QGIS Developers Guide

Release 3.4

QGIS Project

mrt. 15, 2020

Contents

Welkom op de pagina's van QGIS Development. Hier vindt u de regels, gereedschappen en stappen om gemakkelijk en efficiënt code voor QGIS bij te dragen.

CHAPTER 1

QGIS standaarden voor coderen

Deze standaarden zouden door alle ontwikkelaars van QGIS moeten worden gevolgd.

1.1 Klassen

1.1.1 Naamgeving

Een klasse in QGIS begint met Qgs en wordt gevormd met behulp van camel case.

Voorbeelden:

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

1.1.2 Leden

Namen van leden van klassen beginnen met een kleine letter m en worden gevormd met behulp van mixed case.

- `mMapCanvas`
- `mCurrentExtent`

Alle leden van klassen zouden private moeten zijn. Leden van Public-klassen worden STERK ontraden. beveiligde leden zouden moeten worden vermeden als toegang tot het lid moet worden verkregen via subklassen van Python, omdat beveiligde leden niet kunnen worden gebruikt vanuit de Python bindings.

Muteerbare statische leden van klassen zouden moeten beginnen met een kleine letter `s`, maar namen van constante statische leden van klassen zouden allemaal hoofdletters moeten zijn:

- `sRefCounter`
- `DEFAULT_QUEUE_SIZE`

1.1.3 Functies Accessor

Waarden voor leden van klassen zouden moeten worden verkregen door middel van functies accesssor. De functie zou moeten worden benoemd zonder een voorvoegsel get. Functies Accessor voor de twee bovenstaande private leden zouden zijn:

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

Zorg er voor dat accessors juist zijn gemarkeerd met `const`. Waar van toepassing zou dit er toe kunnen leiden dat gecachte waardentypen van variabele leden zijn gemarkeerd met `mutable`.

1.1.4 Functies

Namen van functies beginnen met een kleine letter en worden gevormd met behulp van mixed case. De functienaam zou iets over het doel van de functie moeten zeggen.

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

Voor consistentie met de bestaande API van QGIS en met de API van Qt zouden afkortingen moeten worden vermeden. Bijv. `setDestinationSize` in plaats van `setDestSize`, `setMaximumValue` in plaats van `setMaxVal`.

Acroniemen zouden ook camel case moeten zijn om redenen van consistentie. Bijv. `setXml` in plaats van `setXML`.

1.1.5 Argumenten voor functies

Argumenten voor functies zouden beschrijvende namen moeten hebben. Gebruik geen argumenten van één letter (bijv. `setColor( const QColor& color )` in plaats van `setColor( const QColor& c )`).

Wees uitermate zorgvuldig als argumenten zouden moeten worden doorgegeven door middel van verwijzingen. Tenzij de objecten voor de argumenten klein zijn en eenvoudig zijn te kopiëren (zoals objecten QPoint), zouden zij moeten worden doorgegeven door middel van een verwijzing const. Voor consistentie met de API van Qt dienen zelfs impliciet gedeelde objecten te worden doorgegeven door een verwijzing const (bijv. `setTitle( const QString& title )` in plaats van `setTitle( QString title )`.

1.1.6 Teruggegeven waarden van functies

Geef kleine en eenvoudig eenvoudig te kopiëren objecten als waarden. Grotere objecten zouden moeten worden teruggegeven door een verwijzing const. De enige uitzondering hierop zijn de impliciet gedeelde objecten, die altijd met hun waarde worden teruggegeven. Geef `QObject` of objecten van subklassen terug als pointers.

- `int maximumValue() const`
- `const LayerSet& layers() const`
- `QString title() const` (`QString` is impliciet gedeeld)
- `QList< QgsMapLayer* > layers() const` (`QList` is impliciet gedeeld)
- `QgsVectorLayer *layer() const;` (`QgsVectorLayer` erft `QObject`)
- `QgsAbstractGeometry *geometry() const;` (`QgsAbstractGeometry` is abstract en zal waarschijnlijk moeten worden gecast)

1.2 API-documentatie

Het is vereist om documentatie voor de API te schrijven voor elke klasse, methode, enumeratie en andere code die beschikbaar is in de publieke API.

QGIS gebruikt Doxygen voor documentatie. Schrijf beschrijvende en betekenisvolle opmerkingen die een lezer informatie geven over wat te verwachten, wat gebeurt er in randgevallen en geef hints over andere interfaces die hij zou kunnen gebruiken, goede best practices en voorbeelden van code.

1.2.1 Methoden

Beschrijvingen voor methoden zouden in een beschrijvende vorm moeten worden geschreven, in de 3e persoon. Methoden vereisen een tag `\since` die definieert wanneer zij zijn geïntroduceerd. U zou aanvullende tags `\since` kunnen toevoegen voor belangrijke wijzigingen die later werden geïntroduceerd.

```
/**
 * Cleans the laundry by using water and fast rotation.
 * It will use the provided \a detergent during the washing programme.
 *
 * \returns True if everything was successful. If false is returned, use
 * \link error() \endlink to get more information.
 *
 * \note Make sure to manually call dry() after this method.
 *
 * \since QGIS 3.0
 * \see dry()
 */
```

1.2.2 Variabele leden

Variabele leden zouden normaal gesproken in het gedeelte `private` moeten staan en beschikbaar moeten komen via getters en setters. Één uitzondering hierop is voor data containers zoals voor het rapporteren van fouten. Gebruik in dergelijke gevallen niet het voorvoegsel `m` voor het lid.

```
/**
 * \ingroup core
 * Represents points on the way along the journey to a destination.
 *
 * \since QGIS 2.20
 */
class QgsWaypoint
{
  /**
   * Holds information about results of an operation on a QgsWaypoint.
   *
   * \since QGIS 3.0
   */
  struct OperationResult
  {
    QgsWaypoint::ResultCode resultCode; //!< Indicates if the operation completed␣
↪successfully.
    QString message; //!< A human readable localized error message. Only set if␣
↪the resultCode is not QgsWaypoint::Success.
    QVariant result; //!< The result of the operation. The content depends on the␣
↪method that returned it. \since QGIS 3.2
  };
};
```

1.3 Qt Designer

1.3.1 Gegenereerde klassen

Klassen voor QGIS die worden gegenereerd uit Qt Designer (ui)-bestanden zouden een achtervoegsel Base moeten hebben. Dat identificeert de klasse als een gegenereerde basisklasse.

Voorbeelden:

- `QgsPluginManagerBase`
- `QgsUserOptionsBase`

1.3.2 Dialoogvensters

Alle dialoogvensters zouden helptips moeten implementeren voor alle pictogrammen van de werkbalk en andere relevante widgets. Helptips voegen een grote mate van ontdekkingsdrang toe voor zowel nieuwe als ervaren gebruikers.

Zorg er voor dat de tabvolgorde voor widgets wordt bijgewerkt, iedere keer als de lay-out van het dialoogvenster wijzigt.

1.4 C++-bestanden

1.4.1 Naamgeving

C++ implementatie en headerbestanden zouden respectievelijk de extensie .cpp en .h moeten hebben. De bestandsnaam zou geheel in kleine letters moeten zijn en, in het geval van klassen, moeten overeenkomen met de naam van de klasse.

Voorbeeld: Bronbestanden van de klasse `QgsFeatureAttribute` zijn `qgsfeatureattribute.cpp` en `qgsfeatureattribute.h`

Notitie: In het geval het niet duidelijk is uit het argument hierboven: om een bestandsnaam overeen te laten komen met een naam voor de klasse betekent het impliciet dat elke klasse zou moeten zijn gedeclareerd en geïmplementeerd in zijn eigen bestand. Dit maakt het voor nieuwkomers veel gemakkelijker om te identificeren waar de code gerelateerd is aan een specifieke klasse.

1.4.2 Standaard header en licentie

Elk bronbestand zou een gedeelte header moeten hebben met een patroon als in het volgende voorbeeld:

```
/***************************************************************************
  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.
 *
 ***************************************************************************/
```

Notitie: Er staat een sjabloon voor Qt Creator in GIT. Kopieer het, om het te gebruiken, vanuit `doc/qt_creator_license_template` naar een lokale locatie, pas het mailadres aan en - indien vereist - de naam en configureer QtCreator om het te gebruiken: `Tools` -> `Options` -> `C++` -> `File Naming`.

1.5 Namen van variabelen

Namen van variabelen beginnen met een kleine letter en worden gevormd met behulp van mixed case. Gebruik geen voorvoegsels zoals `my` of `the`.

Voorbeelden:

- `mapCanvas`
- `currentExtent`

1.6 Geënumereerde typen

Geënumereerde typen zouden moeten worden benoemd in CamelCase met een hoofdletter aan het begin, bijv.:

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

Gebruik geen algemene namen voor typen die kunnen conflicteren met andere typen. Gebruik bijvoorbeeld `UnkownUnit` in plaats van alleen `Unknown`

1.7 Globale constanten & macro's

Globale constanten en macro's zouden moeten worden geschreven in hoofdletters, gescheiden door een underscore, bijv.:

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

1.8 Commentaar

Opmerkingen bij klassemethoden zouden een derde persoon indicatieve stijl moeten gebruiken in plaats van de imperatieve stijl:

```
/**
 * Creates a new QgsFeatureFilterModel, optionally specifying a \a parent.
 */
explicit QgsFeatureFilterModel( QObject *parent = nullptr );
~QgsFeatureFilterModel() override;
```

1.9 Qt signalen en slots

Alle verbindingen naar signal/slot zouden moeten worden gemaakt met behulp van de verbindingen "new style" die beschikbaar zijn in Qt5. Meer informatie over dit vereiste is beschikbaar in QEP #77.

Vermijd het gebruiken van Qt auto connect slots (d.i. die welke zijn genaamd `void on_mSpinBox_valueChanged`). Auto connect slots zijn fragiel en gevoelig voor beschadigingen zonder waarschuwing als dialoogvensters opnieuw worden opgebouwd.

1.10 Bewerken

Elke tekstbewerker/IDE kan worden gebruikt om de code van QGIS te bewerken, vooropgesteld dat aan de volgende eisen wordt voldaan.

1.10.1 Tabs

Stel uw bewerker in om tabs te laten zien als spaties. De afstand voor de tab zou moeten zijn ingesteld op 2 spaties.

Notitie: In VIM wordt dit gedaan met `set expandtab ts=2`

1.10.2 Inspringen

Broncode zou moeten worden ingesprongen om de leesbaarheid te verbeteren. Er is een `scripts/prepare-commit.sh` dat de gewijzigde bestanden ophaalt en ze opnieuw laat inspringen met astyle. Dit zou moeten worden uitgevoerd vóór het indienen. U kunt ook `scripts/astyle.sh` gebruiken om individuele bestanden in te laten springen.

Omdat nieuwere versies van astyle anders inspringen dan de gebruikte versie voor een volledige nieuw inspringen van de bron, gebruikt het script een oude versie van astyle, die we op hebben genomen in onze opslagplaats (schakel `WITH_ASTYLE` in cmake in om het op te nemen in de build).

1.10.3 Haakjes

Haakjes zouden op de regel moeten beginnen volgens de volgende expressie:

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

1.11 Compatibiliteit met de API

Er is API documentatie voor C++.

We proberen de API stabiel en achterwaarts compatibel te houden. Opschonen van de API zou op een manier moeten worden gedaan die soortgelijk is aan de broncode voor Qt bijv.

```
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 is 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();
}
```

1.12 SIP-bindingen

Sommige van de SIP-bestanden worden automatisch gemaakt met een toegewezen script.

