Qt RPC Framework

This project came about as a result of a real requirement that needed communication/IPC between a QML/QtQuick application plugin, written in C++ and a test application that needed to interact with it.The test application conformed to the QTestLib convention and I needed a way of sending remote commands, events and input activity to my C++ QML plugin and also reading back results, states and properties from the plugin.

The original aim was to take my C++ backend, which was a QObject subclass and write some custom functions that used a simple socket connection to transfer the function parameters and return types to call the method on the C++ plugin. However, since this requirement was likely to recur quite frequently, I thought the best approach to be to avoid writing custom remote method calls for each new QML plugin that came along and instead, create something that created the code for me on the client/test application side and do the protocol stuff in a standard way and reusable way.

The side benefit of the latter approach was that I would be able to reuse it not just for testing but for any other case where the goal is to call remote Q_INVOKABLE methods on an instance of a QObject subclass anywhere on the network. I then decided to further extend this by making it possible to connect a signal on one object to a slot on an instance of an object somewhere else on the network. This is not a new idea and has been implemented elsewhere. I did feel it was necessary to add this capability to harness the full power of the Qt metaobject system and give the framework a sense of completeness.

One might ask why go to so much trouble to do something simple like call a procedure on a remote object? I would answer that by saying that you could rewrite the same code and go through the same problems each time you do a bespoke remote procedure call. Each class would end up with its own version (undesirable) and it would be necessary to manually edit the implementation of the QObject subclass for which you want to create the proxy.

Much in the same way as moc generates the moc_<implementation>.cpp for your QObject subclasses that contain the Q_OBJECT macro, I needed a similar concept in order to create a stub object on the client side. This would react and behave identically to the real object with the exception that it forwarded all its Q_INVOKABLE calls to the real instance running somewhere else or in a different Qt application on the same machine. To do this at run time would either require function hooking and code injection or I could simply make a pre-compilation tool, similar in nature to moc, which simply re-writes the implementation of my client-side QObject subclass, alters the .pro file for the containing Qt project so that the application can be rebuilt with the "mock" object.

The project provides for two different use cases. In the first, which you can find a fuller description of in the RpcBinding page, models a proxy server for method calls. Let's say you have a QObject subclass which contains methods with the Q_INVOKABLE tag. You want calls to these methods on a local instance to be forwarded to an identical object on a remote node.

The second use case, QtSignalForwarding, extends the above by adding the ability to connect a signal on a local QObject to a slot on an instance of another QObject on a remote node.