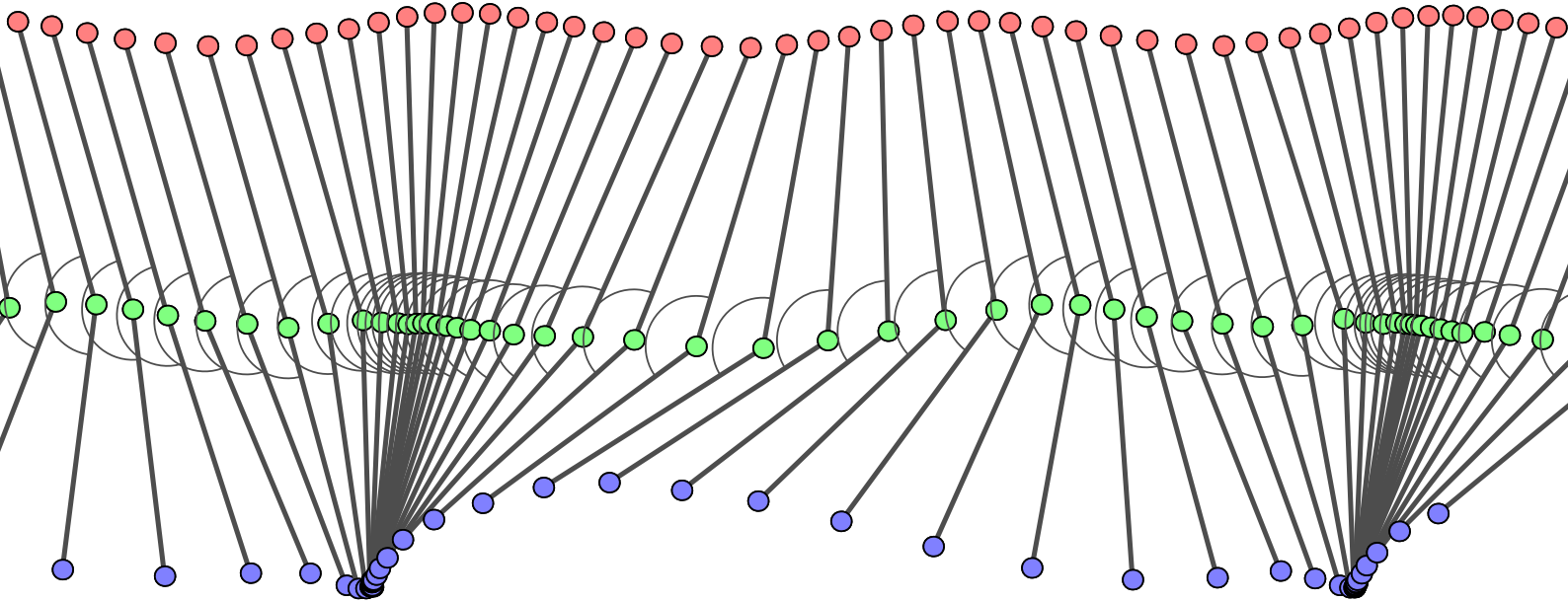


CHALMERS



Implementation of motion capture support in smartphones

*Master's Thesis in the Master Degree Programmes,
Computer Science: Algorithms, Languages and Logic and
Networks and Distributed Systems*

JOHANNES MARTINSSON
REIMUND TROST

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010

Implementation of motion capture support in smartphones

JOHANNES MARTINSSON
REIMUND TROST

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Implementation of motion capture support in smartphones

JOHANNES MARTINSSON
REIMUND TROST

© JOHANNES MARTINSSON, DECEMBER 2010.
© REIMUND TROST, DECEMBER 2010.

Examiner: Ulf Assarsson

CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
An illustration of a leg moving during gait analysis.

Department of Computer Science and Engineering
Göteborg, Sweden December 2010

Abstract

This thesis investigates the feasibility of developing cross-platform smart phone applications that utilize the Qualisys motion capture system. An Application Programming Interface that implements an in-house communication protocol is created for use on smart phones and other portable devices. Furthermore, two iPhone applications that utilizes the API are implemented — a viewfinder and a goniometer.

Keywords: motion capture, smartphone, iPhone, goniometer, visualisation

Sammanfattning

Denna rapport utreder möjligheterna att utveckla plattformsoberoende applikationer för smarta telefoner som utnyttjar Qualisys system för motion capture. Fokus är dock inriktat på utveckling för iPhone. Ett bibliotek för realtidskommunikation med Qualisys Track Manager beskrivs även. Slutligen presenteras två iPhone-applikationer som använder biblioteket — en sökare och en goniometer.

Contents

1	Introduction	1
1.1	Goals	1
1.2	Purpose	2
1.3	Scope	3
2	Background	5
2.1	Motion Capture	5
2.2	Motion Capture at Qualisys	8
2.3	iPhone application development	10
2.4	Android application development	11
2.5	Comparing iPhone and Android development	12
2.6	Developing cross-platform mobile applications	13
3	Previous Work	17
3.1	Motion capture related applications	17
3.2	Traditional use of goniometers	17
4	Implementation	19
4.1	Client library	19
4.2	The Viewfinder	20
4.3	The Goniometer	22
5	Result	29
5.1	Client library	29
5.2	Viewfinder	29
5.3	Goniometer	32
5.4	Discussion	32
6	Future work	35
6.1	Client library and the RTP	35
6.2	Improving the Viewfinder	35
6.3	Improving the Goniometer	36
6.4	Other end-user applications	37
A	Client library specification	42

Abbreviations

AIM	Automatic Identification of Markers
API	Application Programming Interface
BSD	Berkeley Software Distribution
CLAPI	client library API
CSS	Cascading Stylesheets
EMG	Electromyography
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IR	Infra-red
JNI	Java Native Interface
JVM	Java Virtual Machine
LED	Light-Emitting Diode
NDK	Native Development Kit
OHA	Open Handset Alliance
QTM	Qualisys Track Manager
RTP	Real-Time Protocol
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

1 Introduction

The developments made in computer hardware have made it possible for motion capture systems to go from large stationary computers with relatively small setups, to being able to run large setups on laptop computers. This enables more flexible and portable installations. The next step in the development towards even more portable systems is to take advantage of the latest technology in even smaller computers, such as netbooks, tablet PCs and smart phones.

Qualisys¹ is a company that develops camera-based motion capture systems and are now investigating uses for current ultra-portable computers. A full implementation of the current motion capture system in such ultra-portable devices is not being considered, much due to their small screens and limited computational power. However, there is interest in developing end-user applications with very specific functionality where the user can take advantage of the portability these devices offers.

It is mainly applications for the Apple iPhone² that is currently wanted. Future support for other platforms such as the Android³ platform is also under consideration.

This thesis investigates the feasibility of developing cross-platform smart phone applications that utilize the Qualisys motion capture system. The main focus is however on iPhone applications. An Application Programming Interface (API) that implements an in-house communications protocol is created for use on smart phones and other portable devices. Furthermore, two iPhone applications that utilizes the API are implemented — a virtual viewfinder and a virtual goniometer.

1.1 Goals

The thesis project have a number of goals:

¹ For more information about Qualisys, visit <http://qualisys.com>.

² More information on the iPhone can be found at <http://www.apple.com/iphone/>.

³ Android is an open source platform for mobile devices developed by the Open Handset Alliance.

- Implement an API that implements Qualisys' Real-Time Protocol (RTP). The RTP allows clients to stream data in real-time from Qualisys Track Manager (QTM)⁴ software and offers some remote control features. The API should expose the 2D position of markers from all cameras in the current setup, the 3D position and orientation of the markers, the position of rigid bodies as well as analog data, which for example can come from force plates.
- Develop a viewfinder application for viewing 2D markers and video in real-time, as seen by the cameras. The purpose of the Viewfinder application is to be a visual aid during setup and calibration of Qualisys' motion capture system. During setup, cameras are placed and oriented so that they cover a specific area. In order to get optimal capture data it is important that the cameras are oriented so that the captured objects are covered by as many cameras as possible. To avoid occluding objects, cameras are often placed at some height where ladders are required. When setting up cameras in such occasions, it would be of great aid to have a portable viewfinder that shows the view of the camera currently being set up, which is the main motivation for the viewfinder application.
- Develop an end-user application that works as a virtual goniometer. A goniometer is a tool for measuring angles. It is typically used by gait analysts to measure angles on the human body. A common task is to measure the angle at the knee joint of a person during a walk cycle. The application should present the measured angle as well as the minimum and maximum angles recorded during a measurement. The purpose of the goniometer application is not to replace traditional goniometers but rather as a compliment to current tools. The goniometer should have an intuitive GUI and the amount of preparations needed in order to use the application should be minimized.
- Investigate how one can develop applications targeting both the iPhone and Android platforms.

1.2 Purpose

The purpose of this thesis is:

- To learn how to design and develop an API for current handheld devices that communicates with Qualisys real-time protocol.
- To learn how to design and develop smart phone applications, which uses the API.

⁴ For more information on QTM, see the product sheet at http://www.qualisys.com/archive/product_information_pdf/PI_QTM.pdf.

- To fulfill the goals outlined in the Goals section.

The purpose from Qualisys' point of view is to:

- Be able to offer clients an API for application development on smart phones.
- Make setup of the motion capture system easier by being able to view a live stream of marker positions in a hand held device.
- Be able to offer the market's first end-user application for a motion capture system on a smart phone.

1.3 Scope

This project consists of a brief investigation, a large implementation part and this report. In the investigation, available methods for developing cross-platform application for portable devices will be explored. The implementation work makes up the main work of this thesis and will be limited to developing a cross-platform API and two applications for the iPhone.

2 Background

The purpose of this chapter is to introduce the reader to contemporary motion capture techniques and to investigate possible methods for creating cross-platform applications for the iPhone and Android platforms. Moreover a brief comparison between iPhone and Android development is presented.

To conclude, some final thoughts are presented regarding the most suitable cross-platform solution for the purposes of this thesis.

2.1 Motion Capture

Motion capture is the process of recording the position and orientation of objects in a computer-usable form. A common task is to record the motion of humans and animals for use in fields of medicine, sports and entertainment. The motion data can for example be used for gait analysis in biomechanics (Cloete and Scheffer, 2008 and Corazza et al., 2006) and movement optimization of athletes in sports applications (Fradet et al., 2003, Nathan, 2008, Bideau et al., 2004 and Ghasemzadeh and Jafari, 2010). In entertainment, motion capture is typically used for character animation in games and movies (see [figure 2.1](#) for an example). Other uses for motion capture include industrial, military and research applications.



Figure 2.1 Facial motion capture in the movie Avatar.

Motion capture have attracted a lot of research interest since the 1970s and a number of motion capture systems have since been developed. Various techniques have been employed in these systems including mechanical, optical, magnetic, inertial, ultrasonic and hybrids (Bachmann, 2000, Cloete and Scheffer, 2008 and Zhang et al., 2009). The systems can be divided into two categories: marker based and markerless (Corazza et al., 2006).

2.1.1 Marker based solutions

In marker based motion capture, special objects, called markers are placed on the subjects. An object makes a good marker if it's feasible to compute its position, e.g. by applying image analysis on a captured video frame. Motion data is then acquired by computing the position of markers over time. Marker based optical motion capture using reflective markers (see [figure 2.2](#)) is presently the most common technique (Corazza et al., 2006) and it is the approach used at Qualisys.



Figure 2.2 Passive, reflective markers.

A marker is said to be active if it emits a signal in order to announce its position. For example, some active markers uses LED lights to make them more visible on camera. Active markers typically need a power source in order to function, which can make them unsuitable for some situations. Markers that don't emit any signal are called passive markers and are typically cheaper than active markers.

However, several non-optical methods exist where other means of acquiring motion data than using video cameras are used. Methods using magnetic sensors, inertial sensors (also called inertial navigation system) or a combination of the two (hybrid systems) are examples of non-optical alternatives. In these solutions, sometimes referred to as magnetic/inertial orientation tracking systems, there is a sensor in each marker that senses the position and orientation of the marker (see [figure 2.3](#)). Eric Robert Bachmann presents a method where each sensor unit contains a three-axis magnetometer, a three-axis angular rate sensor and a three-axis

accelerometer (Bachmann, 2000). These nine-axes sensors are referred to as MARG (Magnetic, Angular, Rate, Gravity) sensors. The main advantage of non-optical solutions is that they are not as likely to suffer from occluding objects.



Figure 2.3 Xsens MVN motion capture solution using inertial sensors.

However, inertial/magnetic systems have their own drawbacks, such as the drift problem, i.e. small errors that propagate over time. To accommodate for the drift problem, drift compensation is typically applied (Cloete and Scheffer, 2008). Another problem with magnetic/inertial systems is that magnetometers can suffer from interference from metallic objects and electrical sources in the environment that affect the magnetic field (Zhang et al., 2009).

2.1.2 Markerless solutions

As the name implies, there are no markers used in the markerless approach. Instead, movements are computed by other means, e.g. by applying advanced image analysis on regular video frames (Deutscher et al., 2002). Since markerless motion capture can be achieved with a simple video camcorder, without having subjects wear any extra equipment, it is a rather inexpensive method (Deutscher et al., 2002). However, it typically suffers from lower accuracy and slower update rates than marker based solutions, such as a marker based optical system (Corazza et al., 2006). Although, depending on the application, this might not be an issue. For example, a store owner can use a markerless motion capture solution to count the number

of people entering a store by using an existing surveillance camera (Björgvinsson, 2006).

2.2 Motion Capture at Qualisys

Qualisys takes the optical, marker based approach to motion capture. Their system works with both active and passive markers which are recorded by the Qualisys Oqus⁵ cameras. The passive markers consists of retroreflective spheres available in various sizes. The Oqus camera is equipped with Infra-red (IR) LEDs that are placed around the lens of the camera. During measurement, the LEDs are flashing with a certain frequency. The IR-light hits the spherical markers and the light is reflected back to the cameras, projecting bright, circular shapes on the camera's image sensor. The cameras compute the center point of each circle in terms of x- and y-coordinates on its image sensor, and transmits this information in real-time to a host computer running Qualisys Track Manager (QTM). Based on these coordinates QTM can then compute the position of each marker in three dimensions.



