QGIS Developers Guide

Release 2.18

QGIS Project

April 08, 2019

Contents

Bine ați venit în Paginile Dezvoltatorului QGIS. Aici veți găsi regulile, instrumentele și pașii necesari pentru a contribui eficient, și cu ușurință, la codul QGIS.

CHAPTER 1

Standardele de codificare QGIS

Aceste standarde ar trebui să fie urmate de către toți dezvoltatorii QGIS.

1.1 Clase

1.1.1 Nume

Clasele din QGIS sunt prefixate cu Qgs, iar fiecare cuvânt începe cu majusculă.

Exemple:

- `QgsPoint`
- `QgsMapCanvas`
- `QgsRasterLayer`

1.1.2 Membri

Denumirile membrilor claselor sunt prefixate cu litera m și sunt formate utilizând majuscule și minuscule.

- `mMapCanvas`
- `mCurrentExtent`

Toți membrii claselor ar trebui să fie privați. Membrii publici sunt TOTAL nerecomandați. Membrii protejați ar trebui să fie evitați când membrul ar putea fi accesat din subclase Python, de vreme ce membrii protejați nu pot fi utilizați din legăturile Python.

Mutable static class member names should begin with a lower case s, but constant static class member names should be all caps:

- `sRefCounter`
- `DEFAULT_QUEUE_SIZE`

1.1.3 Funcțiile Accesor

Valorile membrilor unei clase vor fi obținute prin intermediul funcțiilor accesor. Numele acestora nu ar trebui să înceapă cu prefixul "get". Funcțiile accesor pentru cei doi membri privați de mai sus ar putea fi:

- `mapCanvas()`
- `currentExtent()`

Asigurați-vă că accesoriile sunt corect marcate cu `const`. Atunci când este cazul, este posibil ca variabilele membru, de tip cache de valoare, să fie marcate cu `mutable`.

1.1.4 Funcții

Numele funcțiilor încep cu literă mică și conțin majuscule și minuscule. Numele funcțiilor ar trebui să indice scopul acestora.

- `updateMapExtent()`
- `setUserOptions()`

În concordanță cu API-urile QGIS și Qt, ar trebui evitate abrevierile. De exemplu: `setDestinationSize` în loc de `setDestSize`, `setMaximumValue` în loc de `setMaxVal`.

De asemenea, acronimele ar trebui să fie denumite folosind CamelCase. De exemplu: `setXml` în loc de `setXML`.

1.1.5 Argumente ale funcțiilor

Function arguments should use descriptive names. Do not use single letter argments (e.g. `setColor( const QColor& color )` instead of `setColor( const QColor& c )`).

Acordați o atenție deosebită momentului când argumentele ar trebui să fie transmise prin referință. Cu excepția cazului în care obiectele argumentului nu au dimensiuni mari și sunt trivial de copiat (cum ar fi obiectele QPoint), ele ar trebui să fie transmise prin referințe constante. Pentru concordanța cu API-ul Qt, chiar și obiectele partajate în mod implicit sunt transmise prin referințe constante (ex.: `setTitle( const QString& title )` în loc de `setTitle( QString title )`.

1.1.6 Valorile Returnate de Funcție

Return small and trivially copied objects as values. Larger objects should be returned by const reference. The one exception to this is implicitly shared objects, which are always returned by value.

- `int maximumValue() const`
- `const LayerSet& layers() const`
- `QString title() const` (QString is implicitly shared)
- `QList< QgsMapLayer* > layers() const` (QList is implicitly shared)

1.2 Qt Designer

1.2.1 Clasele Generate

Clasele QGIS care sunt generate în fișierele produse de Qt Designer (UI) ar trebui să aibă sufixul Base. Acest lucru identifică o clasă de bază, generată.

Exemple:

- `QgsPluginManagerBase`
- `QgsUserOptionsBase`

1.2.2 Dialoguri

Toate dialogurile ar trebui să implementeze ajutorul pentru toate pictogramele barei de instrumente și alte controale grafice relevante. Baloanele cu indicii adaugă foarte mult la descoperirea caracteristicilor, atât pentru utilizatorii noi, cât și pentru cei experimentați.

Asigurați-vă că ordinea filelor pentru controalele grafice este actualizată ori de câte ori se modifică aspectul unui dialog.

1.3 Fișierele C++

1.3.1 Nume

Fișierele antet și de implementare C++ ar trebui să aibă extensiile .h, respectiv .cpp. Numele de fișier ar trebui să conțină doar litere mici și, în cazul claselor, să se potrivească numelui clasei.

Example: Class `QgsFeatureAttribute` source files are `qgsfeatureattribute.cpp` and `qgsfeatureattribute.h`

Note: În cazul în care nu este clară indicația de mai sus, cerința ca numele de fișier să se potrivească numelui de clasă, arată că, implicit, fiecare clasă trebuie să fie declarată și implementată în propriul fișier. Acest lucru facilitează nou-veniților identificarea codului care este specific anumitor clase.

1.3.2 Antetul Standard și Licența

Fiecare fișier sursă trebuie să conțină o secțiune antet, în conformitate cu exemplul următor:

```
/***************************************************************************
  qgsfield.cpp - Describes a field in a layer or table
  --------------------------------------
  Date : 01-Jan-2004
  Copyright: (C) 2004 by Gary E.Sherman
  Email: sherman at mrcc.com
/***************************************************************************
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 ***************************************************************************/
```

Note: Există un șablon pentru Qt Creator în git. Pentru a-l utiliza, copiați-l din `doc/qt_creator_license_template` într-un dosar local, ajustați adresa e-mail și numele - dacă este necesar - apoi configurați QtCreator pentru a-l folosi: *Instrumente* → *Opțiuni* → *C++* → *Denumirea Fișierului*.

1.4 Numele Variabilei

Numele variabilelor locale încep cu o literă mică, utilizând majuscule și minuscule. Nu folosiți prefixe precum `my` sau `the`.

Exemple:

- `mapCanvas`
- `currentExtent`

1.5 Tipurile Enumerate

Tipurile enumerate ar trebui să fie denumite folosind CamelCase, începând cu o majusculă, de ex.:

```
enum UnitType
{
  Meters,
  Feet,
  Degrees,
  UnknownUnit
};
```

Nu utilizați tipuri de nume generice, care vor intra în conflict cu alte tipuri. De ex., folosiți mai degrabă `UnkownUnit` decât `Unknown`

1.6 Constantele locale și Comenzile Macro

Constantele locale și comenzile macro ar trebui să fie scrise cu majuscule, separate cu ajutorul caracterului de subliniere, de ex .:

```
const long GEOCRS_ID = 3344;
```

1.7 Semnale și Sloturi Qt

Toate conexiunile la semnale/sloturi ar trebui să fie făcute folosindu-se legăturile de "stil nou" disponibile în Qt5. Informații suplimentare privind această cerință sunt disponibile în QEP #77.

Evitați folosirea sloturilor cu autoconectare QT (adică pe acelea denumite `void on_mSpinBox_valueChanged`). Sloturile cu autoconectare sunt fragile și predispuse la întrerupere, fără avertisment, dacă dialogurile sunt refactorizate.

1.8 Editarea

Orice editor de text/IDE poate fi folosit pentru a edita codul QGIS, oferind garanția că sunt îndeplinite următoarele cerințe.

1.8.1 Caracterele TAB

Setați în editorul dvs. emularea TAB-ului cu ajutotul spațiilor albe. Ar trebui folosite în acest scop 2 spații.

Note: În vim, acest lucru se realizează prin comanda `set expandtab ts=2`

1.8.2 Indentarea

Source code should be indented to improve readability. There is a `scripts/prepare-commit.sh` that looks up the changed files and reindents them using astyle. This should be run before committing. You can also use `scripts/astyle.sh` to indent individual files.

As newer versions of astyle indent differently than the version used to do a complete reindentation of the source, the script uses an old astyle version, that we include in our repository (enable WITH_ASTYLE in cmake to include it in the build).

1.8.3 Acoladele

Acoladele ar trebui să fie poziționate pe linia următoare expresiei:

```
if(foo == 1)
{
  // do stuff
  ...
}
else
{
  // do something else
  ...
}
```

1.9 Compatibilitatea API-ului

There is API documentation for C++.

Încercăm să păstrăm API-ul stabil și compatibil cu versiunile anterioare. Curățarea API-ului ar trebui să fie făcută într-un mod similar cu al codului sursă Qt, de ex.:

```
class Foo
{
  public:
    /** This method will be deprecated, you are encouraged to use
     *  doSomethingBetter() rather.
     *  @deprecated doSomethingBetter()
     */
    Q_DECL_DEPRECATED bool doSomething();

    /** Does something a better way.
     *  @note added in 1.1
     */
    bool doSomethingBetter();

  signals:
    /** This signal will be deprecated, you are encouraged to
     *  connect to somethingHappenedBetter() rather.
     * @deprecated use somethingHappenedBetter()
     */
#ifndef Q_MOC_RUN
    Q_DECL_DEPRECATED
#endif
    bool somethingHappened();

    /** Something happened
     *  @note added in 1.1
     */
    bool somethingHappenedBetter();
}
```

1.10 Stilul de Codificare

Aici sunt descrise câteva sugestii și sfaturi de programare care vor reduce, sperăm, erorile, timpul de întreținere și dezvoltare.

1.10.1 Generalizați Codul Atunci Când Este Posibil

Decât să duplicați un anumit cod, mai bine luați în considerare consolidarea acestuia într-o funcție unică.

Acest lucru vă va permite să:

- efectuați modificări într-o singură locație în loc de mai multe
- preveniți încurcarea codului
- împiedica apariția, de-a lungul timpului, a diferențelor între secțiunile identice de cod, făcându-le, astfel, mai greu de înțeles și de întreținut de către alții

1.10.2 Se preferă poziționarea Constantelor înaintea Predicatelor

Se preferă poziționarea constantelor la începutul predicatelor.