1.12.1 Voorverwerking header

Alle informatie om op de juiste manier het SIP-bestand te bouwen moet te vinden zijn in het C++ header-bestand. Enkele macro's zijn beschikbaar voor een dergelijke definitie:

- Gebruik `#ifdef SIP_RUN` om alleen code te maken in SIP-bestanden of `#ifndef SIP_RUN` voor alleen code voor C++. Argumenten `#else` worden in beide gevallen afgehandeld.
- Gebruik `SIP_SKIP` om een regel te negeren
- De volgende annotaties worden afgehandeld:
  - `SIP_FACTORY`: `/Factory/`
  - `SIP_OUT`: `/Out/`
  - `SIP_INOUT`: `/In,Out/`
  - `SIP_TRANSFER`: `/Transfer/`
  - `SIP_PYNAME(name)`: `/PyName=name/`
  - `SIP_KEEPREFERENCE`: `/KeepReference/`

  - SIP_TRANSFERTHIS: /TransferThis/
  - SIP_TRANSFERBACK: /TransferBack/

- gedeelten private worden niet weergegeven, behalve wanneer u een argument #ifdef SIP_RUN in dit blok gebruikt.
- SIP_PYDEFAULTVALUE(value) kan worden gebruikt om een alternatieve standaard waarde van de methode van Python te definiëren. Als de standaard waarde een komma , bevat, zou de waarde moeten worden omsloten door enkele aanhalingstekens '
- SIP_PYTYPE(type) kan worden gebruikt om een alternatieve type voor een argument van de methode van Python te definiëren. Als het type een komma , bevat, zou het type moeten worden omsloten door enkele aanhalingstekens '

Een bestand ter demonstratie is te vinden in tests/scripts/sipifyheader.h.

1.12.2 Het SIP-bestand maken

Het SIP-bestand kan worden gemaakt met een toegewezen script. Bijvoorbeeld:

```
scripts/sipify.pl src/core/qgsvectorlayer.h > python/core/qgsvectorlayer.sip
```

Zodra een SIP-bestand is toegevoegd aan een van de bronbestanden (python/core/core.sip, python/gui/gui.sip of python/analysis/analysis.sip), zal het worden beschouwd als automatisch gemaakt. Een test op Travis zal er voor zorgen dat dit bestand up to date is met zijn overeenkomende header.

Oudere bestanden waarvoor het automatisch maken niet is ingeschakeld zijn vermeld in python/auto_sip.blacklist.

1.12.3 Verbeteren van script sipify

Als enkele verbeteringen vereist zijn voor het script sipify, voeg dan de ontbrekende gedeelten toe aan het demo-bestand tests/scripts/sipifyheader.h en maak de verwachte header tests/scripts/sipifyheader.expected.si. Dit zal ook automatisch worden getest op Travis als een eenheidstest van het script zelf.

1.13 Stijl van coderen

Hier worden enkele hints en tips beschreven voor het programmeren die hopelijk het aantal fouten, ontwikkeltijd en onderhoud reduceren.

1.13.1 Waar mogelijk: generaliseer code

Indien u code knipt-en-plakt, of op een andere manier hetzelfde meer dan eens schrijft, overweeg dan om de code te consolideren in één enkele functie.

Dat zal:

- het mogelijk maken wijzigingen op één plaats door te voeren in plaats van op meerdere plaatsen
- helpen bij het tegengaan van overbodige code
- het moeilijker maken voor meerdere kopieën om in de tijd verschillen te ontwikkelen, wat het moeilijker maakt voor anderen om het te begrijpen en te onderhouden

1.13.2 Voorkeur voor het hebben van constanten als eerste in predicaten

Heb een voorkeur voor het als eerste plaatsen van constanten in predicaten.

`0 == value` in plaats van `value == 0`

Dit zal programmeurs helpen om te voorkomen dat zij per ongeluk = gebruiken wanneer zij bedoelen == te gebruiken, wat zeer subtiele logische bugs kan introduceren. De compiler zal een fout genereren als u per ongeluk = gebruikt in plaats van == voor vergelijkingen omdat inherent aan constanten geen waarden kunnen worden toegewezen.

1.13.3 Witruimte kan uw vriend zijn

Toevoegen van spaties tussen operatoren, statements en functies maken het voor mensen gemakkelijker code te parsen.

Wat is gemakkelijker te lezen, dit:

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

of dit:

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

Notitie: `scripts/prepare-commit.sh` zal hier voor zorgen.

1.13.4 Plaats opdrachten op afzonderlijke regels

Het is bij het lezen van code eenvoudig om opdrachten te missen als zij niet aan het begin van de regel staan. Bij het snel doorlezen van code worden vaak regels, die er niet uitzien als zoals u verwacht in de eerste tekens, over te slaan. Het is ook algemeen gebruik om een opdracht te verwachten na een voorwaarde, zoals `if`.

Overweeg:

```
if (foo) bar();

baz(); bar();
```

Het is heel eenvoudig om delen van de beheersstroom te missen. Gebruik in plaats daarvan

```
if (foo)
  bar();

baz();
bar();
```

1.13.5 Laat access modifiers inspringen

Acess modifiers structureren een klasse die is opgedeeld in public API, protected API en private API. Access modifiers zelf groeperen de code binnen deze structuur. Laat de access modifier en declaraties inspringen.

```
class QgsStructure
{
  public:
    /**
     * Constructor
     */
```

```
    explicit QgsStructure();
}
```

1.13.6 Aanbevolen boeken

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

U zou ook echt dit artikel moeten lezen uit Qt Quarterly ove designing Qt style (APIs)

1.14 Vermelding van bijdragen

Zij die nieuwe functies bijdragen worden aangemoedigd om mensen kennis te laten nemen van hun bijdrage door:

- een opmerking toe te voegen in het log van wijzigingen voor de eerste versie waar de code is ingevoegd, in de vorm van:

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

- een blogartikel te schrijven over de nieuwe mogelijkheid en dat toe te voegen aan de QGIS planet https://plugins.qgis.org/planet/
- hun naam toe te voegen aan:
  - https://github.com/qgis/QGIS/blob/release-3_4/doc/CONTRIBUTORS
  - https://github.com/qgis/QGIS/blob/release-3_4/doc/AUTHORS

CHAPTER 2

HIG (Richtlijnen voor de gebruikersinterface)

Het is belangrijk dat de volgende richtlijnen worden gevolgd voor de lay-out en het ontwerpen van GUI's om alle elementen van de grafische gebruikersinterface er consistent uit te laten zien en om de gebruiker instinctief dialoogvensters te laten gebruiken.

1. Groepeer gerelateerde elementen met behulp van groepsvakken: probeer elementen te identificeren die gegroepeerd kunnen worden en gebruik dan groepsvakken met een label om het onderwerp van die groep te identificeren. Vermijd het gebruiken van groepsvakken met slechts één enkel widget / item erin.
2. Geef alleen in labels, Helptips, beschrijvende tekst en andere tekst die geen koptekst of titel is de eerste letter een hoofdletter: Deze zouden als een frase moeten worden geschreven met alleen de eerste letter als hoofdletter, en alle resterende woorden een kleine letter als eerste letter, tenzij zij een zelfstandig naamwoord zijn.
3. Geef alle woorden in titels (groepsvak, tab, lijstweergave kolommen, enzovoort), Functies (menuitems, knoppen), en andere te selecteren items (items voor combinatievak, items voor keuzelijsten, items voor lijsten van de boom, enzovoort hoofdletters): Geef alle woorden hoofdletters, behalve voorzetsels die korter zijn dan vijf letters (bijvoorbeeld, 'with' maar 'Without'), voegwoorden (bijvoorbeeld en, of, maar), en lidwoorden (de, een, het). Het eerste en laatste woord krijgen echter altijd een hoofdletter.
4. Laat labels voor widgets of groepsvakken niet eindigen op ene dubbele punt: Toevoegen van ene dubbele punt veroorzaakt visuele ruis en voegt geen aanvullende betekenis toe, gebruik ze dus niet. Een uitzondering op deze regel is wanneer u twee labels naast elkaar hebt, bijv.: Label1 Plug-in (Pad:) Label2 [/pad/naar/plug-ins]
5. Houd riskante acties weg van acties zonder risico's: Als u acties heeft voor 'delete', 'remove' etc, probeer dan voldoende ruimte te houden tussen de riskante actie en ongevaarlijke acties zodat de gebruikers minder snel per ongeluk op de riskante actie kunnen klikken.
6. Gebruik altijd een QButtonBox voor knoppen 'OK', 'Annuleren' etc: gebruik van een knoppenvak zal er voor zorgen dat de volgorde van de knoppen 'OK' en 'Annuleren' etc, consistent is met het besturingssysteem / locale / desktopomgeving die de gebruiker gebruikt.
7. Tabs zouden niet moeten worden genest. Als u tabs gebruikt, volg dan de stijl van de tabs die wordt gebruikt in QgsVectorLayerProperties / QgsProjectProperties etc. d.i. tabs bovenin met pictogrammen van 22x22.
8. Stapels van widgets zouden zoveel mogelijk moeten worden vermeden. Zij kunnen problemen veroorzaken met lay-outs en onverklaarbare (voor de gebruiker) wijzigingen in grootte van dialoogvensters om widgets te accommoderen die niet zichtbaar zijn.

9. Probeer technische termen te vermijden en gebruik lekentaal als equivalent bijv. gebruik het woord 'Doorzichtbaarheid' in plaats van 'Alfakanaal' (bedrieglijk voorbeeld), 'Tekst' in plaats van 'String' enzovoort.
10. Gebruik pictogrammen consistent. Als u een pictogram of elementen van pictogrammen nodig hebt, neem dan voor assistentie contact op met Robert Szczepanek via de mailinglijst.
11. Plaats lange lijsten met widgets in scrollvakken. Een dialoogvenster zou niet groter moeten zijn dan 580 pixels in hoogte en 1000 pixels in breedte.
12. Geavanceerde opties dienen te worden gescheiden van standaard functionaliteit. Nieuwe gebruikers zouden in staat moeten zijn snel toegang te verkrijgen tot functionaliteit zonder dat zij zich zelf bezig dienen te houden met de complexiteit van geavanceerde mogelijkheden. Geavanceerde mogelijkheden zouden ofwel moeten zijn geplaatst onder een scheidingslijn, of op een afzonderlijke tab.
13. Voeg geen opties toe die niet bijdragen aan de gebruikerservaring. Houd de interface minimalistisch en gebruik logische standaard instellingen
14. Als het klikken op een knop een nieuw dialoogvenster zou openen, zou een ellipsis (...) na de tekst op de knop moeten worden geplaatst. Opmerking: zorg er voor dat u het teken voor Horizontale ellips U+2026 gebruikt in plaats van drie punten.

2.1 Auteurs

- Tim Sutton
- Garry Sherman
- Marco Hugentobler
- Matthias Kuhn

CHAPTER 3

Toegang met Git

Dit deel beschrijft hoe je kunt beginnen met het gebruiken van de QGIS Git repository. Eerst heb je een op je systeem geïnstalleerde Git client nodig.

3.1 Installatie

3.1.1 Installeer Git voor GNU/Linux

Gebruikers van de op Debian gebaseerde distributie kunnen:

```
sudo apt install git
```

3.1.2 Installeren van Git voor Windows

Windows gebruikers kunnen Git verkrijgen via msys git of gebruik git gedistribueerd met cygwin.

3.1.3 Installeer Git voor OSX

Het git project heeft een te downloaden build van GIT. Zorg er voor dat u het pakket krijgt dat overeenkomt met uw processor (meest waarschijnlijk x86_64, alleen de eerste Intel Mac's hadden het pakket i386 nodig).

