

Qt priručnik

(mini)

SADRŽAJ

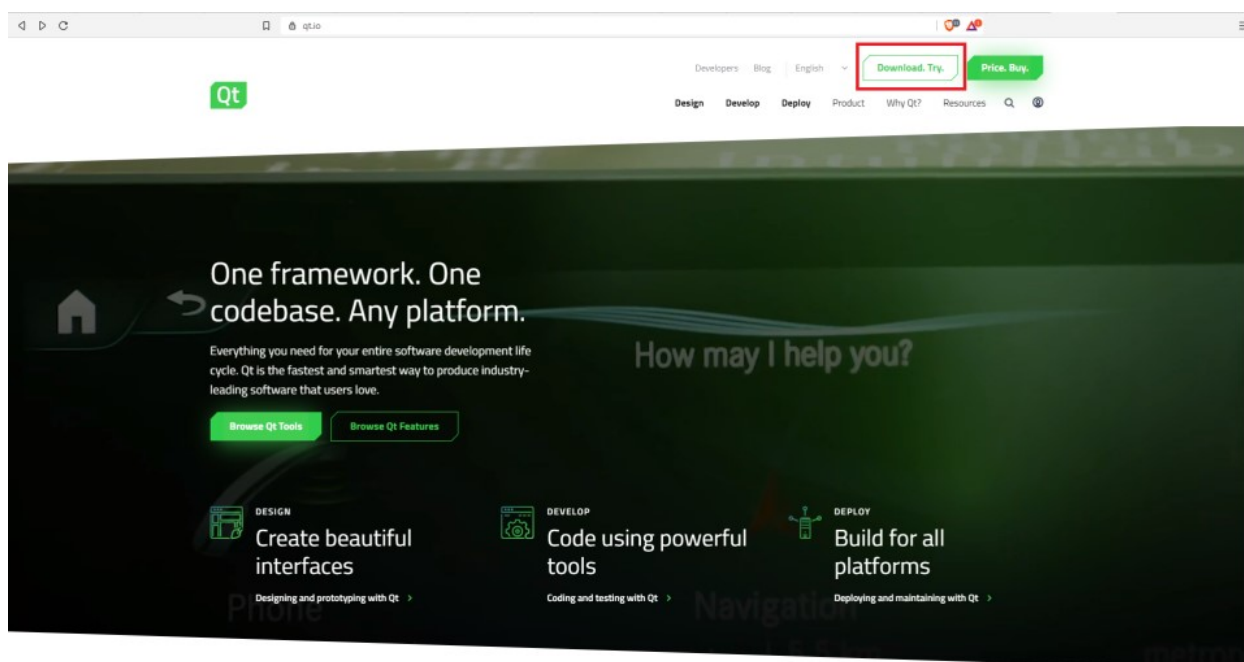
1. Instalacija	1
1.1 Preuzimanje instalera	1
1.2 Instalacioni proces	4
2. Okruženje Qt Creator	6
2.1 Pravljenje novog projekta	6
2.2 Dodavanje postojećeg projekta	14
3. Signali i slotovi	16
3.1 Signali	17
3.2 Slotovi	17
3.3 Poređenje C++ objekta i QObject-a	18
4. Mrežno programiranje u Qt radnom okviru	20
4.1 Klasa QAbstractSocket	20
4.2 Specifičnosti za TCP	24
4.2.1 Klasa QTcpServer	24
4.3 Specifičnosti za UDP	25
4.3.1 Klasa QUdpSocket	25
5. Literatura	27

1. Instalacija

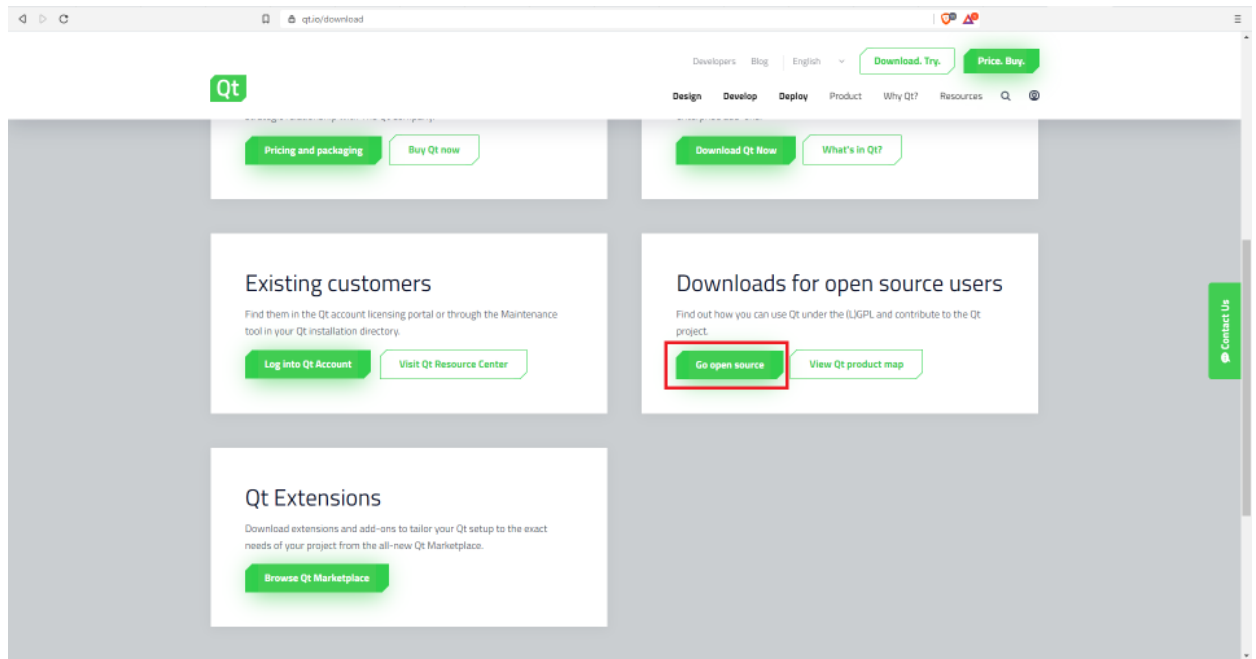
Sa veb prezentacije Qt-a preuzeti i pokrenuti instaler za vašu radnu stanicu:
<https://www.qt.io/>

1.1 Preuzimanje instalera

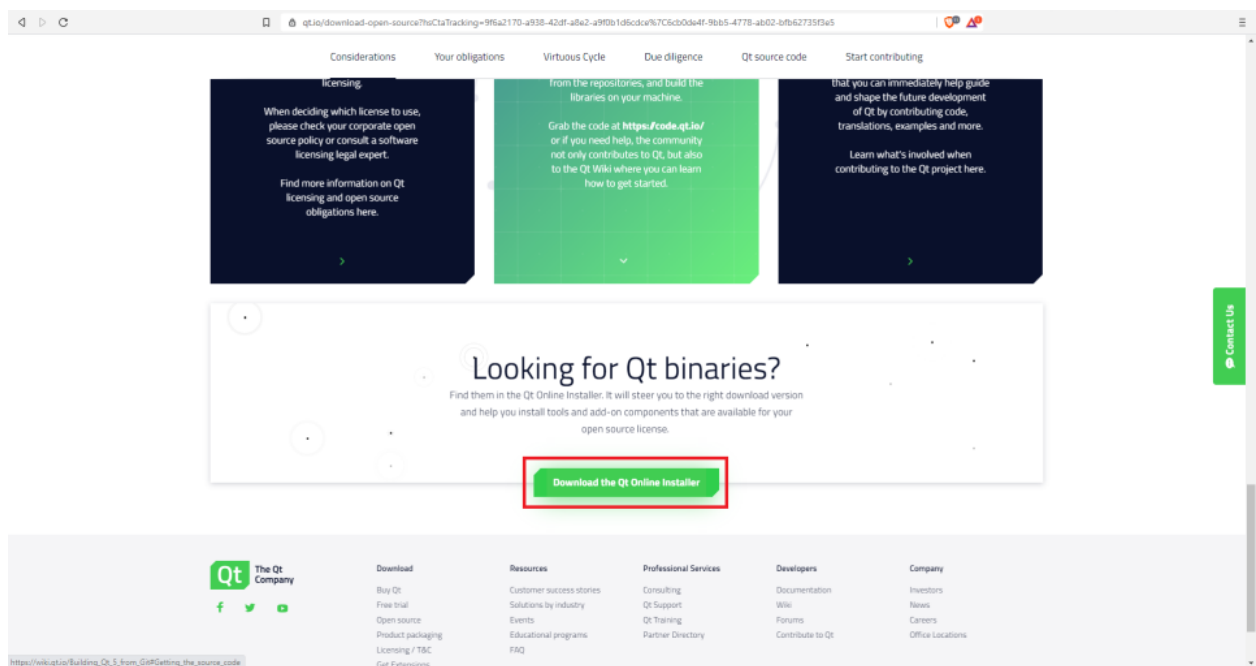
1.



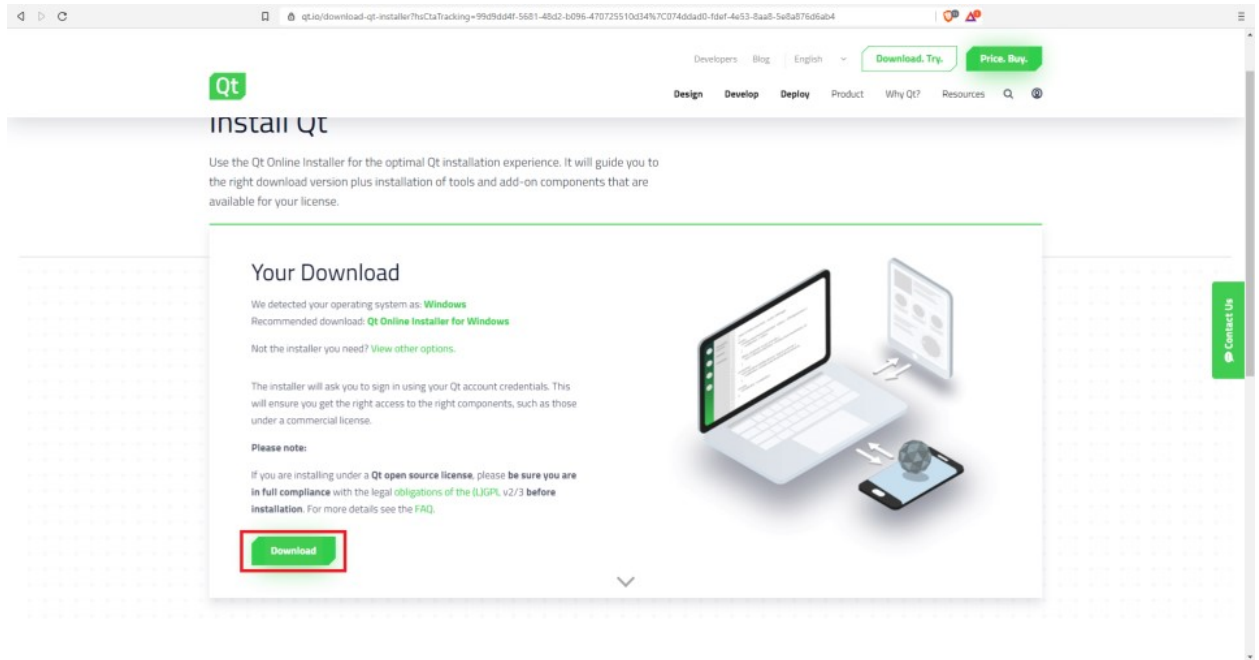
2.



