Oz Robot Puppeteering System

protocol 1.0

by Desmond Germans
Germans Media Technology & Services

1-5-2014: first version

Overview

This document describes the exact protocol between client (controlling computer) and server (the robot) for the Oz Robot Puppeteering System. This system is designed especially for social robotics and supports speech synthesis, speech recognition, face detection and a small variety of concurrent nonverbal communication styles for the robot.

Figure 1. shows a system overview.

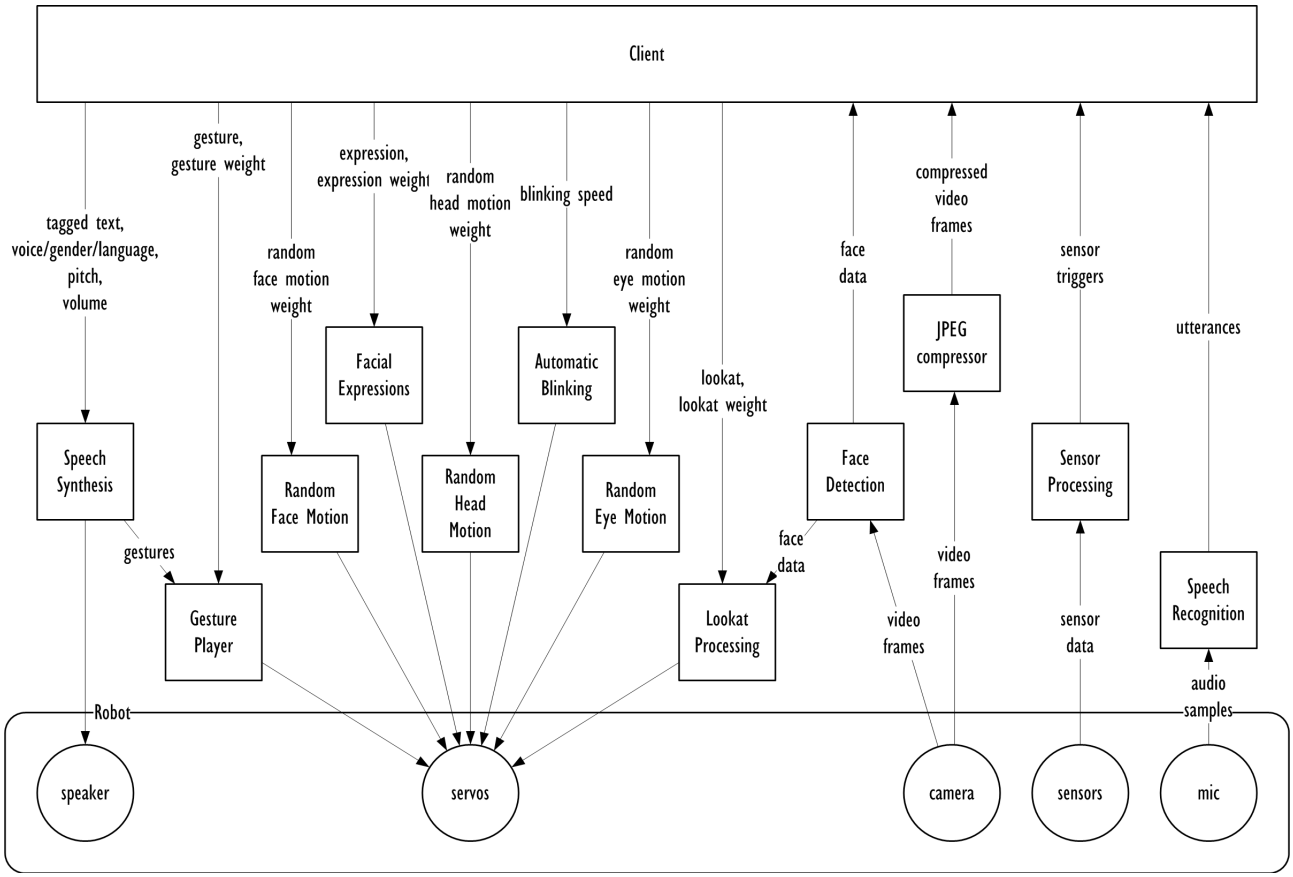

Figure 1. Oz Robot Puppeteering System overview

At the top is the client, in the middle the various software components that make up the system, and at the bottom the robot hardware.

