the output was the XML tree structure similar to basketball.xml shown earlier. The output of the main.c file can be seen below:



```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ ./tree
<?xml version="1.0" encoding="UTF-8"?>
<BasketballTeam>
  <Team>Heat</Team>
  <Conference>East</Conference>
  <Players>
    <Player>
      <Name>John Doe</Name>
      <Age>28</Age>
      <Number>7</Number>
      <Position>Point Guard</Position>
      <Skill>Passing</Skill>
    </Player>
  </Players>
</BasketballTeam>
```

Client and Server

The purpose of the client and server c files was to create a XML in the client.c and be received by the server.c. This XML, when produced, would validate with basketball.xsd schema. Once validated, libpackedobjects would be used to compress the client file and send the XML over to the server. The server would then unpack the compressed XML and then output it to the screen.

As main.c was already created, the client file was easier to create. The additions to the client file where user inputs and a loop for the number of players. This was included in the function basketball_team. The main function then used pointers and used libpackedobjects. The server contained error handling for potential faults in the use of packedobjects, along with the unpacking of the compressed file.

If in a real situation and the file was to be outputted by the server, the server computer would need the correct schema, in this case basketball.xsd. The schema here would act as a key to validate the client.c file.

Below are the client.c, server.c and the output of the two files. In the output of the files, the image shows that there are user inputs for the client (left) and the output of the XML on the server (right):

Client.c:

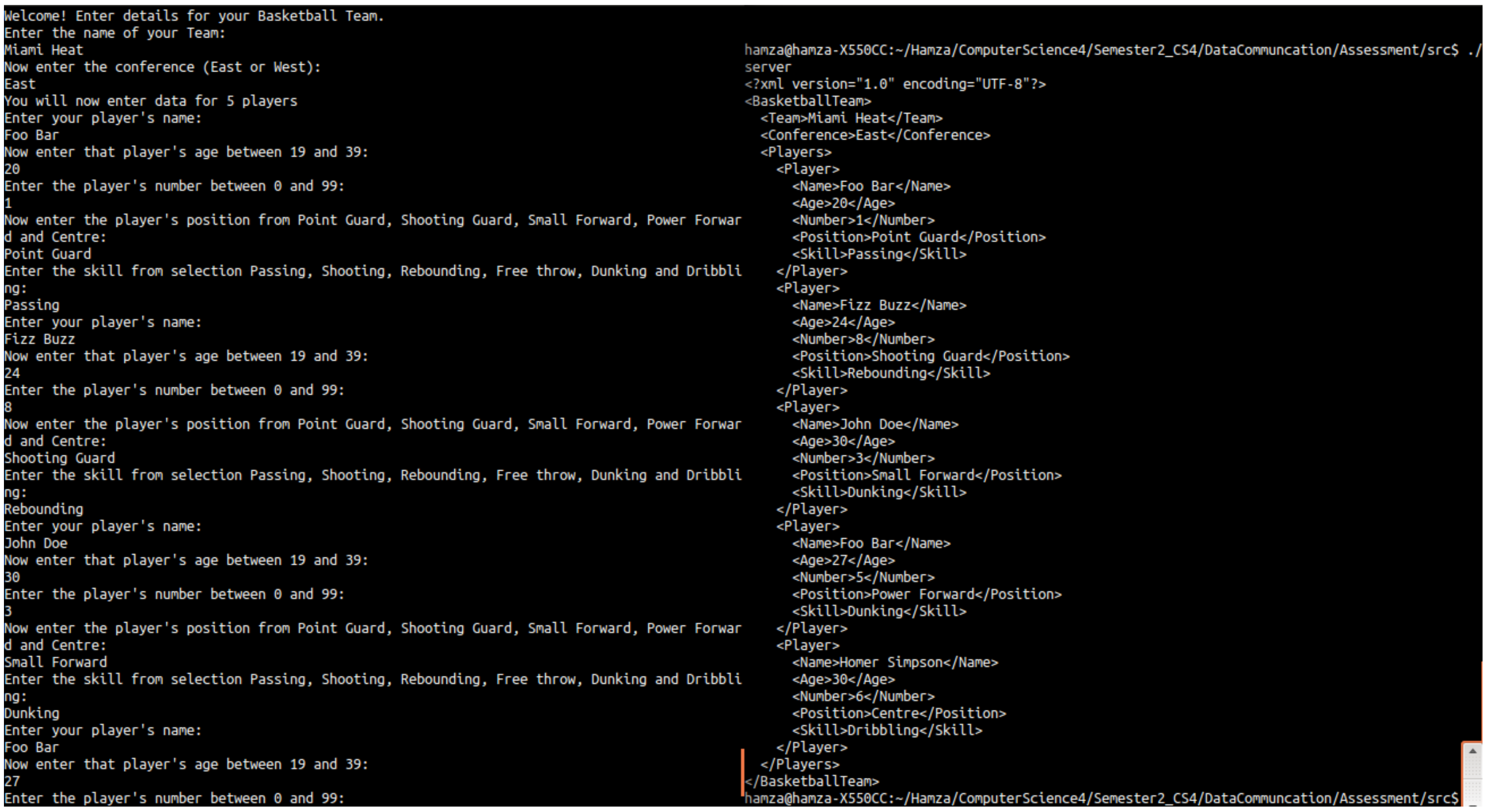
```c
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <packedobjects/packedobjects.h>
#include <libxml/parser.h>
#include <libxml/tree.h>

// Details for the XML
#define XML_SCHEMA "basketball.xsd"
#define MAXCHARS 250
#define NUM_PLAYERS 5

// Local IP and port
#define HOST_IP "127.0.0.1"
#define HOST_PORT 6969


int basketball_team(xmlDocPtr doc)
{
  // Message for user
  printf("Welcome! Enter details for your Basketball Team.\n");

  // Nodes to be used for the XML tree creation
  xmlNodePtr root_node = NULL , node = NULL , node2 = NULL;

  // MAXCHARS to be used for user input
  char buffer[MAXCHARS];

  // i to be used in players loop
  int i = 0;
```

```
// Creating xml tree

// Creating a root node (BasketballTeam) and set it for the whole doc
root_node = xmlNewNode(NULL, BAD_CAST "BasketballTeam");
xmlDocSetRootElement(doc, root_node);

// Clearing buffer ready for use for input
bzero(buffer, MAXCHARS);


// User input for Team
printf("Enter the name of your Team: \n");
// Error message
if(fgets(buffer, MAXCHARS, stdin) == NULL)
{
  fprintf(stderr, "Failed to read your team name\n");
  return (EXIT_FAILURE);
}
// Save input in buffer
strtok(buffer, "\n");
// Making Team node, child of root_node (BasketballTeam)
xmlNewChild(root_node, NULL, BAD_CAST "Team" , BAD_CAST buffer);

// Clean buffer
bzero(buffer, MAXCHARS);

//User input for Conference
printf("Now enter the conference (East or West): \n");
// Error message
if(fgets(buffer, MAXCHARS, stdin) == NULL)
{
  fprintf(stderr, "Failed to read your conference\n");
  return (EXIT_FAILURE);
}
// Save input in buffer
strtok(buffer, "\n");
// Making Conference node, child of root_node (BasketballTeam)
xmlNewChild(root_node, NULL, BAD_CAST "Conference" , BAD_CAST buffer);

// Creating a new parent node (Players)
node = xmlNewChild(root_node, NULL, BAD_CAST "Players", NULL);

// Starting loop for Player child of Players
printf("You will now enter data for 5 players\n");

for (i = 1; i <= NUM_PLAYERS; i++)
{
  // Cleaning buffer at start of loop
  bzero(buffer, MAXCHARS);

  // Creating a new subtree (Player)
  node2 = xmlNewChild(node, NULL, BAD_CAST "Player", NULL);

  // User input for Name
  printf("Enter your player's name: \n");
  // Error message
  if(fgets(buffer, MAXCHARS, stdin) == NULL)
  {
    fprintf(stderr, "Failed to read your player's name\n");
    return (EXIT_FAILURE);
  }
  // Save input in buffer
  strtok(buffer, "\n");
  xmlNewChild(node2, NULL, BAD_CAST "Name", BAD_CAST buffer);

  bzero(buffer, MAXCHARS);
```

```
    // User input for Age
    printf("Now enter that player's age between 19 and 39: \n");
    // Error message
    if(fgets(buffer, MAXCHARS, stdin) == NULL)
    {
      fprintf(stderr, "Failed to read your player's age\n");
      return (EXIT_FAILURE);
    }
    // Save input in buffer
    strtok(buffer, "\n");
    xmlNewChild(node2, NULL, BAD_CAST "Age", BAD_CAST buffer);

    bzero(buffer, MAXCHARS);

    // User input for Number
    printf("Enter the player's number between 0 and 99: \n");
    // Error message
    if(fgets(buffer, MAXCHARS, stdin) == NULL)
    {
      fprintf(stderr, "Failed to read your player's number\n");
      return (EXIT_FAILURE);
    }
    // Save input in buffer
    strtok(buffer, "\n");
    xmlNewChild(node2, NULL, BAD_CAST "Number", BAD_CAST buffer);

    bzero(buffer, MAXCHARS);

    // User input for Position
    printf("Now enter the player's position from Point Guard, Shooting Guard, Small
Forward, Power Forward and Centre: \n");
    // Error message
    if(fgets(buffer, MAXCHARS, stdin) == NULL)
    {
      fprintf(stderr, "Failed to read the player's position\n");
      return (EXIT_FAILURE);
    }
    // Save input in buffer
    strtok(buffer, "\n");
    xmlNewChild(node2, NULL, BAD_CAST "Position", BAD_CAST buffer);

    bzero(buffer, MAXCHARS);

    // User input for player Skill
    printf("Enter the skill from selection Passing, Shooting, Rebounding, Free throw,
Dunking and Dribbling: \n");
    // Error message
    if(fgets(buffer, MAXCHARS, stdin) == NULL)
    {
      fprintf(stderr, "Failed to read the player's skill\n");
      return (EXIT_FAILURE);
    }
    // Save input in buffer
    strtok(buffer, "\n");
    xmlNewChild(node2, NULL, BAD_CAST "Skill", BAD_CAST buffer);
  }
   return EXIT_SUCCESS;
}


int main (int argc, char **argv)
{
  xmlDocPtr doc = NULL;
  packedobjectsContext *pc = NULL;
  char *pdu = NULL;
  ssize_t bytes_sent;
  int       sock;
  struct    sockaddr_in servaddr;

  // setup socket
  if ( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "Error creating socket.\n");
    exit(EXIT_FAILURE);
  }
  // setup addressing
  memset(&servaddr, 0, sizeof(servaddr));
  servaddr.sin_family      = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr(HOST_IP);
  servaddr.sin_port        = htons(HOST_PORT);

  // initialise packedobjects
  if ((pc = init_packedobjects(XML_SCHEMA, 0, 0)) == NULL) {
    printf("failed to initialise libpackedobjects");
    exit(1);
  }

  // create the data
  doc = xmlNewDoc(BAD_CAST "1.0");
  basketball_team(doc);

  // encode the XML DOM
  pdu = packedobjects_encode(pc, doc);
  if (pc->bytes == -1) {
    printf("Failed to encode with error %d.\n", pc->encode_error);
  }

  // make network connection
  if (connect(sock, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    fprintf(stderr, "Error calling connect()\n");
    exit(EXIT_FAILURE);
  }

  // send the pdu across the network
  bytes_sent = send(sock, pdu, pc->bytes, 0);

  if (bytes_sent != pc->bytes) {
    fprintf(stderr, "Error calling send()\n");
    exit(EXIT_FAILURE);
  }

  if ( close(sock) < 0 ) {
    fprintf(stderr, "Error calling close()\n");
    exit(EXIT_FAILURE);
  }

  // free the DOM
  xmlFreeDoc(doc);

  // free memory created by packedobjects
  free_packedobjects(pc);

  xmlCleanupParser();
  return(0);
}
```