`0 == value` în loc de `value == 0`

Acest lucru va ajuta prevenirea folosirii accidentale de = în loc de ==, care poate introduce erori subtile de logică. Compilatorul va genera o eroare dacă folosiți accidental = în loc de == pentru comparații, deoarece nu se pot atribui valori constantelor.

1.10.3 Spațiile Albe Vă Pot Fi De Ajutor

Adăugarea de spații între operatori, declarații și funcții, facilitează utilizatorilor analiza codului.

Care cod este mai ușor de citit? Acesta:

```
if (!a&&b)
```

sau acesta:

```
if ( ! a && b )
```

Note: `scripts/prepare-commit.sh` will take care of this.

1.10.4 Puneți comenzile pe linii separate

La citirea codului, este ușor să omiteți comenzile dacă acestea nu se află la începutul liniei. La parcurgerea rapidă a codului, este normal să omiteți liniile atunci când acestea nu prezintă ceea ce căutați în primele câteva caractere. Este, de asemenea, normal să vă așteptați la o comandă după un condițional ca `if`.

Se ia în considerare următorul cod:

```
if (foo) bar();

baz(); bar();
```

Este foarte ușor să omiteți o parte din fluxul de control. Scrieți, în schimb

```
if (foo)
  bar();

baz();
bar();
```

1.10.5 Indentați modificatorii de acces

Modificatorii de acces structurează o clasă în secțiuni de API public, API protejat și API privat. Rolul lor este de a grupa codul pe această structură. Indentați modificatorii de acces și declarațiile.

```
class QgsStructure
{
  public:
    /**
     * Constructor
     */
     explicit QgsStructure();
}
```

1.10.6 Cărți recomandate

- Effective Modern C++, Scott Meyers
- More Effective C++, Scott Meyers
- Effective STL, Scott Meyers
- Design Patterns, GoF

Ar trebui, de asemenea, să citiți acest articol de la Qt Quarterly despre proiectarea în stilul Qt (API)

1.11 Recunoașterea contribuțiilor

Contribuitorii la noile funcții sunt încurajați să-i informeze pe ceilalți despre contribuția lor prin:

- adăugarea unei note la jurnalul schimbărilor, pentru prima versiune în care a fost încorporat codul, de tipul:

```
This feature was funded by: Olmiomland http://olmiomland.ol
This feature was developed by: Chuck Norris http://chucknorris.kr
```

- writing an article about the new feature on a blog, and add it to the QGIS planet http://plugins.qgis.org/planet/
- adăugarea numelui lor la:
  - https://github.com/qgis/QGIS/blob/master/doc/CONTRIBUTORS
  - https://github.com/qgis/QGIS/blob/master/doc/AUTHORS
  - https://github.com/qgis/QGIS/blob/master/doc/contributors.json

CHAPTER 2

Accesul GIT

Această secțiune vă arată cum să începeți lucrul cu depozitul GIT al QGIS. Înainte de a face acest lucru, trebuie să aveți un client git instalat pe sistemul dumneavoastră.

2.1 Instalarea

2.1.1 Instalarea git pentru GNU/Linux

Utilizatorii distribuției Debian pot efectua:

```
sudo apt-get install git
```

2.1.2 Instalarea git pentru Windows

Windows users can obtain msys git or use git distributed with cygwin.

2.1.3 Instalarea git pentru OSX

The git project has a downloadable build of git. Make sure to get the package matching your processor (x86_64 most likely, only the first Intel Macs need the i386 package).

O dată descărcat, deschideți imaginea discului și executați programul de instalare.

Notă PPC/sursă

Site-ul Git nu oferă compilații PPC. În cazul în care trebuie să construiți o compilație PPC, sau doriți doar ceva mai mult control asupra instalării, trebuie să efectuați singuri compilarea.

Download the source from http://git-scm.com/. Unzip it, and in a Terminal cd to the source folder, then:

```
make prefix=/usr/local
sudo make prefix=/usr/local install
```

Dacă nu aveți nevoie de facilități suplimentare, Perl, Python sau TclTk (GUI), aveți posibilitatea să le dezactivați înainte de a rula comanda make:

```
export NO_PERL=
export NO_TCLTK=
export NO_PYTHON=
```

2.2 Accesarea Depozitului

Pentru a clona QGIS master:

```
git clone git://github.com/qgis/QGIS.git
```

2.3 Selectați o ramură

Pentru a selecta o ramură, de exemplu, ramura 2.6.1:

```
cd QGIS
git fetch
git branch --track origin release-2_6_1
git checkout release-2_6_1
```

Pentru a selecta ramura master:

```
cd QGIS
git checkout master
```

Note: În QGIS păstrăm codul nostru cel mai stabil din ramura versiunii actuale. Ramura master conține codul pentru așa-numita serie de versiuni 'instabile'. Periodic vom ramifica o versiune pe care o vom stabiliza continuu și în care vom încorpora selectiv noi caracteristici.

Parcurgeți fișierul INSTALL din arborele sursă, pentru instrucțiunile specifice privind construirea versiunilor de dezvoltare.

2.4 Sursele documentației QGIS

În cazul în care sunteți interesat în verificarea surselor documentației QGIS:

```
git clone git@github.com:qgis/QGIS-Documentation.git
```

Puteți arunca, de asemenea, o privire la fișierele readme incluse în depozitul cu documentație, pentru mai multe informații.

2.5 Sursele site-ului web QGIS

În cazul în care sunteți interesat în verificarea surselor site-ului web QGIS:

```
git clone git@github.com:qgis/QGIS-Website.git
```

Puteți arunca, de asemenea, o privire la fișierele readme incluse în depozitul site-ului web, pentru mai multe informații.

2.6 Documentația GIT

Parcurgeți următoarele site-uri pentru informații despre cum putți deveni un maestru GIT.

- http://gitref.org
- http://progit.org
- http://gitready.com

2.7 Dezvoltarea în ramuri

2.7.1 Scopul

Complexitatea codului sursă QGIS a crescut considerabil în ultimii ani. Prin urmare, sunt greu de anticipat efectele secundare pe care le va avea adăugarea unei noi funcționalități. În trecut, proiectul QGIS avea cicluri de lansare cu durată mare, deoarece era necesară o cantitate mare de muncă pentru a reface stabilitatea sistemului software, în urma adăugării unor noi caracteristici. Pentru a depăși aceste probleme, QGIS a trecut la un model de dezvoltare în cazul în care noile caracteristici sunt introduse mai întâi în ramurile GIT, și abia apoi are loc fuziunea cu masterul (ramura principală), după ce au fost finisate și au devenit stabile. Această secțiune descrie procedurile de ramificare și de fuzionare din cadrul proiectului QGIS.

2.7.2 Procedura

- **Anunțul inițial pe listele de discuții:** Înainte de a începe, postați un anunț pe lista de discuții a dezvoltatorilor, pentru a vedea dacă un alt dezvoltator lucrează deja la aceeași caracteristică. De asemenea, contactați consilierul tehnic al comitetului (PSC) de coordonare a proiectului. În cazul în care noua caracteristică necesită o modificare a arhitecturii QGIS, este necesară o cerere de comentariu (RFC).

Crearea unei ramuri: Creați o nouă ramură GIT, pentru dezvoltarea noii funcțiuni.

```
git checkout -b newfeature
```

Acum puteți începe dezvoltarea. Dacă aveți de gând să lucrați intensiv în această ramură, și să lucrați alături de alți dezvoltatori, care să aibă drepturi de scriere în depozitul părinte, puteți urca depozitul dvs. în depozitul oficial al QGIS:

```
git push origin newfeature
```

Note: În cazul în care ramura există deja, modificările dvs. vor fi urcate acolo.

Reîncorporați, în mod regulat, în ramura master: Este recomandabil să reintroduceți modificările în ramura master, în mod regulat. Acest lucru face mai ușoară fuziunea ulterioară a ramurii cu masterul. După încorporare trebuie să efectuați `git push -f` asupra depozitului dvs. derivat.

Note: Niciodată nu efectuați `git push -f` în depozitul original! Folosiți această comandă numai pentru ramura în care lucrați.

```
git rebase master
```

2.7.3 Documentation on wiki

It is also recommended to document the intended changes and the current status of the work on a wiki page.

2.7.4 Efectuarea de teste înaintea fuzionării cu ramura master

Când ați terminat cu noua caracteristică și vă mulțumește stabilitatea, faceți un anunț pe lista dezvoltatorilor. Înainte de fuziunea cu masterul, modificările vor fi testate de către dezvoltatori și utilizatori.

2.8 Transmiterea de corecții și a cererilor de actualizare

Iată câteva indicații care vă vor ajuta să obțineți cu ușurință corecții și solicitări de actualizare pentru proiectele QGIS, facilitându-ne gestionarea corecțiilor.

2.8.1 Cererile de actualizare

În general, este mai ușor pentru dezvoltatori dacă trimiteți cereri de actualizare a codului de pe GitHub. Pentru detalii asupra acestui aspect, faceți referire la documentația GitHub pull request.

Dacă faceți o cerere de actualizare vă rugăm să actualizați cu regularitate ramura master din PR, astfel încât un PR să poată actualiza ramura master părinte.

If you are a developer and wish to evaluate the pull request queue, there is a very nice tool that lets you do this from the command line

Vă rugăm să consultați secțiunea de mai jos pentru a 'obține atenție pentru corecția dvs.'. În general, atunci când trimiteți o cerere PR ar trebui să vă asumați responsabilitatea de a o urmări până la finalizare - să răspundeți la întrebările postate de alți dezvoltatori, să identificați un 'susținător' pentru funcționalitatea propusă de dvs., și să transmiteți un memento manierat, uneori, dacă vedeți că PR-ul nu este luat în considerare. Vă rugăm să țineți cont de faptul că proiectul QGIS este condus printr-un efort voluntar, fiind posibil ca oamenii să nu participe instantaneu la PR. Dacă observați că un PR nu beneficiază de atenția pe care o merită, opțiunile de accelerare a aceastuia ar trebui să fie (în ordinea priorității):

- Trimiterea unui mesaj în lista de discuții, pentru a 'populariza' PR-ul, indicând și cât de bine ar fi dacă ar fi inclus în baza de cod.
- Trimiterea unui mesaj persoanei din lista de PR-uri, căreia i-a fost asignată cererea dvs.
- Trimiterea unui mesaj lui Marco Hugentobler (care gestionează lista de PR-uri).
- Trimiterea unui mesaj comitetului director al proiectului, cerându-le să vă ajute la încorporarea PR-ului în codul de bază.