Figure 2.4 Gait analysis on a horse using Qualisys' Motion Capture System.

The cameras can also capture full-frame video with high frame rates, for example 1.3 megapixel images in 500 frames per second. There are a number of cameras available with different resolutions, frame rate and accessories. A measurement setup consists of one or more cameras, a computer running QTM and possibly other accessories such as force plates, EMG equipment et cetera.

⁵ A detailed specification of the camera is available on http://www.qualisys.com/archive/product_information_pdf/PI_Oqus.pdf.



Figure 2.5 Qualisys Oqus 300 camera.

2.2.1 Qualisys Track Manager

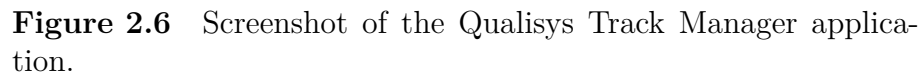
Qualisys Track Manager (QTM) is the host application used with Qualisys' motion capture hardware. The most fundamental function of QTM is real-time marker tracking, i.e. computing each marker's position in the three-dimensional measurement space. These computations are based on the markers' two-dimensional coordinates that QTM receives from the cameras in the system, and are performed using triangulation (Horprasert et al., 1998). The resulting 3D positions are shown in a 3D viewport (see [figure 2.6](#)) that the user can navigate through.

All common tasks like starting and stopping measurements, setting camera exposure and so on can be done in QTM. In addition, the software offers various other features such as plotting graphs and showing live video feeds from connected cameras.

2.2.2 The Real-Time Protocol

QTM also acts as a server which implements an in-house Real-Time Protocol (RTP). The protocol provides a variety of commands that enables third party applications to communicate with QTM over TCP and UDP. Typically, the RTP is used to stream tracked data for use in another application, e.g. Visual3D⁶ and MATLAB (Fehlhaber and Holmberg, 2009).

⁶ The C-Motion Visual3D software is used for biomechanical motion analysis. More information at <http://c-motion.com/products/Visual3D.htm>.



2.3 iPhone application development

There are two types of applications for the iPhone: native applications and web applications (commonly referred to as web apps). Web apps are created using web technologies such as HTML, CSS and Javascript and are run inside the web browser on the phone. Web apps for the iPhone are like ordinary web applications except that they can make use of Apple's proprietary Javascript APIs for the iPhone.

Native iPhone applications on the other hand are mostly written in Objective-C (Wikipedia, 2010a) and compiled and are executed by the iPhone operative system. Native applications have fewer limitations, and can potentially offer higher performance than web applications. For example, access to device functionality like the camera and accelerometer, are currently only available to native applications (Apple, 2010a).

2.3.1 Native applications

Native iPhone applications are mainly written in the Objective-C programming language, which is a strict superset of the C programming language. A description of the language is given on Wikipedia (Wikipedia, 2010a):

Objective-C is a reflective, object-oriented programming language which adds Smalltalk-style messaging to the C programming language.

Objective-C can be seen as a thin layer on top of C that adds object orientation to the language. The object syntax is derived from Smalltalk⁷ and all of the non-object related syntax is identical to that of C.

2.3.2 Programming environment

iPhone development is usually done in Xcode⁸, Apple's Integrated Development Environment (IDE), using the iPhone Software Development Kit (SDK)⁹. Xcode contains an iPhone simulator that lets developers test their software without deploying it on the device. The IDE also contains various tools for debugging and measuring memory and cpu consumption.

Applications can also be written in C and C++. However, most iPhone APIs are written in the Objective-C programming language. By enabling developers to partially write applications in C and C++, parts of an application can be made cross-platform.

2.4 Android application development

The Android platform consists of an open source¹⁰ operating system and a set of tools and frameworks for portable devices developed by the Open Handset Alliance (OHA). Android applications are written in the Java programming language and are executed on the Dalvik virtual machine¹¹.

Application developers can use the Android SDK¹², which provides several APIs. In addition to the Android SDK, the OHA also provides a Native Development

⁷ The Smalltalk programming language: <http://www.smalltalk.org>.

⁸ For more information on Xcode, see <http://developer.apple.com/tools/xcode/>.

⁹ The iPhone Software Development Kit is available on <http://developer.apple.com/iphone/>.

¹⁰ Definition available on <http://www.opensource.org/docs/osd>

¹¹ A JVM optimized for running on portable devices (Wikipedia, 2010b).

¹² The Android SDK is available on <http://developer.android.com/sdk/>.

Kit¹³ (NDK). The NDK is a companion tool to the SDK and lets developers build performance-critical portions of their applications in C or C++, these portions will not be executed on the Dalvik virtual machine. The use of the NDK should be limited to certain situations—including cross-platform support—according to the OHA (Alliance, 2010):

Typical good candidates for the NDK are self-contained, CPU-intensive operations that don't allocate much memory, such as signal processing, physics simulation, and so on. Simply re-coding a method to run in C usually does not result in a large performance increase. The NDK can, however, can be an effective way to reuse a large corpus of existing C/C++ code.

2.5 Comparing iPhone and Android development

An important difference between the iPhone and Android platforms is that the former is based on proprietary software while the latter is based on and released as open source.

2.5.1 Platform implications

At present, you can only develop applications for the iPhone using the Xcode programming environment—which is only available on the Mac OS X operating system—effectively limiting iPhone development to Apple computers¹⁴. The system requirements¹⁵ for Android application development is not as severe.

In order to deploy an application on a device running the iPhone OS¹⁶, the developer must sign-up for the Apple Developer Program, currently priced from \$99 annually. In comparison, developing for the Android platform is free of charge¹⁷.

Another difference is that Apple enforces strict policies on application content and implementation, which must be adhered for an application to be accepted into the Apple App Store. Such policies does not exist for applications submitted to the Android Market.

¹³ The Android NDK is available on <http://developer.android.com/sdk/ndk/>.

¹⁴ Further, the iPhone simulator is only available for Apple's intel-based computers.

¹⁵ For details see <http://developer.android.com/sdk/requirements.html>.

¹⁶ iPhone OS is the operating system running on the Apple iPhone, iPod touch and iPad.

¹⁷ Submitting applications to Android stores is not necessarily free. At present, the only existing Android application store, Android Market, requires developers to pay a one-time fee of \$25. However, other Android stores could have different regulations.

While the implementation specific differences of the two platforms are considerable—much due to the use of different programming languages—both offer similar tools and functionality for developers. In addition, both platforms make it possible to develop applications using the native-code languages C and C++. However, due to Objective-C's lineage, programming in C and C++ for the iPhone has less overhead.

2.6 Developing cross-platform mobile applications

Maintaining different versions of an application for different platforms can be expensive and cumbersome, therefore it is desirable to develop applications that are compatible with several platforms, so called cross-platform applications. Obvious advantages of such applications being shorter development time and lower maintenance costs.

There are several ways to make cross-platform applications for mobile phones. However, because of the different programming languages and APIs of mobile platforms, developing cross-platform applications is not a trivial task. Various tools are available to alleviate development of cross-platform applications, e.g. Appcelerator Titanium Mobile, PhoneGap and Rhomobile Rhodes¹⁸.

2.6.1 Intermediate compatibility layers

Some of the mentioned tools facilitate development of cross-platform applications by use of *intermediate compatibility layers*. These are APIs that add a layer of abstraction on top of the APIs available for each supported mobile platform. The compatibility layer will call different API functions depending on which platform the application is executed on.

There are several disadvantages of using such compatibility layers. One being that they might not always be up to date with the supported platforms. Hence, whenever a new feature is introduced by one of the platform vendors, it will take some time before developers can take advantage of that feature. Moreover, some features (especially ones that are not available on all supported platforms) might not be made available at all. Relying on a third party to provide these compatibility layers also adds a level of risk; development might be discontinued et cetera. Another disadvantage with compatibility layers is that they are not always suitable for CPU-intensive tasks such as signal processing and physics simulation, since it might not be possible to fully utilize the underlying platform.

¹⁸ For more information about Titanium Mobile, PhoneGap and Rhodes, see <http://www.appcelerator.com>, <http://phonegap.com> and <http://rhomobile.com>.

2.6.1.1 Legal uncertainty on iPhone

Recent updates to the iPhone Developer License Agreement (Gruber, 2010) have raised a lot of attention on whether it is allowed to create applications for the iPhone using third party frameworks such as Titanium and PhoneGap. Section 3.3.1 (Apple, 2010b) now reads:

Applications may only use Documented APIs in the manner prescribed by Apple and must not use or call any private APIs. Applications must be originally written in Objective-C, C, C++, or JavaScript as executed by the iPhone OS WebKit engine, and only code written in C, C++, and Objective-C may compile and directly link against the Documented APIs (e.g., Applications that link to Documented APIs through an intermediary translation or compatibility layer or tool are prohibited).

At the time of this writing, it is still unclear how the new license affects the use of frameworks such as Titanium and PhoneGap. However, it does give a hint that the future of these tools are very much in the hands of Apple.

2.6.2 Other solutions

A possible solution to the cross-platform problem is to write applications in languages such as C and C++ using a subset of libraries available on all the target platforms. This, however, might not always be a viable solution because the subset of libraries might not be enough to develop the application. It is not uncommon that each platform has a different API for development of graphical user interfaces, which leaves this solution unable to implement a complete application. However, it could still be viable to implement parts of an application that are logically separable from other parts that are implemented in a platform specific manner.

2.6.3 Cross-platform development conclusions

All things considered, we decided not to use any intermediate compatibility layers. Instead, a selected part of each application was made cross-platform by writing it in the C programming language using a common subset of libraries. This approach enabled us to develop an API that handles the communication with QTM via its RTP. The API is provided to our applications as a static C library, which can be used

directly when developing applications for the iPhone, but which must be wrapped in a JNI-wrapper for Android application development.

We acknowledged that it was not in our interest to invest time in learning a third party intermediate compatibility layer framework that later might turn out to be unsatisfactory for the purposes of this thesis.

3 Previous Work

This chapter describes some related work that this thesis builds upon.

3.1 Motion capture related applications

We were unable to find any smartphone applications related to motion capture created prior to the work conducted in this thesis. Considering that motion capture is a relatively small field of technology and that smartphones are relatively new¹⁹, it's rather unsurprising that no such applications existed beforehand.

3.2 Traditional use of goniometers

There are various kinds of goniometers, each with a specific area of use. For example, there are advanced goniometers specialized in light measurements. For measuring angles on the human body however, gait analysts often use a simple tool that consists of two elongated shapes joined at a joint, as illustrated in [figure 3.1](#). This tool is used by manually rotating the elongated shapes around the joint so they align with two adjoining bone segments on a patient; an effective method for static analysis on a stationary patient — on a moving patient however — it's difficult to get anything more than a rough estimation of the angle in question.

To effectively get accurate data on moving patients, many gait analysts currently take advantage of motion capture technology. The ability to have corresponding tools in ultra portable devices such as smartphones can give greater flexibility.

¹⁹ The first smartphone, Simon, was designed by IBM and released in 1993 (Wikipedia, 2010c).

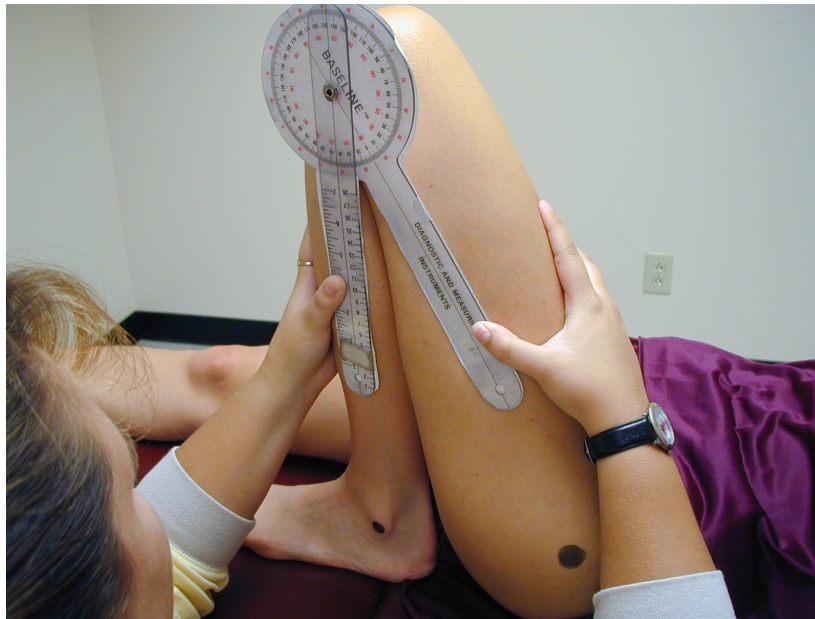


Figure 3.1 A traditional goniometer used for gait analysis.

4 Implementation

This chapter describes implementation details of the client library API (CLAPI), the viewfinder and the goniometer applications. The CLAPI enables real-time communication with the Qualisys Track Manager (QTM) software, and is used by both applications.

4.1 Client library

The client library provides an API to Qualisys' Real-Time Protocol (RTP) and is implemented as a C library. The library can be compiled with most C compilers adhering to the C99 standard, and it is compatible with both the iPhone and Android platforms. On the Android platform an additional JNI wrapper²⁰ is needed to interface with the library. On the iPhone platform the developed application only needs to link against the library. The principal benefit of using the library is to avoid having to implement the network communication and bit unpacking needed to communicate over the RTP, leaving the developers able to focus on the application in development instead.

The main use of the library is retrieving motion capture data from QTM but offers a few other features as well. It can for instance be used to start and stop captures remotely. Captured data is contained in *data frames*, which are a binary representation of the actual data and various headers for housekeeping. Data frames can either be retrieved one at a time or as a stream.

4.1.1 Network communication

Protocol communication is done over TCP, although streaming data frames can be done over UDP to increase performance. The CLAPI use standard BSD sockets to implement the network communication.