Server.c:
```c
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <packedobjects/packedobjects.h>

#define XML_SCHEMA "basketball.xsd"
#define MAX_BUFFER (1000*1024)
#define HOST_IP "127.0.0.1"
#define HOST_PORT 6969

int main(int argc, char **argv)
{
  xmlDocPtr doc = NULL;
  packedobjectsContext *pc = NULL;
  int list_s, conn_s;
  struct sockaddr_in servaddr;
  char buffer[MAX_BUFFER];
  ssize_t bytes_received = 0;

  // setup socket
  if ( (list_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "Error creating listening socket.\n");
    exit(EXIT_FAILURE);
  }
  // setup addressing
  memset(&servaddr, 0, sizeof(servaddr));
  servaddr.sin_family      = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port        = htons(HOST_PORT);

  // setup to accept connections
  if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    fprintf(stderr, "Error calling bind()\n");
    exit(EXIT_FAILURE);
  }
  if ( listen(list_s, 5) < 0 ) {
    fprintf(stderr, "Error calling listen()\n");
    exit(EXIT_FAILURE);
  }
  if ( (conn_s = accept(list_s, NULL, NULL) ) < 0 ) {
    fprintf(stderr, "Error calling accept()\n");
    exit(EXIT_FAILURE);
  }

  // receive the pdu
  bytes_received = read(conn_s, buffer, MAX_BUFFER);

  // initialise packedobjects
  if ((pc = init_packedobjects(XML_SCHEMA, bytes_received, 0)) == NULL) {
    printf("failed to initialise libpackedobjects");
    exit(1);
  }

  // decode the PDU into DOM
  doc = packedobjects_decode(pc, buffer);
  if (pc->decode_error) {
    printf("Failed to decode with error %d.\n", pc->decode_error);
    exit(1);
  }
  // output the received DOM
  packedobjects_dump_doc(doc);

  // free the DOM
  xmlFreeDoc(doc);

  // free memory created by packedobjects
  free_packedobjects(pc);

  xmlCleanupParser();
  return(0);
}
```

