Puerto entre APOLO y MATLAB

Este manual describe brevemente el puerto de comunicación TCP/IP que permite el control de Apolo desde otro programa. Además del protocolo básico se explican los distintos módulos de Matlab que implementan estas funcionalidades.

Universidad Politécnica de Madrid
Centro de Automática y Robótica
EUITI
miguel.hernando@upm.es

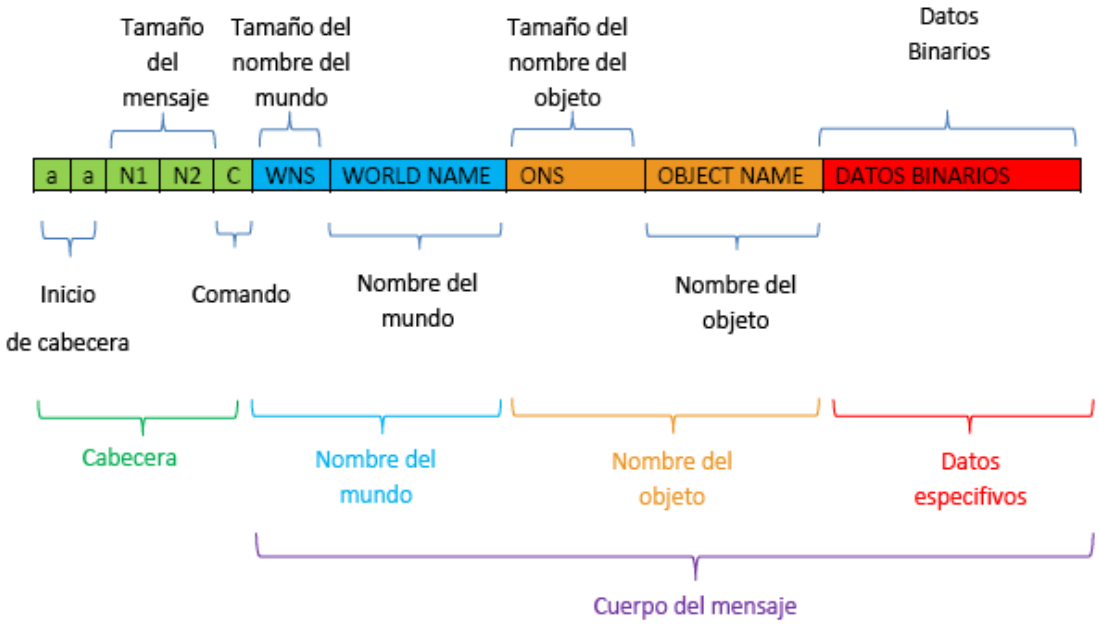
Contenido

PROTOCOLO

Apolo dispone de un puerto de comunicación externo por medio de una conexión TCP/IP, de forma que cualquier sistema capaz de enviar y recibir datos binarios a través de dicho puerto de comunicación, podrá solicitar que se ejecuten comandos en el simulador.

El puerto de comunicaciones utilizado es el **12000**.

Los mensajes aceptados y respondidos por tienen el formato siguiente:



De estos campos, los únicos que aparecerá en todos los posibles mensajes son los relativos a la cabecera. El resto dependerán del tipo de comando que se transmita o del tipo de dato de respuesta. En este manual una cabecera de un comando determinado la indicaremos como HEADER('caracter del comando').

En general todas las **cadenas de caracteres** (nombres) se codifican de la misma forma. El primer caracter indica el tamaño (0 si no se especifica una cadena) , y en caso de que el tamaño sea mayor que cero, a continuación aparecerá la cadena en formato C. El tamaño del nombre incluye el cero de finalización. En este manual, a esta secuencia la denominaremos STRING(nombre)

Los **vectores de doubles**, se codifican siempre con un primer número entero que indica el tamaño y codificado con dos bytes (el primero es la parte menos significativa, y el segundo la más significativa (little endian)), y a continuación se transmiten los 64 bits del estandar de números reales, por orden creciente de cada elemento del vector. En el manual, a esta secuencia la denominaremos V_DOUBLES(num,vector)

Inicio de cabecera: Bytes utilizados para distinguir el comienzo de un mensaje. Todos empezaran por los caracteres 'a' 'a'.
Tamaño del mensaje: N1 indicara el numero de caracteres de 0 a 255, puesto que es el valor máximo que puede adoptar un byte, en caso de que el tamaño del mensaje sea superior a este, se utilizara el byte N2, ampliando el rango a 65536.