²⁰ Java Native Interface is a framework that enables code running in a Java Virtual Machine to call and be called from code running outside of the JVM.

The RTP is mostly plain-text and synchronous, e.g. to get the current version of QTM one sends the command `QTMVersion` to the server, which responds with, for instance, `QTM version is 2.3 (build 464)`. However, not all communication is plain-text and synchronous: data frames are not plain-text and can be streamed. For example, to get the latest data frame, the command `GetCurrentFrame` is sent, as described below.

```
GetCurrentFrame All | ([2D] [2DLin] [3D] [3DRes] [3DNoLabels]
[3DNoLabelsRes] [Analog] [Force] [6D] [6DRes] [6DEuler] [6DEulerRes])
```

The parameters specify which components (see [table 4.1](#)) should be included in the requested data frame. To get a stream of data frames the command `StreamFrames` is sent upon which the server responds by starting to stream data frames.

4.2 The Viewfinder

The viewfinder application is an iPhone application that works as a virtual viewfinder for Qualisys' motion capture cameras. The aim of the application is to ease the setup of a motion capture system.

4.2.1 Scope

The viewfinder implementation is limited to showing a real-time view of markers from a camera's point of view. An user can choose which camera to use, but only one camera view can be seen at a time. The number of markers currently tracked by the selected camera is presented on screen, as is the cameras identification number.

Originally, live video stream support was also planed, but due to limitations in QTM and in the RTP, such video support has not been implemented.

4.2.2 Implementation

The viewfinder uses the CLAPI in order to receive a stream of markers with their respective 2D position for each camera from QTM. Information about each camera's resolution is used so that markers can be presented in the correct position on screen. This is because the RTP defines 2D marker positions to be in absolute pixels, and the target resolution on the iPhone differs from the resolution of the cameras. The markers are drawn in real-time using OpenGL ES 1.1²¹.

There are separate threads for receiving data frames and for rendering the received markers on screen. Users can choose between having the markers presented

²¹ OpenGL ES is a light weight version of OpenGL for use on embedded systems.

Type ID	Name	Description
2	3DnoLabels	Unidentified 3D marker data
10	3DnoLabelsRes	Unidentified 3D marker data with residuals
3	Analog	Analog data from one or more analog devices
4	Force	Data from one or more force plates
5	6D	6D data position and rotation matrix
11	6DRes	6D data position and rotation matrix with residuals
6	6DEuler	6D data position and Euler angles
12	6DEulerRes	6D data position and Euler angles with residuals
7	2D	2D marker data
8	2DLin	Linearized 2D marker data

Table 4.1 A table showing the various component types defined by the RTP.

in a rectangle with the same aspect ratio as the selected camera, or using the whole screen to present markers. The latter option makes better use of the device's small screen, but can be somewhat misleading when setting up the system since the markers relative position to each other is not preserved.

4.3 The Goniometer

The goniometer application is a virtual goniometer for measuring angles, its intended use being measuring leg angles during gait analysis. It works by computing the angle between markers tracked by Qualisys Track Manager. The marker data, is streamed to the application via the CLAPI.

This section presents some implementation details and a few important design decisions that affect the amount of preparation needed in order to start using the application. The decisions also affect the range of angles that can be measured by the application.

4.3.1 Requirements

The requirements of the goniometer application was that it should:

- Compute and show the angle between upper and lower legs of a person in real-time.
- Compute and show the minimum and maximum angles during an ongoing walk cycle.
- Require a minimal amount of preparation.
- Draw a graph of the measured angle that is updated in real-time.

4.3.2 Scope

The goniometer application is meant as a tool for gait analysts, as such, its main use will be to measure the angle at knee joints of humans. However, the implementation in this thesis is limited to compute an approximate angle and not a biomechanically correct angle. To compute a biomechanically correct angle the true joint centre of rotation must be found. This limitation can be considered reasonable since the measurements made by gait analysts using traditional goniometers are also only approximations.

4.3.3 Unlabeled markers

In QTM, markers can be either *labeled* or *unlabeled*. When a marker is labeled, it is given a name so that it can be identified across frames. Unlabeled markers on the other hand cannot be identified across frames in a trivial way.

In order to compute a consistent angle, markers must be properly identified. If they aren't, and the order of markers in a data frame²² changes, it means that at least one marker will be mistaken for another, resulting in the wrong angle being calculated.

Thus, the advantage of using labeled markers is obvious. However, to make use of this advantage, an Automatic Identification of Markers (AIM) model must be setup and used in QTM. Since this requires more preparation work for the user — work that should be kept to a minimum according to the requirements — we decided to use unlabeled markers even though the obvious disadvantage.

Because of this decision, a method of identifying markers must be found and implemented.

4.3.4 Marker configuration

The simplest possible marker configuration for our purposes is a three marker setup — one marker at the hip, knee and ankle respectively. This is the minimum requirement of our application in terms of marker configuration. The angle measurement of the three-marker setup is however limited to an angle in the interval $[0, \pi]$ as only the smallest of the two possible angles are measured, as illustrated in [figure 4.1](#). But since some joints might bend with an angle $> \pi$ it is desirable to be able to measure such angles as well. Therefore, we decided to allow a second marker configuration, where a fourth marker — referred to as the m control marker — is used to determine whether an angle is in $[0, \pi]$ or $(\pi, 2\pi]$. The control marker is placed on the lower leg (see [figure 4.2](#)) and is used to determine the direction of the leg.

4.3.5 Identifying markers

A marker identification algorithm is executed whenever the application receives new marker data, i.e. x, y and z coordinates. For simplicity, the algorithm will be described for the three-marker setup, but works correspondingly for the four-marker setup as well. The purpose of the algorithm is to find which marker corresponds to the hip, knee and ankle markers. For the algorithm to work, it must be initialized.

²² With unlabeled markers the RTP gives no guarantees that the order in which markers appear is consistent across data frames.

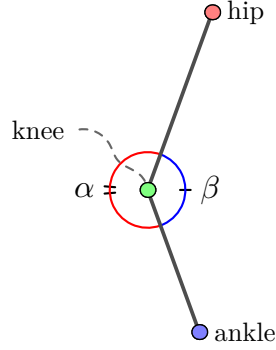


Figure 4.1 An example of a three-marker setup on a leg. Markers are placed on the hip, knee and ankle as indicated by the labels. At the knee joint, there are two possible angles α and β .

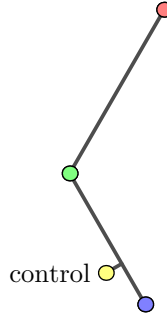


Figure 4.2 An example of a four-marker setup on a leg, with the addition of the control marker.

Initialization is performed by starting the measurement with all markers in an up-right position — i.e. each marker is identified by its vertical position — the highest marker gets assigned to the hip, the second highest to the knee and so on.

On initialization, the hip–knee and knee–ankle distances are saved. The distances are used to identify markers by finding the marker-pairs that best matches the stored distances. Ideally, these marker distances remain constant during a session, however due to gliding and inaccuracies of the tracked data, there will be slight variations in these distances over time.

Identification cannot be based solely on these distances, since for some angles, the data can be ambiguous. Consider the situation illustrated in [figure 4.3](#), where the markers make up an isosceles triangle. If the algorithm were solely based on distances, there are two possible markers that could be the knee joint. This would not be a problem if the tracked data were impeccable, since the two angles would be equal anyway. But due to slight tracking inaccuracies there can be jumps between

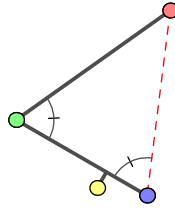


Figure 4.3 Markers forming an isosceles triangle.

the correct knee marker and an incorrectly chosen one when markers are close to forming an isosceles triangle. To accommodate for this, the position of the markers is also taken into account in the identification algorithm. This works by computing how far a marker have moved since the last frame—the further it has moved, the more unlikely it is to be the correct marker.

The algorithm performs an exhaustive search over all possible permutations of markers-joint combinations, and finds the one that best matches the distance and position criterion described above. It can be summarized with the following expression

$$\arg \min_{p \in \mathbf{P}} f(p)$$

where \mathbf{P} is the set of all marker permutations and $f(p)$ is a function that computes a fitness value of the given permutation. A fitness value of 0 is a perfect match, i.e. the hip–knee and knee–ankle distances are equal to their initialized counterparts, and the markers are in the same positions as the previous frame, i.e. they have not moved. For three markers ($n = 3$) $f(p)$ is defined as:

$$f(p) = \sum_{i=1}^n d(p_i, b_i)^2 + \sum_{i=1}^{n-1} (d(p_i, p_{i+1}) - c_i)^2$$

where d is a function that computes distance between two markers, b_i the best matched marker from previous frame and c_i the distances saved from the initialization.

4.3.6 Computing the angle

In the three-marker setup, with markers a , b and c representing the hip, knee and ankle, respectively, the angle at b is given by:

$$\alpha = \arccos \frac{\vec{ab} \cdot \vec{bc}}{|\vec{ab}| |\vec{bc}|} \quad (4.1)$$

Using this method the computed angle ϕ is in $[0, \pi]$ radians due to the definition of \arccos .

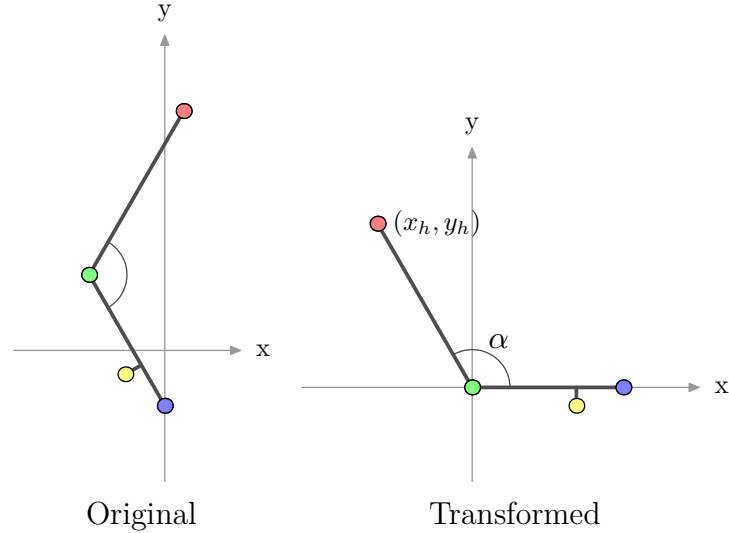


Figure 4.4 Markers in original and transformed coordinate system.

In the four-marker setup, the situation is a little more complex. Instead of using [equation 4.1](#) the angle is computed with [equation 4.2](#) in a coordinate system that is aligned with a plane spanned by the knee, ankle and control markers. This is illustrated with a simplified example (in \mathbb{R}^2 rather than \mathbb{R}^3) in [figure 4.4](#). The angle computed in the four-marker setup is given by

$$\alpha' = \begin{cases} \text{atan2}(x_h, y_h) + 2\pi & \text{if } \alpha < 0 \\ \text{atan2}(x_h, y_h) & \text{otherwise} \end{cases} \quad (4.2)$$

The `atan2` function²³ can only be used for points in \mathbb{R}^2 but since the interesting points are mostly in the same plane that the coordinate system is aligned with, the third coordinate can be ignored.

One problem with this method is that it is only perfect when the markers are placed in the same plane. It is therefore important that markers are placed carefully. Slightly misplaced markers will produce angles that are slightly off, which is acceptable since the purpose of the application is only to give an approximation of the biomechanical angle. To compute a biomechanically correct angle, a better approximation of the true joint center is needed.

²³ The `atan2(y, x)` function is a variation of the arctangent function that takes the quadrant of the point (x, y) into consideration. It is described at <http://en.wikipedia.org/wiki/Atan2>.

5 Result

This chapter presents the finished implementation work conducted in this thesis. For the client library API (CLAPI), an outline of the API specification is given (the complete specification can be found in [appendix A](#)). The finished iPhone applications are presented with screenshots.

5.1 Client library

The essential functionality of the client library API is summarised in [table 5.1](#). The client uses the `qtmAPIInit` function to get an initialised API `struct`. The `struct` is used as an argument in all subsequent library calls. After initialisation, the client connects by calling `qtmConnect`. If a connection is established successfully, the client can communicate with the server via the remaining functions.

5.2 Viewfinder

The finished viewfinder application is shown in [figure 5.1](#). The screenshot shows two people wearing at least 58 markers in total. Each marker is represented by a small circle, here shown in green. The number in the bottom right corner indicates which camera's view is displayed. In the top left corner, the current number of markers visible to the current camera is displayed.

To change camera view, one can either use a slider that pops up when the camera icon at the bottom left is tapped (as shown in [figure 5.2](#)), or by simply flicking to the left or right on the touch screen.

The application supports both landscape and portrait orientation — switching is done automatically whenever the orientation changes (as sensed by the integrated accelerometer). The user can also change a number of settings, e.g. the stream rate of data frames, marker size, marker colour and aspect ratio.



Figure 5.1 Screenshot of the viewfinder application running in landscape mode. The viewfinder is currently showing camera 1 which have tracked 58 markers for the current frame, as indicated in the top and bottom left corners.



Figure 5.2 Screenshot of the viewfinder application showing the slider used for changing camera view. The large number indicates the currently chosen camera.

Function	Description
<code>qtmAPIInit</code>	Get an initialised api struct.
<code>qtmConnect</code>	Connects to a QTM server.
<code>qtmShutdown</code>	Closes the current connection, if one exists.
<code>qtmVersion</code>	Tells the server which version of the protocol it should use.
<code>qtmQTMVersion</code>	Gets the QTM version of the server.
<code>qtmTakeControl</code>	Takes control of the RT interface, which is needed for some commands. Only one client can have control at the time.
<code>qtmReleaseControl</code>	Releases control of the RT interface.
<code>qtmByteOrder</code>	Gets the byte order used by the server.
<code>qtmCheckLicense</code>	Checks if a license key for QTM is valid.
<code>qtmNew</code>	Creates a new measurement.
<code>qtmClose</code>	Closes the current measurement.
<code>qtmStart</code>	Starts a new capture.
<code>qtmStop</code>	Stops the current capture.
<code>qtmCapture</code>	Downloads the latest capture as a C3D file.
<code>qtmGetParameters</code>	Gets the current QTM settings.
<code>qtmGetCurrentFrame</code>	Gets the current frame from the connected QTM server.
<code>qtmStreamFrames</code>	Streams data frames from the connected QTM server.