Na het downloaden op het disk image en start de installer.

PPC/source notitie

De Git site heeft geen PPC installatiepakket. Wanneer je Git wilt installeren op een PPC dan moet je deze zelf compileren waarbij je wel meer controle hebt over de installatie.

Download de bron van https://git-scm.com/. Unzip het, en cd in een Terminal naar de map source, dan:

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

Wanneer je de extra's, Perl, Python of TclTk (GUI), niet nodig hebt kun je deze uit de build met make houden door:

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

3.2 Toegang tot de repository

Om een clone te maken van QGIS master:

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

3.3 Uitchecken van een branch

Om een branch uit te checken, bijvoorbeeld de 2.6.1 branch doe:

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

Om de master branch uit te checken:

```
cd QGIS
git checkout master
```

Notitie: Voor QGIS wordt er voor elke uitgebrachte versie van QGIS een release branch gemaakt. Master bevat de laatste versie van QGIS, de zogenaamde 'unstable' release. Wanneer een nieuwe versie wordt aangemaakt zal er ook een nieuwe branch vanuit master worden aangemaakt, vervolgens wordt deze verder stabiel gemaakt en wordt er selectief nieuwe functionaliteit aan master toegevoegd.

Zie het bestand INSTALL in de root van de broncode voor specifieke instructies voor het bouwen van ontwikkel versies van QGIS.

3.4 QGIS documentatie broncode

Wanneer je de broncode van de QGIS documentatie wilt uitchecken:

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

Je kunt ook de readme lezen welke onderdeel is van de documentatie repository voor meer informatie.

3.5 QGIS website broncode

Wanneer je de broncode van de QGIS website wilt uitchecken:

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

Je kunt ook de readme lezen welke onderdeel is van de website repository voor meer informatie.

3.6 Git documentatie

De volgende websites bevatten informatie om een Git master te worden.

- https://services.github.com/
- https://progit.org
- http://gitready.com

3.7 Ontwikkeling in branches

3.7.1 Doel

De QGIS broncode is toegenomen en ingewikkelder geworden in de afgelopen jaren. Het is dan ook moeilijk om alle negatieve bijwerkingen te overzien die het toevoegen van nieuwe functionaliteit zal hebben. In het verleden had het project QGIS lange release-cyclussen, omdat het veel werk was om het systeem weer stabiel te krijgen na het toevoegen van nieuwe functionaliteit. Om deze problemen te voorkomen is er overgestapt naar een nieuw ontwikkelingsmodel waar de nieuwe functionaliteit eerst werd ontwikkeld in Git Branches en daarna pas in de master branch worden samengevoegd, nadat deze gereed en stabiel zijn. Dit gedeelte beschrijft de procedure branching en samenvoegen van broncode in het project QGIS.

3.7.2 Procedure

- **Initiële aankondiging op de mailinglijst:** Voordat men begint, plaats een aankondiging op de 'qgis-developer' mailing list voor ontwikkelaars om te zien of een andere ontwikkelaar mogelijk al werkt aan dezelfde functionaliteit. Neem ook contact op met de technisch adviseur van het 'Project Steering Committee' (PSC). Als de nieuwe mogelijkheid wijzigingen vereist aan de architectuur van QGIS is een 'request for comment' (RFC) nodig.

Maak een branch: Maak een nieuwe Git branch aan voor de ontwikkeling van de nieuwe functionaliteit.

```
git checkout -b newfeature
```

Nu kunt u beginnen met de ontwikkeling. Als u van plan bent groot uit te pakken in die branch, u werk zou willen delen met andere ontwikkelaars, en schrijftoegang nodig heeft naar de upstream repository, kunt u uw repository pushen naar de officiële QGIS repository met:

```
git push origin newfeature
```

Notitie: Als de branch al bestaat zullen uw wijzigingen daarin worden gepusht.

Voer regelmatig een 'rebase' uit vanuit master: Het wordt aanbevolen om op een regelmatige basis de wijzigingen in de master samen te voegen met de branch met een 'rebase'. Dat maakt het later eenvoudiger om wijzigingen vanuit de branch met een 'merge' toe te voegen in de master branch. Na een 'rebase' gebruik `git push -f` naar uw gevorkte repository.

Notitie: Gebruik nooit `git push -f` naar de origin repository! Gebruik deze opdracht alleen voor het bijwerken van uw werk-branch.

```
git rebase master
```

3.7.3 Testen voor het uitvoeren van een 'merge' om wijzigingen terug naar master te brengen

Wanneer u gereed bent met de ontwikkeling van de nieuwe functionaliteit en tevreden bent met de stabiliteit, maak een aankondiging op de lijst voor ontwikkelaars. Vóór het terug mergen van wijzigingen, zullen de wijzigingen worden getest door ontwikkelaars en gebruikers.

3.8 Het indienen van Patches en Pull Requests

Er zijn een aantal richtlijnen, die u helpen om uw patches en pull requests eenvoudig in QGIS te krijgen, en ons helpen met het verwerken van patches die naar ons worden gestuurd.

3.8.1 Pull Requests

In het algemeen is het voor ontwikkelaars gemakkelijker wanneer u pull requests bij GitHub indient. We beschrijven hier niet het uitvoeren van 'pull requests', maar verwijzen daarvoor naar de documentatie van GitHub over 'pull requests'.

Wanneer u een pull request indient vragen we u om regelmatig wijzigingen vanuit master samen te voegen naar uw PR branch (= pull request branch), zodat wijzigingen vanuit uw PR branch altijd upstream zijn samen te voegen naar de master branch.

Als u een ontwikkelaar bent en de rij van de pull requests wilt bekijken, is er een aardig programma dat u dat laat doen vanaf de opdrachtregel

Bekijk het gedeelte hieronder over 'hoe aandacht voor uw patch te krijgen'. In het algemeen is het zo dat als u een PR indient u ook de verantwoordelijkheid moet nemen om deze te volgen totdat deze is voltooid - beantwoord vragen die door andere ontwikkelaars zijn gepost, zoek een 'kampioen' voor uw mogelijkheid en geef ze af en toe een vriendelijke herinnering als u ziet dat er geen actie wordt ondernomen met betrekking tot uw PR. Onthoud wel dat het project QGIS wordt uitgevoerd door inspanningen van vrijwilligers en mensen zijn mogelijk niet in staat om direct te reageren op uw PR. Als u van mening bent dat uw PR niet de aandacht krijgt die het verdient, zijn uw opties om daar wat vaart in te brengen (in volgorde van prioriteit):

- Stuur een bericht naar de mailinglijst om uw PR 'aan de man te brengen' en hoe mooi het zou zijn om die onderdeel te maken van de basis code.
- Stuur een bericht naar de persoon aan wie jouw PR is toegewezen in de PR lijst.
- Stuur een bericht naar Marco Hugentobler (hij beheert de PR lijst).
- Stuur een bericht naar de project stuur groep (project steering committee) met de vraag of zij hulp kunnen bieden bij het opnemen van jouw PR in de basis code.

Beste manier voor het aanmaken van een pull request.

- Start altijd met het aanmaken van een 'feature branch', branch waarin de nieuwe functionaliteit wordt ontwikkeld, vanuit master.
- Wanneer je ontwikkeld in de 'feature branch', gebruik dan geen merge voor het bijwerken van die branch, maar een rebase zoals beschreven in volgende punt, om zo jouw geschiedenis schoon te houden.
- Voordat je een pull verzoek uitvoert geef eerst de opdrachten `git fetch origin` en `git rebase origin/master` (gegeven origin verwijst naar de 'remote' voor 'upstream' en niet jouw eigen 'remote', controleer jouw `.git/config` of geef de opdracht `git remote -v | grep github.com/qgis`).
- U kunt een git rebase, zoals vermeld in de laatste regel, herhaaldelijk uitvoeren zonder schade (zolang als het enige doel is om de wijzigingen uit uw branch gemerged te krijgen in master).
- Attentie: Na een rebase dient u `git push -f` uit te voeren op uw gevorkte repository. **CORE DEVS: DOE DIT NIET OP DE QGIS PUBLIC REPOSITORY!**

Speciale labels om schrijvers van documentatie te informeren

Naast algemene tags die u kunt toevoegen om uw PR te classificeren, zijn er speciale tags die u kunt gebruiken om automatisch issue rapporten te genereren in de repository van de QGIS-Documentation zodra uw pull request is gemerged:

- `[needs-docs]` is een verzoek aan de schrijvers van documentatie om aanvullende documentatie toe te voegen na een reparatie of uitbreiding op een bestaande functionaliteit.
- `[feature]` in het geval van nieuwe functionaliteit. Het invullen van een goede beschrijving van uw PR is een goed begin.

Ontwikkelaars wordt gevraagd deze labels (niet hoofdlettergevoelig) te gebruiken zodat schrijvers van documentatie issues binnenkrijgen zodat zij een overzicht hebben van werk dat gedaan moeten worden. Ook belangrijk, neem de tijd om een beschrijving toe te voegen, ofwel in de commit of in de documentatie.

Voor het uitvoeren van een merge van een pull request

Optie A:

- klik op de knop Merge (maakt een 'non-fast-forward merge' aan)

Optie B:

- Doe een checkout van de pull request
- Testen (Uiteraard ook vereist voor option A)
- checkout master, git merge pr/1234
- Optioneel: `git pull --rebase`: maakt een "fast-forward, no merge commit" aan. Dit levert een schonere historie op, maar het is moeilijker om de commit ongedaan te maken.
- `git push` (gebruik hier NOOIT de -f optie)

3.9 Naamgeving voor patch bestanden

Als de patch een reparatie is voor een specifieke bug, geef het bestand dan een naam met het nummer van de bug erin bijv. bug777fix.patch, en maak het een bijlage van het originele bug report in GitHub.

Als de bug een verbetering of een nieuwe mogelijkheid is, is het gewoonlijk een goed idee om eerst een ticket in GitHub te maken en uw patch als bijlage daaraan toe te voegen.

3.10 Een patch aanmaken in de top folder van de QGIS broncode

Dit maakt het voor ons gemakkelijker om de patches door te voeren omdat we niet hoeven te navigeren naar een specifieke folder van de broncode om de patch door te voeren. Wanneer ik patches ontvang, evalueer ik ze met behulp van een merge, wanneer de patch in de top-folder staat maakt dit veel gemakkelijker. Hieronder is een voorbeeld gegeven hoe u meerdere gewijzigde bestanden kunt opnemen in uw patch vanuit de top-folder:

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

Dit zal er voor zorgen dat uw master branch in-sync is met de upstream repository, en hoe vervolgens een patch kan worden aangemaakt die de wijzigingen bevat tussen jouw 'feature branch' en de master branch.

3.10.1 Aandacht krijgen voor uw patch

Ontwikkelaars van QGIS zijn drukke mensen. We scannen de inkomende patches voor bugrapporten maar soms missen we dingen. Wees daardoor niet beledigd of bang. Probeer een ontwikkelaar te identificeren die u kan helpen en neem contact met ze op en vraag of zij naar uw patch willen kijken. Als u geen antwoord krijgt, kunt u uw vraag escaleren naar een van de leden van het Project Steering Committee (contactdetails ook beschikbaar in de Technical Resources).

3.10.2 Betracht gepaste zorgvuldigheid

QGIS is gelicenseerd onder de GPL. U dient er u uiterste best voor te doen om te zorgen dat u alleen patches indient waarover geen conflicten kunnen ontstaan met betrekking tot rechten over intellectueel eigendom. Dien ook geen broncode in wanneer u niet tevreden met de gedachte dat deze eveneens beschikbaar komt onder de GPL licentie.