Cele mai bune practici pentru crearea unei cereri de actualizare

- Creați întotdeauna o ramură pentru o funcționalitate, pornind de la ramura master curentă.
- Atunci când codificați ramura pentru o caracteristică, nu "îmbinați" nimic cu acea ramură, ci mai degrabă folosiți comanda rebase, așa cum este descris în punctul următor, pentru a păstra curat istoricul.

- Înaintea creării unei cereri de actualizare a codului efectuați `git fetch origin` și `git rebase origin/master` (datorită faptului că originea reprezintă remote pentru părinte, efectuați `.git/config` sau `git remote -v | grep github.com/qgis`).
- Ați putea executa comanda Git rebase, în mod repetat, fără a provoca nici un prejudiciu (atât timp cât singurul scop al ramificării dvs. este de a obține fuzionarea cu ramura master).
- Atenție: După rebase trebuie să efectuați `git push -f` ' în depozitul ramificat. **DEZVOLTATORI DE BAZĂ: NU FACEȚI ASTA ÎNTR-UN DEPOZIT QGIS PUBLIC**!

Etichete speciale pentru a notifica documentatorii

Pe lângă etichetele comune pe care le puteți adăuga pentru a clasifica PR-urile dvs., există unele speciale, pe care le puteți utiliza pentru a genera automat rapoarte de probleme în depozitul Documentației QGIS, de îndată ce codul dumneavoastră este combinat:

- `[needs-docs]` pentru a aminti creatorilor documentației să ceară adăugarea de informații suplimentare, după corectarea sau îmbunătățirea unei funcționalități existente deja.
- `[feature]` în cazul noilor functionalități. Adăugarea unei bune descrieri PR-ului va reprezenta un bun început.

Rugăm dezvoltatorii să folosească aceste etichete (nu se face diferenţiere între litere mari şi mici), astfel încât creatorii documentației să identifice problemele pe care le au și să aibă o privire de ansamblu asupra lucrurilor de făcut. DAR, faceți-vă, de asemenea, timp pentru a adăuga un text: fie în commit, fie chiar în documentație.

Pentru îmbinarea unei cereri de actualizare

Opţiunea A:

- faceți clic pe butonul de îmbinare (Se creează o îmbinare non-fast-forward)

Opţiunea B:

- Verificați cererea de actualizare
- Testare (De asemenea, necesar pentru opțiunea A, evident)
- checkout master, git merge pr/1234
- Opțional: `git pull --rebase`: Creează fast-forward, fără "merge commit". Istoricul este mai curat, dar este mai greu de anulat îmbinarea.
- `git push` (NICIODATĂ nu folosiți opțiunea -f aici)

2.9 Denumirea fişierului de corecție

If the patch is a fix for a specific bug, please name the file with the bug number in it e.g. bug777fix.patch, and attach it to the original bug report in trac.

If the bug is an enhancement or new feature, its usually a good idea to create a ticket in trac first and then attach your patch.

2.10 Crearea corecției la nivelul superior al directorului sursă QGIS

Procedând în acest mod, aplicarea corecțiilor este mai ușoară, deoarece nu trebuie să navigăm într-un loc specific din arborele sursă pentru aplicarea unei corecții. De asemenea, atunci când primim corecții, de obicei, le vom

evalua folosind îmbinarea, iar aplicarea corecției la nivel directorului superior facilitează acest lucru. Mai jos este un exemplu despre cum se pot include în corecție, din directorul de nivel superior, mai multe fișiere modificate:

```
cd QGIS
git checkout master
git pull origin master
git checkout newfeature
git format-patch master --stdout > bug777fix.patch
```

Acest lucru vă asigură că ramura master este sincronizată cu depozitul părinte, și apoi generează o corecție care conține diferențele dintre ramura noii funcționalități și ramura master.

2.10.1 Obținerea atenției necesare pentru corecția dvs.

QGIS developers are busy folk. We do scan the incoming patches on bug reports but sometimes we miss things. Don't be offended or alarmed. Try to identify a developer to help you - using the Technical Resources and contact them asking them if they can look at your patch. If you don't get any response, you can escalate your query to one of the Project Steering Committee members (contact details also available in the Technical Resources).

2.10.2 Măsuri de Protecţie

QGIS este licențiat sub GPL. Ar trebui să depuneți toate eforturile pentru a vă asigura că trimiteți numai corecții care nu sunt grevate de drepturi de proprietate intelectuală, de natură conflictuală. De asemenea, nu trimiteți un cod care nu se încadrează în GPL.

2.11 Obținerea Accesului de Scriere în GIT

Accesul de scriere în arborele sursei QGIS are loc pe bază de invitație. În mod obișnuit, atunci când o persoană depune mai multe corecții substanțiale (nu există un număr fix), care să demonstreze competența și înțelegerea C++ de bază și a convențiilor de codificare QGIS, unul dintre membrii PSC, sau alți dezvoltatori, pot nominaliza persoana respectivă la acordarea accesului la scriere. Cel care face nominalizarea ar trebui să justifice printr-un paragraf promoțional, motivele pentru care consideră că o persoană ar trebui să aibă acces la scriere. În unele cazuri, vom acorda acces la scriere dezvoltatorilor non C++, de ex. pentru traducători și creatorii de documentație. Chiar și în aceste cazuri, persoanele trebuie să demonstreze o înțelegere a proceselor de modificare a bazei de cod fără producerea de erori, etc.

Note: Din momentul orientării către GIT, suntem mai puțin susceptibili la acordarea accesului de scriere pentru noii dezvoltatori, deoarece este banală partajarea codului din cadrul github, prin bifurcarea QGIS, urmată de emiterea cererilor de actualizare.

Verificați întotdeauna că totul se compilează corect, înaintea efectuării cererilor de actualizare. Încercați să conștientizați posibilele defecțiuni pe care le-ar putea produce actualizările dvs. asupra celor care lucrează pe alte platforme, sau cu versiuni mai vechi/mai noi ale bibliotecilor.

Atunci când se face o actualizare, editorul dvs. va fi indicat (așa cum este definit în variabila de mediu $EDITOR), iar dvs. va trebui să efectuați un comentariu în partea de sus a fișierului (deasupra zonei care spune 'nu efectuați modificări'). Introduceți un comentariu descriptiv și, mai degrabă, efectuați mai multe mici, în cazul în care modificările dintr-un număr mare de fișiere sunt independene. Pe de altă parte, preferăm să actualizați împreună modificările efectuate în fișiere înrudite.

CHAPTER 3

Testarea Unităților

Din noiembrie 2007 am cerut ca toate noile caracteristici care intră în versiunea master să fie însoțite de teste de unitate. Inițial ne-am limitat la qgis_core, apoi vom extinde această cerință pentru alte părți ale bazei de cod, o dată ce dezvoltatorii se vor familiariza cu procedurile pentru testele de unitate, detaliate în secțiunile următoare.

3.1 Cadrul de testare QGIS - o privire de ansamblu

Testele de unitate se efectuează folosind o combinație de QTestLib (biblioteca de testare Qt) și CTest (un cadru pentru compilarea și rularea testelor, ca parte a procesului de compilare CMake). Haideți să aruncăm o privire asupra procesului, înainte de a intra în detalii:

- Să presupunem că există un cod pe care ar trebui să-l testați, cum ar fi o clasă sau o funcție. Adepții programării extreme susțin că imediat ce începeți scrierea unui cod ar trebui să începeți și construirea testelor, astfel că, pe măsură ce implementați codul, să puteți valida imediat, cu ajutorul testului, fiecare nouă parte funcțională adăugată. În practică va trebui, probabil, să scrieți teste pentru codul pre-existent în QGIS, deoarece efectuăm testarea mult prea târziu, după ce logica aplicației a fost deja implementată.
- Creați un test de unitate. Acest lucru se întâmplă în directorul `<QGIS Source Dir>/tests/src/core`, în cazul bibliotecilor de bază. Testul este de fapt un client care creează o instanță a unei clase, apelând unele metode ale acestei clase. Acesta va verifica valorile returnate din fiecare metodă, pentru a se asigura că se potrivesc valorilor așteptate. În cazul în care unul dintre apeluri eșuează, testul de unitate va eșua.
- Includeți macro-urile QtTestLib în clasa dvs. de test. Macrocomenzile sunt procesate de către compilatorul de obiecte meta Qt (MOC) și transformă clasa de testare într-o aplicație executabilă.
- Adăugați o secțiune la fișierul CMakeLists.txt din directorul testelor dvs., care va construi testul.
- Asigurați-vă că ați activat `ENABLE_TESTING` în ccmake / cmakesetup. Astfel, vă veți asigura că testele dumneavoastră se compilează atunci când tastați make.
- Opțional, adăugați datele de testare în `<QGIS Source Dir>/tests/testdata` dacă testul dvs. este dirijat cu ajutorul datelor (de exemplu, trebuie încărcat un fișier shape). Aceste date de testare ar trebui să fie cât mai mici posibil, și ori de câte ori se poate, ar trebui să utilizați seturile de date existente deja acolo. Testele dvs. nu ar trebui să modifice aceste date în sine, ci mai degrabă o copie temporară, pe undeva, dacă este necesar.

- Compilați sursele, apoi efectuați instalarea. Faceți acest lucru utilizând procedura normală `make && (sudo) make install`.
- Rulați testele. Acest lucru se face în mod normal, pur și simplu prin accesarea `make test` după pasul `make install`, deși vom prezenta și alte abordări, care oferă un control mai fin asupra testelor de funcționare.