Table 5.1 The essential functions of the client library API.

5.3 Goniometer

The finished goniometer application is shown in [figure 5.3](#) and [figure 5.4](#). The application have two modes; one with a dial showing the current, minimum and maximum angles, another mode showing a graph with the current and recent angles. Switching between the modes can be done by either tapping the graph and done buttons or by turning the device between the vertical and horizontal positions.

When a three-marker setup is used, the application can only measure angles up to π radians. To indicate that only three markers are available and that measurements thus is limited to the interval $[0, \pi]$ radians, half of the dial graphics is greyed out. As soon as four markers are available, the entire dial is displayed in full brightness to indicate that an angle in the interval $[0, 2\pi]$ radians can be measured, as shown in [figure 5.3](#).

To reset the minimum and maximum angles, the user taps the stop button two times (the first tap halts the measurement, the second tap starts a new measurement).

5.4 Discussion

The work presented in this chapter works as expected and performs well. Most of the original requirements are met, the only notable exception being the lack of live video streaming support in the viewfinder application.

The client library API allows developers to write applications on portable devices that communicate with Qualisys QTM software through the RTP (although the client library API is by no means limited to portable devices).

Both applications use OpenGL ES to achieve adequate frame rates. During testing, the main bottle neck turned out to be the network communication. To get acceptable network performance, an ad-hoc WLAN is recommended. This allows a wireless peer-to-peer connection to be setup between the computer running QTM and the portable device.



Figure 5.3 Screenshot of the goniometer application in portrait mode. The red band around the meter indicates the range of angles measured.

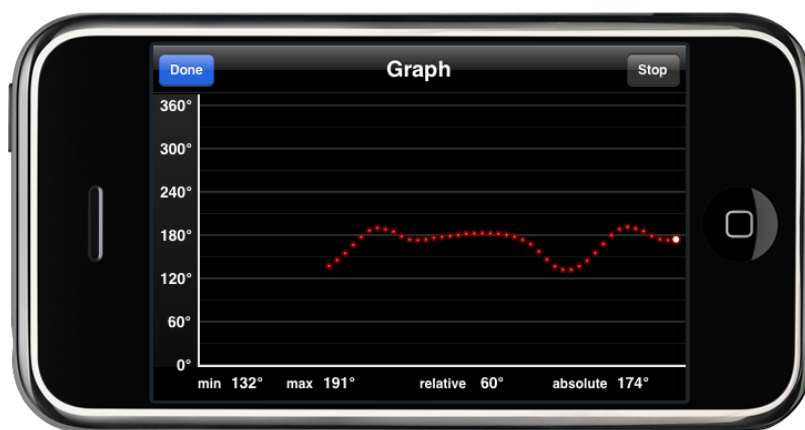


Figure 5.4 Screenshot of the goniometer application in landscape mode during a walk cycle. In this mode, a graph shows the measured angle for approximately the last second.

6 Future work

This chapter discusses possible improvements on the real-time API, the Viewfinder and the goniometer application.

6.1 Client library and the RTP

Both the client library and the Real-Time Protocol (RTP) could be extended to allow applications to change some settings and issue various commands in Qualisys Track Manager (QTM). This would make it possible to develop applications that can act as remote controls for some aspects of QTM. Support in the RTP for streaming live video would also be desirable.

6.2 Improving the Viewfinder

Three areas of interest for further improvement have been identified: live video, 3D view and remote control features. Some of these require extensions to both QTM and the RTP while others are ready to be implemented.

6.2.1 Live video streaming

Live video streaming is a desirable feature in the Viewfinder application, mostly because it would aid camera setup. The video view could also be augmented by an overlaid marker view. However, live video streaming require extensions of the RTP.

6.2.2 Remote control features

The Viewfinder application could be improved further by implementing remote control functionality to change camera lens parameters, i.e. aperture, focus and shutter speed. However, there is no support for this in QTM or current camera models.

(The only remote control features currently possible via the RTP is starting and stopping measurement captures.)

6.2.3 3D view

Instead of drawing the markers in 2D, they could be drawn in a 3D view. This would have the advantage that the user could change the view and navigate freely. However, if a 3D view from the point of view of a camera is desired, the application needs to be aware of the position and orientation of the camera, but this information is only available to QTM after calibration. Hence, no real-time update of camera position and orientation can be achieved with the current technology.

6.3 Improving the Goniometer

In this section, two possible improvements of the goniometer application are suggested.

6.3.1 Estimating the true joint center

The goniometer application computes a rather rough estimation of the angle at a given joint. For gait analysts it might be of interest to improve the application by making it measure a biomechanically correct angle. This can be done by implementing an algorithm that gives a better estimation of the true joint center. Several such methods have been suggested (Ehrig et al., 2006, Frigo and Rabuffetti, 1998 and Andriacchi et al., 1998) and could be implemented in future versions of the application.

6.3.2 AIM support

Support for Automatic Identification of Markers (AIM) was left out to minimise the amount of preparations needed in order to use the application. Under some circumstances however, using AIM can be favorable, therefore support for AIM is desirable. When AIM is used the marker order is consistent across frames. This means that the application doesn't need to do any work in order to identify markers, since that would be done by QTM.

6.3.3 Architecture

The development of the goniometer application could benefit from another architectural design, where the angle and marker calculations are done in a plugin of

QTM. This would ease further development of angle calculations—using estimations of true joint centres, AIM models and so forth—by having a clear separation of calculations and presentation.

Taking into account that the developers at Qualisys do not have any iPhone expertise, it is easy to understand why moving most of the development off the iPhone platform into a more familiar environment would be beneficial, both to Qualisys and their costumers. As the QTM software develops the iPhone application could be frozen while updates are done to the plugin.

The plugin would do all the angle calculations and marker identification, and the iPhone application would communicate to this plugin to get the calculated angle.

6.4 Other end-user applications

The real-time API developed in this thesis can be used to create other interesting applications, either for the iPhone or other devices.

One idea for an application is a virtual camera where markers are placed on the device itself. The position and orientation of the device can then be derived and used as a viewpoint in a virtual three-dimensional environment. Such an application could be useful in entertainment industry applications.

References

- Alliance, O. H. (2010). Android ndk. Last retrieved 2010-04-20. <http://developer.android.com/sdk/ndk/>
- Andriacchi, T. P., Alexander, E. J., Toney, M. K., Dyrby, C. and Sum, J. (1998). A point cluster method for in vivo motion analysis: Applied to a study of knee kinematics. *Journal of Biomechanical Engineering*, 120(6), 743-749.
- Apple (2010a). iphone application programming guide. Last retrieved 2010-04-19. <http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html>
- Apple (2010b). iphone developer program license agreement. Last retrieved 2010-04-26.
- Bachmann, E. R. (2000). Inertial and magnetic tracking of limb segment orientation for inserting humans into synthetic environments. In *Ph.D. Dissertation, Naval Postgraduate School*.
- Bideau, B., Multon, F., Kulpa, R., Fradet, L. and Arnaldi, B. et al. (2004). Using virtual reality to analyze links between handball thrower kinematics and goalkeeper's reactions. *Neuroscience Letters*, 372(1-2), 119 - 122.
- Björgvinsson, T. (2006). Peocounter: people counting software. Master's Thesis, Chalmers University of Technology, Göteborg, Sweden.
- Cloete, T. and Scheffer, C. (2008). Benchmarking of a full-body inertial motion capture system for clinical gait analysis.. *Conf Proc IEEE Eng Med Biol Soc*, 2008, 4579-82.
- Corazza, S., Mündermann, L., Chaudhari, A. M., Demattio, T. and Cobelli, C. et al. (2006). A markerless motion capture system to study musculoskeletal biomechanics: visual hull and simulated annealing approach.. *Ann Biomed Eng*, 34(6), 1019-29.
- Deutscher, J., Blake, A. and Reid, I. (2002). Articulated body motion capture by annealed particle filtering. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, pages 126–133 vol.2.
- Ehrig, R. M., Taylor, W. R., Duda, G. N. and Heller, M. O. (2006). A survey of formal methods for determining the centre of rotation of ball joints. *Journal of Biomechanics*, 39(15), 2798 - 2809.

- Fehlhaber, K. and Holmberg, M. (2009). Implementation of a matlab based real-time demo rig: Control of quadcopter. Master's Thesis, Chalmers University of Technology, Göteborg, Sweden.
- Fradet, L., Kulpa, R., Bideau, B., Multon, F. and Delamarche, P. (2003). Kinematic simulation of handball throwing. In *The Society for Modeling and Simulation International. In Proceedings of European Simulation Multiconference*. London, UK.
- Frigo, C. and Rabuffetti, M. (1998). Multifactorial estimation of hip and knee joint centres for clinical application of gait analysis.. *Gait Post*, 8, 91 - 102.
- Ghasemzadeh, H. and Jafari, R. (2010). Coordination analysis of human movements with body sensor networks a signal processing model to evaluate baseball swings. *IEEE SENSORS JOURNAL*, 00(00).
- Gruber, J. (2010). Why apple changed section 3.3.1. Last retrieved 2010-04-22. http://daringfireball.net/2010/04/why_apple_changed_section_331
- Horprasert, T., Haritaoglu, I. and Harwood, D. (1998). Real-time 3d motion capture. <http://citeseer.ist.psu.edu/horprasert98realtime.html>
- Nathan, A. M. (2008). The effect of spin on the flight of a baseball. *American Journal of Physics*, 76(2), 119-124.
- Wikipedia (2010b). Dalvik virtual machine. Last retrieved 2010-05-17. http://en.wikipedia.org/wiki/Dalvik_virtual_machine
- Wikipedia (2010a). Objective-c. Last retrieved 2010-04-19. <http://sv.wikipedia.org/wiki/Objective-C>
- Wikipedia (2010c). Smartphone. Last retrieved 2010-05-21. <http://sv.wikipedia.org/wiki/Smartphone>
- Zhang, Z., Wu, Z., Chen, J. and Wu, J.-K. (2009). Ubiquitous human body motion capture using micro-sensors. *Pervasive Computing and Communications, IEEE International Conference on*, 0, 1-5.

Index

a

AIM 23, 36

Android 11

c

client library API

goals 2

implementation of 19

specification 29

compatibility layer 13

cross-platform 13, 14

g

goniometer 22, 32

goals 2

improvements of 36

i

intermediate compatibility layer

see compatibility layer

iPhone 10

legal uncertainty 14

m

marker

active 6

passive 6

reflective 6

markers

configuration of 23

unlabeled 23

motion capture 5

marker based 6

markerless 7

o

Objective-C 11

q

Qualisys Track Manager 9

Qualisys Real-time Protocol 9

v

viewfinder 20, 29

goals 2

improvements of 35

Appendices

A Client library specification

QMT RT API

0.1

Generated by Doxygen 1.6.2

Wed Jun 16 09:27:32 2010

Contents

1	Main Page	1
2	Module Index	2
2.1	Modules	2
3	Data Structure Index	2
3.1	Data Structures	2
4	Module Documentation	3
4.1	API	3
4.1.1	Detailed Description	5
4.1.2	Define Documentation	5
4.1.3	Function Documentation	6
5	Data Structure Documentation	12
5.1	QRA Struct Reference	12
5.1.1	Field Documentation	12
5.2	QTM2DData Struct Reference	13
5.3	QTM2DMarker Struct Reference	14
5.4	QTM3DData Struct Reference	14
5.5	QTM3DMarker Struct Reference	15
5.6	QTM6DBody Struct Reference	15
5.7	QTM6DData Struct Reference	16
5.8	QTMAnalogData Struct Reference	17
5.9	QTMAnalogDevice Struct Reference	17
5.9.1	Detailed Description	18
5.10	QTMCamera Struct Reference	18
5.11	QTMComponent Struct Reference	19
5.12	QTMDataFrame Struct Reference	21
5.13	QTMForceData Struct Reference	22
5.14	QTMForcePlate Struct Reference	23
5.15	QTMForcePlateData Struct Reference	24
5.16	QTMMsgHeader Struct Reference	24
5.16.1	Detailed Description	24

5.16.2 Field Documentation	24
5.17 QTMPParam3D Struct Reference	25
5.18 QTMPParam3DData Struct Reference	26
5.19 QTMPParam6D Struct Reference	27
5.20 QTMPParamAnalog Struct Reference	28
5.21 QTMPParamBody Struct Reference	29
5.22 QTMPParamCamera Struct Reference	29
5.23 QTMPParamChannel Struct Reference	30
5.24 QTMPParamDevice Struct Reference	30
5.25 QTMPParamForce Struct Reference	31
5.26 QTMPParamGeneral Struct Reference	32
5.27 QTMPParamLabel Struct Reference	33
5.28 QTMPParamParameters Struct Reference	34
5.29 QTMPParamPlate Struct Reference	35
5.30 QTMPParamPoint Struct Reference	36
5.31 QTMStreamFrequency Struct Reference	36
5.32 QTMVersion Struct Reference	36
5.32.1 Detailed Description	37

1 Main Page

Author:

Johannes Martinsson <w@antiklimax.se>
 Reimund Trost <reimund@code7.se>

Date:

2010-02-05

This API provides an interface to a QTM RT server. It is implemented in C with basic
 BSD sockets.

This API implementation is intended to be compatible with both iPhone and Android
 devices. The idea is to create wrappers for this API in Objective-C and Java.