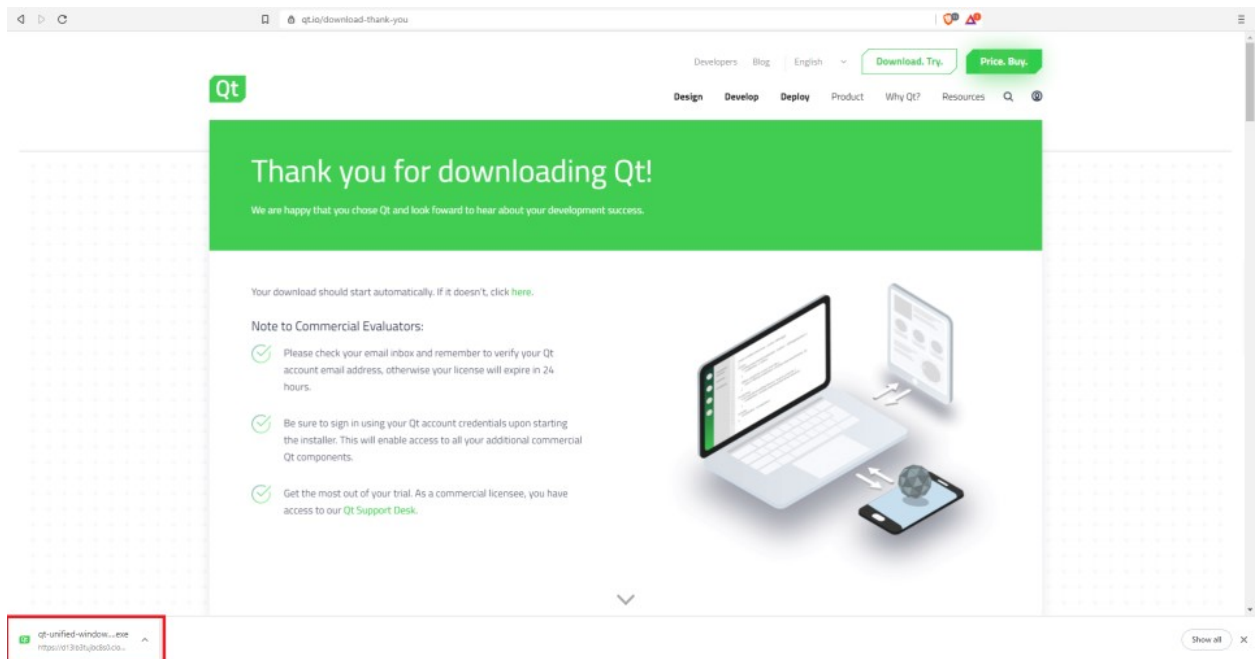
3.



4.



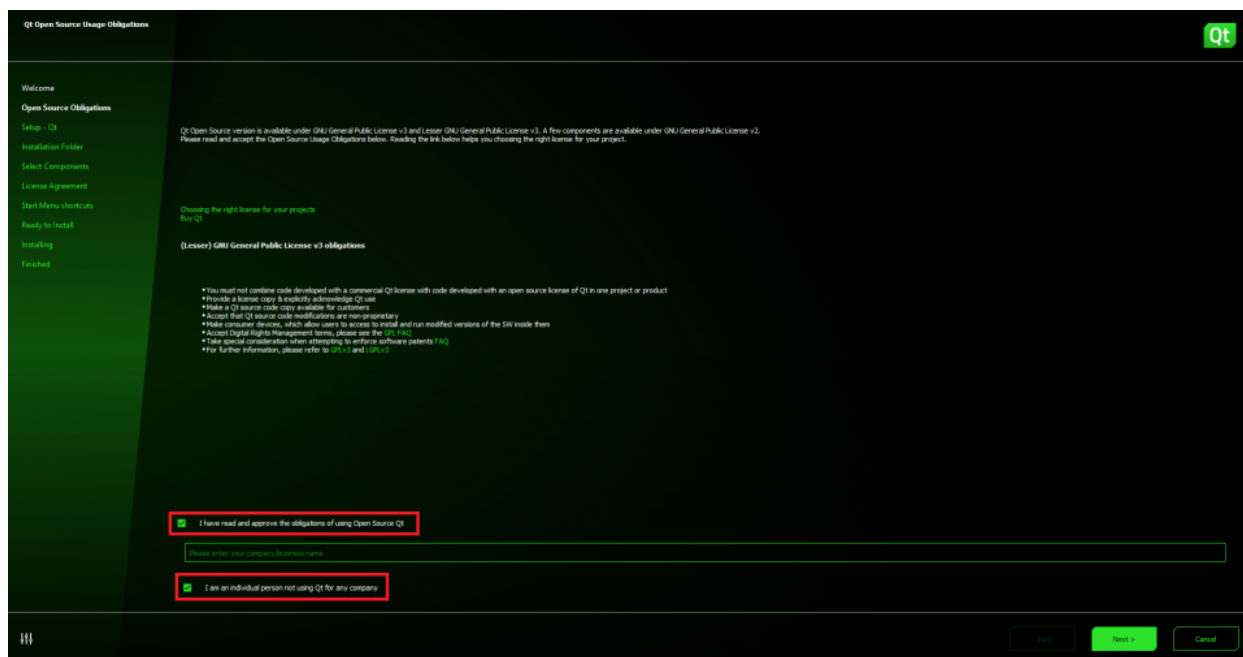
5.



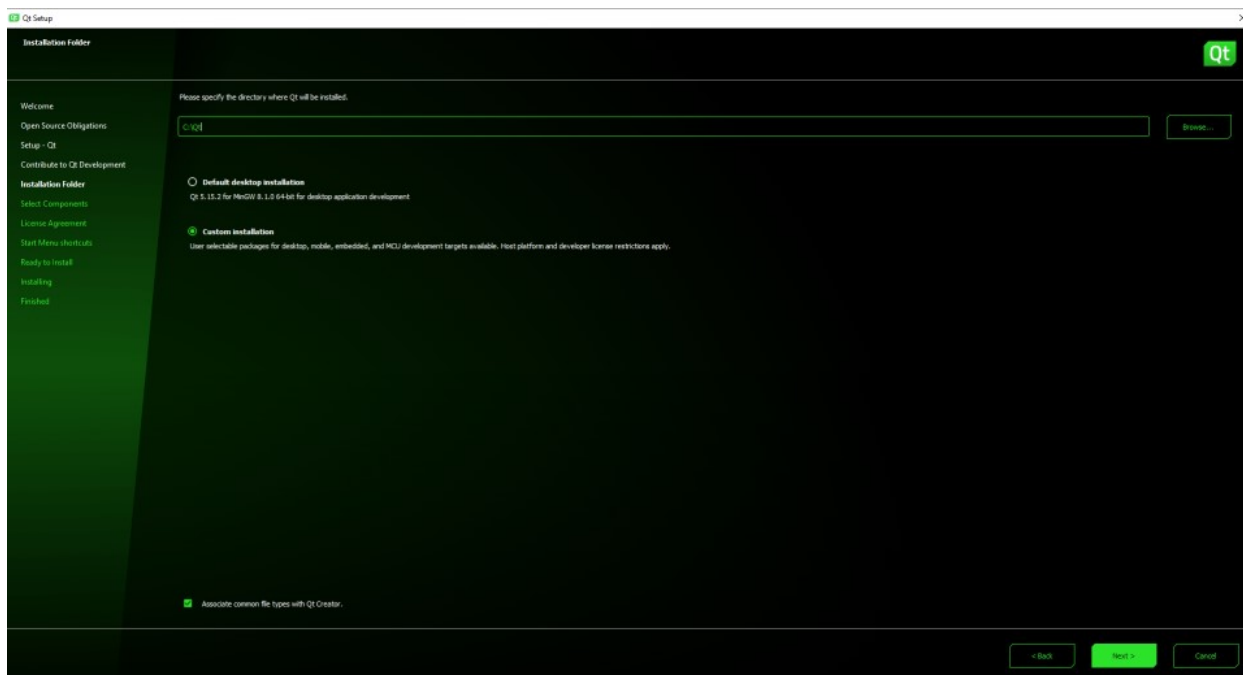
1.2 Instalacioni proces

Prvo je potrebno napraviti korisnički nalog, zatim pratiti sledeće korake:

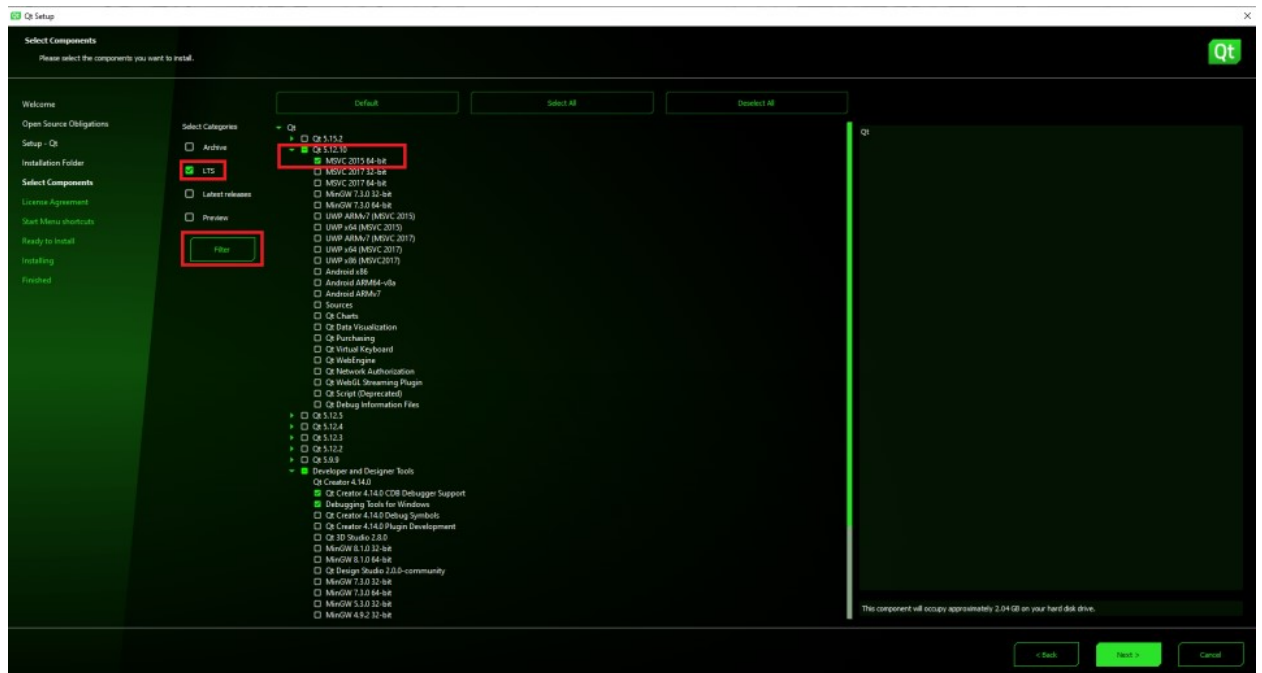
1.