3.11 Het verkrijgen van Git schrijftoegang

Directe schrijftoegang tot de QGIS broncode wordt gegeven door uitnodiging. Wanneer een persoon meerdere (geen vastgesteld aantal) substantiële wijzigingen heeft ingediend die aangeven dat deze C++ en de QGIS coding

conventions beheerst, kan een lid van de PSC of een van de andere ontwikkelaars voorstellen om schrijfrechten te verlenen. Diegene die iemand voordraagt moet schriftelijk een promotiestukje indienen waarom diegene schrijfrechten zou moeten krijgen. In sommige gevallen geven we schrijfrechten aan niet C++ ontwikkelaars bijvoorbeeld aan vertalers en schrijvers van documentatie. In die gevallen moet de persoon in kwestie hebben aangetoond dat deze patches kan indienen en bij voorkeur al enkele substantiele patches ingedient die duidelijk maken dat de persoon wijzigingen kan maken in de basis repository zonder nieuwe problemen te introduceren, enz.

Notitie: Sinds de overgang naar GIT geven we minder snel schrijftoegang aan nieuwe ontwikkelaars omdat het eenvoudig is de code te delen binnen GitHub door een fork (een persoonlijke online kopie van QGIS project repository binnen GitHub) aan te maken en vervolgens wijzigingen via pull requests in te dienen.

Controleer altijd of alles compileert vóór het maken van commit / pull request. Ben er continue bewust van dat jouw commit mogelijk problemen veroorzaakt voor mensen die bouwen op andere platformen met oudere / nieuwere versies van functie bibliotheken.

Tijdens een commit, zal uw teksteditor (zoals gedefinieerd in de omgevingsvariabele $EDITOR) verschijnen en je zou commentaar moeten toevoegen aan het begin van het bestand (Voor de regel met 'don't change this'). Geef beschrijvend commentaar maak liever verscheidene kleinere commits wanneer de wijzigingen over verschillende bestanden niet gerelateerd zijn. Daarnaast prefereren we u om gerelateerde wijzigingen wel in één enkele commit te groeperen indien mogelijk.

CHAPTER 4

Beginnen met QtCreator en QGIS

QtCreator is een nieuwe IDE van de makers van de bibliotheek Qt. Met QtCreator kunt u elk project in C++ bouwen, maar het is echt geoptimaliseerd voor mensen die werken aan op Qt(4) gebaseerde toepassingen (inclusief mobiele apps). Alles wat ik hieronder beschrijf gaat er van uit dat u Ubuntu 11.04 'Natty' gebruikt.

4.1 QtCreator installeren

Dit gedeelte is gemakkelijk:

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

Na de installatie zou u het moeten vinden in uw menu van Gnome.

4.2 Instellen van uw project

Ik ga er van uit dat u al een lokale QGIS kloon heeft die de broncode bevat, en alle benodigde afhankelijkheden etc. voor het bouwen hebt geïnstalleerd. Er zijn gedetailleerde instructies voor *toegang tot git* en installeren van afhankelijkheden.

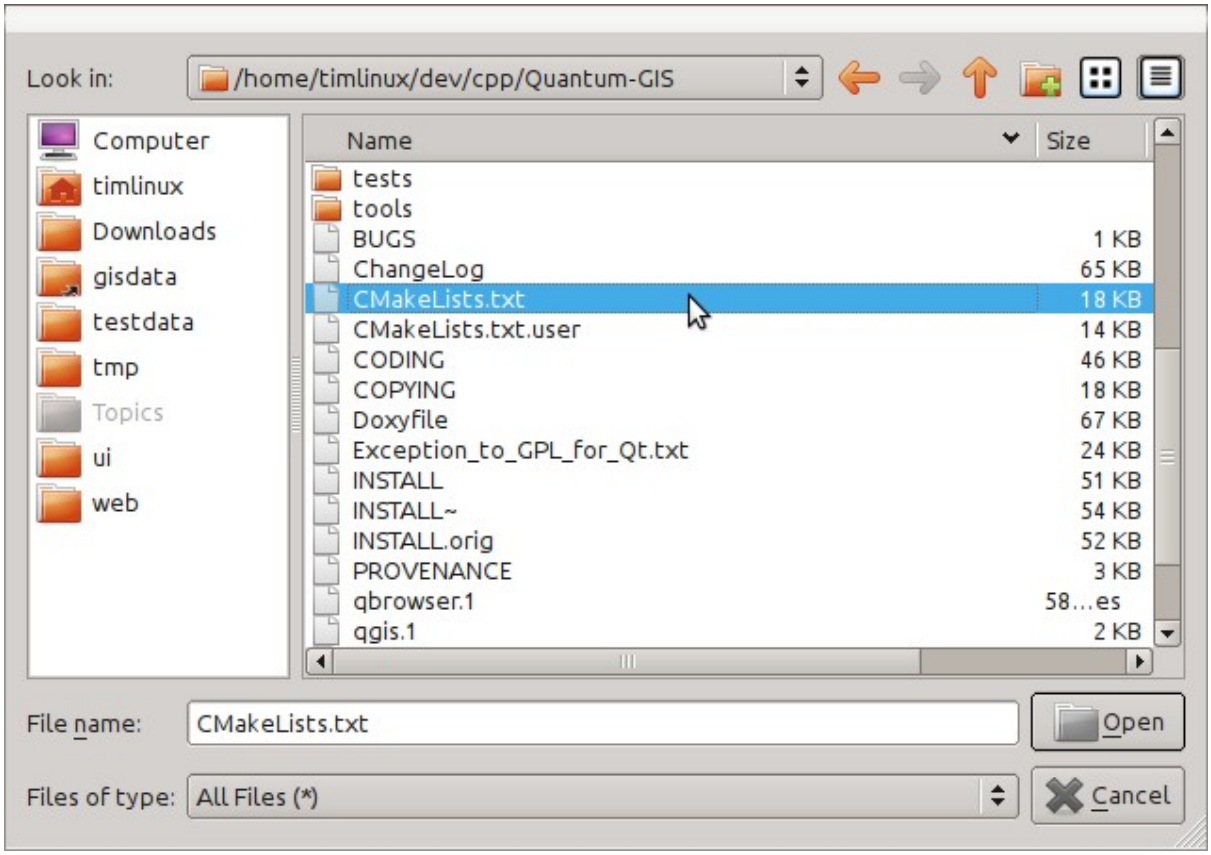
Op mijn systeem heb ik de code uitgecheckt in `$HOME/dev/cpp/QGIS` en de rest van het artikel is geschreven met de aanname dat u deze paden zou moeten bijwerken zoals te doen gebruikelijk voor uw lokale systeem.

Doe bij het opstarten van QtCreator:

File -> Open File or Project

Gebruik dan het resulterende dialoogvenster voor bestandsselectie om naar dit bestand te bladeren en het te openen:

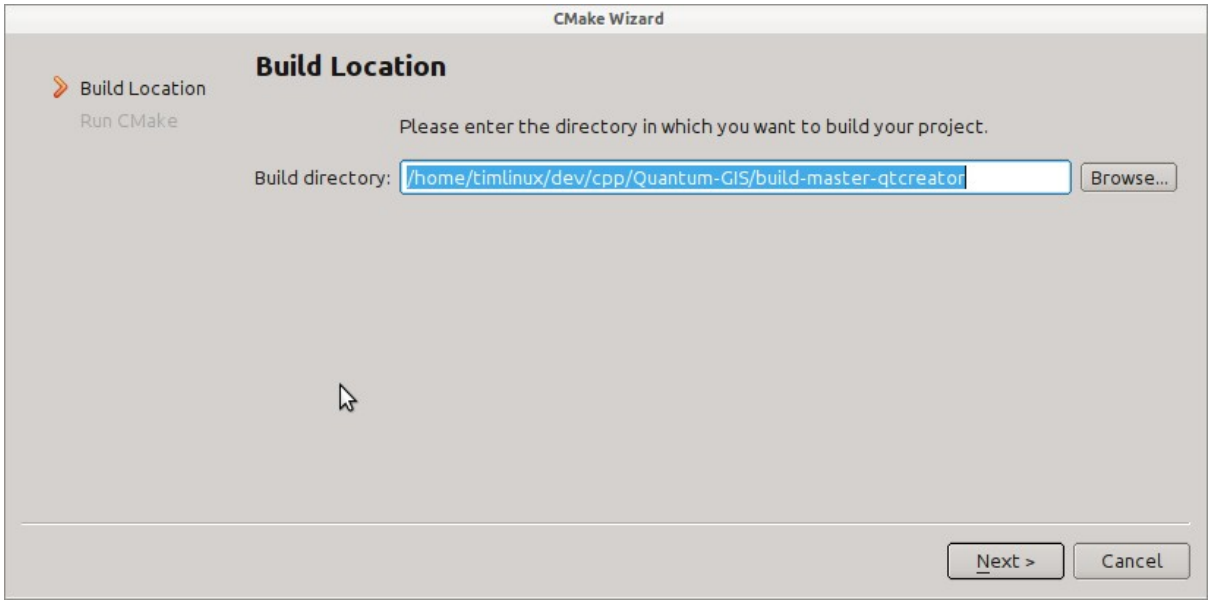
```
$HOME/dev/cpp/QGIS/CMakeLists.txt
```



Vervolgens zult u worden gevraagd naar een locatie voor de build. Ik maakte een specifieke map build voor QtCreator om onder te werken:

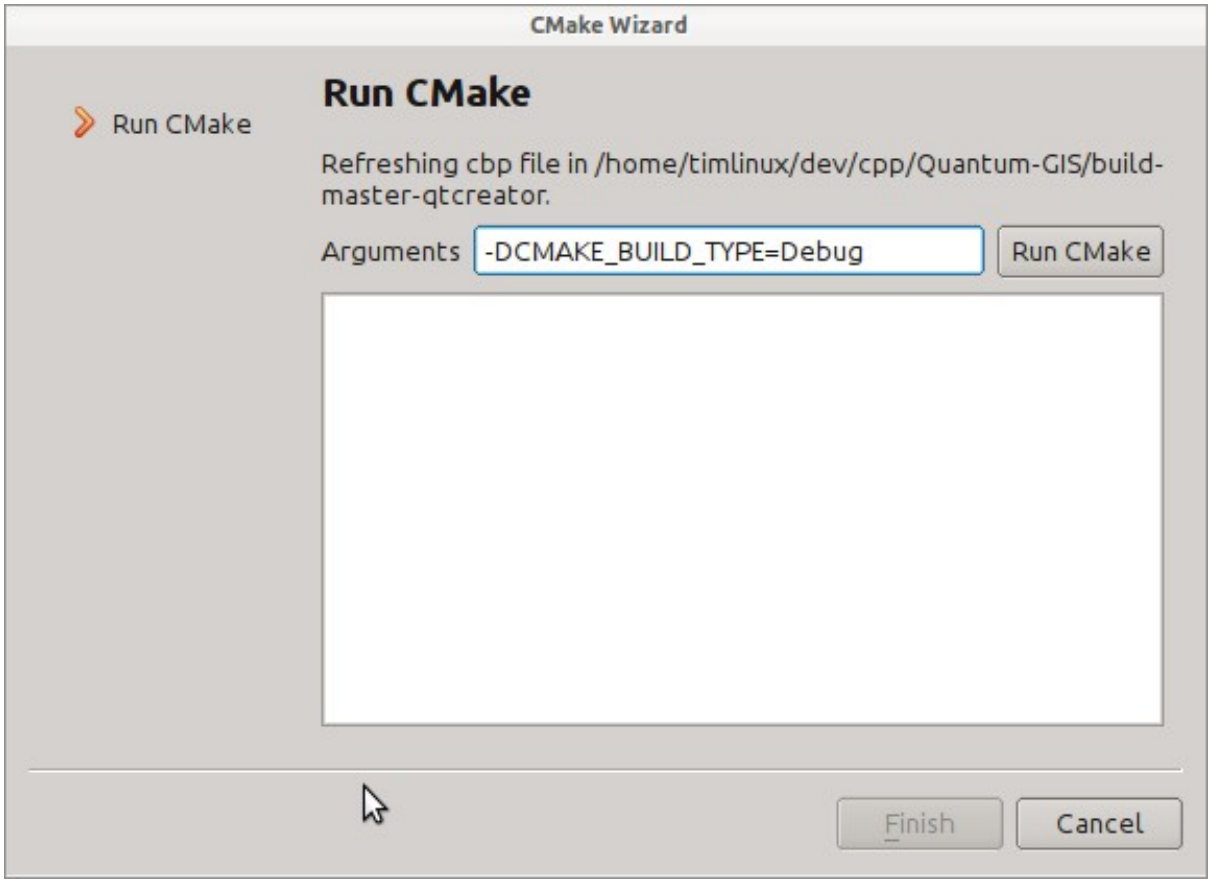
```
$HOME/dev/cpp/QGIS/build-master-qtcreator
```

Het is waarschijnlijk een goed idee om afzonderlijke mappen build te maken voor verschillende branches, als u daarvoor de schijfruimte heeft.