Comando: Byte que indicara el tipo de instrucción que contiene el mensaje, para que el receptor sepa como interpretar el contenido de este.

Tamaño del nombre del mundo: Byte que indica el número de caracteres de los que consta el nombre del mundo.
Nombre del mundo: Conjunto de bytes que contienen los caracteres que forman el nombre del mundo, el último carácter es un 0
Tamaño del nombre del objeto: Byte que indica el número de caracteres de los que consta el nombre del objeto.
Nombre del objeto: Byte que indica el número de caracteres de los que consta el nombre del objeto., el ultimo carácter es un cero.

Datos binarios: datos característicos de cada mensaje para la manipulación del objeto en cuestión.

Ejemplo:

El tamaño 46 se descompone en:

5 bytes de cabecera

8+1 bytes para el nombre del mundo

14+1 bytes para el nombre del robot

1 byte para el indicador de numero de articulaciones

2x8 bytes para codificar los dos doubles

COMANDOS DISPONIBLES

apoloCheckRobotJoints

funcionalidad:

Esta función permite mover las articulaciones de un robot articular y chequear si en la configuración resultante el robot colisiona con los objetos del entorno.

prototipo :

```
ret=apoloCheckRobotJoints(robot,values,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot e.g: 'Puma560'
- *values*: vector de números reales. Su rango determinará cuantos ejes se intentarán establecer.
- *world*: mundo al que pertenece el robot que se quiere mover. En caso de omitirse, se moverá el primer robot que se encuentre bajo el nombre indicado si se recorren en orden creciente los mundos abiertos en Apolo.

valor de retorno:

1 **-** en caso de que el robot colisione con el entorno
0- si la configuración es libre de colisión

Ejemplo:

```
res=apoloCheckRobotJoints('Puma560',[0.5 0.23 -0.1],'world1');
if(res==1)
      display('colisiona');
else
      display('no colisiona');
end
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('j')+STRING(WORLD)+STRING(ROBOT)+DATOS_BINARIOS

DATOS BINARIOS: BYTE(n=tamaño vector)+n*(DOUBLE(VALOR[i])

información trama de respuesta:

Para el valor true: HEADER con el COMANDO 'T'
Para el valor false: HEADER con el COMANDO 'F'

apoloGetLocation

funcionalidad:

Esta función obtiene la localización espacial de un objeto (habitualmente un robot).

prototipo :

```
ret=apoloGetLocation(robot,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot u el objeto e.g: 'Puma560'
- *world*: mundo al que pertenece el elemento que se quiere mover. Omisible.

valor de retorno:

Double [6] - retorna un vector de 6 componenetes. Las tres primeras corresponden a la posición, mientras que las tres ultimas a los ángulos rpy en radianes.

Ejemplo:

```
>> apoloGetLocation('pieza')
ans =
   -0.2500    0.2000    0.0600    1.5708    0.6912    0.9425
>>
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('G')+STRING(WORLD)+STRING(OBJECT)

DATOS BINARIOS:

información trama de respuesta:

HEADER ('T')+ V_DOUBLES(6,posicion y orientacion)

apoloGetLocationMRobot

funcionalidad:

Esta función obtiene la posicion de un vehículo con ruedas

prototipo :

```
ret=apoloGetLocationMRobot(robot,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot e.g: 'Neo'
- *world*: mundo al que pertenece el elemento que se quiere mover. Omisible.

valor de retorno:

***Double [4]* -** retorna un vector de 4 componentes. Las tres primeras corresponden a la posición, mientras que la última es el ángulo de rotación en Z..

Ejemplo:

```
>> apoloGetLocationMRobot('Pioneer3AT')
ans =
    0.7000    2.5000         0         0
>>
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('g')+STRING(WORLD)+STRING(ROBOT)

DATOS BINARIOS:

información trama de respuesta:

HEADER ('T')+ V_DOUBLES(4,[x y z rz])

apoloMoveMRobot

funcionalidad:

Esta mueve un robot movil a la velocidad indicada durante un tiempo tambien establecido. El robot aplicará las restricciones cinemáticas y geométricas. Por tanto si colisionase, se quedaría parado en la última posición sin colisión.

prototipo :

```
function apoloMoveMRobot(robot,speeds,time,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot e.g: 'Neo'
- *speeds*: vector de dos doubles que son las consignas de velocidad (normalmente avance y rotación)
- *time*: tiempo en segundos que durará la acción. Conviene indicar pasos pequeños para evitar que el robot se "salte" o cheque contra objetos por razones de discretización de la trayectoria.
- *world*: cadena de caracteres indicación del mundo. Omitible

Ejemplo:

```
>> apoloMoveMRobot('Pioneer3AT',[0.1 0.0],0.1)
>> apoloUpdate
>>
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('m')+STRING(WORLD)+STRING(ROBOT)+DATOS_BINARIOS

DATOS BINARIOS: DOUBLE(speed1)+DOUBLE(speed2)+DOUBLE(time)

apoloPlace

funcionalidad:

Posiciona un objeto en la posición y orientación indicada independientemente de si colisiona o no con el entorno.

prototipo :

```
function apoloPlace (object,pos,or,world)
```

parámetros:

- *object*: cadena de caracteres que identifica el elemento a mover e.g: 'pieza'
- *pos: vector de 3 doubles que indican la posición*
- *or: vector de 3 doubles que indican la orientación en radianes (rpy)*
- *world*: cadena de caracteres indicación del mundo. Omisible

retorno:

Ejemplo:

```
>> apoloPlace ('pieza',[1 1 1],[0.5 0 0])
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('P')+STRING(WORLD)+STRING(OBJECT)+DATOS_BINARIOS

DATOS BINARIOS:
DOUBLE(x)+DOUBLE(y)+DOUBLE(z)+(DOUBLE(rx)+DOUBLE(ry)+DOUBLE(rz)

apoloPlaceMRobot

funcionalidad:

intenta posicionar un robot movil en la posición x y z , y la orientación dada por el angulo. Para ello *deja caer* desde la altura indicada el robot y comprueba si la posición de caida es valida por el número de apoyos y porque el cuerpo del robot no colisione.

prototipo :

```
function ret=apoloPlaceMRobot(robot,pose,angle,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot e.g: 'Neo'
- *pose*: vector de tres doubles que son las coordenadas de posición
- *angle*: angulo en radianes de rotación en Z
- *world*: cadena de caracteres indicación del mundo. Omisible

retorno:

- *1, en caso de tener éxito, 0 en caso contrario.*

Ejemplo:

```
>> apoloPlaceMRobot('Pioneer3AT',[0.7 0.7 0],0.23)
ans =
    1
>>
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('p')+STRING(WORLD)+STRING(ROBOT)+DATOS_BINARIOS

DATOS BINARIOS: DOUBLE(x)+(DOUBLE(y)+DOUBLE(z)+DOUBLE(rz)

apoloPlaceXYZ

funcionalidad:

Posiciona un objeto en las coordenadas XYZ independientemente de si colisiona o no con el entorno..

prototipo :

```
function apoloPlaceXYZ(object,x,y,z,world)
```

parámetros:

- *object*: cadena de caracteres que identifica el elemento a mover e.g: 'pieza'
- *x,y,z: doubles que indican la posición*
- *world*: cadena de caracteres indicación del mundo. Omisible

retorno:

Ejemplo:

```
>> apoloPlaceXYZ('Pioneer3AT',1,1,1)
```

Información de la trama TCP/IP COMANDO:

TRAMA: la misma que apoloPlace pero con rx=ry=rz=0

DATOS BINARIOS:

apoloSetRobotJoints

funcionalidad:

Esta función permite mover las articulaciones de un robot articular, pero no valida si colisiona o no.

prototipo :