2.



3.

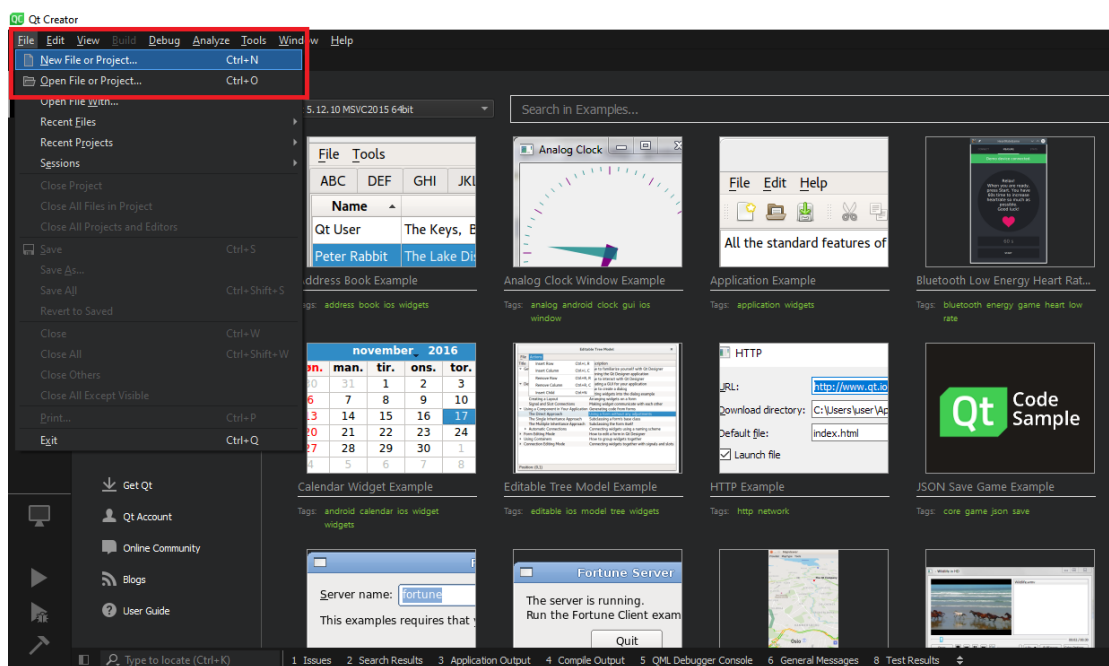


Preostaje samo prihvatiti uslove korišćenja i odabrati dodatne opcije (da li će se napraviti prečica u Start meniju itd.).

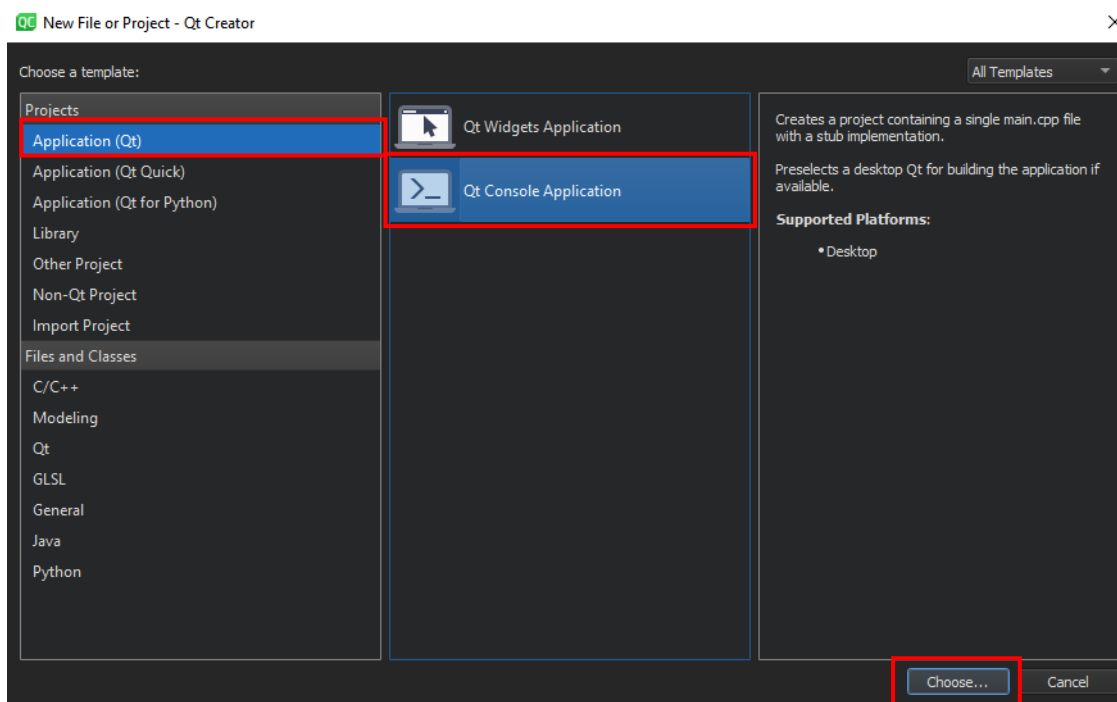
2. Okruženje Qt Creator

2.1 Pravljenje novog projekta

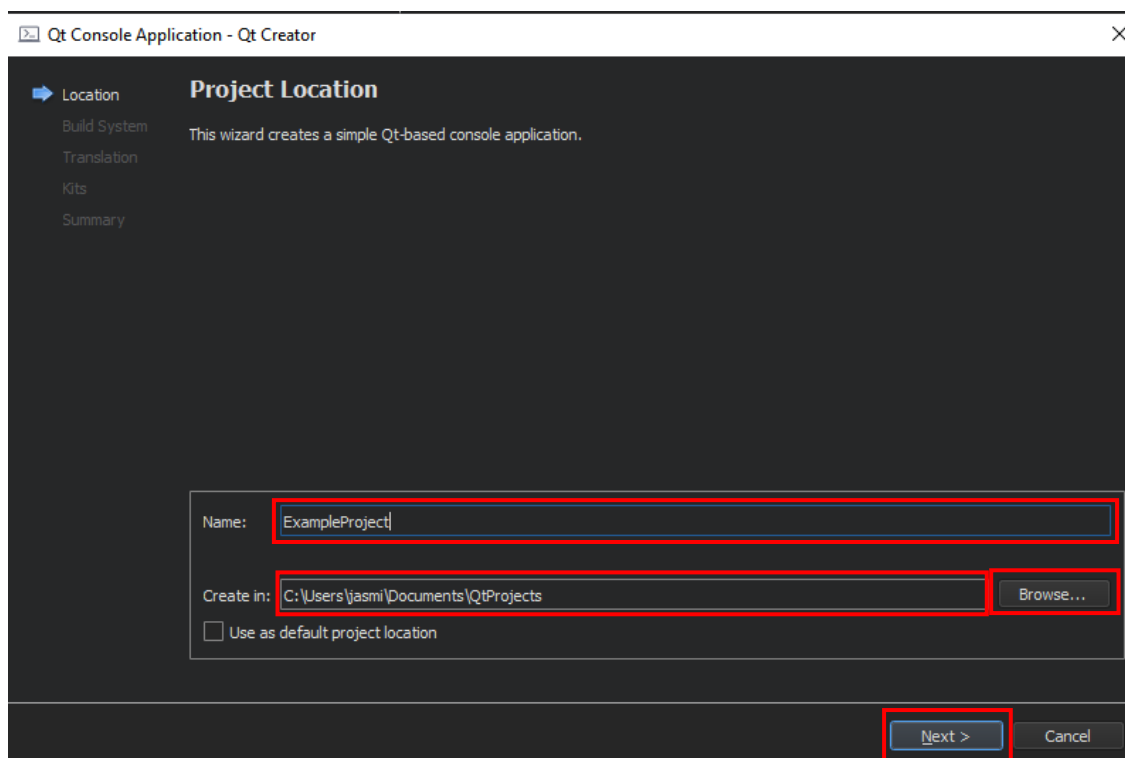
Nov projekat se pravi klikom na **File > New File or Project...** nakon čega će se otvoriti novi prozor za pravljenje projekta.



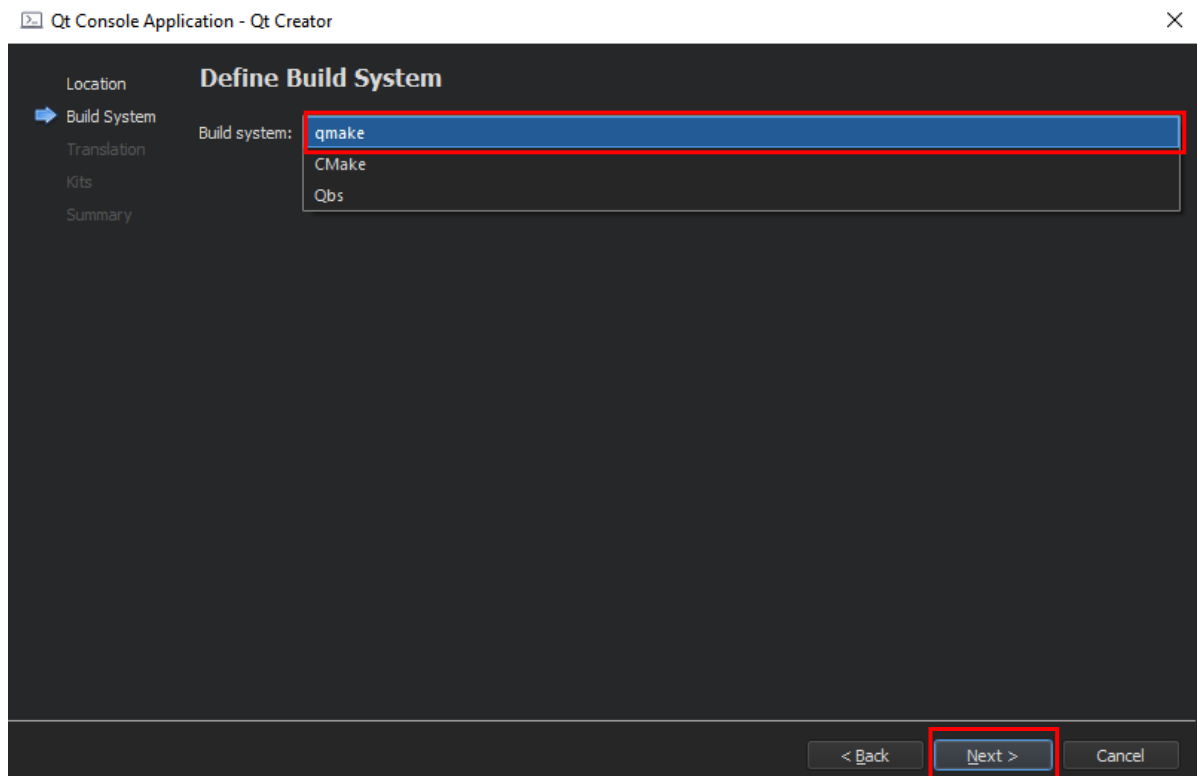
U prozoru za podešavanje novog projekta će se ponuditi opcija za tip projekta. Postoji više opcija (Qt *Widget* aplikacija, Qt konzolna aplikacija, prazan projekat, projekat koji nije zasnovan na Qt-u itd.). Pokazaćemo primer kako se pravi obična Qt konzolna aplikacija. U polju **Projects** potrebno je odabrati opciju **Application (Qt)**, a zatim odabrati **Qt Console Application** opciju s desne strane i zatim pritisnuti dugme **Choose...** u donjem desnom uglu.