2 Module Index

2.1 Modules

Here is a list of all modules:

API	3
-----	---

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

QRA	12
QTM2DData	13
QTM2DMarker	14
QTM3DData	14
QTM3DMarker	15
QTM6DBody	15
QTM6DData	16
QTMAnalogData	17
QTMAnalogDevice (Data will be layed out one float after another in the data member)	17
QTMCamera	18
QTMComponent	19
QTMDataFrame	21
QTMForceData	22
QTMForcePlate	23
QTMForcePlateData	24
QTMMsgHeader (Header used in all packets sent to the server)	24
QTMParam3D	25
QTMParam3DData	26

QTMPParam6D	27
QTMPParamAnalog	28
QTMPParamBody	29
QTMPParamCamera	29
QTMPParamChannel	30
QTMPParamDevice	30
QTMPParamForce	31
QTMPParamGeneral	32
QTMPParamLabel	33
QTMPParamParameters	34
QTMPParamPlate	35
QTMPParamPoint	36
QTMPStreamFrequency	36
QTMPVersion (Generic structure to represent a version with a major and a minor part)	36

4 Module Documentation

4.1 API

Main API group.

Defines

- #define **STREQ**(a, b) (strcmp((a), (b)) == 0)
- #define **DPRINT**(msg, err)

Functions

- [QRA qtmAPIInit](#) (void)
Get an handle to the QTM RT API.
- QTMErrorCode [qtmConnect](#) ([QRA](#), const char *, int, int)
Connect to a QTM RT server.

- QTMErrorCode [qtmShutdown](#) (QRA)
Shutdown the connection.
- QTMErrorCode [qtmVersion](#) (QRA, QTMVersion *, int, int)
Set or query protocol version.
- QTMErrorCode [qtmQTMVersion](#) (QRA, QTMVersion *)
Get the version of QTM connected to.
- QTMErrorCode [qtmTakeControl](#) (QRA)
Make this client the master client.
- QTMErrorCode [qtmReleaseControl](#) (QRA)
Make this client a regular client.
- QTMErrorCode [qtmByteOrder](#) (QRA, int *)
Get the current byte order of the protocol.
- QTMErrorCode [qtmCheckLicense](#) (QRA, const char *)
Check a license code against QTM.
- QTMErrorCode [qtmGetLastEvent](#) (QRA, QTMEvent *)
Get the latest event that occurred in QTM.
- QTMErrorCode [qtmNew](#) (QRA)
Create a new measurement in QTM.
- QTMErrorCode [qtmClose](#) (QRA)
Close current measurement in QTM.
- QTMErrorCode [qtmStart](#) (QRA)
Start a new capture in QTM.
- QTMErrorCode [qtmStop](#) (QRA)
Stop an ongoing capture in QTM.
- QTMErrorCode [qtmSave](#) (QRA api, const char *filename)
Save the current capture QTM.
- QTMErrorCode [qtmGetCapture](#) (QRA, char *)
Gets a C3D file with the latest captured QTM measurement.
- QTMErrorCode [qtmGetParameters](#) (QRA, QTMPParamParameters *, int)
Get measurement parameters from server.
- QTMErrorCode [qtmGetCurrentFrame](#) (QRA, QTMDDataFrame *, int)

Get the current frame of real-time data from QTM.

- QTMErrorCode [qtmStreamFrames](#) (QRA, int(*w)(QTMDDataFrame *, void *), void *, bool, [QTMStreamFrequency](#), int, char *, int)

Streams data frames from QTM.

- const char * [qtmErrorToString](#) (QTMErrorCode)

Translate error codes into a error message.

- const char * [qtmErrorToError](#) (QTMErrorCode)

Translate error codes into its define.

- void [qtmFreeDataFrameComponents](#) (QTMDDataFrame *)

Free memory allocated to components in receiveFrame.

- void [qtmFreeDataFrame](#) (QTMDDataFrame *)

Free memory allocated to a frame (including frame components).

- void [qtmFree3D](#) (QTMParm3D *p)

Free memory allocated by a call to qtmParse3D.

- void [qtmFree6D](#) (QTMParm6D *p)

Free memory allocated by a call to qtmParse6D.

- void [qtmFreeAnalog](#) (QTMParmAnalog *p)

Free memory allocated by a call to qtmParseAnalog.

- void [qtmFreeForce](#) (QTMParmForce *p)

Free memory allocated by a call to qtmParseForce.

- void [qtmFreeGeneral](#) (QTMParmGeneral *p)

Free memory allocated by a call to qtmParseGeneral.

4.1.1 Detailed Description

Main API group.

4.1.2 Define Documentation

4.1.2.1 #define DPRINT(msg, err)

Value:

```
(fprintf(stderr, \
    "[DEBUG %s:%d in %s]: %s (%d)\n", \
    __FILE__, __LINE__, __func__, (msg), (err)))
```

4.1.3 Function Documentation

4.1.3.1 QRA qtmAPIInit (void)

Get an handle to the QTM RT API. Use this function to get an initiated structure containing function pointers to all the API functions.

Returns:

A structure with function pointers.

4.1.3.2 QTMErrorCode qtmByteOrder (QRA *api*, int * *endian*)

Get the current byte order of the protocol.

Parameters:

endian Will contain the endian, BIG_ENDIAN or LITTLE_ENDIAN.

Returns:

QTM_OK on success, QTM_NO_PARSE or an error code from qtmAsciiInteract.

4.1.3.3 QTMErrorCode qtmCheckLicense (QRA *api*, const char * *licenseKey*)

Check a license code against QTM.

Parameters:

licenseKey code to check.

Returns:

QTM_LICENSE_PASS, QTM_LICENSE_FAIL or QTM_NO_PARSE. (Might also return an error code from qtmInteract.)

4.1.3.4 QTMErrorCode qtmClose (QRA *api*)

Close current measurement in QTM. Only possible to issue if client is master.

Returns:

QTM_CLOSING_CONNECTION on success, QTM_NO_CONNECTION, QTM_MUST_BE_MASTER, QTM_NO_PARSE or an error code from qtmInteract.

4.1.3.5 QTMErrorCode qtmConnect (QRA *api*, const char * *host*, int *port*, int *timeout*)

Connect to a QTM RT server.

Parameters:

host Host to connect to, should be an IP address. (For now.)

port Port to connect to, should probably be 22223.

timeout Timeout in seconds.

Returns:

QTM_CONNECTED on success or QTM_NO_CONNECTION, QTM_NO_PARSE on error.

4.1.3.6 const char* qtmErrorToError (QTMErrorCode *err*)

Translate error codes into its define.

Parameters:

err The error code to translate.

Returns:

The define representation of the error code.

4.1.3.7 const char* qtmErrorToString (QTMErrorCode *err*)

Translate error codes into a error message.

Parameters:

err The error code to translate.

Returns:

The string representation of the error code.

4.1.3.8 QTMErrorCode qtmGetCapture (QRA *api*, char * *buf*)

Gets a C3D file with the latest captured QTM measurement.

Parameters:

buf The buffer to write the C3D file to.

Returns:

An error code indicating the status of the call.

4.1.3.9 QTMErrorCode qtmGetCurrentFrame (QRA *api*, QTMDDataFrame * *frame*, int *componentsToGet*)

Get the current frame of real-time data from QTM.

Parameters:

frame Will contain the data components.

componentsToGet Bitmasked field of which components to get.

See also:

QTMPParameter.

Returns:

QTM_OK on success. QTM_NO_MEASUREMENT, QTM_WRONG_TYPE or an error code from qtmInteract.

4.1.3.10 QTMErrorCode qtmGetLastEvent (QRA *api*, QTMEvent * *event*)

Get the latest event that occurred in QTM. This function can only deal with events without event data.

Parameters:

event Will contain the latest event.

4.1.3.11 QTMErrorCode qtmGetParameters (QRA *api*, QTMPParamParameters * *params*, int *paramsToGet*)

Get measurement parameters from server.

Parameters:

params Struct to hold the requested pieces of data.

paramsToGet Which parts to get, bitmasked bitfield.

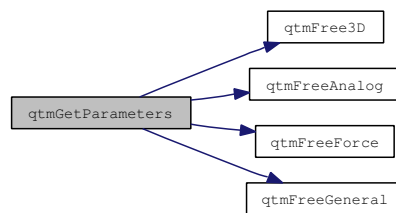
See also:

QTMParameter

Returns:

Returns QTM_OK on success. QTM_NO_MEASUREMENT or an error code from qtmInteract or any of the qtmParse-functions.

Here is the call graph for this function:



4.1.3.12 QTMErrorCode qtmNew (QRA *api*)

Create a new measurement in QTM. Only possible to issue if client is master.

Returns:

QTM_NEW_CONNECTION on success, QTM_ALREADY_CONNECTED, QTM_MUST_BE_MASTER, QTM_NO_PARSE or an error code from qtmInteract.

4.1.3.13 QTMErrorCode qtmQTMVersion (QRA *api*, QTMVersion * *ver*)

Get the version of QTM connected to.

Parameters:

ver Will contain the QTM version.

Returns:

QTM_OK on success or QTM_NO_PARSE if version could not be parsed. Might also return an error code from qtmAsciiInteract.

4.1.3.14 QTMErrorCode qtmReleaseControl (QRA *api*)

Make this client a regular client.

Returns:

QTM_IS_CLIENT on success or an error code from qtmAsciiInteract.

4.1.3.15 QTMErrorCode qtmSave (QRA *api*, const char * *filename*)

Save the current capture QTM. Only possible to issue if client is master.

Parameters:

filename Filename to save under, must be less than 255 characters long. If filename doesn't end with '.qtm' it will be added.

Returns:

QTM_OK on success, QTM_NO_MEASUREMENT, QTM_MUST_BE_MASTER, QTM_FAIL or an error code from qtmAsciiInteract.

4.1.3.16 QTMErrorCode qtmShutdown (QRA *api*)

Shutdown the connection. Basically amounts to a close system call on the socket.

4.1.3.17 QTMErrorCode qtmStart (QRA *api*)

Start a new capture in QTM. Only possible to issue if client is master.

Returns:

QTM_STARTING_MEASUREMENT on success, QTM_NO_MEASUREMENT, QTM_NO_CONNECTION, QTM_MUST_BE_MASTER, QTM_NO_PARSE or an error code from qtmInteract.

4.1.3.18 QTMErrorCode qtmStop (QRA *api*)

Stop an ongoing capture in QTM. Only possible to issue if client is master.

Returns:

QTM_STOPPING_MEASUREMENT on success, QTM_NO_MEASUREMENT, QTM_MUST_BE_MASTER, QTM_NO_PARSE or an error code from qtmInteract.

4.1.3.19 QTMErrorCode qtmStreamFrames (QRA api, int*)(QTMDDataFrame *, void *)f, void * anything, bool shouldFree, QTMStreamFrequency freq, int port, char * host, int comps)

Streams data frames from QTM.

Parameters:

f Function that handles a single received frame, should return a negative number if streaming should stop.

anything A void pointer that will be passed to *f* when called.

free Should be set to false if the streamed data frames should not be automatically freed. If it is set to false it is imperative that the caller takes care of freeing each data frame (e.g. using qtmFreeDataFrameComponents).

freq Stream frequency specifier,

See also:

[QTMStreamFrequency](#)

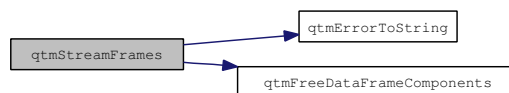
Parameters:

port If non-negative streaming will be done on UDP to this port.

host Used when streaming over UDP, specifies which host to stream to.

comps Which components to ask QTM for.

Here is the call graph for this function:



4.1.3.20 QTMErrorCode qtmTakeControl (QRA api)

Make this client the master client.

Returns:

QTM_IS_MASTER on success, QTM_NO_PARSE or an error code from qtmInteract.

4.1.3.21 QTMErrorCode qtmVersion (QRA api, QTMVersion * ver, int major, int minor)

Set or query protocol version. If either major or minor is negative, only a query is performed.

Parameters:

- ver* Will contain the protocol version.
- major* If major and minor is set to -1 let the server choose protocol, otherwise this will be the major part of the protocol version.
- minor* Minor part of protocol version.

5 Data Structure Documentation

5.1 QRA Struct Reference

Data Fields

- char [host](#) [253]
The host name or IP-address of the QTM host.
- int [port](#)
UDP port to use if applicable.
- int [tcpSocket](#)
TCP socket for QTM communication.
- int [udpSocket](#)
UDP socket for streaming data frames.
- struct sockaddr_in [sa](#)
Support for network communication.
- int(* [eventHandler](#))(QTMEvent, void *)
Event handler function.

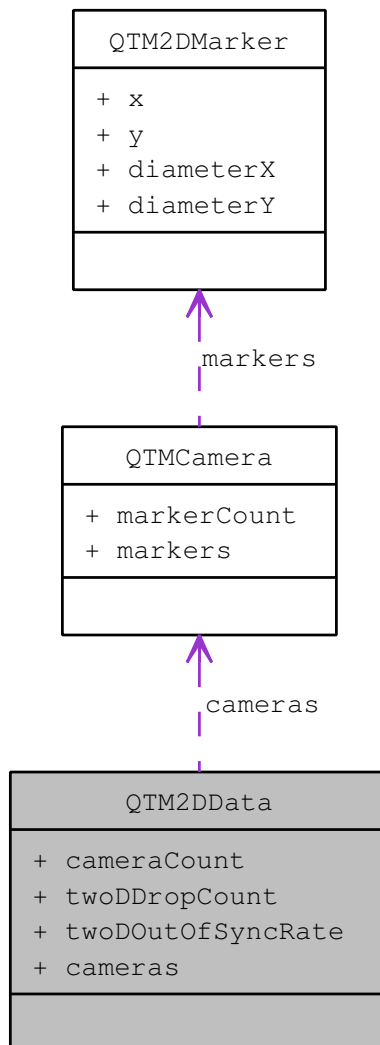
5.1.1 Field Documentation

5.1.1.1 int(* QRA::eventHandler)(QTMEvent, void *)

Event handler function. If this function is not set all events will be silently ignored. If it is set, whenever an event is encountered in the network communication with QTM this function will be called with the event and the void pointer given in the registerEventHandler function.