Vervolgens zult u worden gevraagd of u opties voor CMake build heeft die u wilt doorgeven aan CMake. We zullen CMake vertellen dat we een debug build willen door deze optie toe te voegen:

```
-DCMAKE_BUILD_TYPE=Debug
```

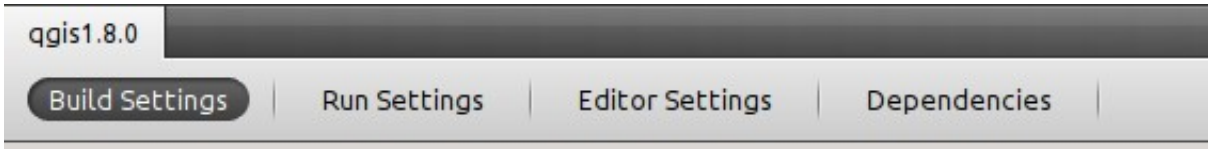


Dat is de basis ervan. Wanneer u de Wizard voltooid, zal QtCreator beginnen met het scannen van de boom van de bron voor ondersteuning van automatisch aanvullen en op de achtergrond nog wat huishoudelijk werk doen. We willen echter nog een aantal dingen aanpassen voordat we beginnen te bouwen.

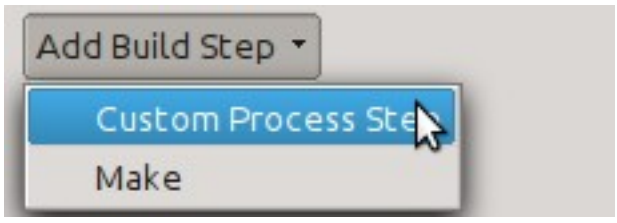
4.3 Instellen van de omgeving voor uw build

Klik op het pictogram ‘Projects’ aan de linkerzijde van het venster van QtCreator.

Selecteer de tab build settings (normaal gesproken standaard actief).



We willen nu een aangepaste stap voor het proces toevoegen. Waarom? Omdat QGIS momenteel alleen kan worden uitgevoerd vanuit een map install, niet vanuit zijn map build, dus moeten we er voor zorgen dat het is geïnstalleerd wanneer we het bouwen. Klik, onder 'Build Steps', op de combinatieknop 'Add BuildStep' en kies 'Custom Process Step'.



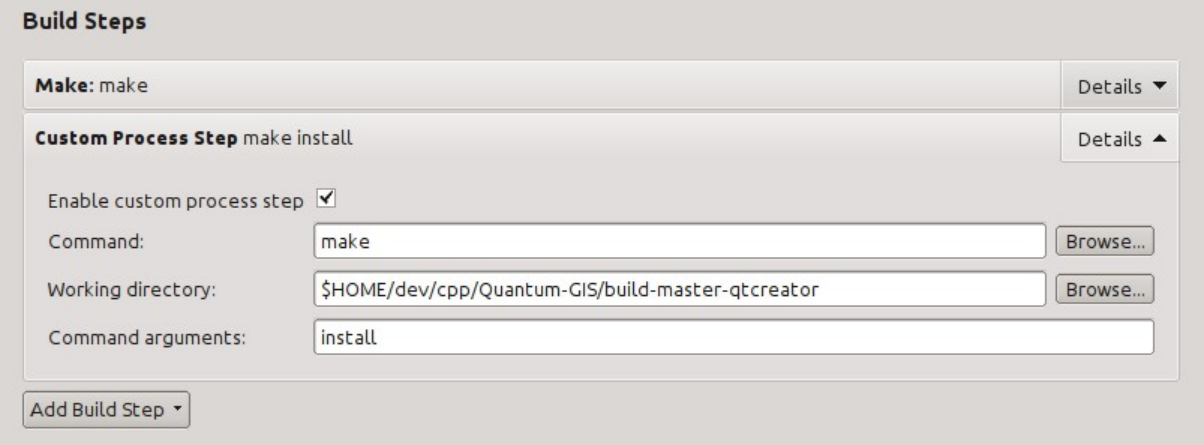
Nu stellen we de volgende details in:

Enable custom process step: [yes]

Command: make

Working directory: $HOME/dev/cpp/QGIS/build-master-qtcreator

Command arguments: install



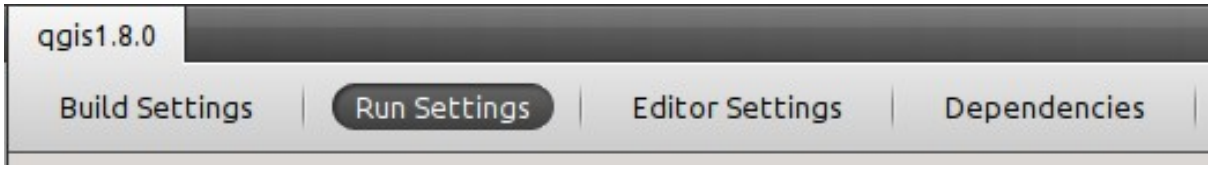
U bent bijna gereed om te bouwen. Nog slechts één opmerking: QtCreator heeft schrijfrechten nodig voor het voorvoegsel install. Standaard (wat ik hier gebruik) zal QGIS worden geïnstalleerd in `/usr/local/`. Voor mijn doelen op mij machine voor ontwikkelen, gaf ik mijzelf schrijfrechten voor de map /usr/local.

Klik op het pictogram van de grote hamer aan de linker onderkant van het venster om het bouwen te beginnen.

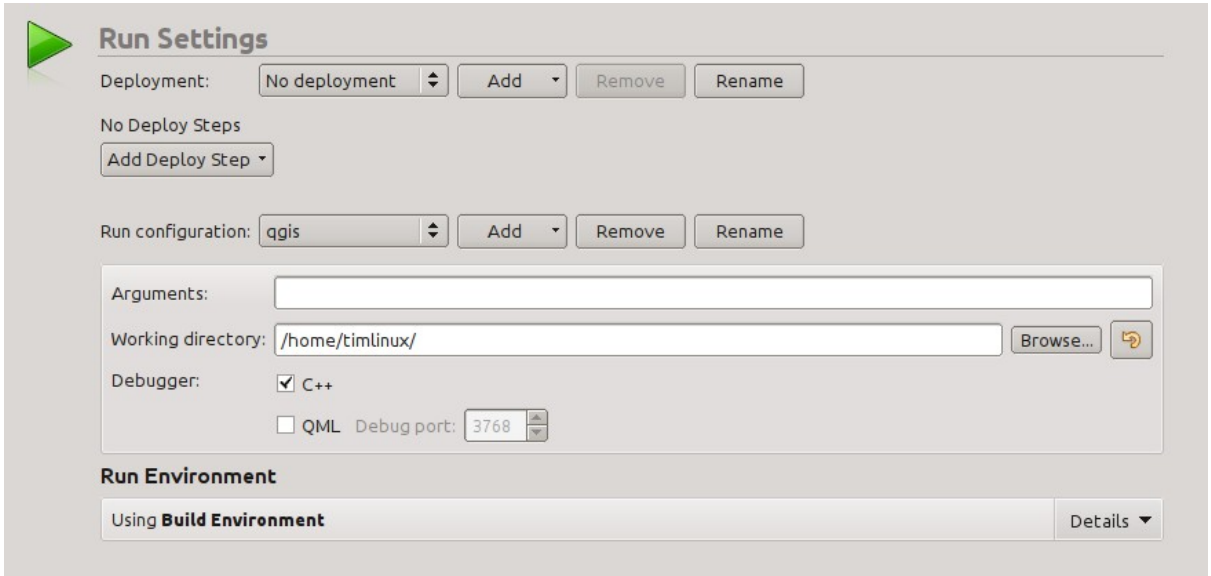
4.4 Instellen van uw omgeving voor uitvoeren

Zoals hierboven vermeld kunnen we QGIS niet direct uitvoeren vanuit de map build, dus moeten we een aangepast doel voor uitvoeren maken om QtCreator te vertellen om QGIS uit te voeren vanuit de map install (in mijn geval `/usr/local/`). Ga, om dat te doen. terug naar het scherm voor configuratie van het project.

Selecteer nu de tab ‘Run Settings’



We dienen de standaard instellingen voor uitvoeren bij te werken om niet de configuratie voor het uitvoeren van ‘qgis’ te gebruiken maar een aangepaste.



Klik, om dat te doen, op de combinatieknop ‘Add v’ naast de combinatieknop Run configuration en kies ‘Custom Executable’ boven uit de lijst.

Stel nu in het gebied Properties de volgende details in:

Executable: /usr/local/bin/qgis

Arguments :

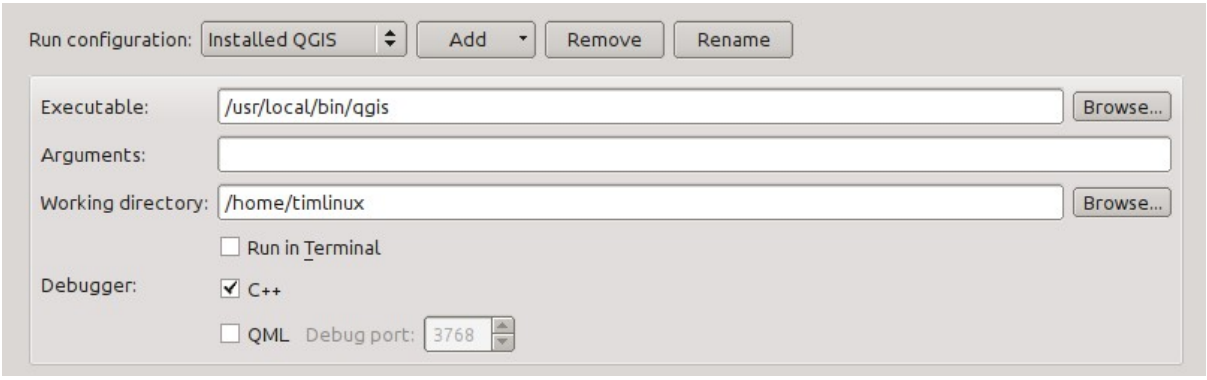
Working directory: $HOME

Run in terminal: [no]

Debugger: C++ [yes]

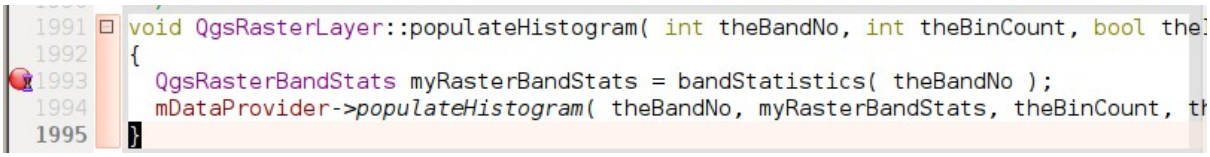
Qml [no]

Klik dan op de knop 'Rename' en geef uw aangepaste executable een betekenisvolle naam bijv. 'Geïnstalleerde QGIS'

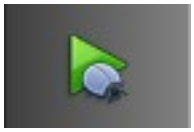


4.5 Uitvoeren en debuggen

Nu bent u gereed om QGIS uit te voeren en te debuggen. Open eenvoudigweg een bronbestand en klik in de kolom links om een breekpunt in te stellen.