Având imaginea de ansamblu în minte, vom intra un pic în detalii. O mare parte din configurație este realizată deja în CMake, precum și în alte locuri din arborele sursei, astfel încât tot ce mai trebuie să faceți este simplu - să scrieți testele de unitate!

3.1.1 Crearea unui test de unitate

Crearea unui test de unitate este ușoară - de obicei, veți face acest lucru doar prin crearea un singur fișier .cpp (fișierul .h nu este utilizat), și implementând toate metodele de testare sub formă de metode publice care returnează valori vide. Pentru demonstrație, vom folosi o clasă de test simplă pentru QgsRasterLayer, de-a lungul secțiunii care urmează. Prin convenție vom numi testul nostru cu același nume ca și clasa de testare, dar prefixat cu 'Test'. Așa că testul va fi stocat într-un fișier denumit testqgsrasterlayer.cpp iar clasa în sine va fi TestQgsRasterLayer. În primul rând vom adăuga informațiile privind drepturile de autor:

```
/***************************************************************************
 testqgsvectorfilewriter.cpp
 ---------------------------------------
  Date : Friday, Jan 27, 2015
  Copyright: (C) 2015 by Tim Sutton
  Email: tim@kartoza.com
 ***************************************************************************
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 ***************************************************************************/
```

În continuare vom folosi declarațiile de includere, necesare testelor pe care ne propunem să le rulăm. Există una specială, care ar trebui să existe pentru toate testele:

```
#include <QtTest/QtTest>
```

Mai departe, continuați implementarea normală a clasei, incluzând antetele de care aveți nevoie:

```
//Qt includes...
#include <QObject>
#include <QString>
#include <QObject>
#include <QApplication>
#include <QFileInfo>
#include <QDir>

//qgis includes...
#include <qgsrasterlayer.h>
#include <qgsrasterbandstats.h>
#include <qgsapplication.h>
```

Din moment ce combinăm atât declarația clasei cât și implementarea într-un singur fișier, urmează declarația clasei. Vom începe cu documentația doxygen. Fiecare caz de testare trebuie să fie documentat în mod corespunzător. Folosim directiva doxygen ingroup, astfel încât toate Unitățile de Testare apar ca module în documentația Doxygen generată. Urmează apoi o scurtă descriere a unității de testare, clasa trebuind să moștenească QObject și să includă macrocomanda Q_OBJECT.

```
/** \ingroup UnitTests
 * This is a unit test for the QgsRasterLayer class.
 */

class TestQgsRasterLayer: public QObject
{
    Q_OBJECT
```

Toate metodele noastre de testare sunt implementate ca sloturi private. Cadrul de lucru QtTest va apela succesiv fiecare metodă slot privată din clasa de test. Există patru metode 'speciale' care, dacă sunt implementate, vor fi apelate la începutul testului de unitate (initTestCase), și la sfârșitul testului unitate (cleanupTestCase). Înainte de apelarea fiecărei metode de testare va fi apelată metoda init(), iar după fiecare metodă de testare este apelată metoda cleanup(). Aceste metode sunt foarte utile întrucât vă permit alocarea și curățarea resurselor înainte de rularea fiecărui test, și a unității de testare în ansamblul său.

```
private slots:
  // will be called before the first testfunction is executed.
  void initTestCase();
  // will be called after the last testfunction was executed.
  void cleanupTestCase(){};
  // will be called before each testfunction is executed.
  void init(){};
  // will be called after every testfunction.
  void cleanup();
```

Apoi urmează metodele de testare, care nu ar trebui să preia parametri, dar ar trebui să se întoarcă valoarea vidă. Metodele vor fi apelate în ordinea declarației. Vom implementa două metode aici, care ilustrează două tipuri de teste. În primul caz, pentru a testa, în general, dacă diferitele părți ale clasei lucrează, vom folosi o abordare de testare funcțională. Încă o dată, programatorii extremi pledează pentru scrierea acestor teste înainte de implementarea clasei. Apoi, pe măsură ce lucrați la implementarea clasei dvs. rulați iterativ testele de unitate. Cât mai multe funcții de testare ar trebui să se finalizeze cu succes în timp ce implementarea clasei progresează, iar după ce întregul test de unitate se încheie corect, noua dvs. clasă este gata, dispunând de un mod repetabil de validare.

În mod normal, testele dumneavoastră de unitare vor acoperi doar API-ul public al clasei, nefiind necesară scrierea testelor pentru accesori şi mutatori. Dacă s-ar întâmpla ca un acccesor sau un mutator să nu funcționeze conform așteptărilor, veți implementa un test de regresie, pentru a verifica acest lucru (așa cum se poate vedea mai departe).

```
//
// Functional Testing
//

/** Check if a raster is valid. */
void isValid();

// more functional tests here ...
```

În continuare, vom implementa testele noastre de regresie. Implementarea acestora ar trebui să fie făcută în așa fel, încât să poată reproduce condițiile particulare ale oricărei erori. De exemplu, am primit recent un raport prin e-mail, în care se preciza că numărul de celule din rastere este întotdeauna mai mic cu 1, lucru care denatura toate statisticile pentru benzile raster. Am înregistrat eroarea (prin tichetul #832) și apoi am creat un test de regresie care replica defecțiunea, cu ajutorul unui set de date de test restrâns (un raster 10x10). Apoi am rulat testul, certificând veridicitatea informației (faptul că numărul de celule a fost de 99 în loc de 100). Am corectat eroarea, am rulat iarăși testul de unitate și trecând cu succes de testul de regresie. Am urcat în baza de cod testul de regresie, alături de codul corect. Ulterior, dacă cineva se va bloca la acest pas, vom putea identifica imediat porțiunea de cod care a regresat. În plus, pe viitor, chiar înainte de a înregistra modificările în baza codului principal, rulând aceste teste, ne vom asigura că schimbările aduse nu au efecte adverse neașteptate - cum ar fi întreruperea funcționalității existente.

Mai există un beneficiu al prezenței testelor de regresie - acestea pot aduce economie de timp. Dacă ați depanat vreodată o eroare care a implicat efectuarea de modificări în codul sursă, urmată de rularea aplicației și efectuarea unei serii de pași extrem de complicați pentru a reproduce problema, va fi imediat evident că simpla implementare

a testului de regresie înainte de corectarea erorii permite automatizarea testării, ceea ce reprezintă o eliminare eficientă a defectelor.

Pentru a implementa testul de regresie, ar trebui să urmați convenția de denumire a regresiei<TicketID> pentru funcțiile de testare. În cazul în care nu există deja un tichet Redmine pentru regresie, ar trebui să creați mai întâi unul. Această abordare facilitează persoanei care rulează un test de regresie nereușit, obținerea de informații suplimentare.

```
//
// Regression Testing
//

/** This is our second test case...to check if a raster
 *  reports its dimensions properly. It is a regression test
 *  for ticket #832 which was fixed with change r7650.
 */
void regression832();

// more regression tests go here ...
```

În cele din urmă, în declarația clasei test puteți declara, în mod privat, datele membru și metodele ajutătoare de care ați putea avea nevoie în testele de unitate. În cazul nostru, vom declara un QgsRasterLayer * care poate fi folosit de către oricare dintre metodele noastre de testare. Stratul raster va fi creat în funcția initTestCase(), care este rulat înainte de oricare alte teste, iar apoi va fi distrus de către cleanupTestCase(), care se execută după toate testele. Prin declararea metodelor ajutătoare (care pot fi apelate prin diverse funcții de testare) în mod privat, vă veți asigura că nu vor fi apelate în mod automat de către executabilul QTest, care este creat în urma compilării testului nostru.

```
  private:
    // Here we have any data structures that may need to
    // be used in many test cases.
    QgsRasterLayer * mpLayer;
};
```

Astfel se termină declarația clasei noastre. Implementarea are loc pur și simplu, în partea de jos a fișierului. Mai întâi are loc inițializarea și curățarea funcțiilor:

```
void TestQgsRasterLayer::initTestCase()
{
  // init QGIS's paths - true means that all path will be inited from prefix
  QString qgisPath = QCoreApplication::applicationDirPath ();
  QgsApplication::setPrefixPath(qgisPath, TRUE);
#ifdef Q_OS_LINUX
  QgsApplication::setPkgDataPath(qgisPath + "/../share/qgis");
#endif
  //create some objects that will be used in all tests...

  std::cout << "PrefixPATH: " << QgsApplication::prefixPath().toLocal8Bit().data() << std::endl;
  std::cout << "PluginPATH: " << QgsApplication::pluginPath().toLocal8Bit().data() << std::endl;
  std::cout << "PkgData PATH: " << QgsApplication::pkgDataPath().toLocal8Bit().data() << std::endl
  std::cout << "User DB PATH: " << QgsApplication::qgisUserDbFilePath().toLocal8Bit().data() << st

  //create a raster layer that will be used in all tests...
  QString myFileName (TEST_DATA_DIR); //defined in CmakeLists.txt
  myFileName = myFileName + QDir::separator() + "tenbytenraster.asc";
  QFileInfo myRasterFileInfo ( myFileName );
  mpLayer = new QgsRasterLayer ( myRasterFileInfo.filePath(),
  myRasterFileInfo.completeBaseName() );
}

void TestQgsRasterLayer::cleanupTestCase()
{
  delete mpLayer;
```

```
}
```

Funcția de inițializare de mai sus ilustrează câteva lucruri interesante.

1. Trebuie să setăm manual calea către datele aplicației QGIS, astfel încât resursele, cum ar fi srs.db, să poată fi găsite în mod corect.
2. În al doilea rând, acesta este un test dirijat cu ajutorul datelor, astfel încât trebuie să oferim o modalitate de a localiza, în mod generic, fișierul `tenbytenraster.asc`. Acest lucru a fost realizat prin utilizarea definiția de compilator `TEST_DATA_PATH`. Definiția este creată în fișierul de configurare `CMakeLists.txt` din `<QGIS Source Root>/tests/CMakeLists.txt`, fiind disponibilă pentru toate testele de unitate QGIS. Dacă aveți nevoie de date pentru testul dumneavoastră, urcați-le în `<QGIS Source Root>/tests/testdata`. Ar trebui să urcați aici doar seturi de date foarte mici. În cazul în care testul trebuie să modifice datele, ar trebui să faceți mai întâi o copie a acestora.