5.2 QTM2DData Struct Reference

Collaboration diagram for QTM2DData:



Data Fields

- `int32_t cameraCount`
- `uint16_t twoDDropCount`

- uint16_t **twoDOutOfSyncRate**
- [QTMCamera](#) * **cameras**

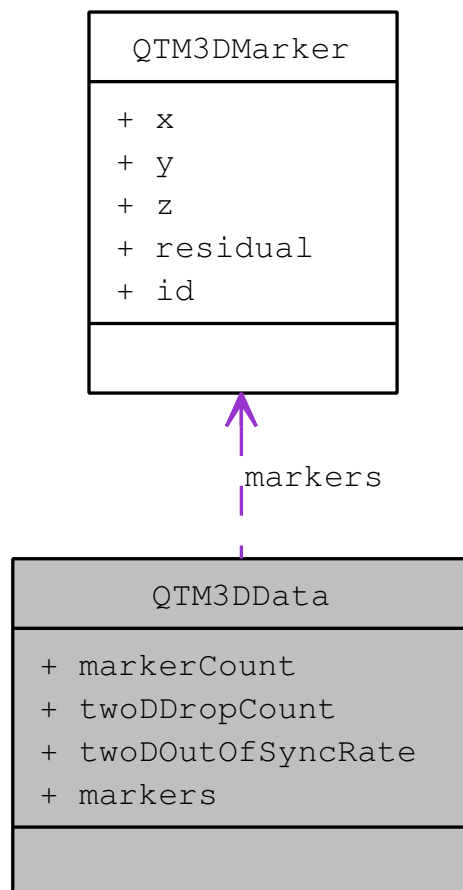
5.3 QTM2DMarker Struct Reference

Data Fields

- int32_t **x**
- int32_t **y**
- int16_t **diameterX**
- int16_t **diameterY**

5.4 QTM3DData Struct Reference

Collaboration diagram for QTM3DData:



Data Fields

- uint32_t **markerCount**
- uint16_t **twoDDropCount**
- uint16_t **twoDOutOfSyncRate**
- [QTM3DMarker](#) * **markers**

5.5 QTM3DMarker Struct Reference**Data Fields**

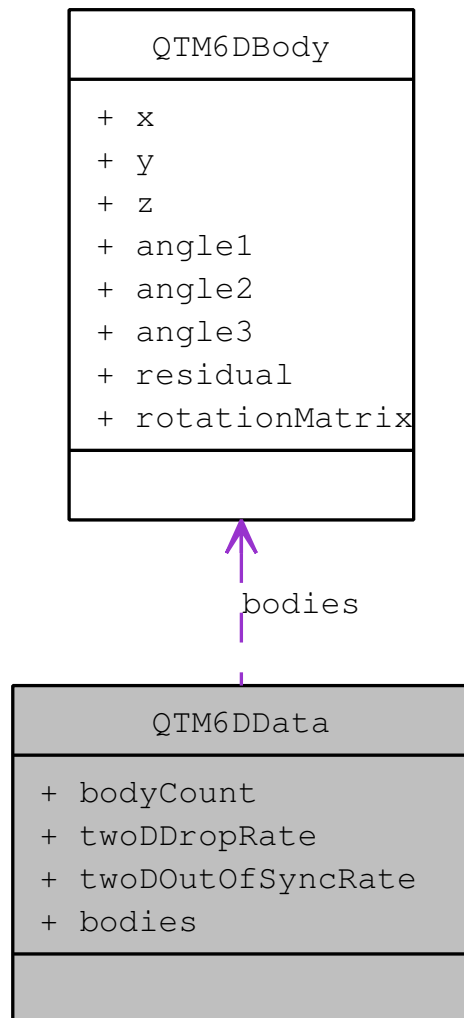
- double **x**
- double **y**
- double **z**
- float **residual**
- uint32_t **id**

5.6 QTM6DBody Struct Reference**Data Fields**

- double **x**
- double **y**
- double **z**
- double **angle1**
- double **angle2**
- double **angle3**
- float **residual**
- double * **rotationMatrix**

5.7 QTM6DData Struct Reference

Collaboration diagram for QTM6DData:

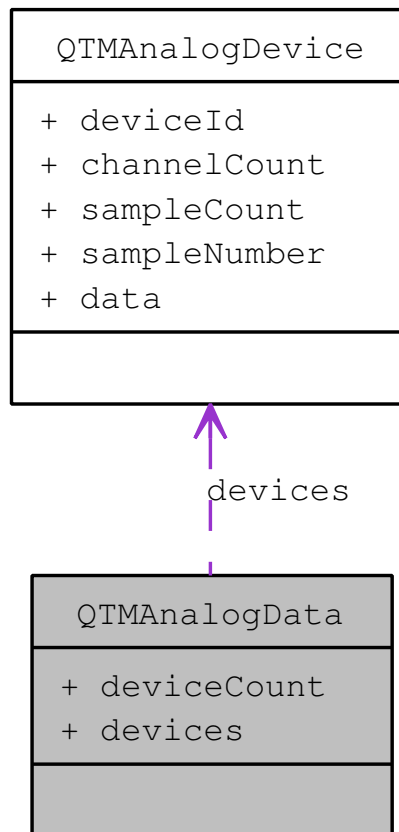


Data Fields

- `uint32_t` **bodyCount**
- `uint16_t` **twoDDropRate**
- `uint16_t` **twoDOutOfSyncRate**
- [QTM6DBody](#) * **bodies**

5.8 QTMAalogData Struct Reference

Collaboration diagram for QTMAalogData:



Data Fields

- `int deviceCount`
- `QTMAalogDevice * devices`

5.9 QTMAalogDevice Struct Reference

Data will be layed out one float after another in the data member.

Data Fields

- `int deviceId`
- `int channelCount`
- `int sampleCount`

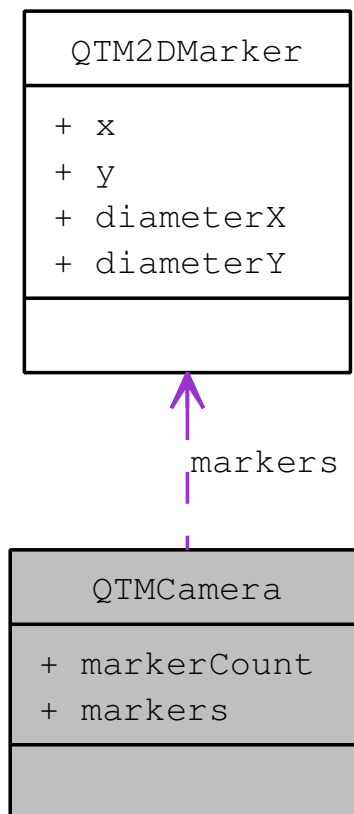
- int **sampleNumber**
- float * **data**

5.9.1 Detailed Description

Data will be layed out one float after another in the data member. First comes the first channel's samples, then the second channel's and so on.

5.10 QTMCamera Struct Reference

Collaboration diagram for QTMCamera:

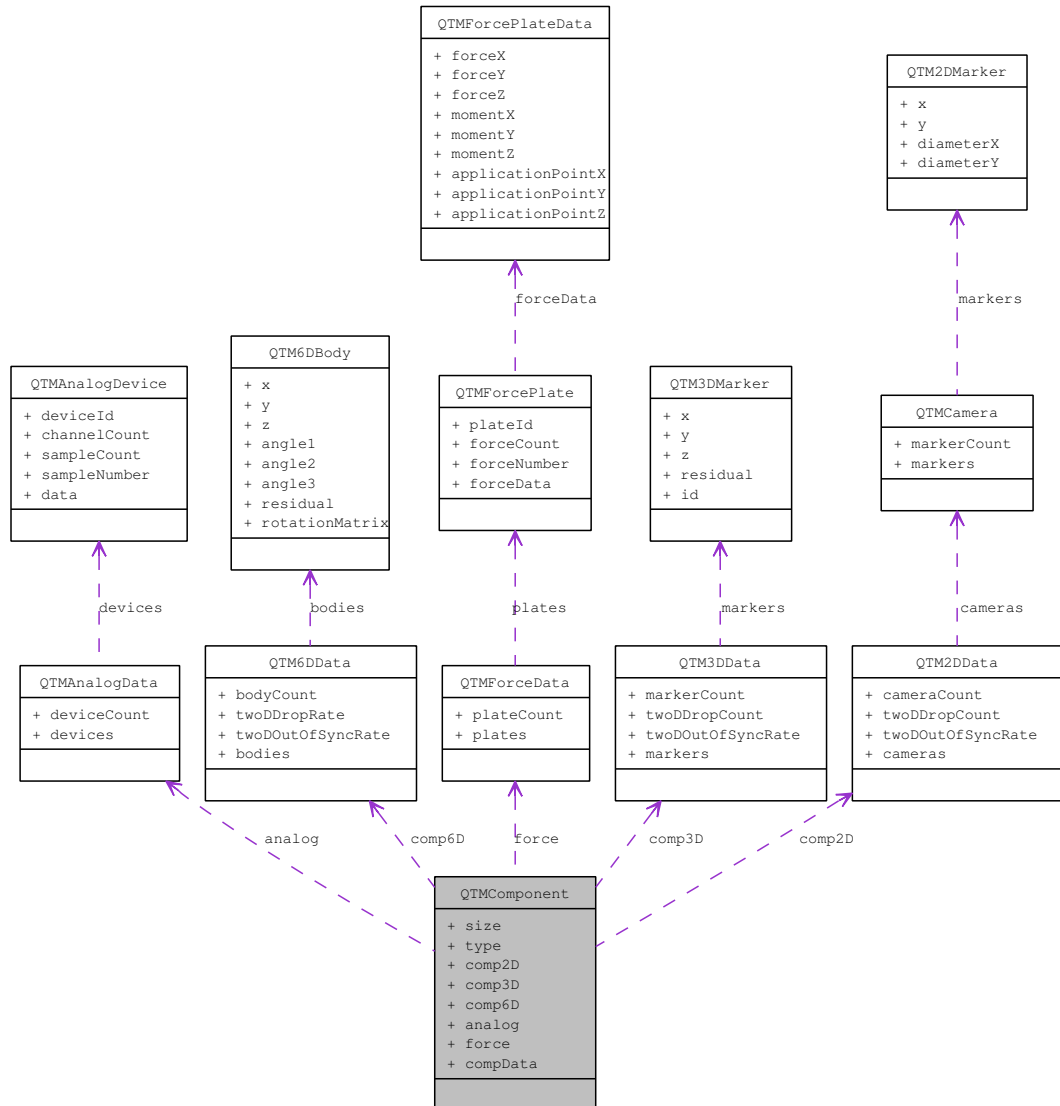


Data Fields

- int32_t **markerCount**
- [QTM2DMarker](#) * **markers**

5.11 QTMComponent Struct Reference

Collaboration diagram for QTMComponent:



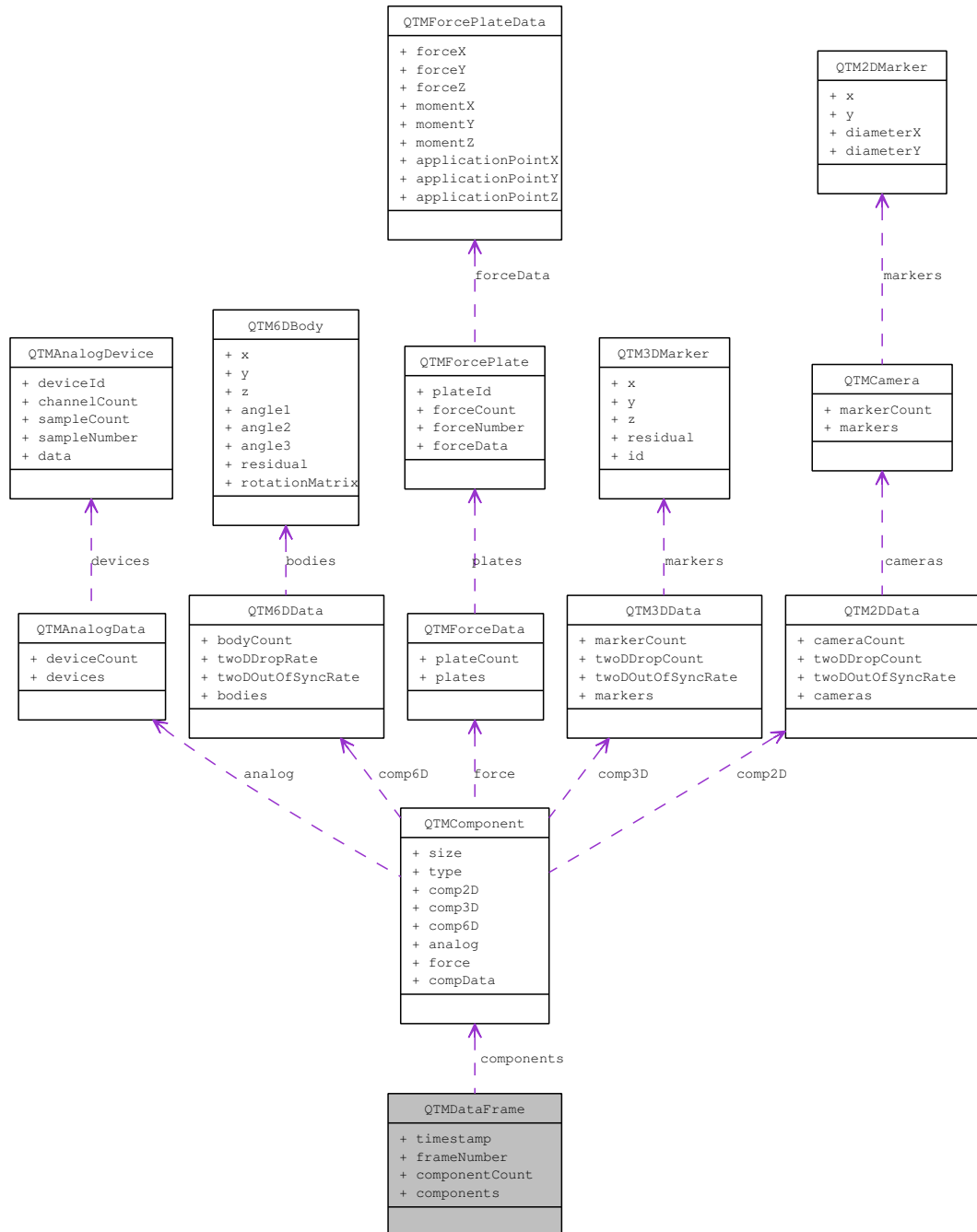
Data Fields

- `uint32_t` **size**
- `uint32_t` **type**
- union {
 - [QTM2DData](#) **comp2D**
 - [QTM3DData](#) **comp3D**
 - [QTM6DData](#) **comp6D**

```
    QTMAalogData analog
    QTMForceData force
} compData
```