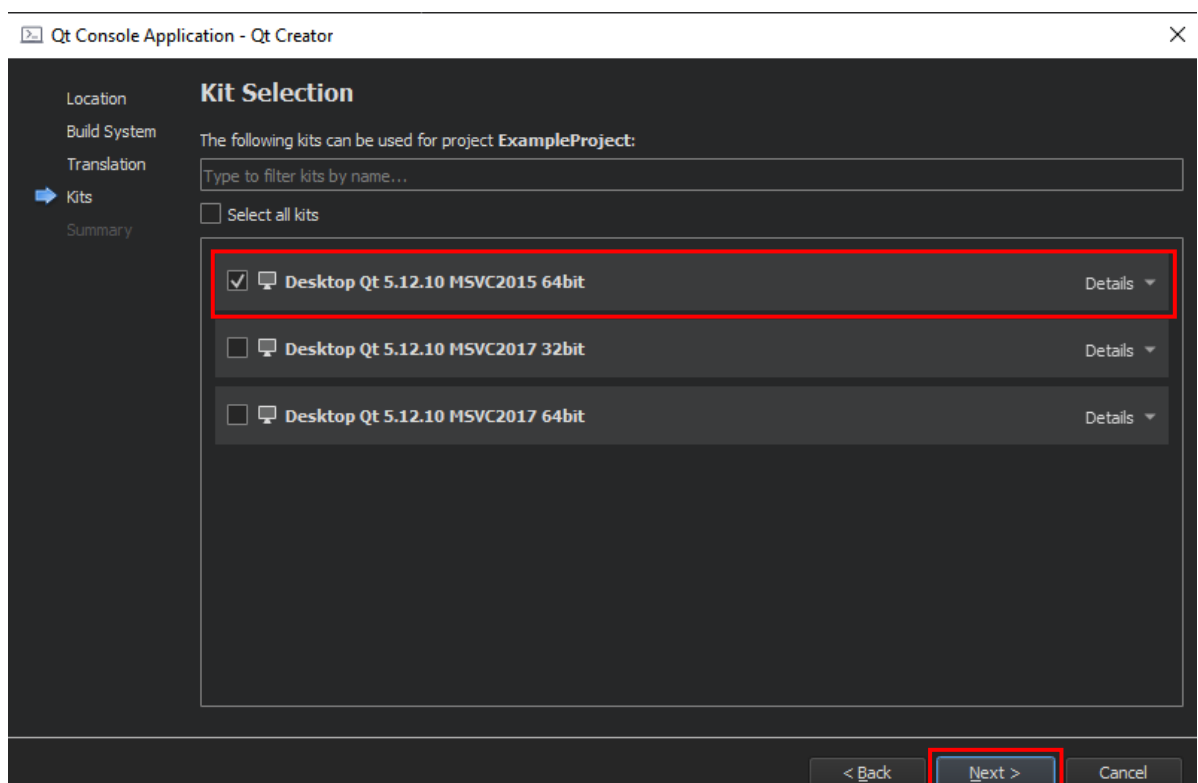
Zatim je potrebno uneti ime projekta i odabrati lokaciju (upisivanjem putanje do foldera ili klikom na dugme **Browse...**) na kojoj će se projekat nalaziti na disku i pritisne se dugme **Next >**. Preporučuje se pravljenje posebnog direktorijuma na disku samo za Qt projekte.



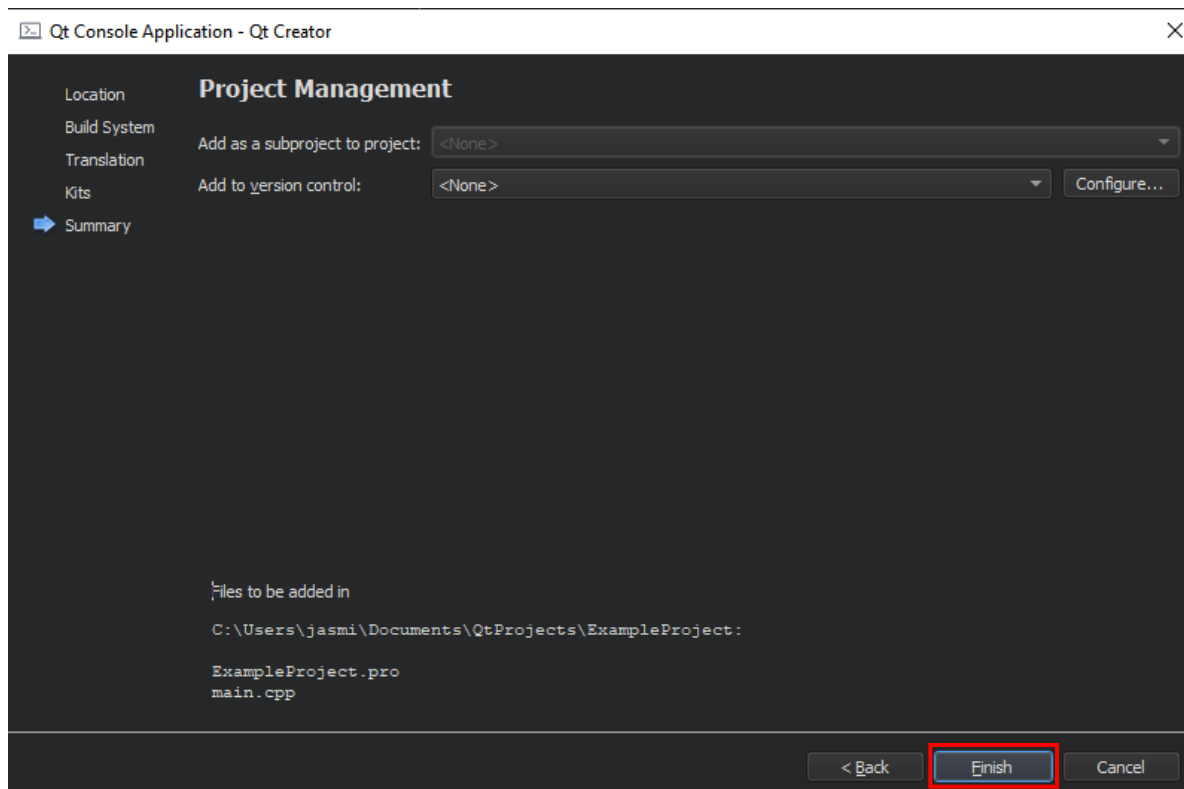
Sledeće je potrebno odabrati *build* sistem. U zavisnosti od instaliranih komponenti, moguće je odabrati jedan od više *build* sistema. Mi ćemo se zadržati na Qt-ovom *build* sistemu, **qmake**. Nakon odabranog *build* sistema, prelazi se na sledeći korak podešavanja projekta. Sledeći prozor će biti *Translation* ukoliko planiramo da prevodimo UI elemente na drugi jezik pomoću Qt-ovog alata *Qt Linguist*. Nama ovo nije od interesa, pa samo ostavljamo prazno i prelazimo na sledeći korak.



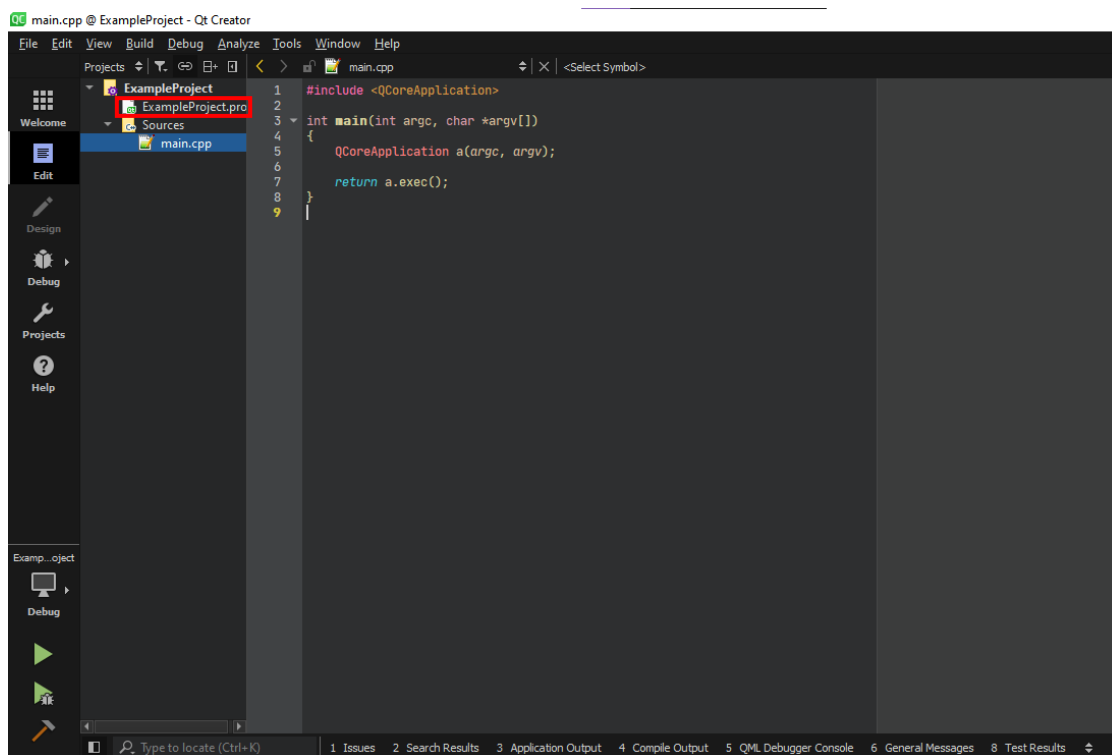
U sledećem prozoru, *Kits*, potrebno je odabrati alate koje želimo da koristimo. Izabrati jedan alat (ili više alata – zbog kompatibilnosti sa različitim verzijama) s kojima želimo da radimo.



U sledećem prozoru će se naći sažetak podešavanja našeg projekta i mogućnost da podesimo neki od sistema za kontrolu verzija (opciono). Klikom na **Finish**, naš projekat će biti napravljen.