Communications between client and server is done over a TCP connection. The protocol is very simple and direct, and relies on an uninterupted reliable full-duplex stream between client and server. There is no error recovery and no security.

All data is transmitted in big-endian format. This means that of a longer integer (uint16, uint32, uint64, etc.) the

individual bytes are transmitted in hi-to-lo order, like this:

Type	Transmitted as	Example data	Transmitted bytes
char/int8/uint8	no change	`0x01`	`0x01`
int16/uint16	first the high byte, then the low byte	`0x0123`	`0x01 0x23`
int32/uint32	first the highest byte, ..., finally the lowest byte	`0x01234567`	`0x01 0x23 0x45 0x67`
float	treat binary as uint32	0.390625 (`0x43A4B200`)	`0x43 0xA4 0xB2 0x00`
string	first uint16 to indicate number of bytes, then the bytes encoded as UTF-8	`"Hello"`	`0x00 0x05 0x47 0x65 0x6C 0x6C 0x6F`

Initialization

The client connects to the server (typically port 7331) and does a version request. The server responds with the version number and the server's short and long names as follows:

A connection is established between client and server.

The client sends `MSG_VERSION_REQUEST`, followed by the version number:

Offset	Type	Description
0	uint32	`MSG_VERSION_REQUEST` (`0x00000001`)
4	uint32	requested version number (currently `0x01000000`)

The server responds with the supported version number and two strings:

Offset	Type	Description
0	uint32	version number (currently `0x01000000`)
4	uint16	number of bytes in short name string (n)
6	char[n]	short name string
6 + n	uint16	number of bytes in long name string (m)
8 + n	char[m]	long name string

If the version number does not match, the client may choose to abort, or fallback to an earlier supported version.

When the version numbers match, the server starts transmitting video frames, detected faces and understood utterances to the client. The client should handle all of the messages accordingly and asynchronously.

At any time, the client can reset the server by sending `MSG_RESET` as follows:

Offset	Type	Description
0	uint32	`MSG_RESET` (`0x00000002`)

Speech Synthesis

The client can send text string to the server for parsing and speaking out loud. The text string can contain gesture tags to trigger specific gestures inside the speech. This way, the server can raise eyebrows at specific words, nod where it is appropriate in the sentence, etc.

Sending text to the robot goes as follows:

Offset	Type	Description
0	uint32	`MSG_SPEAK` (`0x00000003`)
4	uint16	number of bytes in text string (n)
6	char[n]	text string

The server immediately starts speaking. When gestures are triggered from the text, the server sends back `MSG_SPEAK_GESTURE` messages, and when the server is finished, it sends a `MSG_SPEAK_DONE` message to indicate readiness.

The `MSG_SPEAK_GESTURE` message looks like this:

Offset	Type	Description
0	uint32	`MSG_SPEAK_GESTURE`
4	uint32	gesture ID

For the possible gesture IDs, see Gestures section.

The `MSG_SPEAK_DONE` message looks like this:

Offset	Type	Description
0	uint32	`MSG_SPEAK_DONE` (`0x00000014`)

Next to this, the client can request a different language, gender, voice, pitch and volume. To change language/gender/voice, the client sends a `MSG_VOICE_GENDER_LANGUAGE` message:

Offset	Type	Description
0	uint32	`MSG_VOICE_GENDER_LANGUAGE` (`0x00000004`)
4	uint32	voice/gender/language combination

At the moment, the only valid combination is `VCL_DEFAULT` (`0x00000000`)

To change the pitch, the client sends a `MSG_VOICE_PITCH` message:

Offset	Type	Description
0	uint32	`MSG_VOICE_PITCH` (`0x00000005`)
4	float	new pitch (0.5..2.0), default is 1.0

Finally, to change the volume, the client sends a `MSG_VOICE_VOLUME` message:

Offset	Type	Description
0	uint32	`MSG_VOICE_VOLUME` (`0x00000006`)
4	float	new volume (0.0..1.0), default is 1.0

Speech Recognition

Speech recognition is fully automatic. The server informs the client whenever it heard something it understood. It does so via the `MSG_UTTERANCE` to the client. The message contains one or more strings with associated confidence, like so:

Offset	Type	Description
0	uint32	`MSG_UTTERANCE` (`0x00000017`)
4	uint32	number of options (n)
8	option[n]	The options (see below)

Each option is transmitted as follows:

Offset	Type	Description
0	float	confidence factor (0.0..1.0)
4	uint16	number of bytes in text string (n)
6	char[n]	text string

The first option always has the highest confidence.