Start nu QGIS onder de debugger door te klikken op het pictogram met een insect erop in de linker onderzijde van het venster.



CHAPTER 5

Testen van eenheden

Vanaf november 2007 vereisen we dat alle nieuwe mogelijkheden die in master gaan worden vergezeld door een test van eenheden. Initieel hadden we deze eis beperkt tot qgis_core, en we zullen deze eis uitbreiden naar andere delen van de codebasis als mensen eenmaal bekend zijn met de procedures voor het testen van eenheden, uitgelegd in de gedeelten die volgen.

5.1 Het test framework voor QGIS - een overzicht

Testen van eenheden wordt uitgevoerd met behulp van een combinatie van QTestLib (de testbibliotheek van Qt) en CTest (een framework voor het compileren en uitvoeren van testen als deel van het CMake buildproces). Laten we eens naar een overzicht van het proces kijken, voordat we dieper op de details ingaan:

1. Er is enige code die u wilt testen, bijv. een klasse of functie. Extreme experts van programmeren stellen voor dat de code nog niet zou moeten zijn geschreven wanneer u begint met het bouwen van uw tests, en dan, als u uw code implementeert, kunt u onmiddellijk elk nieuw functionele gedeelte dat u toevoegt kunnen valideren met uw test. In de praktijk dient u waarschijnlijk testen te schrijven voor reeds bestaande code in QGIS omdat we met een test framework beginnen ruim nadat veel logica voor de toepassing al is geïmplementeerd.

2. U maakt een test voor eenheden. Dit gebeurt onder `<QGIS Source Dir>/tests/src/core` in het geval van de core lib. De test is in de basis een cliënt die een instantie van ene klasse maakt en enkele methoden van die klasse aanroept. Het zal de teruggave voor elke methoden controleren om er voor te zorgen dat het overeenkomt met de verwachte waarde. Als één van de aanroepen mislukt, zal de eenheid mislukken.
3. U neemt macro's van QtTestLib op in uw testklasse. Deze macro wordt verwerkt door de Qt meta object compiler (moc) en breidt uw testklasse uit naar een uitvoerbare toepassing.
4. U voegt een gedeelte toe aan CMakeLists.txt in uw map met testen die uw test zal bouwen.
5. Zorg er voor dat u `ENABLE_TESTING` heeft ingeschakeld in ccmake / cmakesetup. Dat zal er voor zorgen dat uw testen in feite worden gecompileerd als u make typt.
6. U voegt optioneel testgegevens toe aan `<QGIS Source Dir>/tests/testdata` als uw test aangedreven wordt door gegevens (bijv. dient een shapefile te laden). Deze testgegevens zouden zo klein mogelijk moeten zijn en waar mogelijk zou u de reeds daar aanwezige gegevens moeten gebruiken. Uw testen zouden nooit die gegevens in situ moeten aanpassen, maar in plaats daarvan ergens een tijdelijke kopie moeten maken, indien nodig.
7. U compileert uw bronnen en installeert. Doe dit met behulp van de normale procedure `make && (sudo) make install`.
8. U voert uw testen uit. Dit wordt normaal gesproken eenvoudig gedaan door `make test` te doen na de stap `make install`, hoewel we wel andere benaderingen uitleggen die meer fijnmazige controle over het uitvoeren van testen bieden.

Met dat overzicht in gedachten zullen we een beetje meer ingaan op de details. We hebben al veel van de configuratie voor u gedaan in CMake en op andere plaatsen in de boom van de bron, dus alles wat u nog zou moeten doen zijn de eenvoudige gedeelten - testen voor eenheden schrijven!

5.2 Een test voor eenheden maken

Het maken van een test voor eenheden is eenvoudig - gewoonlijk zult u dit doen door één enkel `.cpp`-bestand te maken (er wordt geen `.h`-bestand gebruikt) en implementeer al uw testmethoden als publieke methoden die void teruggeven. We gebruiken, ter illustratie, een eenvoudige testklasse voor `QgsRasterLayer` in het gedeelte dat hierop volgt. Volgens conventie zullen we onze test dezelfde naam geven als de klasse die getest wordt, maar met het voorvoegsel 'Test'. Dus onze test-implementatie gaat in een bestand genaamd `testqgsrasterlayer.cpp` en de klasse zelf zal zijn `TestQgsRasterLayer`. Eerst voegen we onze standaard banner voor auteursrechten toe:

```
/**************************************************************************
 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.
 *
 **************************************************************************/
```

Vervolgens beginnen we met het vermelden van onze includes die nodig zijn voor de testen die we willen uitvoeren. Er is één speciale include die alle testen zouden moeten hebben:

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

Naast dat u gewoon doorgaat met het implementeren van uw klasse zoals gewoonlijk, er headers in opnemend die u nodig zou kunnen hebben:

```
//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>
```

Omdat we zowel de declaratie van de klasse als de implementatie in één enkel bestand hebben, is het volgende de declaratie van de klasse. We beginnen met onze documentatie voor doxygen. Elke test zou juist gedocumenteerd moeten zijn. We gebruiken het doxygen ingroup directive zodat alle UnitTests als een module verschijnen in de gegenereerde documentatie voor Doxygen. Daarna komt ene korte beschrijving van de test van de eenheid en de klasse moet erven van QObject en de macro Q_OBJECT bevatten.

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

class TestQgsRasterLayer: public QObject
{
    Q_OBJECT
```

Alle onze testmethoden worden geïmplementeerd als private slots. Het framework QtTest zal op volgorde vervolgens elke methode private slot in de testklasse aanroepen. Er zijn vier 'speciale' methoden die, indien geïmplementeerd, zullen worden aangeroepen aan het begin van de test van de eenheid (`initTestCase`), aan het einde van de test van de eenheid (`cleanupTestCase`). Vóórdat elke testmethode wordt aangeroepen, wordt de methode `init()` aangeroepen en na elke testmethode wordt de methode `cleanup()` aangeroepen. Deze methoden zij handig omdat zij u in staat stellen bronnen toe te wijzen en op te schonen, voorafgaande aan elke test, en de test van de eenheid als geheel.

```
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();
```

Dan komen uw testmethoden, die geen van allen parameters zouden moeten hebben en void zouden moeten teruggeven. De methoden zullen worden aangeroepen in de volgorde van hun declaratie. We implementeren hier twee methoden die twee typen van testen illustreren.

In het eerste geval willen we in het algemeen testen of de verschillende delen van de klasse werken. We kunnen een functionele benadering voor het testen gebruiken. Nogmaals, zeer ervaren programmeurs zouden er misschien voor pleiten deze testen te implementeren vóór het implementeren van de klasse. Als u zich dan door uw implementatie van de klasse heen werkt, voert u achtereenvolgens uw testen van de eenheid uit. Meer en meer testfuncties zouden met succes moeten voltooien als het werk aan uw implementatie van de klasse vordert, en wanneer de gehele test van de eenheid slaagt, is uw nieuwe klasse klaar en is nu compleet met een te herhalen manier om hem te valideren.

Gewoonlijk zouden uw testen voor eenheden alleen de public API van uw klasse behandelen, en normaal gesproken behoeft u geen testen te schrijven voor accessors en mutators. Als het zou gebeuren dat een acccessor of

mutator niet werkt zoals u verwacht, zou u normaal gesproken een *regressietest* implementeren om hierop te controleren.

```
//
// Functional Testing
//

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

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

5.2.1 Een regressietest implementeren

Vervolgens implementeren we onze regressietesten. Regressietesten zouden moeten worden geïmplementeerd om de voorwaarden van een bepaald probleem te kunnen reproduceren. Bijvoorbeeld:

1. We ontvingen per e-mail een rapport dat de telling voor cellen in een er steeds 1, naast zat, wat alle statistieken voor de rasterbanden onbetrouwbaar maakte.
2. We openden een rapport voor het probleem (ticket #832)
3. We maakten een regressietest die het probleem reproduceerde met behulp van een kleine test-gegevensset (een raster van 10x10).
4. We voerden de test uit en verifieerden inderdaad dat hij faalde (de telling van de cellen kwam uit op 99 in plaats van op 100).
5. Hierna repareerden we het probleem en voerden de eenheidstest opnieuw uit en de regressietest werd met succes uitgevoerd. We dienden de regressietest samen in met de oplossing van het probleem. Als nu iemand in de toekomst de broncode opnieuw breekt, kunnen we onmiddellijk identificeren dat de code achteruit is gegaan.

   Beter nog, voordat wijzigingen in de toekomst worden ingediend, zou het uitvoeren van onze testen er voor zorgen dat dat onze wijzigingen geen onverwachte neveneffecten hebben - zoals het breken van bestaande functionaliteit.

Er is nog een voordeel van regressietests - zij kunnen u tijd besparen. Als u ooit een bug oploste die mede bestond uit het maken van wijzigingen aan de bron, en daarna de toepassing uitvoerde en een reeks gecompliceerde stappen uitvoerde om het probleem te repliceren, zal het onmiddellijk duidelijk zijn dat het eenvoudigweg implementeren van uw regressietest vóór het oplossen van het probleem het automatiseren van het testen voor oplossingen voor het probleem op een efficiënte manier laat uitvoeren.

U zou, voor het implementeren van uw regressietest, de conventie voor namen van **regressie<TicketID>** moeten volgen voor uw testfuncties. Als er geen ticket bestaat voor de regressie zou u er eerst een moeten maken. Het gebruiken van deze benadering stelt de persoon die een mislukte regressietest uitvoerde in staat eenvoudig door te gaan en meer informatie te zoeken.

```
//
// 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 ...
```

Tenslotte kunt u in uw declaratie van de testklasse nog persoonlijk enkele gegevensleden en hulpmethoden declareren die uw test voor eenheid nodig zou kunnen hebben. In ons geval zal ik een `QgsRasterLayer *` declareren die kan worden gebruikt door elk van onze testmethoden. De rasterlaag zal worden gemaakt in de

functie `initTestCase()` die wordt uitgevoerd vóór enige andere test, en dan worden vernietigd met behulp van `cleanupTestCase()` die na elke test wordt uitgevoerd. Door het persoonlijk declareren van hulpmethoden (die aangeroepen kunnen worden door verscheidene testfuncties), kunt u er voor zorgen dat zij niet automatisch zullen worden uitgevoerd door de uitvoerbare QTest die wordt gemaakt wanneer we onze test compileren.

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

Dat beëindigt onze declaratie van de klasse. De implementatie is eenvoudigweg opgenomen in hetzelfde bestand hieronder. Eerst onze functies init en cleanup:

```
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() << std::endl;

  //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;
}
```

Bovenstaande functie init illustreert een aantal interessante dingen.