Output:



```
Welcome! Enter details for your Basketball Team.
Enter the name of your Team:
Miami Heat
Now enter the conference (East or West):
East
You will now enter data for 5 players
Enter your player's name:
Foo Bar
Now enter that player's age between 19 and 39:
20
Enter the player's number between 0 and 99:
1
Now enter the player's position from Point Guard, Shooting Guard, Small Forward, Power Forwar
d and Centre:
Point Guard
Enter the skill from selection Passing, Shooting, Rebounding, Free throw, Dunking and Dribbli
ng:
Passing
Enter your player's name:
Fizz Buzz
Now enter that player's age between 19 and 39:
24
Enter the player's number between 0 and 99:
8
Now enter the player's position from Point Guard, Shooting Guard, Small Forward, Power Forwar
d and Centre:
Shooting Guard
Enter the skill from selection Passing, Shooting, Rebounding, Free throw, Dunking and Dribbli
ng:
Rebounding
Enter your player's name:
John Doe
Now enter that player's age between 19 and 39:
30
Enter the player's number between 0 and 99:
3
Now enter the player's position from Point Guard, Shooting Guard, Small Forward, Power Forwar
d and Centre:
Small Forward
Enter the skill from selection Passing, Shooting, Rebounding, Free throw, Dunking and Dribbli
ng:
Dunking
Enter your player's name:
Foo Bar
Now enter that player's age between 19 and 39:
27
Enter the player's number between 0 and 99:
```

```
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$ ./
server
<?xml version="1.0" encoding="UTF-8"?>
<BasketballTeam>
  <Team>Miami Heat</Team>
  <Conference>East</Conference>
  <Players>
    <Player>
      <Name>Foo Bar</Name>
      <Age>20</Age>
      <Number>1</Number>
      <Position>Point Guard</Position>
      <Skill>Passing</Skill>
    </Player>
    <Player>
      <Name>Fizz Buzz</Name>
      <Age>24</Age>
      <Number>8</Number>
      <Position>Shooting Guard</Position>
      <Skill>Rebounding</Skill>
    </Player>
    <Player>
      <Name>John Doe</Name>
      <Age>30</Age>
      <Number>3</Number>
      <Position>Small Forward</Position>
      <Skill>Dunking</Skill>
    </Player>
    <Player>
      <Name>Foo Bar</Name>
      <Age>27</Age>
      <Number>5</Number>
      <Position>Power Forward</Position>
      <Skill>Dunking</Skill>
    </Player>
    <Player>
      <Name>Homer Simpson</Name>
      <Age>30</Age>
      <Number>6</Number>
      <Position>Centre</Position>
      <Skill>Dribbling</Skill>
    </Player>
  </Players>
</BasketballTeam>
hamza@hamza-X550CC:~/Hamza/ComputerScience4/Semester2_CS4/DataCommuncation/Assessment/src$
```