Eye Video

After initialization, the server will start to transmit compressed video frames from the camera. The frames are compressed as regular JPEG images, and transmitted as follows:

Offset	Type	Description
0	uint32	`MSG_VIDEO_FRAME` (`0x00000019`)
4	uint32	Number of bytes in compressed image data (n)
8	uint8[n]	Compressed image data

The client should always process the incoming `MSG_VIDEO_FRAME` messages, but could ofcourse immediately discard the data if no video is used.

Face Detection

The server will automatically detect if the camera image contains a face, and build up a list of faces it finds. When it succesfully detects one or more faces, it sends the `MSG_FACES` message back to the client as follows:

Offset	Type	Description
0	uint32	`MSG_FACES` (`0x00000018`)
4	uint32	Number of faces found (n)
8	face[n]	The faces (see below)

Each face is transmitted as follows:

Offset	Type	Description
0	float	confidence factor (0.0..1.0)
4	float	X-coordinate of face (-1.0..1.0, -1.0 is far left, 1.0 is far right)
8	float	Y-coordinate of face (-1.0..1.0, -1.0 is far up, 1.0 is far down)
12	float	width of face (0.0..1.0, 1.0 is full video width)
16	float	height of face (0.0..1.0, 1.0 is full video height)
20	uint16	number of bytes in name string (n)
22	char[n]	name string

Servo Mixer

The server moves using several servo controllers controlling all the body, face and hand servos. With Oz, servo control is a mix of a variety of subsystems that all operate simultaneously (Figure 2).

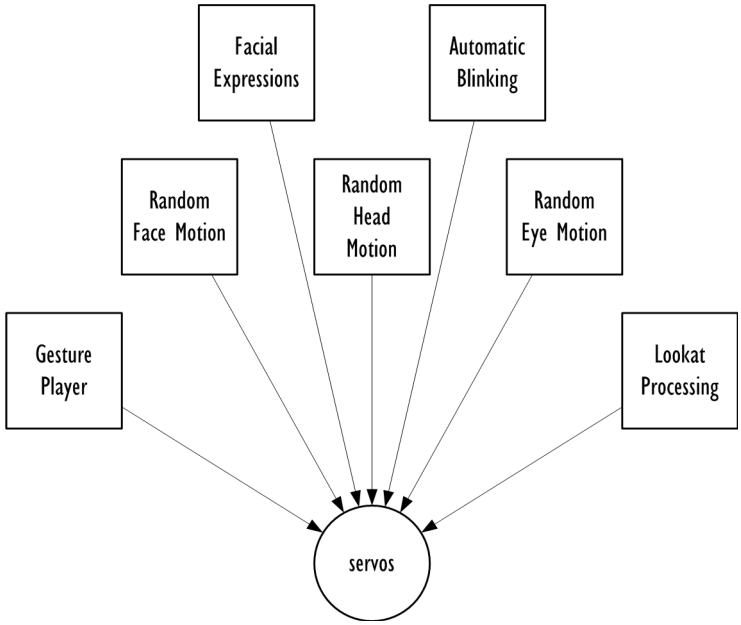

Figure 2. The various subsystems that mix together into the final servo motions.

The client can specify a weighing factor for each of the 7 subsystems. All weights are defined between 0.0 and 1.0, where 0.0 means no contribution and 1.0 means full contribution. The weights can be specified by the client:

Offset	Type	Description
0	uint32	weight message (see below)
4	float	weighing factor (0.0..1.0)

The weight messages for the various weights are:

Message	Description
`MSG_GESTURE_WEIGHT` (`0x00000008`)	gesture weight (default is 1.0)
`MSG_EXPRESSION_WEIGHT` (`0x0000000A`)	facial expression weight (default is 1.0)
`MSG_RANDOM_HEAD_WEIGHT` (`0x0000000C`)	random head motion weight (default is 0.0)
`MSG_RANDOM_EYES_WEIGHT` (`0x0000000D`)	random eyes motion weight (default is 0.0)
`MSG_RANDOM_FACE_WEIGHT` (`0x0000000E`)	random face motion weight (default is 0.0)
`MSG_LOOKAT_WEIGHT` (`0x00000013`)	lookat weight (default is 1.0)