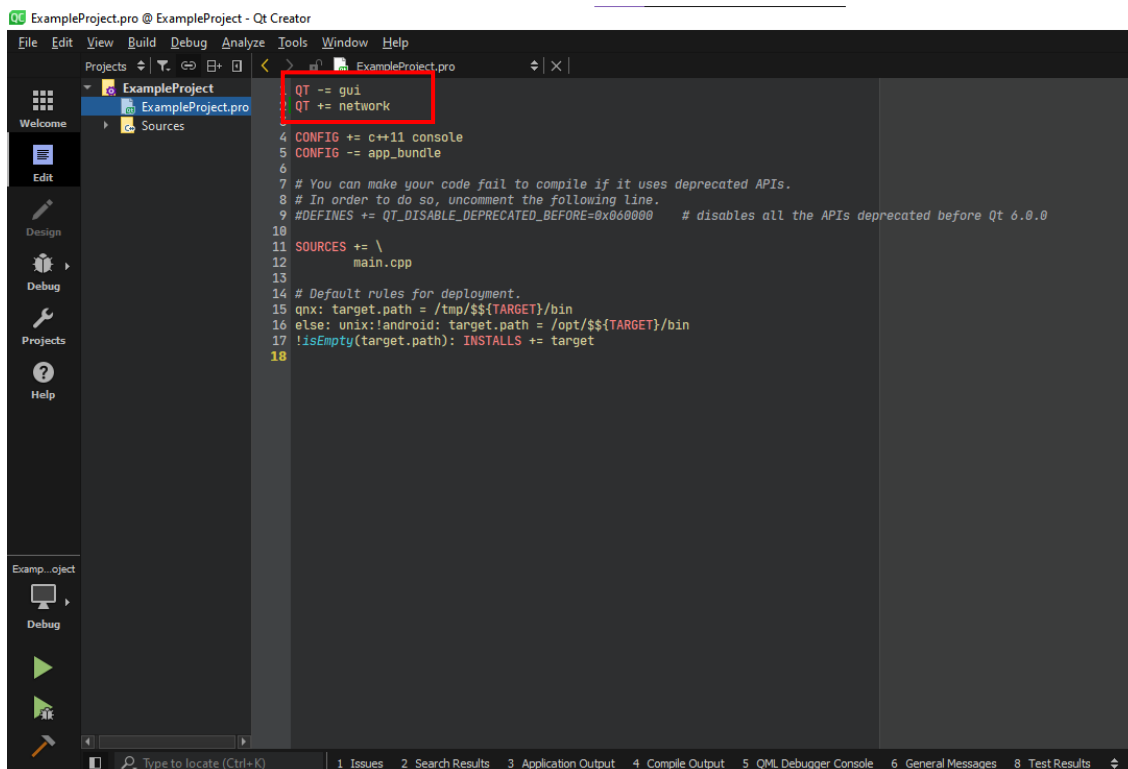


Nakon ovoga, projekat je napravljen i otvoriće se `main.cpp` datoteka vašeg projekta. U prozoru **Projects** prikazaće se upravo napravljen projekat i njegova struktura. Među ostalim datotekama, nalazi se i datoteka sa ekstenzijom `.pro`, koja predstavlja konfiguracionu datoteku sa informacijama koje *qmake* treba da poznaje kako bi pravilno odradio *build* proces.

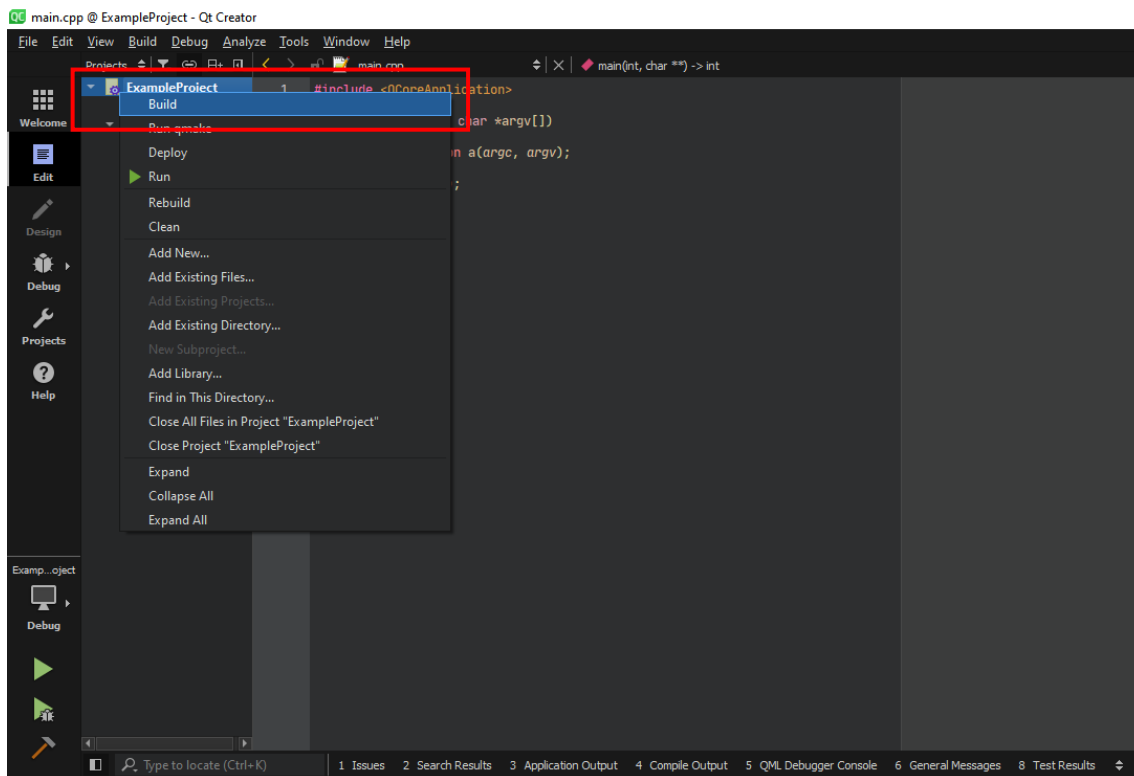


Qt Creator će sam napraviti `.pro` projektnu datoteku, u kojoj se nalaze *qmake* pravila za projekat. Obično će se samo okruženje pobrinuti za dodavanje izvornih datoteka i datoteka zaglavlja u promenljive odgovorne za njih, **SOURCES** i **HEADERS**. Takođe, promenljiva **QT**

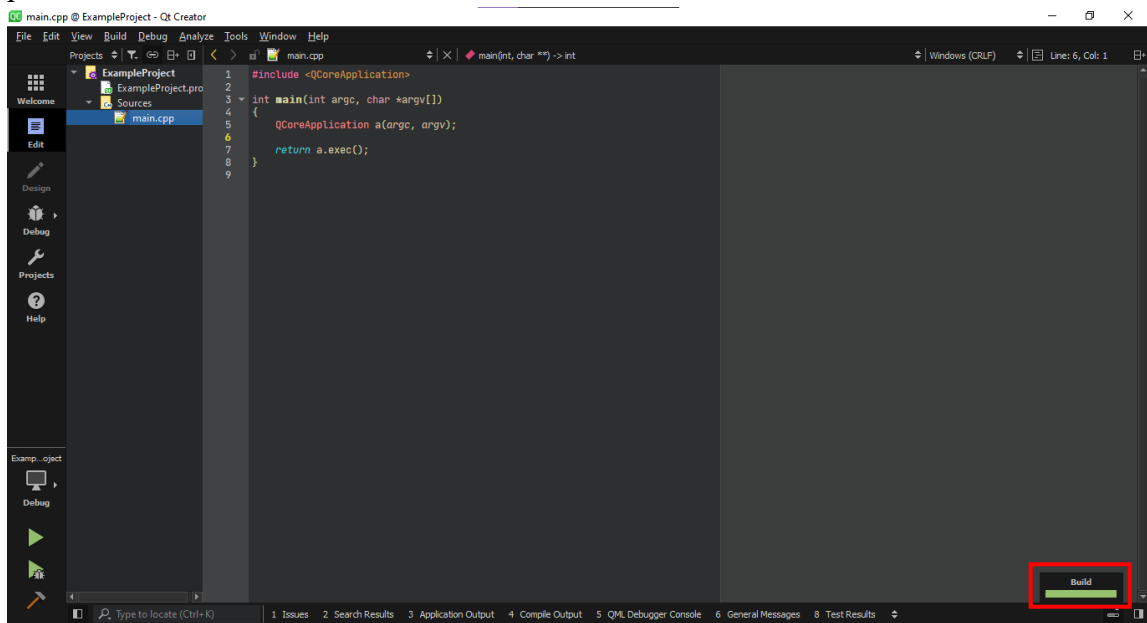
treba da sadrži sve Qt module koji će biti korišćeni. Podrazumevano, **QT** promenljiva sadrži *core* i *gui* module. Dodatne module je moguće uključiti sa += ili isključiti dodate sa -=. Module je obično potrebno ručno uključiti/isključiti.



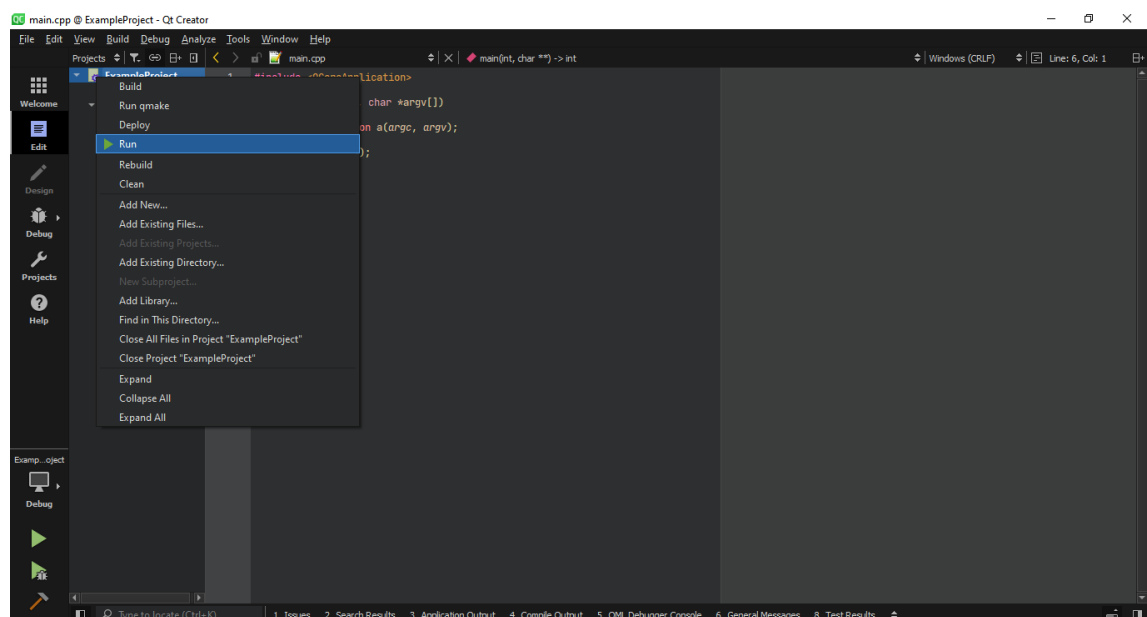
Build proces projekta se može pokrenuti desnim klikom na projekat u **Projects** prozoru, pa zatim klikom na **Build** ili pritiskom kombinacije tastera **Ctrl + B**.