5.12 QTMDDataFrame Struct Reference

Collaboration diagram for QTMDDataFrame:

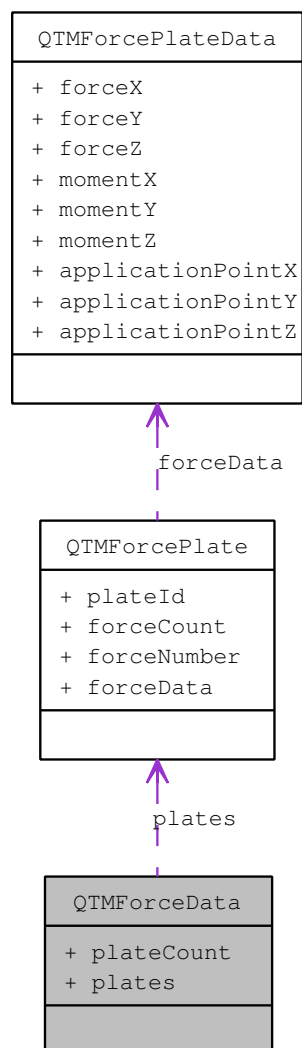


Data Fields

- int64_t **timestamp**
- int32_t **frameNumber**
- int32_t **componentCount**
- [QTMComponent](#) * **components**

5.13 QTMForceData Struct Reference

Collaboration diagram for QTMForceData:

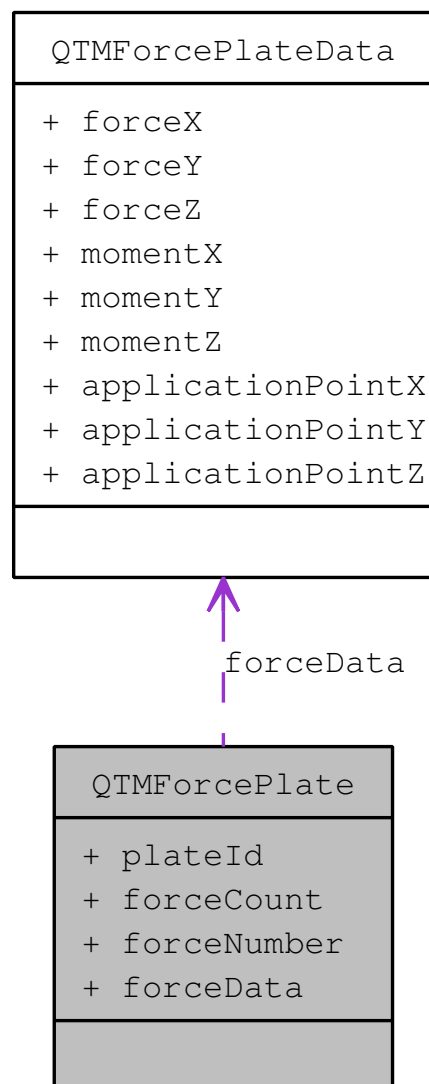


Data Fields

- `int32_t` **plateCount**
- [QTMForcePlate](#) * **plates**

5.14 QTMForcePlate Struct Reference

Collaboration diagram for QTMForcePlate:

**Data Fields**

- `int32_t` **plateId**

- `int32_t forceCount`
- `int32_t forceNumber`
- `QTMForcePlateData * forceData`

5.15 QTMForcePlateData Struct Reference

Data Fields

- `float forceX`
- `float forceY`
- `float forceZ`
- `float momentX`
- `float momentY`
- `float momentZ`
- `float applicationPointX`
- `float applicationPointY`
- `float applicationPointZ`

5.16 QTMMsgHeader Struct Reference

Header used in all packets sent to the server.

Data Fields

- `uint32_t size`
Total size of the packet including this header.
- `uint32_t type`
Type of the packet.

5.16.1 Detailed Description

Header used in all packets sent to the server.

5.16.2 Field Documentation

5.16.2.1 `uint32_t QTMMsgHeader::type`

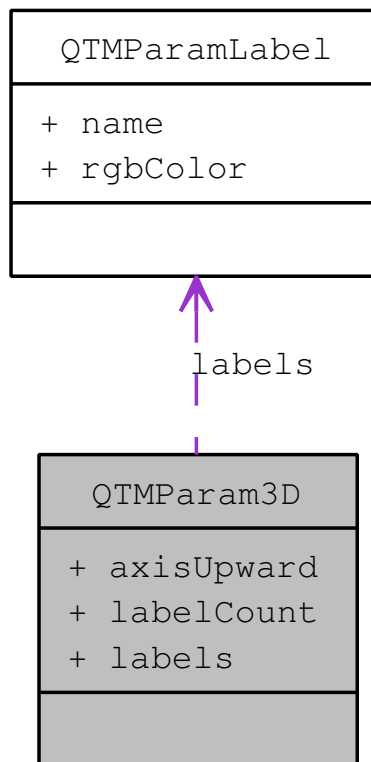
Type of the packet.

See also:

`MsgType`

5.17 QTMPParam3D Struct Reference

Collaboration diagram for QTMPParam3D:

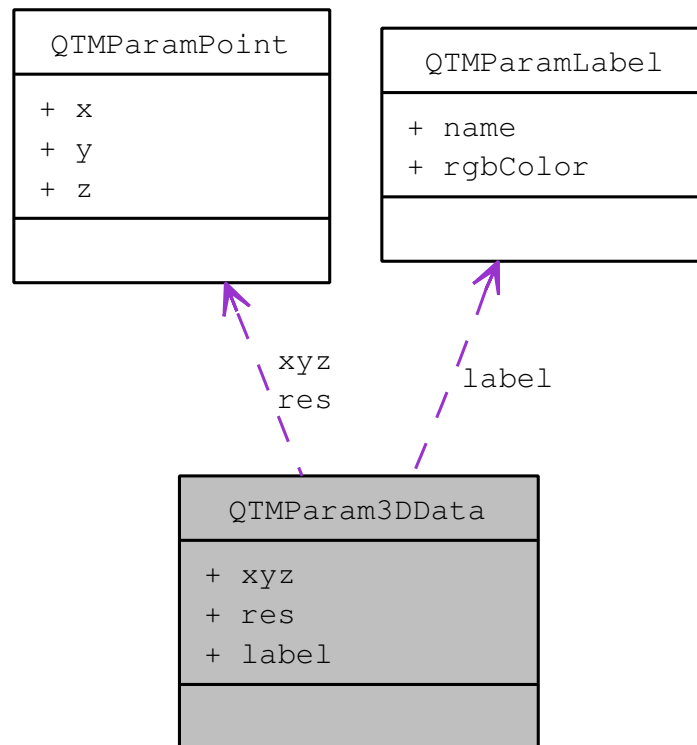


Data Fields

- QTMPParamAxis **axisUpward**
- int **labelCount**
- [QTMPParamLabel](#) * **labels**

5.18 QTMParm3DData Struct Reference

Collaboration diagram for QTMParm3DData:

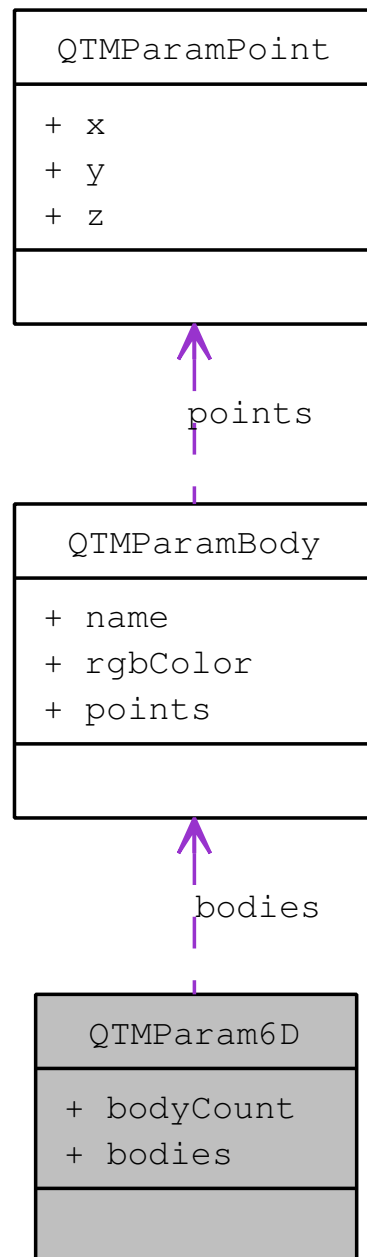


Data Fields

- [QTMParmPoint](#) `xyz`
- [QTMParmPoint](#) `res`
- [QTMParmLabel](#) `label`

5.19 QTMPParam6D Struct Reference

Collaboration diagram for QTMPParam6D:



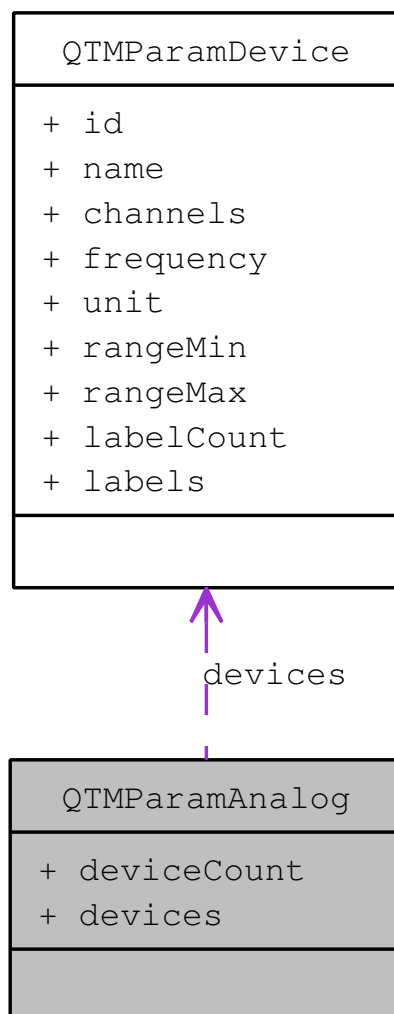
Data Fields

- `int bodyCount`

- [QTMPParamBody](#) * bodies

5.20 QTMPParamAnalog Struct Reference

Collaboration diagram for QTMPParamAnalog:

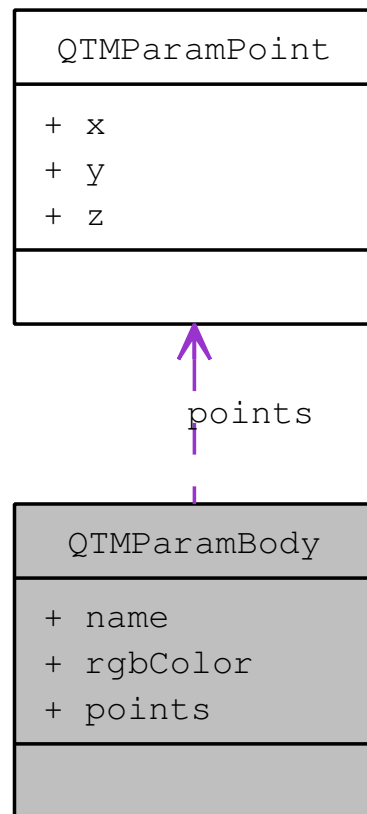


Data Fields

- int deviceCount
- [QTMPParamDevice](#) * devices

5.21 QTMParmBody Struct Reference

Collaboration diagram for QTMParmBody:



Data Fields

- `char * name`
- `unsigned int rgbColor`
- `QTMParmPoint * points`

5.22 QTMParmCamera Struct Reference

Data Fields

- `unsigned int id`
- `QTMParmCameraModel model`
- `unsigned int serial`
- `unsigned int markerResolutionWidth`
- `unsigned int markerResolutionHeight`

- unsigned int **markerFOVLeft**
- unsigned int **markerFOVTop**
- unsigned int **markerFOVRight**
- unsigned int **markerFOVBottom**
- unsigned int **videoResolutionWidth**
- unsigned int **videoResolutionHeight**
- unsigned int **videoFOVLeft**
- unsigned int **videoFOVTop**
- unsigned int **videoFOVRight**
- unsigned int **videoFOVBottom**
- double **positionX**
- double **positionY**
- double **positionZ**
- double **positionRotation** [9]