Qt oferă, de asemenea, alte câteva mecanisme interesante pentru testarea dirijată cu ajutorul datelor, așa că, dacă vă interesează să aflați mai multe despre acest subiect, consultați documentația Qt.

În continuare, haideți să ne uităm la testul nostru funcțional. Testul isValid() verifică dacă stratul raster a fost încărcat corect în initTestCase. QVERIFY este o macrocomandă Qt pe care o puteți utiliza pentru a evalua o condiție de testare. Qt prevede alte câteva macro-uri care se pot utiliza în testele dumneavoastră, printre care:

- *QCOMPARE (actual, expected)*
- *QEXPECT_FAIL (dataIndex, comment, mode)*
- *QFAIL (message)*
- *QFETCH (type, name)*
- *QSKIP (description, mode)*
- *QTEST (actual, testElement)*
- *QTEST_APPLESS_MAIN (TestClass)*
- *QTEST_MAIN (TestClass)*
- *QTEST_NOOP_MAIN ()*
- *QVERIFY2 (condition, message)*
- *QVERIFY (condition)*
- *QWARN (message)*

Unele dintre aceste macro-uri sunt utile numai atunci când se utilizează cadrul de lucru Qt, pentru testarea cu ajutorul datelor (a se vedea documentația Qt, pentru mai multe detalii).

```
void TestQgsRasterLayer::isValid()
{
  QVERIFY ( mpLayer->isValid() );
}
```

În mod normal, testele funcționale ar putea acoperi toată gama de funcționalități a claselor dvs. API publice, acolo unde este fezabil. Testele funcționale fiind gata, ne putem uita la exemplul nostru de test de regresie.

Având în vedere că problema #832 raportează un număr eronat de celule, scrierea testului nostru constă pur și simplu în folosirea macrocomenzii QVERIFY, pentru a verifica dacă numărul de celule corespunde valorii așteptate:

```
void TestQgsRasterLayer::regression832()
{
  QVERIFY ( mpLayer->getRasterXDim() == 10 );
  QVERIFY ( mpLayer->getRasterYDim() == 10 );
  // regression check for ticket #832
  // note getRasterBandStats call is base 1
  QVERIFY ( mpLayer->getRasterBandStats(1).elementCountInt == 100 );
}
```

Având toate funcțiile de testare a unităților implementate, rămâne să adăugăm un ultim lucru clasei noastre de testare:

```
QTEST_MAIN(TestQgsRasterLayer)
#include "testqgsrasterlayer.moc"
```

Scopul acestor două linii este de a semnala compilatorului moc Qt că are de-a face cu un QtTest (acesta va genera o metodă principală, care, la rândul ei, va apela fiecare test funcțional. Ultima linie reprezintă definiția de includere pentru sursele generate de MOC. Ar trebui să înlocuiți 'testqgsrasterlayer' cu numele, scris cu litere mici, al clasei dvs.

3.2 Adăugarea testului de unitate în CMakeLists.txt

Adăugarea testului de unitate în sistemul de compilare reprezintă doar o chestiune de editare a fișierului CMakeLists.txt din directorul de testare, prin clonarea unuia dintre blocurile de testare existente, și apoi prin înlocuirea numele clasei de test din interiorul său. De exemplu:

```
# QgsRasterLayer test
ADD_QGIS_TEST(rasterlayertest testqgsrasterlayer.cpp)
```

3.3 Adăugarea macocomenzii ADD_QGIS_TEST

Vom parcurge pe scurt aceste linii, pentru a explica ceea ce fac, dar dacă nu vă interesează, trebuie să faceți doar pasul explicat în secțiunea de mai sus.

```
MACRO (ADD_QGIS_TEST testname testsrc)
SET(qgis_${testname}_SRCS ${testsrc} ${util_SRCS})
SET(qgis_${testname}_MOC_CPPS ${testsrc})
QT4_WRAP_CPP(qgis_${testname}_MOC_SRCS ${qgis_${testname}_MOC_CPPS})
ADD_CUSTOM_TARGET(qgis_${testname}moc ALL DEPENDS ${qgis_${testname}_MOC_SRCS})
ADD_EXECUTABLE(qgis_${testname} ${qgis_${testname}_SRCS})
ADD_DEPENDENCIES(qgis_${testname} qgis_${testname}moc)
TARGET_LINK_LIBRARIES(qgis_${testname} ${QT_LIBRARIES} qgis_core)
SET_TARGET_PROPERTIES(qgis_${testname}
PROPERTIES
# skip the full RPATH for the build tree
SKIP_BUILD_RPATHTRUE
# when building, use the install RPATH already
# (so it doesn't need to relink when installing)
BUILD_WITH_INSTALL_RPATH TRUE
# the RPATH to be used when installing
INSTALL_RPATH ${QGIS_LIB_DIR}
# add the automatically determined parts of the RPATH
# which point to directories outside the build tree to the install RPATH
INSTALL_RPATH_USE_LINK_PATH true)
IF (APPLE)
# For Mac OS X, the executable must be at the root of the bundle's executable folder
INSTALL(TARGETS qgis_${testname} RUNTIME DESTINATION ${CMAKE_INSTALL_PREFIX})
ADD_TEST(qgis_${testname} ${CMAKE_INSTALL_PREFIX}/qgis_${testname})
ELSE (APPLE)
INSTALL(TARGETS qgis_${testname} RUNTIME DESTINATION ${CMAKE_INSTALL_PREFIX}/bin)
ADD_TEST(qgis_${testname} ${CMAKE_INSTALL_PREFIX}/bin/qgis_${testname})
ENDIF (APPLE)
ENDMACRO (ADD_QGIS_TEST)
```

Să ne uităm un pic, mai în detaliu, la liniile individuale. Mai întâi definim lista surselor pentru testul nostru. Din moment ce avem doar un singur fișier sursă (urmând metodologia I descrisă mai sus, în care declarația și definiția de clasă rezidă în același fișier) vom efectua o declarație simplă:

```
SET(qgis_${testname}_SRCS ${testsrc} ${util_SRCS})
```

Din moment ce clasa noastră de test trebuie să fie executată prin intermediul compilatorului meta-obiect (MOC) din Qt, trebuie să indicăm acest lucru printr-o pereche de linii:

```
SET(qgis_${testname}_MOC_CPPS ${testsrc})
QT4_WRAP_CPP(qgis_${testname}_MOC_SRCS ${qgis_${testname}_MOC_CPPS})
ADD_CUSTOM_TARGET(qgis_${testname}moc ALL DEPENDS ${qgis_${testname}_MOC_SRCS})
```

În continuare vom indica lui cmake faptul că trebuie creeze un executabil din clasa de test. Am efectuat acest lucru în secțiunea anterioară, pe ultima linie a implementării clasei, unde am inclus ieșirile MOC direct în clasa noastră de testare, astfel că îi vom oferi (printre altele), o metodă principală, astfel încât clasa să poate fi compilată ca un executabil:

```
ADD_EXECUTABLE(qgis_${testname} ${qgis_${testname}_SRCS})
ADD_DEPENDENCIES(qgis_${testname} qgis_${testname}moc)
```

În continuare, trebuie să specificăm orice dependențe față de anumite biblioteci. Momentan, clasele au fost implementate cu o dependență catch-all QT_LIBRARIES, dar vom înlocui această clauză doar cu acele biblioteci Qt de care are nevoie fiecare clasă. Desigur, de asemenea, trebuie să conectați și bibliotecile QGIS relevante, în conformitate cu cerințele testului de unitate.

```
TARGET_LINK_LIBRARIES(qgis_${testname} ${QT_LIBRARIES} qgis_core)
```

În continuare vom spune lui cmake să instaleze testele în același loc cu cel al binarelor QGIS. Acest lucru îl vom elimina pe viitor, astfel încât testele să poată rula direct din interiorul arborelui sursă.

```
SET_TARGET_PROPERTIES(qgis_${testname}
PROPERTIES
# skip the full RPATH for the build tree
SKIP_BUILD_RPATHTRUE
# when building, use the install RPATH already
# (so it doesn't need to relink when installing)
BUILD_WITH_INSTALL_RPATH TRUE
# the RPATH to be used when installing
INSTALL_RPATH ${QGIS_LIB_DIR}
# add the automatically determined parts of the RPATH
# which point to directories outside the build tree to the install RPATH
INSTALL_RPATH_USE_LINK_PATH true)
IF (APPLE)
# For Mac OS X, the executable must be at the root of the bundle's executable folder
INSTALL(TARGETS qgis_${testname} RUNTIME DESTINATION ${CMAKE_INSTALL_PREFIX})
ADD_TEST(qgis_${testname} ${CMAKE_INSTALL_PREFIX}/qgis_${testname})
ELSE (APPLE)
INSTALL(TARGETS qgis_${testname} RUNTIME DESTINATION ${CMAKE_INSTALL_PREFIX}/bin)
ADD_TEST(qgis_${testname} ${CMAKE_INSTALL_PREFIX}/bin/qgis_${testname})
ENDIF (APPLE)
```

În cele din urmă s-a folosit `ADD_TEST` pentru a înregistra testul cu cmake / ctest. Aici este locul în care se întâmplă magia - înregistrarea clasei cu ctest. Dacă vă amintiți, în prezentarea generală de la începutul acestei secțiuni, folosim QtTest și CTest împreună. Recapitulând, QtTest adaugă o metodă principală unității de testare și gestionează apelurile metodelor de testare din cadrul clasei. De asemenea, furnizează unele comenzi macro, cum ar fi `QVERIFY`, pe care le putem utiliza pentru rularea testelor folosind condiții. Rezultatul unității de testare QtTest este un executabil pe care îl putem rula din linia de comandă. Cu toate acestea, atunci când avem o suită de teste și dorim să rulăm individual fiecare executabil, și în plus, să integrăm testele de funcționare în procesul de compilare, folosim CTest.