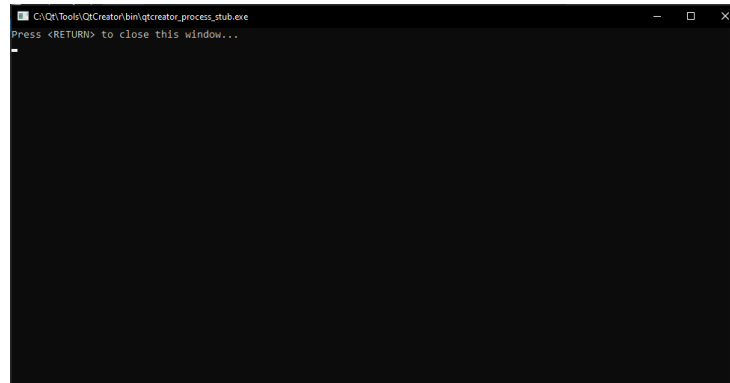
Ukoliko je projekat uspešno prošao kroz *build* proces, u donjem desnom uglu će *progress bar* postati zelen.



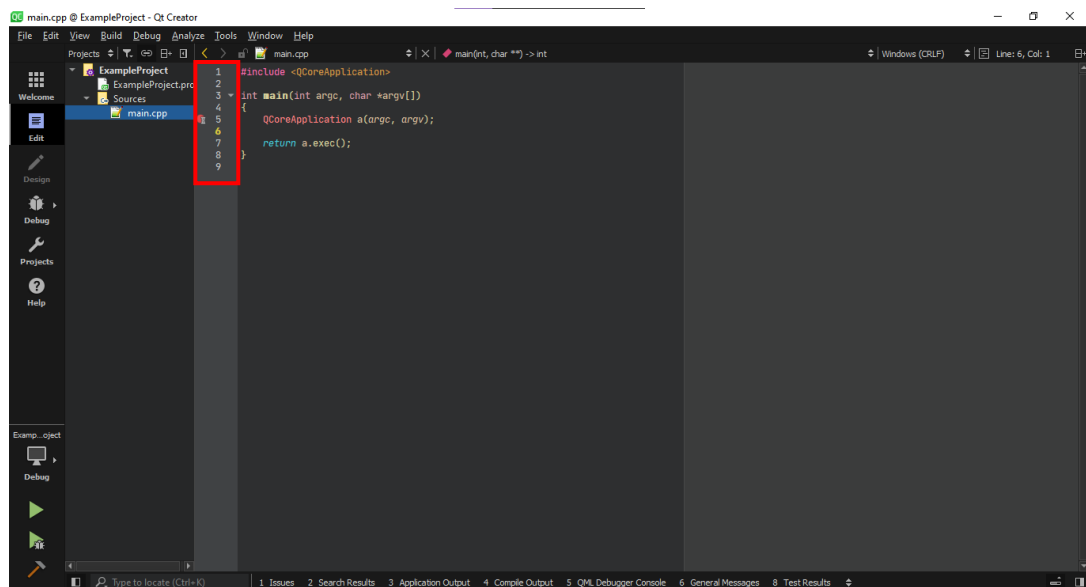
Program se pokreće desnim klikom na projekat, i zatim klikom na opciju **Run** ili pritiskom kombinacije tastera **Ctrl + R**.



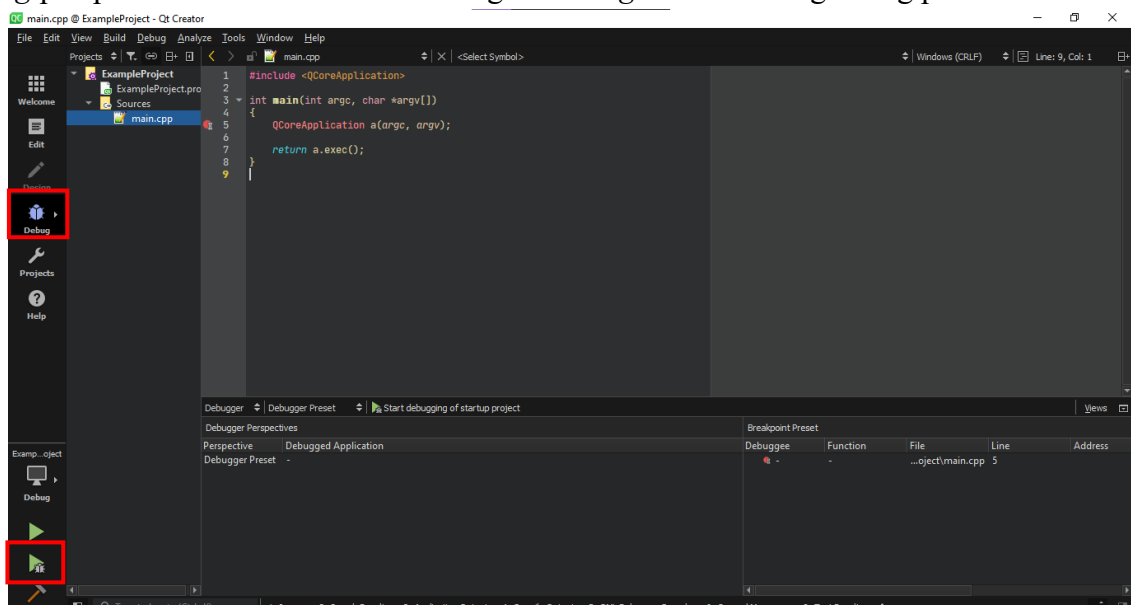
Prekid konzolnog Qt programa se realizuje kombinacijom tastera **Ctrl + C**.



Moguće je i kontrolisano izvršavanje programa (*debug*). Tačka prekida (*breakpoint*) se postavlja levim klikom levo od broja linije na kojoj ju je potrebno postaviti.



Debug perspektiva se otvara klikom na dugme *Debug* s leve strane glavnog prozora.



Kontrolisano izvršavanje programa se pokreće klikom na odgovarajuću ikonicu u donjem levom uglu, klikom na **Debug > Start Debugging > Start Debugging of Startup Project** ili pritiskom tastera **F5**.

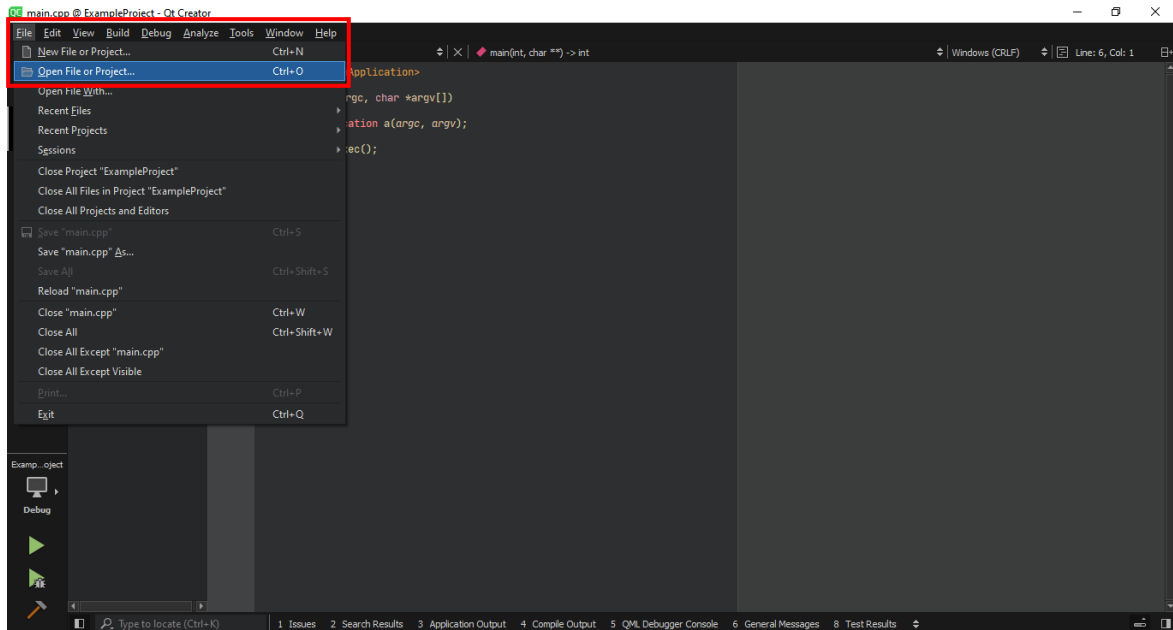
Opcije za izvršavanje korak po korak, sa ulaskom u funkciju ili bez, nastavak izvršavanja do sledeće tačke prekida i terminaciju procesa su predstavljene sa:



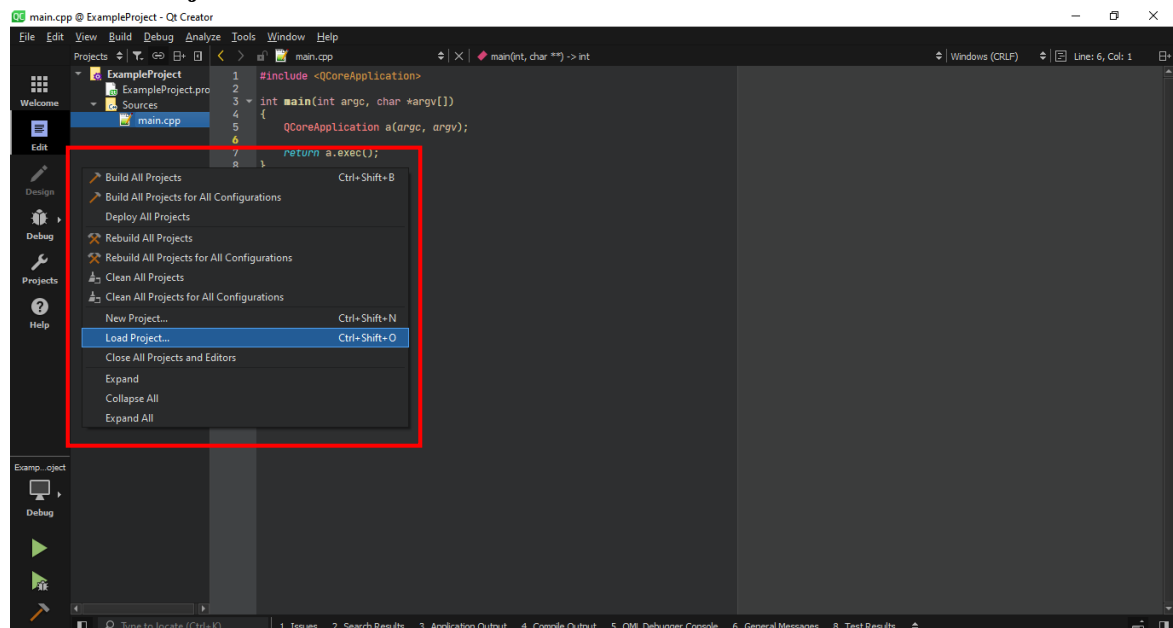
2.2 Dodavanje postojećeg projekta

Postoje 2 mogućnosti:

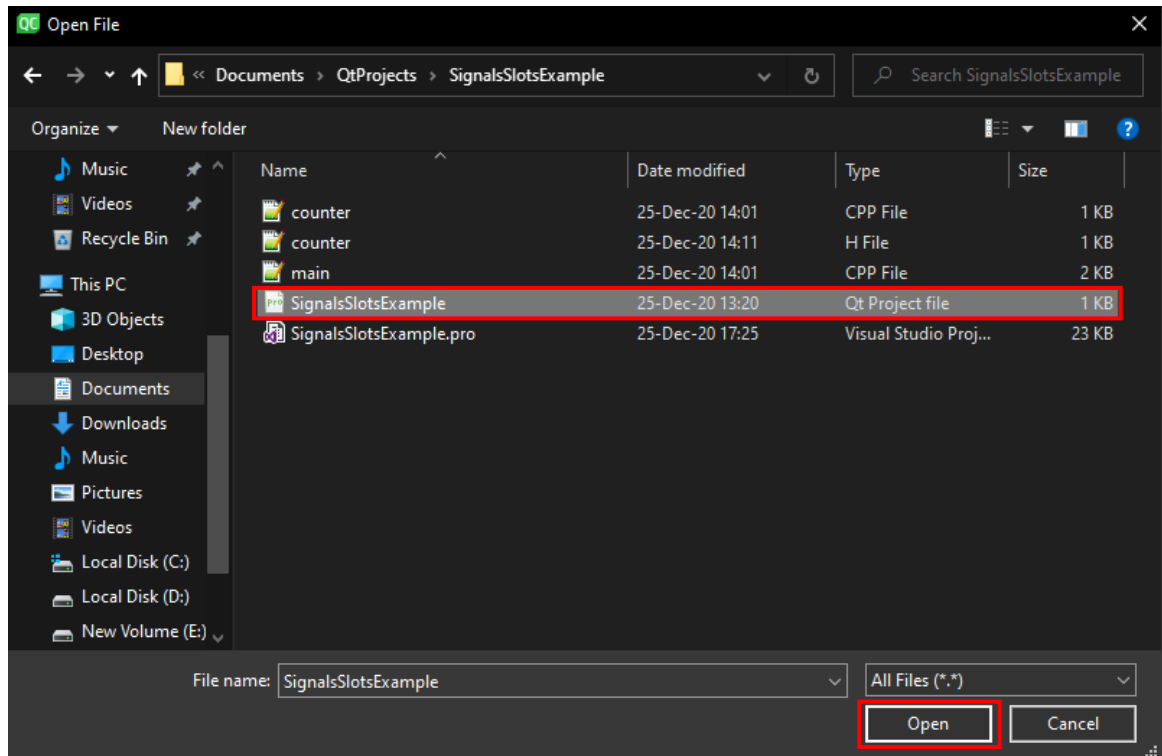
- Klikom na **File > Open File or Project**



- Desnim klikom na prazno mesto u prozoru **Projects**, pa klikom na opciju **Load Projects...**



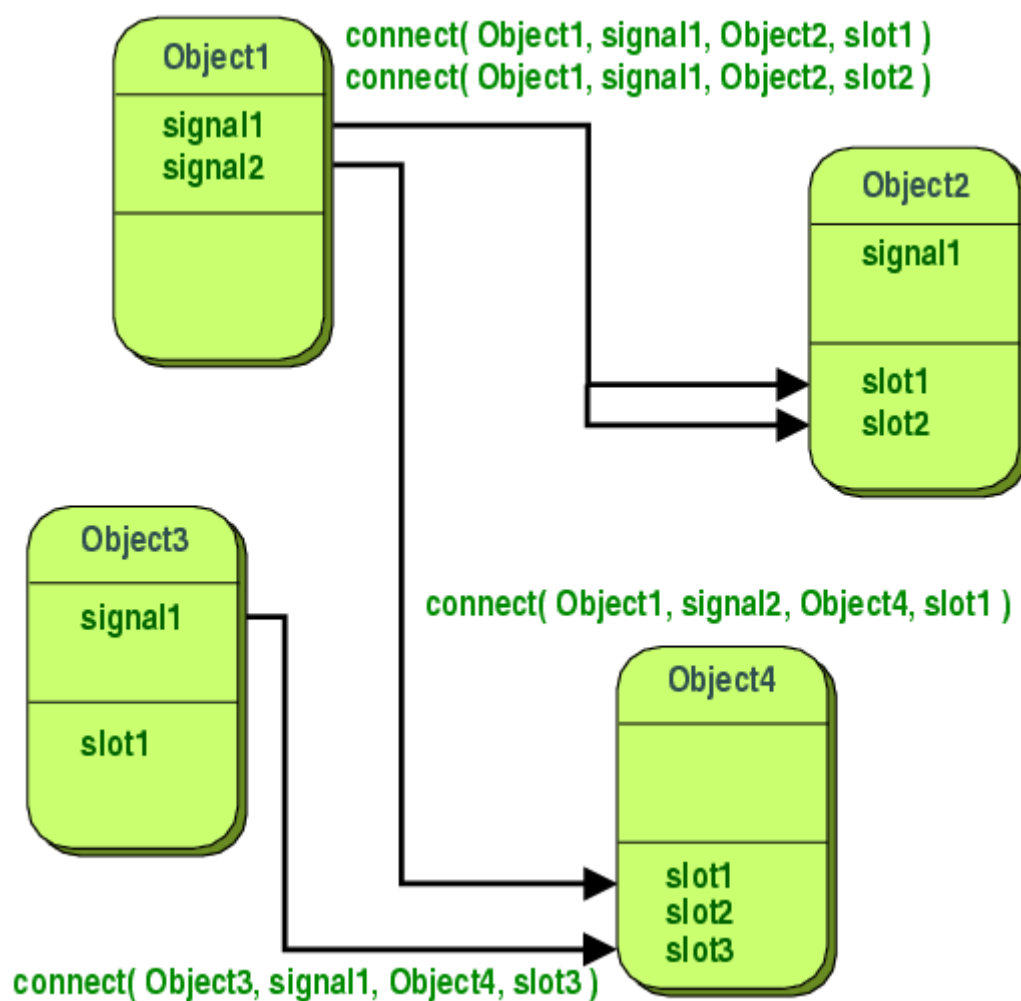
Zatim je potrebno odabrati Qt projektnu datoteku željenog projekta na disku.



3. Signali i slotovi

Mehanizam signala i slotova u Qt-u je veoma moćan mehanizam za komunikaciju koju uvodi Qt. Sličan je konceptu *callback*-a.

Signal se emituje kada se desi određen događaj. Slot je funkcija koja se izvršava kao reakcija na taj signal. Ove veze se vrše „*per instance*“, tj. signal jedne instance objekta vezujemo sa slotom jedne instance neke druge (ili iste) klase, a ne sve instance ovih klasa.



Mehanizam signal-slot je *type-safe*: potpisi signala i slota moraju da se podudaraju (odnosno, slot može i imati manje argumenata, dodatni će se ignorisati). Ovo omogućava proveru poklapanja tipova i detekciju nepoklapanja, kako pri kompajliranju, tako i tokom izvršavanja (*runtime*) ukoliko se iskoriste makroi **SIGNAL ()** i **SLOT ()**.

Takođe, slot radi potpuno nezavisno od signala, tj. ne zna da li uopšte postoje signali povezani za njega. Ovo omogućava da se softverske komponente razvijaju što je opštije moguće i nezavisno jedna od druge.

3.1 Signali

Signale automatski generiše *moc* i ne implementiraju se u izvornoj datoteci. Nikada nemaju povratnu vrednost. Ukoliko se više slotova veže za jedan signal, izvršavaju se redom kojim su bili povezani, kada se signal emituje.

Signale emituju objekti kada se promeni njihovo interno stanje na neki način koji može biti od značaja za taj objekat. Signali su metode sa javnim modifikatorom pristupa (*public*) i mogu se emitovati iz bilo kog dela programa, međutim, preporuka je da se to radi samo iz klase koja definiše taj signal i njenih potklasa.

3.2 Slotovi

Slot se poziva kada se emituje signal koji je vezan za njega. To su obične C++ funkcije i mogu se tako i pozvati – jedino što je posebno u vezi sa njima je da se mogu vezati za signal.

Pošto su to obične C++ funkcije, prate normalna C++ pravila kada su pozvani direktno (npr. što se tiče modifikatora pristupa). Međutim, neki slot, iako proglašen privatnim, može biti pobuđen iz bilo kog dela programa, putem mehanizma slot-signal. Odnosno, signal emitovan iz instance proizvoljne klase može da pobudi privatnu funkciju proglašenu za slot koja pripada njoj nepoznatoj klasi. Upravo ovde se vidi prednost ovog mehanizma – onaj koji emituje signal, i onaj koji emituje slot su „labavo povezani“, tj. ta dva objekta ne moraju da znaju ništa jedan o drugom.

Slotovi mogu biti virtuelne metode i mogu se redefinisati u nasleđenim klasama.

3.3 Poređenje C++ objekta i QObject-a

Uporedimo običnu C++ klasu sa klasom zasnovanom na `QObject` klasi.

```
class Counter
{
public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }
    void setValue(int value);

private:
    int m_value;
};
```

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

Korisnički definisana klasa, ukoliko želi da koristi signale i slotove, *property*-je, i sl. mora da direktno ili indirektno nasledi klasu `QObject`. Potrebno je i uključiti zaglavlje `<QObject>`.

- `Q_OBJECT` – makro potreban *moc*-u da u pozadini obradi `QObject` specifične stvari
- Slotovi se deklarišu nakon ključne reči `slots`:
- Signali se deklarišu nakon ključne reči `signals`:

Signali i slotovi se povezuju pozivom funkcije `QObject::connect()`.

```
bool QObject::connect(const QObject * sender, const char * signal, const QObject * receiver, const char * method, Qt::ConnectionType type = Qt::AutoConnection)
```

Kada objekat `sender` emituje signal `signal`, biće pozvana metoda `method` nad objektom `receiver`. Tip konekcije može ostati na podrazumevanoj vrednosti, tj. da se odredi kada se signal emituje.

Signal se emituje koristeći ključnu reč `emit`.

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

U ovom primeru, objekat `a` ćemo povezati s objektom `b`, tako da, kada se promeni vrednost polja `m_value` u objektu `a`, da se isto dogodi i u objektu `b`. Primetiti da se obrnuto ne dešava, tj. da promena vrednosti tog polja u objektu `b`, neće dovesti do promene u objektu `a`, jer takvu vezu nismo uspostavili.

```
Counter a, b;
QObject::connect(&a, &Counter::valueChanged,
                 &b, &Counter::setValue);

a.setValue(12);    // a.value() == 12, b.value() == 12
b.setValue(48);    // a.value() == 12, b.value() == 48
```

Primer je detaljnije prokomentarisan u priloženom projektu *SignalsSlotsExample*.