```
function apoloSetRobotJoints(robot,values,world)
```

parámetros:

- *robot*: cadena de caracteres que identifica el robot e.g: 'Puma560'
- *values*: vector de números reales. Su rango determinará cuantos ejes se intentarán establecer.
- *world*: mundo al que pertenece el robot que se quiere mover. En caso de omitirse, se moverá el primer robot que se encuentre bajo el nombre indicado si se recorren en orden creciente los mundos abiertos en Apolo.

Ejemplo:

```
>>apoloSetRobotJoints('Puma560',[0.5 0.23 -0.1],'world1');
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('J')+STRING(WORLD)+STRING(ROBOT)+DATOS_BINARIOS

DATOS BINARIOS: BYTE(n=tamaño vector)+n*(DOUBLE(VALOR[i])

apoloUpdate

funcionalidad:

Actualiza la visualización de un mundo en Apolo. En caso de omitirse World, actualizará todos los mundos cargados en Apolo.

prototipo :

```
function apoloUpdate(world)
```

parámetros:

- *world*: cadena de caracteres indicación del mundo. Omisible

retorno:

Ejemplo:

```
>> apoloUpdate('World1')
>> apoloUpdate
>>
```

Información de la trama TCP/IP COMANDO:

TRAMA: HEADER('U')+STRING(World)

ANEXO:

ApoloMessage.h

```
#pragma once

#define AP_NONE 0
#define AP_SETJOINTS 'J'
#define AP_PLACE 'P'
#define AP_CHECKJOINTS 'j'
#define AP_UPDATEWORLD 'U'
#define AP_TRUE 'T'
#define AP_FALSE 'F'
#define AP_PLACE_WB 'p'
#define AP_MOVE_WB 'm'
#define AP_GETLOCATION 'G'
#define AP_GETLOCATION_WB 'g'
#define AP_DVECTOR 'D'
#define AP_LINK_TO_ROBOT_TCP 'L'
/*************************************************************/
/*This class implements the protocol for easily connect to apolo
  An apolo message pointers to an external buffer. Is simply an interpreter
  of raw data. Therefore, use have to be carefull.
*/
class ApoloMessage
{
	char *pData;//pointer to a byte secuence that has a message (header+size+type+specific data)
	char *world,*name,*bindata; //utility fields to avoid reinterpretation
	int size;
	char type;

	ApoloMessage(char *buffer,int size,char type);
public:
	static int writeSetRobotJoints(char *buffer, char *world, char *robot, int num, double *values);
	static int writeCheckColision(char *buffer, char *world, char *robot, int num, double *values);
	static int writeUpdateWorld(char *buffer, char *world);
	static int writeBOOL(char *buffer, bool val);
	static int writePlaceObject(char *buffer, char *world,char *object, double *xyzrpy);
	static int writePlaceWheeledBase(char *buffer, char *world,char *robot, double *xyzy);
	static int writeMoveWheeledBase(char *buffer, char *world,char *robot, double *sp_rs_t);
//speed,rot_speed,time
	static int writeGetLocation(char *buffer, char *world,char *object);
	static int writeGetLocationWheeledBase(char *buffer, char *world,char *robot);
	static int writeDoubleVector(char *buffer, int num, double *d);
	static int writeLinkToRobotTCP(char *buffer, char *world,char *robot,char *object);
	static ApoloMessage *getApoloMessage(char **buffer, int max);

	char *getWorld(){return world;}
	char *getObjectName(){return name;}
	char getType(){return type;}
	int getSize(){return size;}
	int getIntAt(int offset);
	int getUInt16At(int offset);
	double getDoubleAt(int offset);
	char getCharAt(int offset);
	char *getStringAt(int offset);


};
```

