Brief look on the Qt Remote Objects

Timo Aarnipuro 1.10.2018 Public

What is Qt Remote Objects?

- A Qt module that has been in technology preview state since 5.10
- The Qt Remote Objects (QtRO) module provides an easy way to share Qt APIs between processes and devices.
- Planned to be a stable release in the upcoming Qt 5.12
- Provides a way to share a QObject based class without modifying the class itself

Remote Objects

- A Remote Object Source is the QObject that is responsible for the implementation of the exposed API.
- Remote object replicas are proxy objects that have (approximately) the same API as the Source QObject they are replicating. There are a few additional properties and signals to make it possible to detect when the Replica is initialized or if it loses the connectivity to the Source object.

Connection between Remote Objects

- There are two connection types between source and replicas implemented at the moment: QTcpServer and QLocalServer
- The registry provides a simpler way to establish these connections. Every node that wants to be part of the registry's network connects to the registry.
- Qt 5.12 supports QIoDevice, which allows usage of SSL sockets for example.

Usage

- The replica can be generated dynamically or at compile time by using the repc compiler.
- The Replica Compiler (repc) generates QObject header files based on an API definition file. The file (called a "rep" file) uses a specific (text) syntax to describe the API.
- Dynamically created replica objects are empty before a connection is established to the source object.
- C++ calls to dynamically created replica objects are handled through QMetaObject.

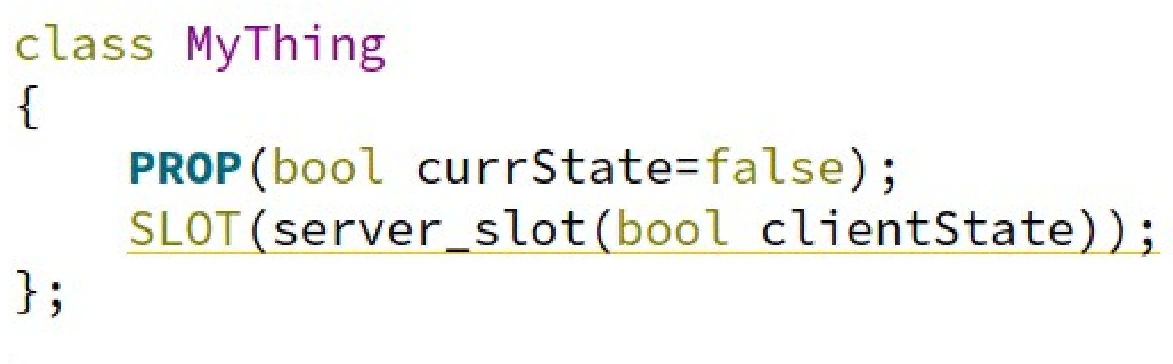
Usage (source side)

- Create the Source object that will be replicated to other nodes (with or without using repc, the Qt Remote Objects Compiler).
- (Optional) Create the Registry. If not used, direct connections are required.
- Create a host node so the source object can be shared.
- Call the node's enableRemoting() function to share the source object.

Usage (replica side)

- (Optional) Use repc to generate a Replica header for your project.
- Create the node that will connect with the Source host node.
- Call the node's acquire() function to create a pointer to a replica.

Example .rep file



```
class MyThing
{
    PROP(bool currState=false);
    SLOT(server_slot(bool clientState));
};
```

- The Replica Compiler creates property change signals automatically
- For example the above currState will have currStateChanged signal generated automatically.

Possible use cases (from GENIVI Wiki)

- +Navigation - map display from IVI to cluster
- +Entertainment (album art)
- -Remote "application" display
- +Telephone book (avatars/photos)
- -Video content (e.g. in cluster, when not driving)
- +Graphical information, triggered by context (people, location, …)
- -Mirroring / Projection mode style CE-device integration and SmartDeviceLink style

Possible use cases (continued)

- +Remote HMI, (car to CE-device)
- + "remote control" for media playback
- + controlling autonomous drive function
- -In-car camera to CE-device (e.g. security from remote location)
- +/-"Home screen" in a large combined HMI, the content could come from different sources.
- -RSE - Media served from one ECU to simple devices in the back seat.
- +/-Synchronized animation - e.g. content moving from one display to another (and consequently between ECUs)

Other use cases

- Consider a sensor, a global positioning system (GPS) receiver for instance. In QtRO terms, the Source would be the process that directly interacts with the GPS hardware and derives your current location. The location would be exposed as QObject properties, and the periodic updates to the location would update the properties and emit property changed signals.
- Settings like night mode or color theme could be shared with remote objects and controlled from multiple domains.

Other use cases

- Come see our multi-screen demo later tonight. It has a remote control interface that has been implemented using Qt Remote Objects.

Thank you!