3.4 Compilarea testului de unitate

Pentru a compila testul de unitate rewbuie doar să vă asigurați că `ENABLE_TESTS = true` în configurația cmake. Există două moduri de a face acest lucru:

1. Rulați `ccmake ..` (sau `cmakesetup ..` sub windows) și setați, în mod interactiv, fanionul `ENABLE_TESTS` pe `ON`.
2. Adăugați o linie de comandă pentru fanion în cmake ex.: `cmake -DENABLE_TESTS=true ..`

La urmă, doar compilați normal QGIS, iar testele ar trebui să se compileze, la rândul lor.

3.5 Rularea testelor dumneavoastră

Cel mai simplu mod de a rula testele este ca parte a procesului normal de compilare:

```
make && make install && make test
```

Comanda de testare va invoca CTest, care va rula fiecare test care a fost înregistrat cu directiva ADD_TEST CMake, descrisă mai sus. Rezultatul tipic al testului va arăta astfel:

```
Running tests...
Start processing tests
Test project /Users/tim/dev/cpp/qgis/build
## 13 Testing qgis_applicationtest***Exception: Other
## 23 Testing qgis_filewritertest *** Passed
## 33 Testing qgis_rasterlayertest*** Passed

## 0 tests passed, 3 tests failed out of 3

The following tests FAILED:
## 1- qgis_applicationtest (OTHER_FAULT)
Errors while running CTest
make: *** [test] Error 8
```

Dacă un test eșuează, puteți utiliza comanda ctest pentru a-i examina mai îndeaproape cauza. Utilizați `opţiunea -R` pentru a specifica un regex pentru testele pe care doriți să le rulați, și un ‘‘ -V‘‘ pentru a obține o ieșire detaliată:

```
$ ctest -R appl -V

Start processing tests
Test project /Users/tim/dev/cpp/qgis/build
Constructing a list of tests
Done constructing a list of tests
Changing directory into /Users/tim/dev/cpp/qgis/build/tests/src/core
## 13 Testing qgis_applicationtest
Test command: /Users/tim/dev/cpp/qgis/build/tests/src/core/qgis_applicationtest
********* Start testing of TestQgsApplication *********
Config: Using QTest library 4.3.0, Qt 4.3.0
PASS : TestQgsApplication::initTestCase()
PrefixPATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/../
PluginPATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/..//lib/qgis
PkgData PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/..//share/qgis
User DB PATH: /Users/tim/.qgis/qgis.db
PASS : TestQgsApplication::getPaths()
PrefixPATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/../
PluginPATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/..//lib/qgis
PkgData PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/..//share/qgis
User DB PATH: /Users/tim/.qgis/qgis.db
QDEBUG : TestQgsApplication::checkTheme() Checking if a theme icon exists:
QDEBUG : TestQgsApplication::checkTheme()
/Users/tim/dev/cpp/qgis/build/tests/src/core/..//share/qgis/themes/default//mIconProjectionDisable
FAIL!: TestQgsApplication::checkTheme() '!myPixmap.isNull()' returned FALSE. ()
Loc: [/Users/tim/dev/cpp/qgis/tests/src/core/testqgsapplication.cpp(59)]
PASS : TestQgsApplication::cleanupTestCase()
Totals: 3 passed, 1 failed, 0 skipped
********* Finished testing of TestQgsApplication *********
```

```
-- Process completed
***Failed

## 0 tests passed, 1 tests failed out of 1

The following tests FAILED:
## 1- qgis_applicationtest (Failed)
Errors while running CTest
```

Well that concludes this section on writing unit tests in QGIS. We hope you will get into the habit of writing test to test new functionality and to check for regressions. Some aspects of the test system (in particular the CMakeLists.txt parts) are still being worked on so that the testing framework works in a truly platform way. I will update this document as things progress.

CHAPTER 4

Pregătirea lucrului cu QtCreator și QGIS

QtCreator este un nou IDE produs de dezvoltatorii bibliotecii Qt. Folosind QtCreator puteți construi orice proiect C++, dar este cu adevărat optimizat pentru persoanele care lucrează cu aplicații bazate pw Qt(4) (inclusiv aplicații mobile). În acest articol am presupus că rulați Ubuntu 11.04 'Natty'.

4.1 Instalarea aplicației QtCreator

Această parte este ușoară:

```
sudo apt-get install qtcreator qtcreator-doc
```

După instalare, ar trebui să-l găsiți în meniul gnome.

4.2 Configurarea proiectului dvs.

Presupunem că aveți deja o clonă de QGIS locală, care conține codul sursă, și că ați instalat toate dependențele necesare compilării etc. Există instrucțiuni detaliate pentru *accesul la git* și instalarea dependențelor.

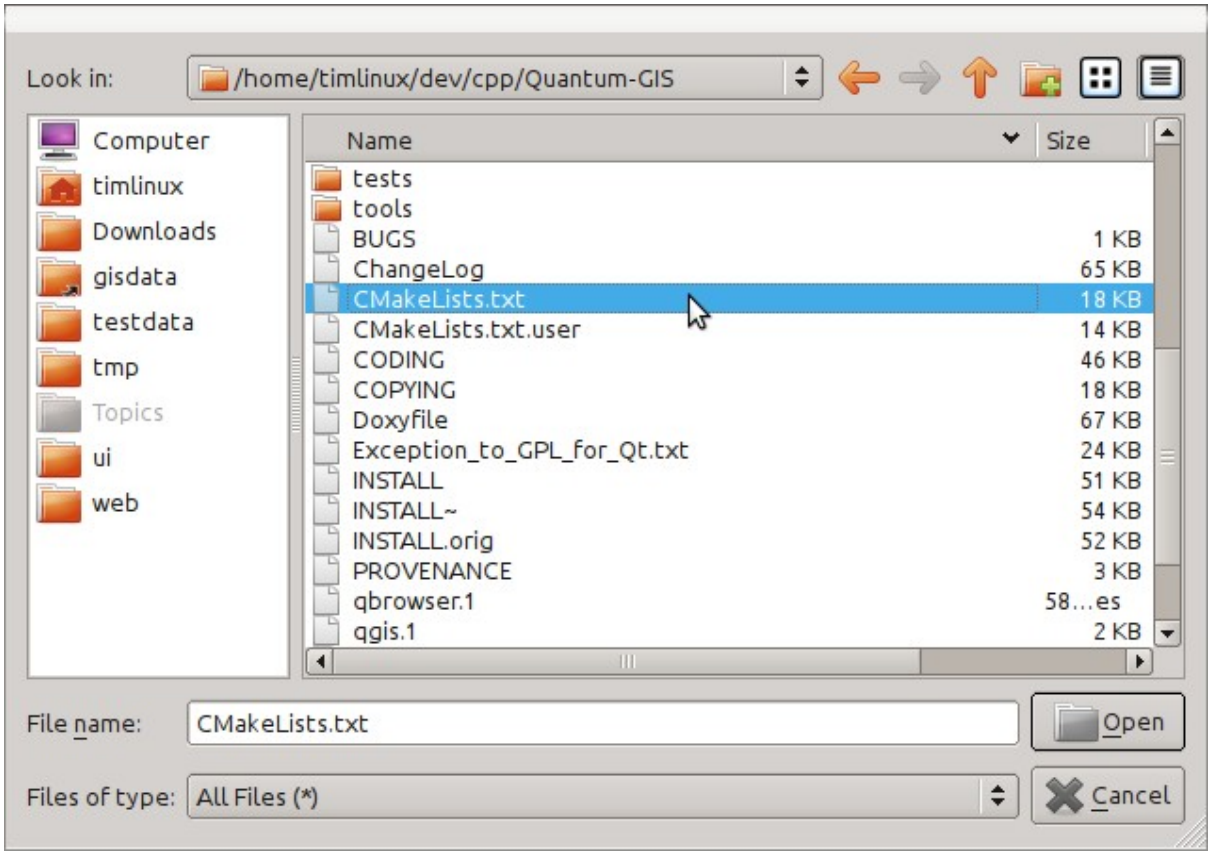
Pe sistemul de test, codul se află în `$HOME/dev/cpp/QGIS` iar restul articolului este scris presupunând că veți actualiza aceste căi, după caz, pentru sistemul dumneavoastră.

Pentru lansarea QtCreator:

Fișier -> Deschidere Fișier sau Proiect

Apoi, selectați în dialogul care se va deschide, următorul fișier:

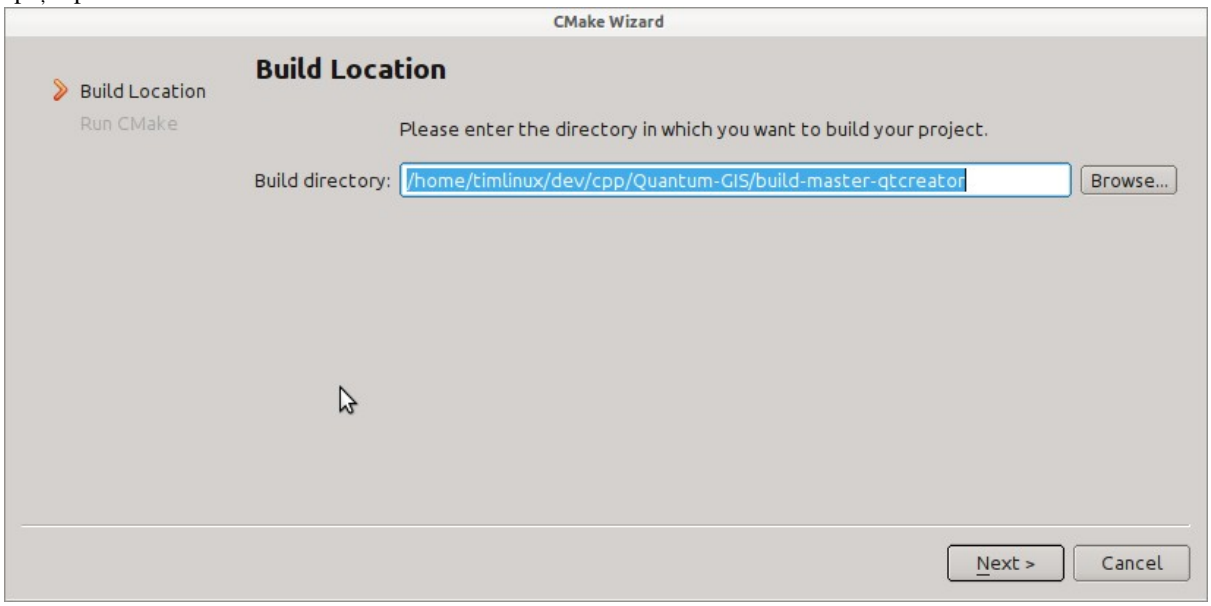
```
$HOME/dev/cpp/QGIS/CMakeLists.txt
```