Git Log

For the assignment, a git repository was created to systematically upload the code. With the files that were uploaded, commit messages where used to identify what was being created and then updated while development was taking place. The end products of these commit messages were the code shown earlier in this assignment. The following are the commit messages:

```
commit fd3d03e12079678b521f06c38472a877b1599849
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Tue May 5 13:04:04 2015 +0100

    Assessment: Client/Server: Have updated the client.c file warning where the return statement for backetball_team function did not have a return statement. The files compile and run successfully.

commit 2da789b3e4c2270b6aa8287ce43aa5992b4d9518
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Tue May 5 11:38:43 2015 +0100

    Assement: Client/ Server: Created the client server programs. Both compile with no errors, but there is a warning for the client file.

commit c339e3f3f947696753dd16fc3f792aa1b06aae04
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Mon May 4 22:09:37 2015 +0100

    Assessment: C file, fixed warning message where doc pointer had more than one node. On compilation, no warning or error messages occur.

commit c7435130626a821c269bc2b6fd14f76abaeef63f
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Mon May 4 17:00:24 2015 +0100

    Assessment: C, created a c file that makes an xml tree. But it does not adhere to libpackedobjects.

commit 61c2875e00df715e03baef4b5014a464b3fd4550
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Mon May 4 12:56:53 2015 +0100

    Assessment: XML/ XSD, updated the xml xsd and .po file just for an example.

commit d2fa5a2be0a516c5eb22675b62262d7ca846cb29
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Tue Apr 28 14:16:11 2015 +0100

    Assessment: XML/XSD: Adding main.c file which is meant to create an XML file which was uploaded previously. For some reason I am getting too many error. Which is intreguing as it follows the same code that was can be previously seen in this rep (Student2) Waiting to fix.

commit c23deb257a4f991b313aad55c607ab039ca0906a
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
```

```
Date:   Tue Apr 28 12:59:27 2015 +0100

    Assessment: XML/XSD: Updated the schema file to have a number type for the player number.
The schema now has an updated tree. Also commit contains an template XML file to check whether
the XML validates (which it has).

commit 8d8b040ad64e8dce1cc59c29ac4510fc5f4a965e
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Thu Apr 23 19:07:01 2015 +0100

    Assessment: XML/XSD: *NOTE changed commit heading to avoid confusion. Added the structure
that the XML that will be coded should conform to.

commit 1b44a373e68a9ac82b248b5caa8a74bb14fdd039
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Thu Apr 23 18:49:53 2015 +0100

    XML Schema: The schema file called basketball.xsd is being uploaded. The restriction for the
elements that are to be used in the XML file have been created. The tree structure is still to be
completed in this .xsd file.

commit 3d6eb3db6e9d1c92babe3036eb4def806f862a4a
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Fri Apr 3 18:42:06 2015 +0100

    Programming XML with C: The subdirectories contain c files that when executed, produce the XML
tree that was programmed. The vidlib direcotry contains a main.c, .xml and .xsd schema.

commit 925b1966dad2cce2229a87d3b6dd1065e1323e93
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Fri Apr 3 18:39:33 2015 +0100

    XML: The Directory contains 3 sub directories. Each sub directory containn one XML file along with
its schema key. These activties gave an introduction to the basics of xml and schema files.

commit 7fb7fa1c676a4e7f32d1b94601a5d53572b88358
Author: Hamza <https://HamzaB93@bitbucket.org/HamzaB93/uwl_compsci4_lab.git>
Date:   Thu Mar 12 09:28:54 2015 +0000

    Testing upload
```