Lookat Functionality

The found face data can be used to have the robot automatically deal with looking at the person it is talking to. The client can specify one of several lookat modes as follows:

Offset	Type	Description
0	uint32	`MSG_LOOKAT` (`0x00000012`)
4	uint32	lookat mode (see below)

The available modes are:

Mode	Description
`LOOKAT_FORWARD` (`0x00000000`)	The robot looks forward (default)
`LOOKAT_DOWN` (`0x00000001`)	The robot looks down
`LOOKAT_UP` (`0x00000002`)	The robot looks up
`LOOKAT_LARGEST` (`0x00000003`)	The robot follows the largest (closest) face
`LOOKAT_SMALLEST` (`0x00000004`)	The robot follows the smallest (furthest) face
`LOOKAT_AWAY` (`0x00000005`)	The robot tries to avoid any face

Gestures

Gestures are small servo animations to convey a non-verbal communication signal. The client can start a gesture, or a gesture can be started from the speech synthesizer. To start a gesture, the client sends:

Offset	Type	Description
0	uint32	`MSG_GESTURE` (`0x00000007`)
4	uint32	Gesture ID (see below)

The gestures Ids and corresponding text tags are as follows:

ID	Tag	Description
`GESTURE_BROW_RAISE_LONG` (`0x00000000`)	`[braisel]`	Long raise of the eyebrows
`GESTURE_BROW_RAISE_SHORT` (`0x00000001`)	`[braises]`	Short raise of the eyebrows
`GESTURE_FROWN_LONG` (`0x00000002`)	`[frownl]`	Long frown of the eyebrows
`GESTURE_FROWN_SHORT` (`0x00000003`)	`[frowns]`	Short frown of the eyebrows
`GESTURE_YES_LONG` (`0x00000004`)	`[yesl]`	Long yes-motion
`GESTURE_YES_SHORT` (`0x00000005`)	`[yess]`	Short yes-motion
`GESTURE_YES_LONG_3X` (`0x00000006`)	`[yesl3]`	Triple long yes-motion
`GESTURE_YES_SHORT_3X` (`0x00000007`)	`[yess3]`	Triple short yes-motion
`GESTURE_NO_LONG` (`0x00000008`)	`[nol]`	Long no-motion
`GESTURE_NO_SHORT` (`0x00000009`)	`[nos]`	Short no-motion
`GESTURE_NO_LONG_3X` (`0x0000000A`)	`[nol3]`	Triple long no-motion
`GESTURE_NO_SHORT_3X` (`0x0000000B`)	`[nos3]`	Triple short no-motion
`GESTURE_INDIAN_WOBBLE` (`0x0000000C`)	`[wob]`	Indian head wobble

When a gesture animation is done, the server sends `MSG_GESTURE_DONE` to the client to indicate readiness. It does so for gestures started with `MSG_GESTURE` as well as gestures started from speaking. The server is capable of playing multiple gestures simultaneously. For instance, speaking `"[braisel][nos]Nope."` will have the server say `"Nope."` with raised eyebrows and a small head shake.

Facial Expressions

Next to lookat processing and gestures, the client can request a more permanent facial expression:

Offset	Type	Description
0	uint32	MSG_EXPRESSION (0x00000009)
4	uint32	facial expression ID (see below)

The facial expression IDs are:

Mode	Description
EXPRESSION_NEUTRAL (0x00000000)	neutral expression
EXPRESSION_HAPPY (0x00000001)	smile and raised eyebrows
EXPRESSION_SAD (0x00000002)	sad and frowning
EXPRESSION_EVIL (0x00000003)	smile and frowning
EXPRESSION_AFRAID (0x00000004)	Sad and raised eyebrows

Automatic Motions

As described in the servo mixing section, there are several automatic motions that the client can mix together. The server can automatically blink with the eyelids, randomly move the eye gaze, randomly move smiling and frowning and randomly move the entire head.

To indicate the automatic blinking speed, the client sends:

Offset	Type	Description
0	uint32	`MSG_AUTO_BLINK_SPEED` (`0x0000000B`)
4	float	New automatic blinking speed in times per second (0.0..10.0), default is 2.0

Client APIs

There are various client APIs available for Oz, all presenting a similar set of methods to the programming environment.