ApoloMessage.cpp

```
#include "apoloMessage.h"
#include <string.h>
/**UTILITY UNIONS FOR CONVERSIONS**/
union double2byteConversor
{
		char bytes[8];
		double real;
};

union int2byteConversor
{
		char bytes[4];
		int integer;
};
typedef unsigned char uchar;

/**
writes intp buffer the message for moving a robot in a worl
if world=null... is equivalent to any.
returns the message size
**/
inline int writeString(char *buffer, char *cad){
	int n=0,len;
	if(cad!=0){ //not null
		if(cad[0]==0)buffer[n++]=0;//empty string
		else{
			len=1+(uchar)strlen(cad);
			((uchar *)buffer)[n++]=(uchar)((len>255)?255:len);
			for(int i=0;i<len-1;i++)buffer[n++]=cad[i];
			buffer[n++]=0;
		}
	}else buffer[n++]=0;
	return n;
}
inline int writeDouble(char *buffer, double val){
	double2byteConversor aux;
	aux.real=val;
	for(int i=0;i<8;i++)buffer[i]=aux.bytes[i];
	return 8;
}
inline int writeUInt16(char *message, int &num)
{
	if(num>65535)num=65535;
	if(num<0)num=0;
	((uchar *)message)[0]=(uchar)(num%255);
	((uchar *)message)[1]=(uchar)(num/255);
	return 2;
}
inline void insertSize(char *message, int size)//size including the header
{
	((uchar *)message)[2]=(uchar)(size%255);
	((uchar *)message)[3]=(uchar)(size/255);
}
//tamaño minimo de mensaje es 5
inline int writeHeader(char*buffer,char command) //escribe la cabecera
{
	int n=0;
	buffer[0]='a';
	buffer[1]='a';
	insertSize(buffer,5);
	buffer[4]=command;
	return 5;
}
```

```
int ApoloMessage::writeSetRobotJoints(char *buffer, char *world, char *robot, int num, double *values)
{
	int n=0;
	n+=writeHeader(buffer,AP_SETJOINTS);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,robot);//robot
	if(num<0)num=0;
	if(num>255)num=255;
	((uchar *)buffer)[n++]=(uchar)num;//num joints
	for(int i=0;i<num;i++)
		n+=writeDouble(buffer+n,values[i]);
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writePlaceObject(char *buffer, char *world,char *object, double *xyzrpy)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_PLACE);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,object);//object
	for(i=0;i<6;i++)
		n+=writeDouble(buffer+n,xyzrpy[i]);
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writeMoveWheeledBase(char *buffer, char *world,char *robot, double *sp_rs_t)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_MOVE_WB);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,robot);//robot
	for(i=0;i<3;i++)//speed, rot speed, time
		n+=writeDouble(buffer+n,sp_rs_t[i]);
	insertSize(buffer,n);
	return n;
}
int  ApoloMessage::writePlaceWheeledBase(char *buffer, char *world,char *robot, double *xyzy)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_PLACE_WB);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,robot);//robot
	for(i=0;i<4;i++)//x,y,z, rot z
		n+=writeDouble(buffer+n,xyzy[i]);
	insertSize(buffer,n);
	return n;
}
//the same message But changes the command id
int ApoloMessage::writeCheckColision(char *buffer, char *world, char *robot, int num, double *values)
{
	int n=writeSetRobotJoints(buffer,world,robot,num,values);
	buffer[4]=AP_CHECKJOINTS;
	return n;
}
int ApoloMessage::writeGetLocation(char *buffer, char *world,char *object)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_GETLOCATION);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,object);//robot
	insertSize(buffer,n);
	return n;
}
```

```
int ApoloMessage::writeGetLocationWheeledBase(char *buffer, char *world,char *robot)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_GETLOCATION_WB);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,robot);//robot