În continuare vi se va cere o locație pentru compilare. Vom crea un director specific compilațiilor QtCreator:

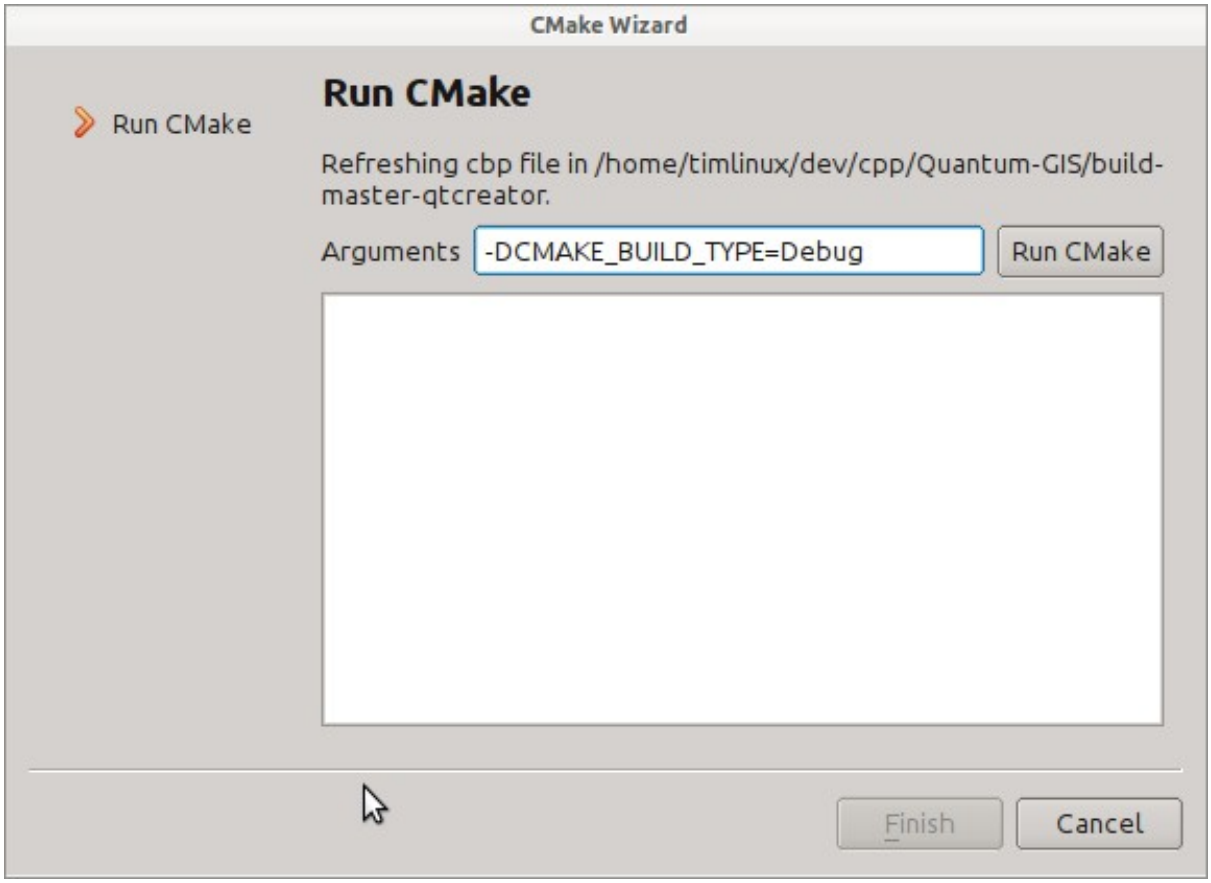
```
$HOME/dev/cpp/QGIS/build-master-qtcreator
```

Probabil că reprezintă o idee bună crearea unor directoare separate pentru diversele ramuri, dacă aveți suficient spațiu pe disc.



În continuare, vi se vor cere opțiunile de compilare pentru CMake. Îi vom indica lui CMake că dorim o compilare de depanare, prin adăugarea acestei opțiuni:

```
-DCMAKE_BUILD_TYPE=Debug
```

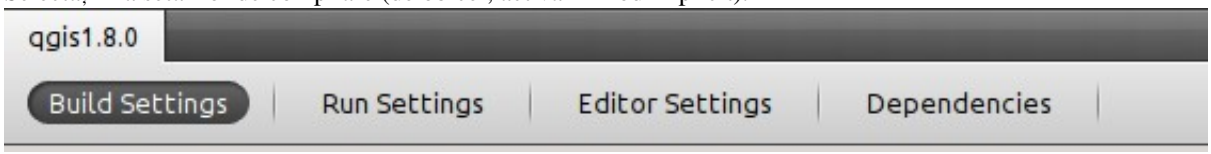


Astea sunt elementele de bază. La definitivare, QtCreator va începe scanarea arborelui sursă pentru autocompletare, și pentru alte operațiuni de întreținere în fundal. Dorim să mai stabilim câteva lucruri înainte de a începe compilarea.

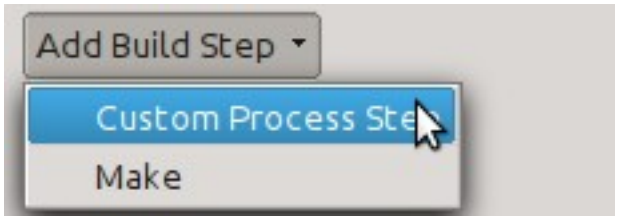
4.3 Instalarea mediului de compilare

Faceți clic pe pictograma 'Proiecte', din partea stângă a ferestrei QtCreator.

Selectați fila setărilor de compilare (de obicei, activă în mod implicit).



Acum, dorim să adăugăm un pas de procesare personalizat. De ce? Pentru că aplicația QGIS poate rula numai dintr-un director de instalare, diferit de directorul de compilare, deci trebuie să ne asigurăm că este instalat după fiecare compilare. În conformitate cu 'Pașii de Compilare', faceți clic pe butonul 'Adăugare BuildStep' și alegeți 'Pas de Procesare Personalizat'.



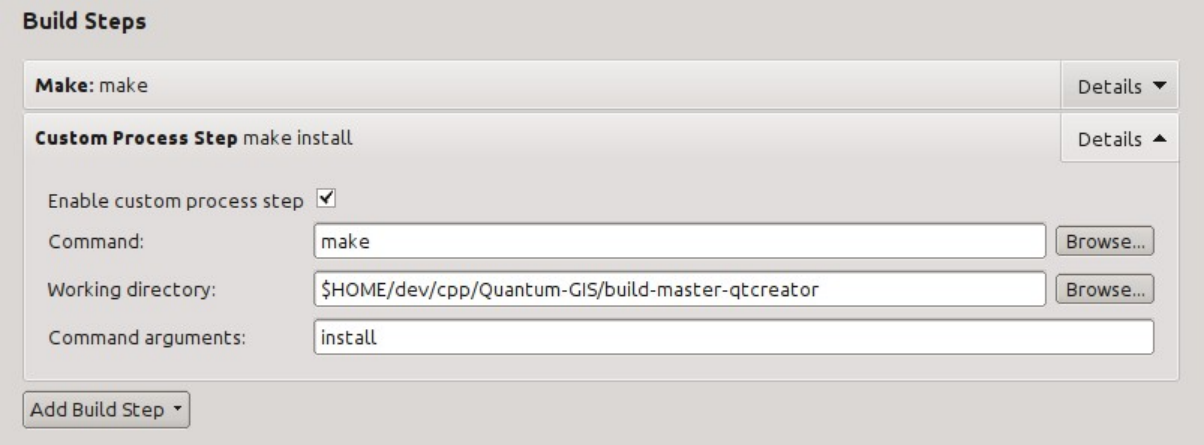
Acum, am stabilit următoarele detalii:

Activare pas de procesare personalizat: [da]

Comanda: make

Directorul de lucru: $HOME/dev/cpp/QGIS/build-master-qtcreator

Argumentele comenzii: install



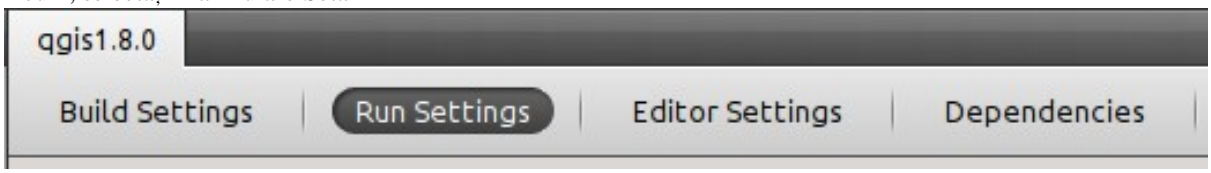
Sunteți aproape gata pentru compilare. Doar o singură notă: QtCreator va avea nevoie de permisiuni de scriere asupra prefixului de instalare. În mod implicit (așa cum am procedat aici) QGIS va fi instalat în `/usr/local/`. În scopuri de dezvoltare, există permisiuni de scriere pentru directorul /usr/local.