C Client

This is the most basic client, with a very straightforward interface. The API defines the following:

```
typedef void* OZ;

OZ oz_connect(char* hostname,int port);  // connect to server, returns null if failed
void oz_disconnect(OZ oz);  // disconnect from server
void oz_reset(OZ oz);  // reset server
char* oz_name(OZ oz);  // return server short name
char* oz_long_name(OZ oz);  // return server long name
void oz_speak(OZ oz,char* text);  // start speaking
int oz_is_speaking(OZ oz);  //
void oz_set_voice_gender_language(OZ oz,int vgl);
int oz_get_voice_gender_language(OZ oz);
void oz_set_volume(OZ oz,float v);
float oz_get_volume(OZ oz);
void oz_set_pitch(OZ oz,float p);
float oz_get_pitch(OZ oz);
void oz_gesture(OZ oz,int id);
int oz_is_gesturing(OZ oz);
void oz_set_gesture_weight(OZ oz,float w);
float oz_get_gesture_weight(OZ oz);
void oz_set_expression(OZ oz,int id);
int oz_get_expression(OZ oz);
void oz_set_expression_weight(OZ oz,float w);
float oz_get_expression_weight(OZ oz);
void oz_set_auto_blink_speed(OZ oz,float s);
float oz_get_auto_blink_speed(OZ oz);
void oz_set_random_head_weight(OZ oz,float w);
float oz_get_random_head_weight(OZ oz);
void oz_set_random_eyes_weight(OZ oz,float w);
float oz_get_random_eyes_weight(OZ oz);
void oz_set_random_face_weight(OZ oz,float w);
float oz_get_random_face_weight(OZ oz);
void oz_set_servo(OZ oz,int id,float pos);
float oz_get_servo(OZ oz,int id);
void oz_set_all_servos(OZ oz,float* pos);
void oz_get_all_servos(OZ oz,float* pos);
void oz_set_direct_weight(OZ oz,float w);
float oz_get_direct_weight(OZ oz);
void oz_set_lookat(OZ oz,int id);
int oz_get_lookat(OZ oz);
void oz_set_lookat_weight(OZ oz,float w);
float oz_get_lookat_weight(OZ oz);
```

C++ Qt5 Client

This is a client that can integrate seamlessly with Qt applications in C++. The API defines the following:

```
struct Utterance
{
    int options;
    struct
    {
        float confidence;
        char text[1024];
    } option[16];
    Utterance();
    Utterance(const Utterance& a);
    Utterance& operator=(const Utterance& a);
};

struct Faces
{
    int faces;
    struct
    {
        float confidence;
        float x,y;
        float xsize,ysize;
        char identity[1024];
    } face[16];
    Faces();
    Faces(const Faces& a);
    Faces& operator=(const Faces& a);
};

class QClient1 : public QObject
{
    Q_OBJECT

public:
    QClient1();
    ~QClient1();

    // network connection
    void connectToHost(const QString& hostName,quint16 port,QIODevice::OpenMode openMode =
QIODevice::ReadWrite,QAbstractSocket::NetworkLayerProtocol protocol = QabstractSocket::AnyIPProtocol);  // connect to server
    void connectToHost(const QHostAddress& address,quint16 port,QIODevice::OpenMode openMode = QIODevice::ReadWrite);  // connect to
server
    bool waitForConnected(int msecs = 30000);  // wait for connection to establish
    void disconnectFromHost();                 // disconnect from server

    // administrative
    void reset();        // reset server
    QString name();      // return short name of server
    QString longName();  // return long name of server

    // speaking
    void speak(QString& text);             // speak text
    void waitForSpeak();                   // wait for speech to finish
    bool speaking();                       // return true if server is speaking
    void setVoiceGenderLanguage(int vgl);  // set new voice/gender/language
    int voiceGenderLanguage();             // get current voice/gender/language
    void setVoiceVolume(float volume);     // set new volume
    float voiceVolume();                   // return current volume
    void setVoicePitch(float pitch);       // set new pitch
    float voicePitch();                    // return current pitch

    // gestures
    void gesture(int id);                 // start gesture
    void waitForGesture();                // wait for gesture to finish
    bool gesturing();                     // return true if gesture is playing
    void setGestureWeight(float weight);  // set new gesture weighing factor
    float gestureWeight();                // return current gesture weighing factor

    // facial expression
    void setExpression(int expr);            // set new facial expression
    int expression();                        // return current facial expression
    void setExpressionWeight(float weight);  // set new expression weighing factor
    float expressionWeight();                // return current expression weighing factor

    // automatic motion
    void setAutoBlinkSpeed(float speed);     // set new automatic blink speed
    float autoBlinkSpeed();                  // return current automatic blink speed
    void setRandomHeadWeight(float weight);  // set new random head motion weighing factor
    float randomHeadWeight();                // return current random head motion weighing factor
    void setRandomEyesWeight(float weight);  // set new random eyes motion weighing factor
    float randomEyesWeight();                // return current random eyes motion weighing factor
    void setRandomFaceWeight(float weight);  // set new random face motion weighing factor
    float randomFaceWeight();                // return current random face motion weighing factor

    // direct servo control
    void setServo(int id,float pos);     // set new direct servo position
    float servo(int id);                 // return currently set direct servo position
    void setAllServos(float* pos);       // set all direct servo positions
    void allServos(float* pos);          // return all direct servo positions
    void setDirectWeight(float weight);  // set new direct servo position weighing factor
    float directWeight();                // return current direct servo position weighing factor

    // looking at faces
    void setLookAt(int target);          // set new lookat target
    int lookAt();                        // return current lookat target
    void setLookAtWeight(float weight);  // set new lookat weighing factor
    float lookAtWeight();                // return current lookat weighing factor

    // signals
signals:
```

```
    void connected();                                   // triggered when connection is established
    void disconnected();                                // triggered when connection is lost
    void speakStart();                                  // triggered when speaking starts
    void speakDone();                                   // triggered when speaking is done
    void gestureStart(int id);                          // triggered when gesture starts
    void gestureDone();                                 // triggered when gesture is done
    void speakGesture(int id);                          // triggered when speech starts a gesture
    void utterance(Utterance& a);                       // triggered when server hears new utterances
    void faces(Faces& a);                               // triggered when server sees new faces
    void jpegVideoFrame(unsigned char* jpeg,int length);  // triggered when new eye camera frame arrives
};
```