1. We moesten handmatig het gegevenspad naar de toepassing van QGIS instellen zodat bronnen zoals `srs.db` op de juiste manier worden gevonden.
2. Ten tweede is dat een door gegevens gedreven test dus moesten we een manier verschaffen om generiek het bestand `tenbytenraster.asc` te lokaliseren. Dit werd bereikt door met behulp van de compiler het `TEST_DATA_PATH` te definiëren. De definitie wordt gemaakt in het configuratiebestand `CMakeLists.txt` onder `<QGIS Source Root>/tests/CMakeLists.txt` en is beschikbaar voor alle testen van eenheden in QGIS. Als u gegevens voor uw test dient te testen, plaats het dan onder `<QGIS Source Root>/tests/testdata`. U zou hier slechts hele kleine gegevenssets moeten plaatsen. Als uw test de testgegevens dient aan te passen zou het daar eerst een kopie van moeten maken.

Qt verschaft ook enige andere interessante mechanismen voor gegevens gedreven testen, als u dus meer wilt weten over dit onderwerp, consulteer dan de documentatie van Qt.

Laten we vervolgens eens kijken naar onze functionele test. De test `isValid()` controleert eenvoudigweg of de rasterlaag juist werd geladen in de initTestCase. QVERIFY is een macro van Qt die u kunt gebruiken om de de

voorwaarde van de test te evalueren. er zijn ook nog ene paar andere macro's die Qt verschaft om te gebruiken bij uw testen, inclusief:

- 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*)

Enkele van deze macro's zijn alleen nuttig bij het gebruiken van het Qt framework voor gegevens gedreven testen (bekijk de documentatie van Qt voor meer details).

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

Normaal gesproken zouden uw functionele testen het gehele bereik van de functionaliteit behandelen van uw klassen voor de public API waar mogelijk. met onze functionele testen uit de weg kunnen we kijken naar het voorbeeld van onze regressietest.

Omdat het probleem in bug #832 een rapport is over een foutieve telling van cellen, is het schrijven van onze test eenvoudigweg een geval van QVERIFY gebruiken om te controleren of de telling van de cellen voldoet aan de verwachte waarde:

```
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 );
}
```

Met alle functies voor de test van de eenheid geïmplementeerd, is er één laatste ding dat we moeten doen om onze testklasse toe te voegen:

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

Het doel van deze twee regels is om een signaal af te geven naar Qt's moc dat dit een QtTest is (het zal een hoofdmethode genereren die op zijn beurt elke testfunctie aanroept. De laatste regel is de include voor de door MOC gegenereerde bronnen. U zou `testqgsrasterlayer` moeten vervangen door de naam van uw klasse in kleine letters.

5.3 Afbeeldingen vergelijken voor testen van renderen

Renderen van afbeeldingen op verschillende omgevingen kan subtiele verschillen produceren wegens platform-specifieke implementaties (bijv. verschillend renderen van lettertypen en algoritmen voor antialiasing), voor de beschikbare lettertypen op het systeem en om andere onduidelijke redenen.

Wanneer een test voor renderen wordt uitgevoerd op Travis en mislukt, zoek dan naar de link met het streepje aan de uiterste onderzijde van het log van Travis. Deze link zal u meenemen naar een pagina van cdash, waar u de gerenderde vs verwachte afbeeldingen kunt zien, naast een afbeeldingen met "verschillen" die in rood pixels accentueert die niet overeen kwamen met de afbeelding waarnaar verwezen werd.

Het systeem voor het testen van eenheden van QGIS heeft ondersteuning voor het toevoegen van "masker"afbeeldingen, die worden gebruikt om aan te geven wanneer een gerenderde afbeelding af zou kunnen wijken van de afbeelding waarnaar verwezen wordt. Een maskerafbeelding is een afbeelding (met dezelfde naam en als de afbeelding waarnaar verwezen wordt, maar inclusief het achtervoegsel **_mask.png**), en zou van dezelfde dimensies moeten zijn als de afbeelding waarnaar verwezen wordt. In een maskerafbeelding geven de pixelwaarden aan hoeveel die individuele pixel mag afwijken van de afbeelding waarnaar verwezen wordt, dus een zwarte pixel geeft aan dat de pixel in de gerenderde afbeelding moet exact overeenkomen met dezelfde pixel in de afbeelding waarnaar verwezen wordt. Een pixel met RGB 2, 2, 2 betekent dat de gerenderde afbeelding tot maximaal 2 kan variëren in zijn waarden voor RGB in de afbeelding waarnaar verwezen wordt, en een volledig witte pixel (255, 255, 255) betekent dat de pixel effectief wordt genegeerd bij het vergelijken van de verwachte en gerenderde afbeeldingen.

Een utility-script om maskerafbeeldingen te genereren is beschikbaar als `scripts/generate_test_mask_image.py`. Dit script wordt gebruikt door het pad van een afbeelding waarnaar verwezen moet worden (bijv. `tests/testdata/control_images/annotations/expected_annotation_fillstyle/expected_annotation_fillstyle.png`) en het pad naar uw gerenderde afbeelding door te geven.

Bijv.

```
scripts/generate_test_mask_image.py tests/testdata/control_images/annotations/
↪expected_annotation_fillstyle/expected_annotation_fillstyle.png /tmp/path_to_
↪rendered_image.png
```

U kunt het pad naar de afbeelding waarnaar verwezen moet worden verkorten door in plaats daarvan een gedeelte van de testnaam door te geven, bijv.

```
scripts/generate_test_mask_image.py annotation_fillstyle /tmp/path_to_rendered_
↪image.png
```

(Dit verkorten werkt alleen als er één enkele overeenkomende afbeelding, waarnaar verwezen moet worden, wordt gevonden. Als meerdere overeenkomsten worden gevonden dient u het volledige pad naar de afbeelding waarnaar verwezen moet worden op te geven.)

Het script accepteert ook URL's voor HTTP voor de gerenderde afbeelding. U kunt dus een URL van een gerenderde afbeelding direct vanuit de pagina met resultaten van cdash kopiëren en in het script plakken.

Wees voorzichtig bij het maken van maskerafbeeldingen - u zou altijd de maskerafbeelding moeten bekijken en witte gebieden in de afbeeldingen nader moeten bekijken. Omdat die pixels worden genegeerd, zorg er dan voor dat die witte afbeeldingen geen belangrijke gedeelten van de afbeelding waarnaar verwezen wordt bedekken – anders heeft uw eenheidstest geen betekenis!

Op dezelfde wijze kunt u handmatig gedeelten van het masker "wit maken" als u ze met opzet wilt uitsluiten van de test. Dit kan nuttig zijn voor bijv. testen met een mix van symbool- en tekstrenderen (zoals testen van de legenda), waar de eenheidstest niet is ontworpen om de gerenderde tekst te testen en u niet wilt dat de test onderworpen wordt aan kruis-platform verschillen bij het renderen van tekst.

U zou de klasse `QgsMultiRenderChecker` of een van zijn subklassen moeten gebruiken om afbeeldingen te vergelijken met eenheidstesten van QGIS.

Hier zijn enkele tips om de robuustheid van testen te verbeteren:

1. Schakel antialiasing uit als u kunt, dit minimaliseert kruis-platform verschillen bij renderen.
2. Zorg er voor dat afbeeldingen waarnaar verwezen wordt "blokkig"... zijn, d. i. geen lijnen met een breedte van 1 px hebben of andere fijne objecten, en gebruik grote, vette lettertypen (14 punten of meer wordt aanbevolen).
3. Soms maken testen enigszins afwijkende grootten van afbeeldingen (bijv. testen voor renderen van legenda's, waar de grootte van de afbeelding afhankelijk is van de grootte voor renderen van het lettertype - wat onderwerp is voor kruis-platform verschillen). Gebruik, om hier rekening mee te houden, `QgsMultiRenderChecker::setSizeTolerance()` en specificeer het maximale aantal pixels die de gerenderde afbeelding in breedte en hoogte mag afwijken van de afbeelding waarnaar verwezen wordt.
4. Gebruik geen transparante achtergronden in afbeelding waarnaar verwezen wordt (CDash ondersteunt ze niet). Gebruik in plaats daarvan `QgsMultiRenderChecker::drawBackground()` om een patroon voor een schaakbord te tekenen voor de achtergrond van de afbeelding waarnaar verwezen wordt.
5. Wanneer lettertypen vereist zijn, gebruik dan het lettertype dat is gespecificeerd in `QgsFontUtils::standardTestFontFamily()` ("QGIS Vera Sans").

5.4 Uw test voor eenheden toevoegen aan CMakeLists.txt

Toevoegen van uw test voor eenheden aan het bouwsysteem is eenvoudigweg een geval van het bewerken van `CMakeLists.txt` in de map test, klonen van één van de bestaande tekstblokken, en dan de naam van uw testklasse daar invullen. Bijvoorbeeld:

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

5.4.1 De macro ADD_QGIS_TEST uitgelegd

We zullen kort door deze regels gaan om uit te leggen wat zij doen, maar als u daar niet in geïnteresseerd bent, doe dan alleen de stap die is uitgelegd in bovenstaand gedeelte.

```
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)
```

Laten we eens iets meer in detail kijken naar de individuele regels. Eerst definiëren we de lijst met bronnen voor onze test. Omdat we slechts één bronbestand hebben (aldus de methodologie volgend die hierboven is beschreven waar declaratie van de klasse en definitie in hetzelfde bestand staan) is het een eenvoudig argument:

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

Omdat onze testklasse moet worden uitgevoerd door de Qt meta object compiler (moc) dienen we ook een aantal regels te verschaffen om dat mogelijk te maken:

```
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})
```

Vervolgens vertellen we cmake dat het een executable moet maken vanuit de testklasse. Onthoud dat in het vorige gedeelte op de laatste regel van de implementatie van de klasse we de uitvoer voor MOC direct opnamen in onze testklasse, zodat het (naast ander dingen) een hoofdmethode zal geven zodat de klasse als een executable kan worden gecompileerd:

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

Vervolgens dienen we enkele afhankelijkheden voor bibliotheken te specificeren. Op dit moment worden klassen geïmplementeerd met een catch-all QT_LIBRARIES afhankelijkheid, maar we zullen er aan werken om dat te vervangen door de specifieke bibliotheken van Qt die elke klasse alleen nodig heeft. Natuurlijk dient u ook de relevante bibliotheken van QGIS te koppelen, zoals vereist door uw test voor eenheden.

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

Vervolgens vertellen we cmake om de testen te installeren op dezelfde plaats als waar de binaries van QGIS zelf staan. Dit is iets waarvan ik van plan ben om dit in de toekomst te verwijderen zodat de testen direct vanuit de boom van de bron kunnen worden uitgevoerd.

```
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)
```

Tenslotte gebruikt bovenstaande `ADD_TEST` om de test te registreren met cmake / ctest. Hier is waar de beste magie gebeurt - we registreren de klasse met ctest. Als u zich het overzicht nog herinnert dat we gaven in het begin van dit gedeelte, gebruiken we zowel QtTest als CTest samen. Recapitulerend, QtTest voegt een hoofdmethode

toe aan uw test voor eenheden en behandelt het aanroepen van uw testmethoden binnen de klasse. Het verschaft ook enkele macro's zoals `QVERIFY` die u kunt gebruiken om te testen op mislukkingen van de door de testen gebruikte voorwaarden. De uitvoer van een QtTest test voor eenheden is een executable die u kunt uitvoeren vanaf de opdrachtregel. Wanneer u echter een suite van testen heeft en u wilt elke executable op zijn beurt uitvoeren, en beter nog uitvoerende testen integreren in het bouwproces, is de CTest wat we gebruiken.