Pentru a porni compilarea, faceți clic pe acea pictogramă cu un ciocan mare, în partea din stânga-jos a ferestrei.

4.4 Instalarea mediului de dezvoltare

După cum s-a menționat mai sus, nu putem rula QGIS în mod direct, din directorul de compilare, așa că trebuie să oferim aplicației QtCreator posibilitatea de a rula QGIS din directorul de instalare (în cazul nostru `/usr/local/`). Pentru a face acest lucru, reveniți la ecranul de configurare a proiectelor.

Acum, selectați fila 'Rulare Setări'



Trebuie să actualizăm setările implicite de rulare, personalizând configurația de rulare 'qgis'.

Pentru aceasta, faceți clic pe butonul 'Add v' de lângă caseta de configurare Run, apoi alegeți 'Executabil Personalizat' din partea de sus a listei.

Acum, în zona proprietăților setați următoarele detalii:

Executabil: /usr/local/bin/qgis

Argumente :

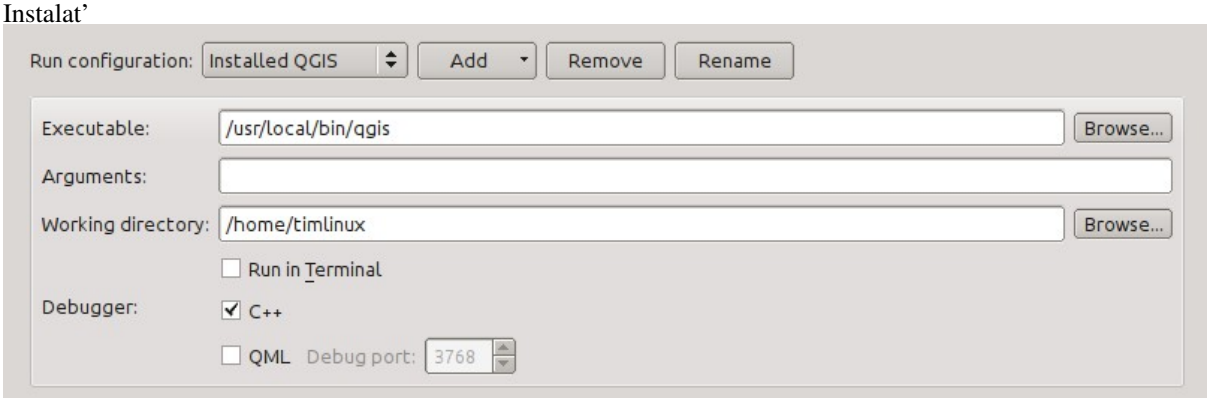
Directorul de lucru: $HOME

Rulare în terminal: [nu]

Depanator: C++ [da]

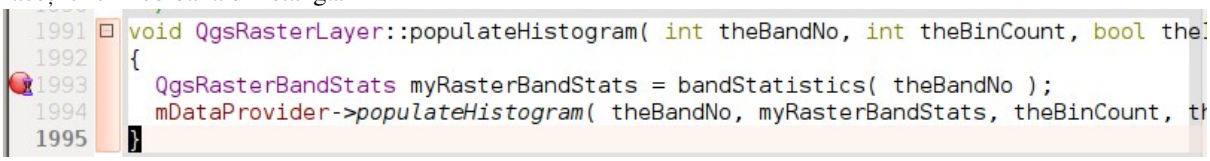
Qml [nu]

Apoi, faceți clic pe butonul 'Redenumire' și dați-i executabilului personalizat un nume sugestiv, cum ar fi 'QGIS Instalat'

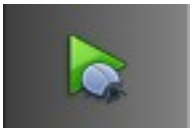


4.5 Rulare și depanare

Acum puteți rula și depana QGIS. Pentru a seta un punct de pauză, pur și simplu deschideți un fișier sursăm apoi faceți clic în coloana din stânga.



Lansați QGIS în modul de depanare, făcând clic pe pictograma corespunzătoare, în partea din stânga-jos a ferestrei.



CHAPTER 5

HIG (Ghidul Interfeței cu Utilizatorul)

In order for all graphical user interface elements to appear consistant and to all the user to instinctively use dialogs, it is important that the following guidelines are followed in layout and design of GUIs.

1. Elementele aflate în relație cu un grup folosesc casetele de grup: Încercați identificarea elementelor care pot fi grupate împreună, apoi utilizați casetele de grup alături de o etichetă, pentru a identifica subiectul acelui grup. Evitați utilizarea casetelor de grup care conțin doar un singur control grafic / element.
2. Capitalise first letter only in labels: Labels (and group box labels) should be written as a phrase with leading capital letter, and all remaining words written with lower case first letters
3. Nu încheiați etichetele componentelor de interfață sau a casetelor de grup cu două puncte: Adăugarea a două puncte produce un efect vizual neplăcut și nu conferă un sens suplimentar, astfel încât nu se vor folosi. O excepție de la această regulă apare atunci când două etichete apar una lângă alta, cum ar fi: Label1 Plugin (Path:) Label2 [/calea/spre/plugin-uri]
4. Keep harmful actions away from harmless ones: If you have actions for 'delete', 'remove' etc, try to impose adequate space between the harmful action and innocuous actions so that the users is less likely to inadvertantly click on the harmful action.
5. Utilizați întotdeauna un QButtonBox pentru butoanele 'OK', 'Cancel' etc: Folosirea controlului grafic respectiv va garanta că ordinea butoanelor 'OK' și 'Cancel', etc, este în concordanță cu sistemul de operare / setările regionale / desktop-ul pe care îl folosește utilizatorul.
6. Evitați imbricarea filelor. În cazul în care utilizați filele, folosiți genul de file utilizate pentru QgsVectorLayerProperties / QgsProjectProperties etc, adică file poziționate în partea superioară, având pictograme de 22x22.
7. Stivuirea controalelor grafice ar trebui să fie evitată, dacă este posibil. Acest lucru produce confuzie și conduce la redimensionarea inexplicabilă (pentru utilizator) a dialogurilor, pentru a se potrivi controalelor grafice care nu sunt vizibile.
8. Try to avoid technical terms and rather use a laymans equivalent e.g. use the word 'Transparency' rather than 'Alpha Channel' (contrived example), 'Text' instead of 'String' and so on.
9. Utilizați o iconografie consistentă. În cazul în care aveți nevoie de o pictogramă sau de elemente de pictogramă, vă rugăm să-l contactați pe Robert Szczepanek, pe lista de discuții pentru asistență.
10. Poziționați listele lungi de controale grafice în casete cu derulare. Nici un dialog nu ar trebui să depășească 580 de pixeli în înălțime și 1000 de pixeli în lățime.
11. Separați opțiunile avansate față de cele de bază. Utilizatorii începători ar trebui să poată accesa rapid elementele necesare pentru activitățile de bază, fără a se preocupa de complexitatea caracteristicilor avansate. Funcțiunile avansate ar trebui să fie amplasate fie sub o linie de demarcație, fie într-o filă separată.
12. Nu adăugați opțiuni de dragul de a avea o mulțime de opțiuni. Încercați să mențineți o interfață cu utilizatorul minimalistă, și utilizați judicios setările implicite.
13. If clicking a button will spawn a new dialog, an ellipsis (...) should be suffixed to the button text.

5.1 Autori

- Tim Sutton (autor și editor)
- Gary Sherman
- Marco Hugentobler
- Matthias Kuhn

CHAPTER 6

Testarea Conformității cu OGC

The Open Geospatial Consortium (OGC) provides tests which can be run free of charge to make sure a server is compliant with a certain specification. This chapter provides a quick tutorial to setup the WMS tests on an Ubuntu system. A detailed documentation can be found at the OGC website.

6.1 Configurarea testelor de conformitate cu WMS 1.3 and WMS 1.1.1

```
sudo apt-get install openjdk-8-jdk maven
cd ~/src
git clone https://github.com/opengeospatial/teamengine.git
cd teamengine
mvn install
mkdir ~/TE_BASE
export TE_BASE=~/TE_BASE
unzip -o ./teamengine-console/target/teamengine-console-4.11-SNAPSHOT-base.zip -d $TE_BASE
mkdir ~/te-install
unzip -o ./teamengine-console/target/teamengine-console-4.11-SNAPSHOT-bin.zip -d ~/te-install
```

Descărcați și instalați testul WMS 1.3.0

```
cd ~/src
git clone https://github.com/opengeospatial/ets-wms13.git
cd ets-wms13
mvn install
```

Descărcați și instalați testul WMS 1.1.1

```
cd ~/src
git clone https://github.com/opengeospatial/ets-wms11.git
cd ets-wms11
mvn install
```

6.2 Proiectul de testare

Pentru testele WMS, datele pot fi descărcate și încărcate într-un proiect QGIS:

```
wget http://cite.opengeospatial.org/teamengine/about/wms/1.3.0/site/data-wms-1.3.0.zip
unzip data-wms-1.3.0.zip
```

Then create a QGIS project according to the description in http://cite.opengeospatial.org/teamengine/about/wms/1.3.0/site/. To run the tests, we need to provide the GetCapabilities URL of the service later.

6.3 Rularea testului WMS 1.3.0

```
export PATH=/usr/lib/jvm/java-8-openjdk-amd64/bin:$PATH
export TE_BASE=$HOME/TE_BASE
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
cd ~/te-install
./bin/unix/test.sh -source=$HOME/src/ets-wms13/src/main/scripts/ctl/main.xml
```

6.4 Rularea testului WMS 1.1.1

```
export PATH=/usr/lib/jvm/java-8-openjdk-amd64/bin:$PATH
export TE_BASE=$HOME/TE_BASE
export ETS_SRC=$HOME/ets-resources
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
cd ~/te-install
./bin/unix/test.sh -source=$HOME/src/ets-wms11/src/main/scripts/ctl/wms.xml
```