Java Client

This is a straightforward Java client. The API defines the following:

C# Client

This is a straightforward Mono/C# client. The API defines the following:

Python Client

This is a straightforward Python client. The API defines the following:

Node.js Client

This is a Javascript client for use with node.js servers. The API defines the following:

Java Android Client

This is a client that can integrate seamlessly with Android applications. The API defines the following:

The Importantest Main Header File With Constants And Definitions And Shit

This is the main C/C++ header file used in this version of the protocol. All client APIs, as well as server implementations are derived from it.

```
// OZ - Robot Puppeteering System Client Protocol, up to version 1.0
// (C) Copyrights by Desmond Germans
// 2014 Germans Media Technology & Services

#ifndef _OZ_PROTOCOL_H_
#define _OZ_PROTOCOL_H_


namespace oz
{


// the port
enum
{
        OZ_PORT = 7331
};


// servo ids
enum
{
    // body
    SERVO_WAIST = 0,

    // left arm
    SERVO_SHOULDER_PITCH_LEFT,
    SERVO_SHOULDER_ROLL_LEFT,
    SERVO_ELBOW_YAW_LEFT,
    SERVO_ELBOW_PITCH_LEFT,

    // right arm
    SERVO_SHOULDER_PITCH_RIGHT,
    SERVO_SHOULDER_ROLL_RIGHT,
    SERVO_ELBOW_YAW_RIGHT,
    SERVO_ELBOW_PITCH_RIGHT,

    // left leg
    SERVO_HIP_ROLL_LEFT,
    SERVO_HIP_YAW_LEFT,
    SERVO_HIP_PITCH_LEFT,
    SERVO_KNEE_PITCH_LEFT,
    SERVO_ANKLE_PITCH_LEFT,
    SERVO_ANKLE_ROLL_LEFT,

    // right leg
    SERVO_HIP_ROLL_RIGHT,
    SERVO_HIP_YAW_RIGHT,
    SERVO_HIP_PITCH_RIGHT,
    SERVO_KNEE_PITCH_RIGHT,
    SERVO_ANKLE_PITCH_RIGHT,
    SERVO_ANKLE_ROLL_RIGHT,

    // head
    SERVO_NECK_YAW,
    SERVO_NECK_ROLL,
    SERVO_NECK_PITCH,
    SERVO_BROWS,
    SERVO_EYELIDS,
    SERVO_EYES_PITCH,
    SERVO_JAW,
    SERVO_EYE_YAW_RIGHT,
    SERVO_SMILE_LEFT,
    SERVO_EYE_YAW_LEFT,
    SERVO_SMILE_RIGHT,

    // left hand
    SERVO_WRIST_LEFT,
    SERVO_GRASP_LEFT,

    // right hand
    SERVO_WRIST_RIGHT,
    SERVO_GRASP_RIGHT,

    MAX_SERVOS  // last servo indicator, do not use
};


// voice/gender/language specifications
enum
{
    VGL_DEFAULT = 0
};


// gestures
enum
{
    GESTURE_BROW_RAISE_LONG = 0,
    GESTURE_BROW_RAISE_SHORT,
    GESTURE_FROWN_LONG,
    GESTURE_FROWN_SHORT,
    GESTURE_YES_LONG,
    GESTURE_YES_SHORT,
    GESTURE_YES_LONG_3X,
    GESTURE_YES_SHORT_3X,
```

```
    GESTURE_NO_LONG,
    GESTURE_NO_SHORT,
    GESTURE_NO_LONG_3X,
    GESTURE_NO_SHORT_3X,
    GESTURE_INDIAN_WOBBLE,

    MAX_GESTURES
};


// facial expressions
enum
{
    EXPRESSION_NEUTRAL = 0,
    EXPRESSION_HAPPY,
    EXPRESSION_SAD,
    EXPRESSION_EVIL,