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writeUpdateWorld(char *buffer, char *world)
{
	int n=0;
	n+=writeHeader(buffer,AP_UPDATEWORLD);//command
	n+=writeString(buffer+n,world);//world
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writeLinkToRobotTCP(char *buffer, char *world,char *robot,char *object)
{
	int n=0,i;
	n+=writeHeader(buffer,AP_LINK_TO_ROBOT_TCP);//command
	n+=writeString(buffer+n,world);//world
	n+=writeString(buffer+n,robot);//robot
	n+=writeString(buffer+n,object);//robot
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writeDoubleVector(char *buffer, int num, double *d)
{
	int n=0;

	n+=writeHeader(buffer,AP_DVECTOR);//command
	n+=writeUInt16(buffer+n,num);
	for(int i=0;i<num;i++)
		n+=writeDouble(buffer+n,d[i]);
	insertSize(buffer,n);
	return n;
}
int ApoloMessage::writeBOOL(char *buffer, bool val)
{
	int n=0;
	char command=AP_FALSE;
	if(val)command=AP_TRUE;
	n+=writeHeader(buffer,command);//command
	insertSize(buffer,n);
	return n;
}
```

```
ApoloMessage::ApoloMessage(char *buffer,int size,char type)
{
	char *aux;
	pData=buffer;
	this->size=size;
	this->type=type;
	if(type==AP_NONE)return;
	world=bindata=name=0;
	switch(type)
	{//command with world and name
		case AP_SETJOINTS:
		case AP_CHECKJOINTS:
		case AP_PLACE:
		case AP_PLACE_WB:
		case AP_MOVE_WB:
		case AP_GETLOCATION_WB:
		case AP_GETLOCATION:
		case AP_LINK_TO_ROBOT_TCP:
			if(pData[5]!=0){
				world=pData+6;
				aux=world+((uchar *)pData)[5];
			}else aux=pData+6;
			if(aux[0]!=0){
				name=aux+1;
				aux=name+((uchar *)aux)[0];
			}else aux++;
			bindata=aux;
		break;
		case AP_UPDATEWORLD://commands with world only
			if(pData[5]!=0){
				world=pData+6;
				aux=world+((uchar *)pData)[5];
			}else aux=pData+6;
			bindata=aux;
		break;
		default: //commands without world neither
			bindata=pData+5;
		break;
	}

}
//se considera que el buffer contiene mensajes completos (pueden ser varios)... si son parciales, se desecharán
ApoloMessage *ApoloMessage::getApoloMessage(char **buffer, int max)
{
	int i=0;
	while(i+4<max){
		if(((*buffer)[i]=='a')&&((*buffer)[i+1]=='a'))
		{
			int size=(((uchar *)(*buffer))[i+2])+(((uchar *)(*buffer))[i+3])*255;
			char  type=(*buffer)[i+4];
			//si el mensaje es correcto... crea el mensaje y situa el puntero al final
			//ojo... el mensaje mantiene unos punteros sobre el buffer original. El mensaje no
reserva memoria
			if(i+size<=max){
				ApoloMessage *message=new ApoloMessage((*buffer)+i,size,type);
				*buffer=*buffer+size;
				return message;
			}//si no lo es retorna null
			else return 0;

		}
	i++;
	}
	return 0;
}
```

```
int ApoloMessage::getIntAt(int offset)
{
	int2byteConversor aux;
	if(offset+(bindata-pData)+4>size)return 0;
	for(int i=0;i<4;i++)aux.bytes[i]=bindata[offset+i];
	return aux.integer;
}
int ApoloMessage::getUInt16At(int offset)
{
	if(offset+(bindata-pData)+2>size)return 0;
	return (((uchar *)(bindata))[offset])+(((uchar *)(bindata))[offset+1])*255;
}
double ApoloMessage::getDoubleAt(int offset)
{
	double2byteConversor aux;
	if(offset+(bindata-pData)+8>size)return 0;
	for(int i=0;i<8;i++)aux.bytes[i]=bindata[offset+i];
	return aux.real;
}
char ApoloMessage::getCharAt(int offset)
{
	if(offset+(bindata-pData)+1>size)return 0;
	return bindata[offset];
}
char *ApoloMessage::getStringAt(int offset)
{
	if(offset+(bindata-pData)+1>size)return 0;
	uchar tam=((uchar *)(bindata))[offset];
	if(tam==0)return 0;
	if(offset+(bindata-pData)+tam+1>size)return 0;
	else return bindata+offset+1;
}
```