5.23 QTMPParamChannel Struct Reference

Data Fields

- unsigned int **index**
- double **conversionFactor**

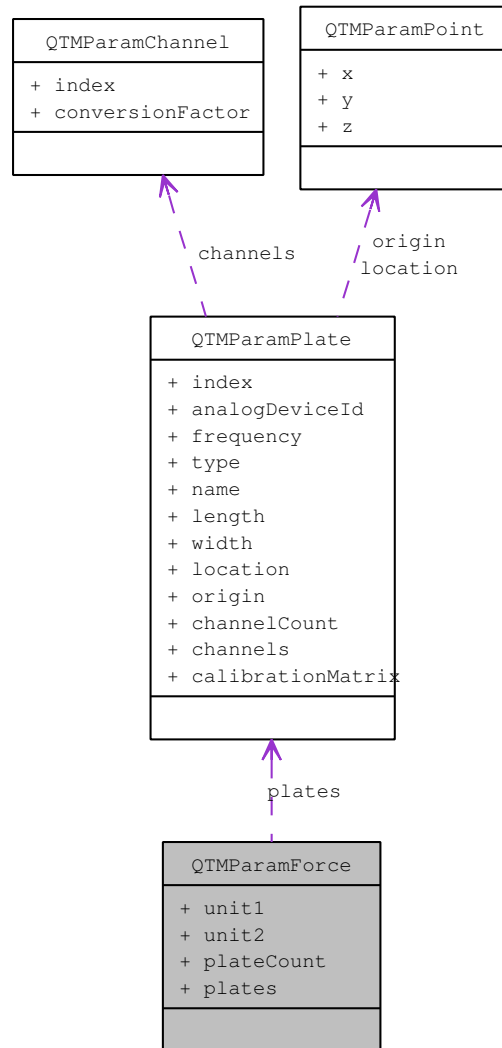
5.24 QTMPParamDevice Struct Reference

Data Fields

- unsigned int **id**
- char * **name**
- unsigned int **channels**
- unsigned int **frequency**
- QTMPParamUnit **unit**
- int **rangeMin**
- int **rangeMax**
- int **labelCount**
- char ** **labels**

5.25 QTMParmForce Struct Reference

Collaboration diagram for QTMParmForce:

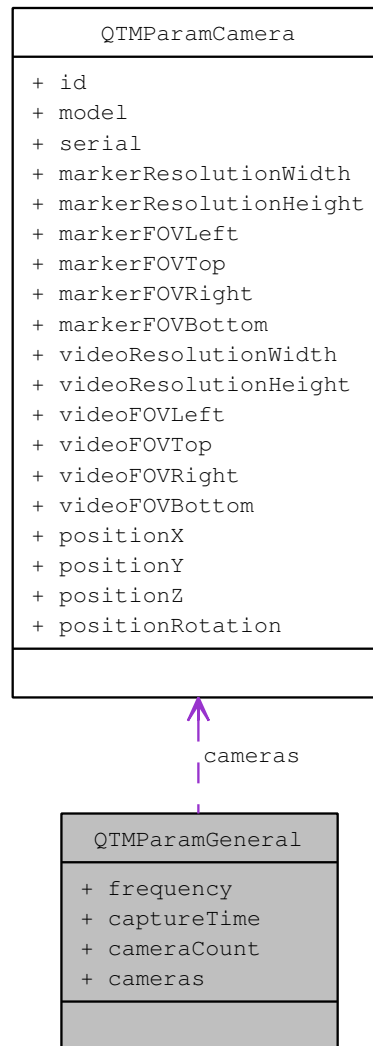


Data Fields

- char * **unit1**
- char * **unit2**
- int **plateCount**
- [QTMParmPlate](#) * **plates**

5.26 QTMParmGeneral Struct Reference

Collaboration diagram for QTMParmGeneral:



Data Fields

- unsigned int **frequency**
- double **captureTime**
- int **cameraCount**
- [QTMParmCamera](#) * **cameras**

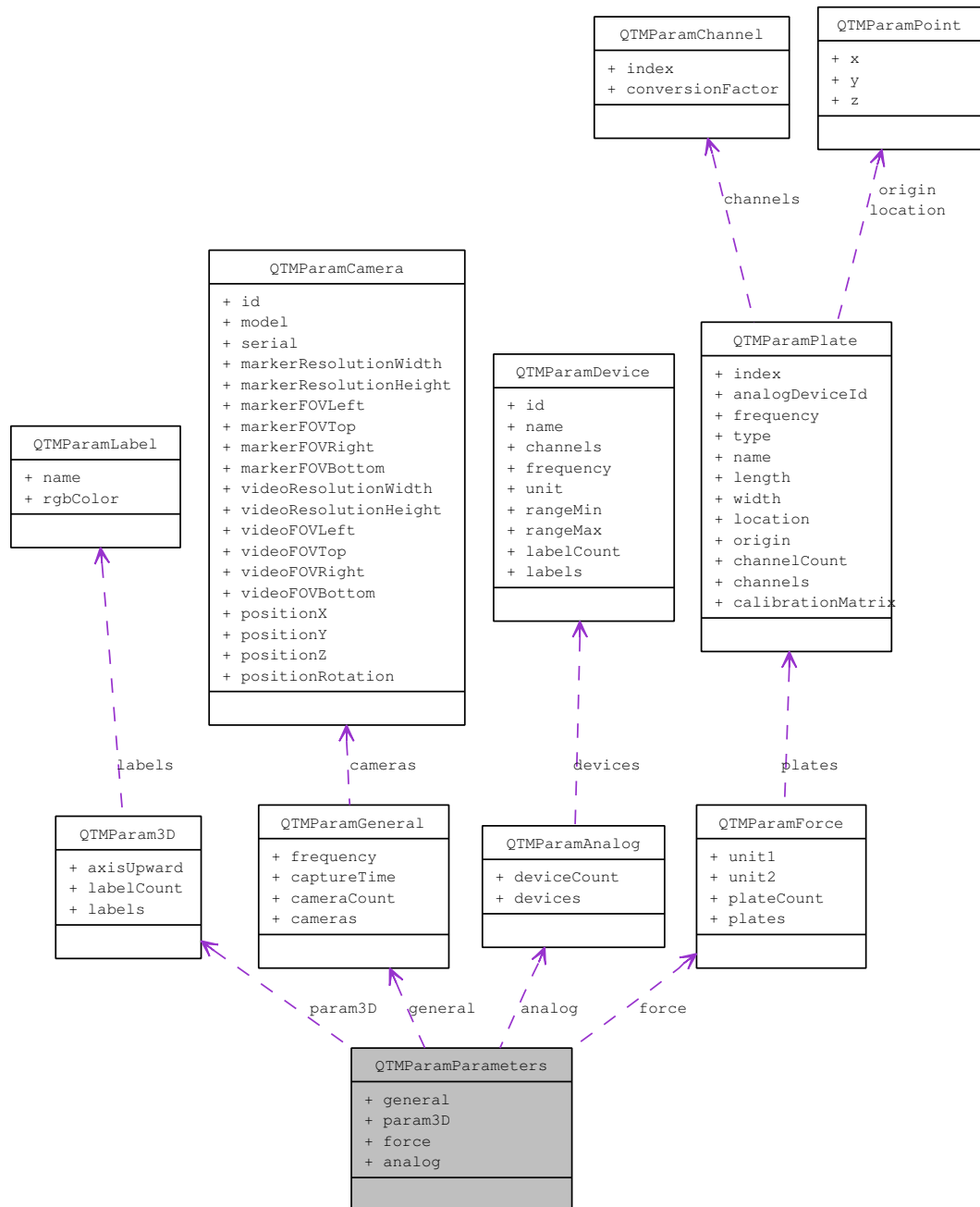
5.27 QTMPParamLabel Struct Reference

Data Fields

- char * **name**
- unsigned int **rgbColor**

5.28 QTMParmParameters Struct Reference

Collaboration diagram for QTMParmParameters:

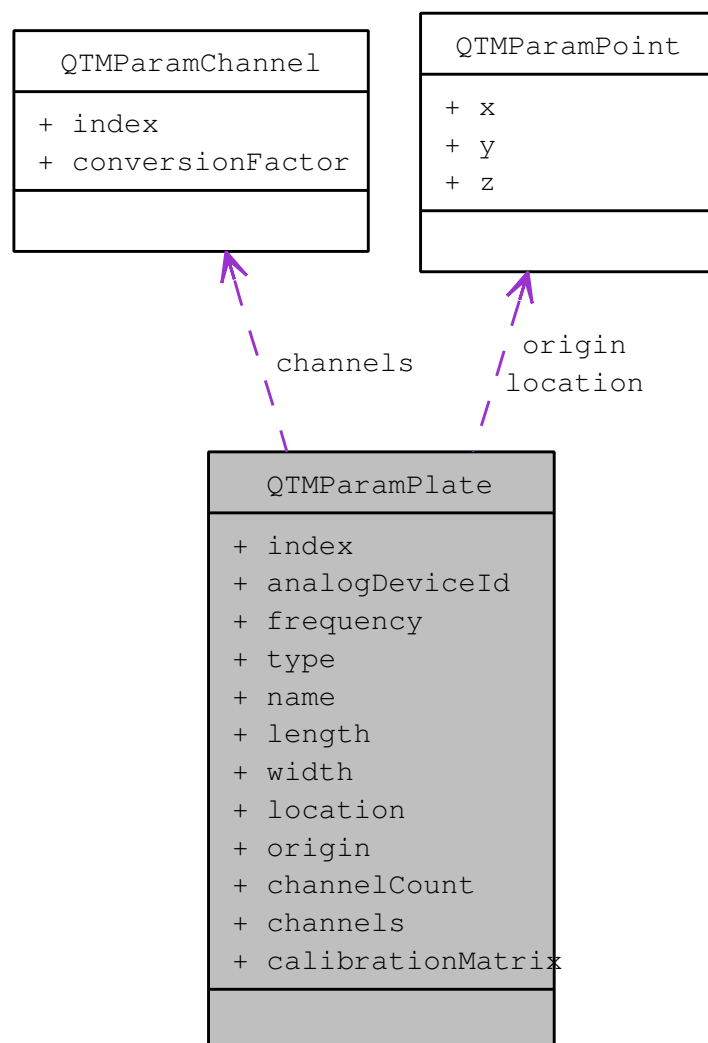


Data Fields

- [QTMParmGeneral](#) * **general**
- [QTMParm3D](#) * **param3D**
- [QTMParmForce](#) * **force**
- [QTMParmAnalog](#) * **analog**

5.29 QTMParmPlate Struct Reference

Collaboration diagram for QTMParmPlate:



Data Fields

- unsigned int **index**
- unsigned int **analogDeviceId**
- unsigned int **frequency**
- char * **type**
- char * **name**
- unsigned int **length**
- unsigned int **width**
- [QTMParmPoint](#) **location** [4]
- [QTMParmPoint](#) **origin**
- int **channelCount**
- [QTMParmChannel](#) * **channels**
- double **calibrationMatrix** [6][6]

5.30 QTMParmPoint Struct Reference**Data Fields**

- double **x**
- double **y**
- double **z**

5.31 QTMStreamFrequency Struct Reference**Public Types**

- enum { **QTM_STREAM_FREQUENCY_DIVISOR**, **QTM_STREAM_FREQUENCY**, **QTM_STREAM_ALL_FRAMES** }

Data Fields

- enum QTMStreamFrequency:: { ... } **kind**
- uint16_t **value**

5.32 QTMVersion Struct Reference

Generic structure to represent a version with a major and a minor part.

Data Fields

- unsigned int **major**
- unsigned int **minor**

5.32.1 Detailed Description

Generic structure to represent a version with a major and a minor part.

Index

- API, [3](#)
 - DPRINT, [5](#)
 - qtmAPIInit, [5](#)
 - qtmByteOrder, [5](#)
 - qtmCheckLicense, [5](#)
 - qtmClose, [6](#)
 - qtmConnect, [6](#)
 - qtmErrorToError, [6](#)
 - qtmErrorToString, [7](#)
 - qtmGetCapture, [7](#)
 - qtmGetCurrentFrame, [7](#)
 - qtmGetLastEvent, [8](#)
 - qtmGetParameters, [8](#)
 - qtmNew, [8](#)
 - qtmQTMVersion, [9](#)
 - qtmReleaseControl, [9](#)
 - qtmSave, [9](#)
 - qtmShutdown, [9](#)
 - qtmStart, [10](#)
 - qtmStop, [10](#)
 - qtmStreamFrames, [10](#)
 - qtmTakeControl, [11](#)
 - qtmVersion, [11](#)
- DPRINT
 - API, [5](#)
- eventHandler
 - QRA, [12](#)
- QRA, [12](#)
 - eventHandler, [12](#)
- QTM2DData, [13](#)
- QTM2DMarker, [14](#)
- QTM3DData, [14](#)
- QTM3DMarker, [15](#)
- QTM6DBody, [15](#)
- QTM6DData, [16](#)
- QTMAnalogData, [17](#)
- QTMAnalogDevice, [17](#)
- qtmAPIInit
 - API, [5](#)
- qtmByteOrder
 - API, [5](#)
- QTMCamera, [18](#)
- qtmCheckLicense
 - API, [5](#)
- qtmClose
 - API, [6](#)
- QTMComponent, [19](#)
- qtmConnect
 - API, [6](#)
- QTMDataFrame, [21](#)
- qtmErrorToError
 - API, [6](#)
- qtmErrorToString
 - API, [7](#)
- QTMForceData, [22](#)
- QTMForcePlate, [23](#)
- QTMForcePlateData, [24](#)
- qtmGetCapture
 - API, [7](#)
- qtmGetCurrentFrame
 - API, [7](#)
- qtmGetLastEvent
 - API, [8](#)
- qtmGetParameters
 - API, [8](#)
- QTMMsgHeader, [24](#)
 - type, [24](#)
- qtmNew
 - API, [8](#)
- QTMParam3D, [25](#)
- QTMParam3DData, [26](#)
- QTMParam6D, [27](#)
- QTMParamAnalog, [28](#)
- QTMParamBody, [29](#)
- QTMParamCamera, [29](#)
- QTMParamChannel, [30](#)
- QTMParamDevice, [30](#)
- QTMParamForce, [31](#)
- QTMParamGeneral, [32](#)
- QTMParamLabel, [33](#)
- QTMParamParameters, [34](#)
- QTMParamPlate, [35](#)
- QTMParamPoint, [36](#)
- qtmQTMVersion
 - API, [9](#)
- qtmReleaseControl
 - API, [9](#)
- qtmSave
 - API, [9](#)
- qtmShutdown

- API, [9](#)
- qtmStart
 - API, [10](#)
- qtmStop
 - API, [10](#)
- qtmStreamFrames
 - API, [10](#)
- QTMStreamFrequency, [36](#)
- qtmTakeControl
 - API, [11](#)
- QTMVersion, [36](#)
- qtmVersion
 - API, [11](#)
- type
 - QTMMsgHeader, [24](#)