    EXPRESSION_AFRAID
};


// lookat targets
enum
{
    LOOKAT_FORWARD = 0,
    LOOKAT_DOWN,
    LOOKAT_UP,
    LOOKAT_LARGEST,
    LOOKAT_SMALLEST,
    LOOKAT_AWAY
};


// messages
enum
{
    MSG_NOTHING = 0,

    // admin
    MSG_REQUEST_VERSION,  // request particular version of the API, dword = version (0x01000000 = 1.0), robot returns version as
answer, followed by name and long name
    MSG_RESET,            // reset robot

    // speaking
    MSG_SPEAK,                  // speak, text = utterance
    MSG_VOICE_GENDER_LANGUAGE,  // set voice/gender/language, dword = VCL_* (default = VCL_DEFAULT)
    MSG_VOICE_VOLUME,           // set voice volume, float = volume (0..1, default = 1)
    MSG_VOICE_PITCH,            // set voice pitch, float = pitch (0..2, default = 1)

    // gesturing
    MSG_GESTURE,         // manually trigger gesture, dword = GESTURE_*
    MSG_GESTURE_WEIGHT,  // set gesture weight, float = weight (0..1, default = 1)

    // facial expression
    MSG_EXPRESSION,         // set facial expression, dword = EXPRESSION_* (default = EXPRESSION_NEUTRAL)
    MSG_EXPRESSION_WEIGHT,  // set facial expression weight, float = weight (0..1, default = 1)

    // automatic motion
    MSG_AUTO_BLINK_SPEED,    // set automatic blink speed, float = speed in Hz (0..10, default = 0, no blinking)
    MSG_RANDOM_HEAD_WEIGHT,  // set random head motion weight, float = weight (0..1, default = 0)
    MSG_RANDOM_EYES_WEIGHT,  // set random eyes motion weight, float = weight (0..1, default = 0)
    MSG_RANDOM_FACE_WEIGHT,  // set random face motion weight, float = weight (0..1, default = 0)

    // direct servo control
    MSG_SERVO,          // set servo position, dword = id, float = position (0..1, default = 0)
    MSG_ALL_SERVOS,     // set all servo positions, { float = position (0..1, default = 0) }
    MSG_DIRECT_WEIGHT,  // set direct servo control weight, float = weight (0..1, default = 0)

    // looking at faces
    MSG_LOOKAT,         // set lookat target, dword = LOOKAT_* (default = LOOKAT_FORWARD)
    MSG_LOOKAT_WEIGHT,  // set lookat weight, float = weight (0..1, default = 0)

    // events from robot
    MSG_SPEAK_DONE,     // speak is done
    MSG_GESTURE_DONE,   // gesture is done
    MSG_SPEAK_GESTURE,  // speak-triggered gesture, dword = GESTURE_*
    MSG_UTTERANCE,      // utterance found, dword = count, { float = confidence, text = utterance }
    MSG_FACES,          // new face recognition state, dword = count, { float = confidence, float = x, float = y, float = xsize,
float = ysize, text = identity }
    MSG_VIDEO_FRAME,    // single video frame from the robot, dword = length, data blob
};


}


#endif
```