4. Mrežno programiranje u Qt radnom okviru

Radni okvir Qt poseduje modul *network* koji nudi, kako klase niskog nivoa za rad sa mrežnim utičnicama (*sockets*), tako i klase višeg nivoa koje su sposobne da obavljaju neke složene mrežne operacije koristeći uobičajene protokole. Mi ćemo se ovde zadržati na osnovnim klasama za rad s mrežnim utičnicama koje koriste *TCP* i *UDP* protokol. Da bi se koristio, ovaj modul je potrebno dodati **QT** promenljivoj okruženja u .pro datoteci (QT += network).

4.1 Klasa QAbstractSocket

Ovu klasu nasleđuju klase `QTcpSocket` i `QUdpSocket`, koje se koriste za TCP, odnosno UDP veze. Sledi pregled bitnijih signala i metoda, koji se mogu koristiti za obe ove nasleđene klase.

Zaglavlje: `<QAbstractSocket>`

Nasleđuje: `QIODevice`

Signali:

- `void QAbstractSocket::connected()`

Signal	Opis
<code>connected</code>	Emituje se nakon uspešnog uspostavljanja veze pomoću <code>connectToHost()</code> .
Parametri	Opis
-	-

- `void QAbstractSocket::disconnected()`

Signal	Opis
<code>disconnected</code>	Emituje se nakon raskida veze na utičnici.
Parametri	Opis
-	-

- `void QAbstractSocket::error(QAbstractSocket::SocketError socketError)`

Signal	Opis
<code>error</code>	Emituje se ukoliko dođe do greške.
Parametri	Opis
<code>socketError</code>	Tip greške.

- `void QAbstractSocket::stateChanged(QAbstractSocket::SocketState socketState)`

Signal	Opis
<code>stateChanged</code>	Emituje se svaki put kada <code>QAbstractSocket</code> promeni stanje.
Parametri	Opis
<code>socketState</code>	Novo stanje.

Bitnije metode:

- `bool QAbstractSocket::bind(const QHostAddress &address, quint16 port = 0, QAbstractSocket::BindMode mode = DefaultForPlatform)`

Metoda	Opis
<code>bind</code>	Povezuje sa adresom <code>address</code> na prolazu <code>port</code> , koristeći <code>mode</code> kao režim.
Parametri	Opis
<code>address</code>	IP adresa s kojom je potrebno povezati utičnicu.
<code>port</code>	Broj prolaza s kojim se treba povezati utičnicu. Ukoliko nije specificiran, dodeliće se nasumičan broj.
<code>mode</code>	Režim povezivanja: deljeni, ekskluzivan, itd. (<i>pogledati dokumentaciju za više detalja</i>)
Povratna vrednost	Opis
<code>bool</code>	Da li je povezivanje završeno uspešno ili ne.

- `void QAbstractSocket::connectToHost(const QString &hostName, quint16 port, QIODevice::OpenMode openMode = ReadWrite, QAbstractSocket::NetworkLayerProtocol protocol = AnyIPProtocol)`

Metoda	Opis
<code>connectToHost</code>	Povezuje sa adresom <code>address</code> na prolazu <code>port</code> , koristeći <code>mode</code> kao režim.
Parametri	Opis
<code>hostName</code>	IP adresa s kojom je potrebno povezati utičnicu. Može biti u u formi stringa (npr. 192.168.0.125) ili data kao <i>hostname</i> (npr. google.com).

port	Broj prolaza hosta.
openMode	Režim pristupa utičnici (pisanje, čitanje, pisanje i čitanje). Vrednosti mogu biti: <code>ReadOnly</code> , <code>WriteOnly</code> , <code>ReadWrite</code> , <code>Append</code> itd.
protocol	Parametar služi da se specificira protokol mrežnog sloja koji se koristi: <code>IPv4</code> ili <code>IPv6</code> . Vrednosti mogu biti: <code>IPv4Protocol</code> , <code>IPv6Protocol</code> ili <code>AnyIPProtocol</code> .

- `QHostAddress QAbstractSocket::peerAddress() const`

Metoda	Opis
peerAddress	Vraća adresu povezanog <i>peer</i> -a.
Parametri	Opis
-	-
Povratna vrednost	Opis
QHostAddress	Adresa povezanog <i>peer</i> -a.

- `quint16 QAbstractSocket::peerPort() const`

Metoda	Opis
peerPort	Vraća broj prolaza povezanog <i>peer</i> -a.
Parametri	Opis
-	-
Povratna vrednost	Opis
quint16	Broj prolaza.

- `void QAbstractSocket::close()`

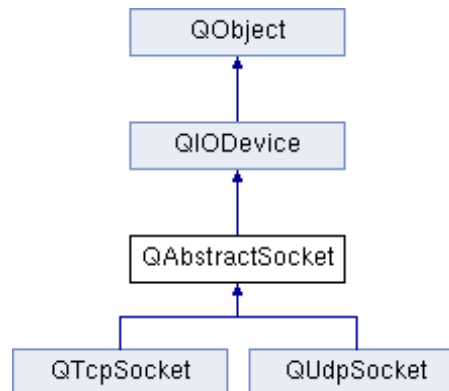
Metoda	Opis
close	Redefiniše istoimenu operaciju roditeljske kase <code>QIODevice</code> i poziva <code>disconnectFromHost()</code> .
Parametri	Opis
-	-

Slanje, odnosno prijem podataka se može realizovati pomoću `read()` i `write()` funkcija za ulazno-izlazne uređaje klase `QIODevice` koju nasleđuje klasa `QAbstractSocket`. Za ostatak metoda, uvedenih enumeracija, tipova, nasleđenih metoda, itd. pogledati dokumentaciju.

Primeri implementacije *TCP* i *UDP* klijenta i servera dati su u priloženim projektima *TCPClient*, *TCPServer*, *UDPClient* i *UDPServer*.

Sledi kratak pregled klasa i njihovih osobina i metoda koje se koriste za mrežno programiranje zasnovano na *TCP* i *UDP* protokolima. Biće izložene samo neke od najbitnijih klasa, za dodatne informacije pogledati zvaničnu dokumentaciju.

Klasni dijagram klasa `QAbstractSocket`, `QTcpSocket` i `QUdpSocket`:



4.2 Specifičnosti za TCP

Što se tiče klase `QTcpSocket`, ne uvode se dodatne funkcionalnosti. Uvodi se klasa `QTcpServer`, koja omogućava prijem nadolazećih *TCP* konekcija i rukovanje njima.

4.2.1 Klasa `QTcpServer`

Zaglavlje: `<QTcpServer>`

Nasleduje: `QObject`

Signali:

- `void QTcpServer::acceptError(QAbstractSocket::SocketError socketError)`

Signal	Opis
<code>acceptError</code>	Emituje se nakon ukoliko dođe do greške pri prihvatanju nove konekcije.
Parametri	Opis
<code>socketError</code>	Opis greške.

- `void QTcpServer::newConnection()`

Signal	Opis
<code>newConnection</code>	Emituje se nakon što se registruje da je dostupna nadolazeća konekcija.
Parametri	Opis
-	-

Bitnije metode:

- `bool QTcpServer::listen(const QHostAddress &address = QHostAddress::Any, quint16 port = 0)`

Metoda	Opis
<code>listen</code>	Govori serveru da započne slušanje na adresi <code>address</code> i broju prolaza <code>port</code> .
Parametri	Opis
<code>address</code>	IP adresa interfejsa na kom server treba da sluša za nadolazećim konekcijama. Ukoliko je postavljeno na <code>QHostAddress::Any</code> , server će slušati na svim interfejsima.
<code>port</code>	Broj prolaza na kojoj server treba da sluša. Ukoliko je prosleđeno 0, bira se nasumičan broj prolaza.
Povratna vrednost	Opis
<code>bool</code>	Da li je operacija obavljena uspešno ili ne.

- `QTcpSocket *QTcpServer::nextPendingConnection()`

Metoda	Opis
<code>nextPendingConnection</code>	Kada postoji konekcija koja čeka prijem, pozivom ove metode se dobavlja pokazivač na povezanu utičnicu.
Parametri	Opis
-	-
Povratna vrednost	Opis
<code>QTcpSocket</code>	Pokazivač na utičnicu prihvaćene konekcije, odnosno <code>nullptr</code> ukoliko ne postoje nadolazeće konekcije.

- `void QTcpServer::close()`

Metoda	Opis
<code>close</code>	Zaustavlja slušanje servera za nadolazećim konekcijama.
Parametri	Opis
-	-

4.3 Specifičnosti za UDP

4.3.1 Klasa QUdpSocket

Zaglavlje: `<QUdpSocket>`

Nasleđuje: `QAbstractSocket`

Signali: nasleđeni od klasa `QAbstractSocket` i `QIODevice`

Bitnije metode:

- `qint64 QUdpSocket::writeDatagram(const QByteArray &datagram, const QHostAddress &host, quint16 port)`

Metoda	Opis
<code>writeDatagram*</code>	Funkcija za slanje datagrama na adresu <code>host</code> sa brojem prolaza <code>port</code> .
Parametri	Opis
<code>datagram</code>	Niz bajtova koji je potrebno poslati.
<code>host</code>	IP adresa na koju se šalje.
<code>port</code>	Broj prolaza.
Povratna vrednost	Opis
<code>qint64</code>	Broj poslatih bajtova, odnosno -1 u slučaju greške.

***NAPOMENA:** ovu funkciju ne treba pozivati nad utičnicom koja je vezana pomoću `connectToHost()`, već je potrebno pozvati metodu `write()` nasleđenu od `QIODevice`

- `qint64 QUdpSocket::readDatagram(char *data, qint64 maxSize, QHostAddress *address = nullptr, quint16 *port = nullptr)`

Metoda	Opis
<code>readDatagram</code>	Funkcija za prijem datagrama.
Parametri	Opis
<code>data</code>	Bafer za prijem podataka.
<code>maxSize</code>	Maksimalan broj bajta koji se može primiti.
<code>address</code>	Promenljiva u kojoj se skladišti IP adresa pošiljaoca datagrama.
<code>port</code>	Promenljiva u kojoj se skladišti broj prolaza pošiljaoca datagrama.
Povratna vrednost	Opis
<code>qint64</code>	Veličina primljenog datagrama u bajtima, odnosno -1 u slučaju neuspeha.

Postoji još nekoliko varijanti ovih metoda (sa različitim parametrima, povratnim vrednostima, itd.) koje realizuju istu radnju prijema/slanja datagrama. Takođe, postoje i metode za rad sa *multicast* i *broadcast* tipom usmeravanja. Za dodatne informacije konsultovati zvaničnu dokumentaciju.

5. Literatura

- [1] Signals & Slots | Qt Core 5.12.10 (n.d.). doc.qt.io. Retrieved December 25, 2020, from <https://doc.qt.io/qt-5.12/signalsandslots.html>
- [2] Qt Documentation | Home. (n.d.). doc.qt.io. Retrieved December 25, 2020, from <https://doc.qt.io/>
- [3] Network Programming with Qt | Qt Network 5.12.10 (n.d.). doc.qt.io. Retrieved December 26, 2020, from <https://doc.qt.io/qt-5.12/qtnetwork-programming.html>