5.5 Uw test voor eenheden bouwen

Voor het bouwen van de test voor eenheden dient u er alleen voor te zorgen dat `ENABLE_TESTS=true` in de configuratie cmake. Er zijn twee manieren om dat te doen:

1. Voer `ccmake ..` uit (of `cmakesetup ..` onder Windows) en stel interactief de vlag `ENABLE_TESTS` in op `ON`.
2. Voeg een vlag voor de opdrachtregel toe aan cmake bijv. `cmake -DENABLE_TESTS=true ..`

Anders dan dat, bouw QGIS gewoon zoals normaal en de testen zouden meegebouwd moeten worden.

5.6 Voer uw testen uit

De eenvoudigste manier om de testen uit te voeren is als deel van uw normale bouwproces:

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

De opdracht `make test` zal CTest activeren dat elke test zal uitvoeren die werd geregistreerd met behulp van de ADD_TEST CMake directive beschreven hierboven. Normale uitvoer van `make test` zal er uitzien zoals dit:

```
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
```

Als een test mislukt, kunt u de opdracht ctest gebruiken om meer nader te bekijken waarom het mislukt is. Gebruik de optie `-R` om een regex te specificeren voor de testen die u wilt uitvoeren en `-V` om uitgebreide uitvoer te krijgen:

```
$ 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//
↪mIconProjectionDisabled.png
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
```

5.6.1 Debuggen met eenheidstesten

Voor eenheidstesten in C++ voegt QtCreator automatisch doelen voor uitvoeren toe, dus u kunt ze starten in de debugger.

Het is ook mogelijk eenheidstesten voor Python te starten vanuit QtCreator met GDB. Hiervoor dient u te gaan naar *Projects* en te kiezen voor *Run* onder *Build & Run*. Voeg dan een nieuwe `Run configuration` toe met de executable `/usr/bin/python3` en de argumenten voor de opdrachtregel ingesteld op het pad van het bestand voor Python van de eenheidstest, bijv. `/home/user/dev/qgis/QGIS/tests/src/python/test_qgsattributeformeditorwidget.py`.

Wijzig nu ook de `Run Environment` en voeg 3 nieuwe variabelen toe:

Variabele	Waarde
PYTHONPATH	[build]/output/python/:[build]/output/python/plugins:[source]/tests/src/python
QGIS_PREFIX_PATH	[build]/output
LD_LIBRARY_PATH	[build]/output/lib

Vervang `[build]` door de map van uw build en `[source]` door de map van uw bron.

5.6.2 Veel plezier

Welnu, dat is het einde voor dit gedeelte over het schrijven van testen voor eenheden in QGIS. We hopen dat u ook testen zult gaan schrijven om nieuwe functionaliteiten te testen en om te controleren op regressies. Aan sommige aspecten van het testsysteem (in het bijzonder de delen voor `CMakeLists.txt`) wordt nog steeds gewerkt zodat het framework voor testen werkt op een echte platform-onafhankelijke manier.

CHAPTER 6

Testen van algoritmes voor Processing

6.1 Testen voor algoritmes

Notitie: De originele versie van deze instructies is beschikbaar op https://github.com/qgis/QGIS/blob/release-3_4/python/plugins/processing/tests/README.md

QGIS verschaft verscheidene algoritmes in het framework Processing. U kunt die lijst uitbreiden met uw eigen algoritmes en, net als met elke nieuwe mogelijkheid, is het toevoegen van testen vereist.

U kunt, om algoritmes te testen, items toevoegen aan `testdata/qgis_algorithm_tests.yaml` of `testdata/gdal_algorithm_tests.yaml` indien van toepassing.

Dit bestand wordt opgemaakt met yaml syntax.

Een basistest verschijnt onder de sleutel op het hoogste niveau `tests` en ziet er uit als dit:

```
- name: centroid
  algorithm: qgis:polygoncentroids
  params:
    - type: vector
      name: polys.gml
  results:
    OUTPUT_LAYER:
      type: vector
      name: expected/polys_centroid.gml
```

6.1.1 Hoe te doen

Volg deze stappen om een nieuwe test toe te voegen:

1. Voer het algoritme dat u wilt testen uit in QGIS vanuit de Toolbox van Processing. Als het resultaat een vectorlaag is, dan bij voorkeur GML, met zijn XSD, als uitvoer voor zijn ondersteuning van gemixte typen geometrie en goede leesbaarheid. Verplaats de uitvoer naar `python/plugins/processing/tests/testdata/expected`. Gebruik bij voorkeur voor invoerlagen dat wat al in de map `testdata` staat. Als u extra gegevens nodig hebt, plaats die dan in `testdata/custom`.
2. Wanneer u het algoritme hebt uitgevoerd, ga naar *Processing* → *Geschiedenis* en zoek naar het algoritme dat u zojuist hebt uitgevoerd.
3. Klik met rechts op het algoritme en klik op *Test maken*. Een nieuw venster zal openen met een tekstdefinitie.
4. Open het bestand `python/plugins/processing/tests/testdata/algorithm_tests.yaml` en kopieer de tekstdefinitie naar hier.

De eerste tekenreeks van de opdracht gaat naar de sleutel `algorithm`, de volgende naar `params` en de laatste(n) naar `results`.

Bovenstaande laat zich vertalen als

```
- name: densify
  algorithm: qgis:densifygeometriesgivenaninterval
  params:
    - type: vector
      name: polys.gml
    - 2 # Interval
  results:
    OUTPUT:
      type: vector
      name: expected/polys_densify.gml
```

Het is ook mogelijk testen te maken voor scripts van Processing. Scripts zouden moeten worden geplaatst in de submap `scripts` in de map voor de testdata `python/plugins/processing/tests/testdata/`. De bestandsnaam van het script zou overeen moeten komen met de naam van het scriptalgoritme.

6.1.2 Parameters en resultaten

Triviale typen parameters

Parameters en resultaten worden gespecificeerd als lijsten of woordenboeken:

```
params:
  INTERVAL: 5
  INTERPOLATE: True
  NAME: A processing test
```

of

```
params:
  - 2
  - string
  - another param
```

Typen parameters voor lagen

U zult vaak lagen dienen te specificeren als parameters. U dient, om een laag te specificeren, te specificeren:

- het type, d.i. `vector` of `raster`
- een naam, met een relatief pad zoals `expected/polys_centroid.gml`

Dit is hoe het er in actie uitziet:

```
params:
  PAR: 2
  STR: string
  LAYER:
    type: vector
    name: polys.gml
  OTHER: another param
```

Typen parameters voor bestanden

Als u een extern bestand nodig hebt voor de test van het algoritme, dient u het type 'file' te specificeren en het (relatieve) pad naar het bestand in zijn 'naam':

```
params:
  PAR: 2
  STR: string
  EXTFILE:
    type: file
    name: custom/grass7/extfile.txt
  OTHER: another param
```

Resultaten

Resultaten worden op zeer soortgelijke manier gespecificeerd.

Basis vectorbestanden

Het zou niet minder triviaal kunnen zijn

```
OUTPUT:
 name: expected/qgis_intersection.gml
 type: vector
```

Voeg de verwachte GML- en XSD-bestanden toe aan de map.

Vector met tolerantie

Soms maken verschillende platforms enigszins verschillende resultaten die nog steeds acceptabel zijn. In dit geval (maar ook alleen dan) mag u ook aanvullende eigenschappen gebruiken om te definiëren hoe een laag is vergeleken.

Voor het afhandelen van een bepaalde tolerantie voor de waarden van de uitvoer kunt u een eigenschap `compare` specificeren voor een uitvoer. De eigenschap compare mag bepaalde sub-eigenschappen bevatten voor `fields`. Die bevatten informatie over hoe precies een bepaald veld is vergeleken (`precision`) of een veld kan zelfs in zijn geheel worden ge‘‘skip‘‘t. Er bestaat een speciale veldnaam `__all__` die een bepaalde tolerantie op alle velden zal toepassen. Er is een andere eigenschap `geometry` die ook een `precision` accepteert die wordt toegepast op elk punt.

```
OUTPUT:
 type: vector
 name: expected/abcd.gml
 compare:
   fields:
     __all__:
       precision: 5 # compare to a precision of .00001 on all fields
     A: skip # skip field A
   geometry:
     precision: 5 # compare coordinates with a precision of 5 digits
```

Rasterbestanden

Rasterbestanden worden vergeleken met een hash checksum. Die wordt berekent als u een test maakt uit de Geschiedenis van Processing.

```
OUTPUT:
 type: rasterhash
 hash: f1fedeb6782f9389cf43590d4c85ada9155ab61fef6dc285aaeb54d6
```

Bestanden

U kunt de inhoud van een uitvoerbestand vergelijken met een verwijzingsbestand voor het verwachte resultaat

```
OUTPUT_HTML_FILE:
 name: expected/basic_statistics_string.html
 type: file
```

Of u kunt een of meer reguliere expressies gebruiken die zullen worden vergeleken met de inhoud van het bestand

```
OUTPUT:
 name: layer_info.html
 type: regex
 rules:
   - 'Extent: \(-1.000000, -3.000000\) - \(11.000000, 5.000000\)'
   - 'Geometry: Line String'
   - 'Feature Count: 6'
```

Mappen

U kunt de inhoud van een map voor de uitvoer vergelijken met een verwijzingsmap voor het verwachte resultaat

```
OUTPUT_DIR:
 name: expected/tiles_xyz/test_1
 type: directory
```

6.1.3 Context voor het algoritme

er zijn nog een aantal definities die de context van het algoritme kunnen aanpassen - deze mogen worden gespecificeerd in het hoogste niveau van de test:

- `project` - zal een gespecificeerd projectbestand van QGIS laden voordat het algoritme wordt uitgevoerd. Indien niet gespecificeerd zal het algoritme worden uitgevoerd met een leeg project
- `project_crs` - overschrijft het standaard project-CRS - bijv. `EPSG:27700`
- `ellipsoid` - overschrijft de standaard ellipsoïde voor het project die wordt gebruikt voor metingen, bijv. `GRS80`

6.1.4 Testen lokaal uitvoeren

```
ctest -V -R ProcessingQgisAlgorithmsTest
```

of één van de volgende waarden die zijn vermeld in de CMakelists.txt

CHAPTER 7

OGC testen van aanpassingen

Het Open Geospatial Consortium (OGC) verschaft testen die gratis kunnen worden uitgevoerd om er zeker van te zijn dat een server voldoet aan een bepaalde specificatie. Dit hoofdstuk verschaft een snelle handleiding om de testen voor de WMS in te stellen op een systeem van Ubuntu. Gedetailleerde documentatie is te vinden op de website van OGC.

7.1 Instellen van testen voor aanpassingen voor WMS 1.3 en WMS 1.1.1

```
sudo apt 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
```

Download and install WMS 1.3.0 test

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

Test voor WMS 1.1.1 downloaden en installeren

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

7.2 Testproject

Voor de testen van WMS kunnen gegevens worden gedownload en geladen in een project van QGIS:

```
wget https://cite.opengeospatial.org/teamengine/about/wms/1.3.0/site/data-wms-1.3.
↪0.zip
unzip data-wms-1.3.0.zip
```

Maak dan een project voor QGIS overeenkomstig de beschrijving in https://cite.opengeospatial.org/teamengine/about/wms/1.3.0/site/. We moeten later de URL van de GetCapabilities van de service opgeven om de testen uit te kunnen voeren.

7.3 Uitvoeren van de test voor 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
```

7.4 Uitvoeren van de test voor 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
```