seven.monkeys.swe@gmail.com

Definizione di Prodotto

Informazioni sul documento

Nome documento	Definizione di Prodotto
Versione	v2.0.0
Data redazione	2014-02-15
Redattori	• Magnabosco Nicola • Bissacco Nicolò • Adami Alberto • Martignago Jimmy
Verificatori	• Feltre Beatrice • Luisetto Luca
Approvazione	• Scapin Davide
Lista distribuzione	• *Seven Monkeys* • *Prof. Tullio Vardanega* • *Prof. Riccardo Cardin*
Uso	Interno

Sommario

Architettura di dettaglio dell'applicazione Romeo

Diario delle Modifiche

Modifica	Autore & Ruolo	Data	Versione
Approvazione documento	Luisetto Luca *Responsabile di Progetto*	2014-06-03	v2.0.0
Verifica documento prima dell'approvazione	Adami Alberto *Verificatore*	2014-05-31	v1.2.0
Modifiche effettuate in base alle indicazioni del verificatore	Magnabosco Nicola *Progettista*	2014-05-27	v1.1.1
Verifica documento prima dell'approvazione	Adami Alberto *Verificatore*	2014-05-24	v1.1.0
Aggiornato tracciamento	Scapin Davide *Progettista*	2014-04-20	v1.0.8
Sistemati diagrammi di sequenza come da segnalazione in revisione di qualifica	Martignago Jimmy *Progettista*	2014-04-15	v1.0.7
Aggiunti metodi, signal e attributi ad alcune classi	Feltre Beatrice *Progettista*	2014-05-05	v1.0.6
Aggiunta appendice per i programmatori per la realizzazione delle feature come segnalato in revisione di qualifica	Feltre Beatrice *Progettista*	2014-05-03	v1.0.5
Aggiunta di classi alcune classi nel package model e view	Scapin Davide *Progettista*	2014-04-24	v1.0.4
Rimozione di classi non utilizzate	Magnabosco Nicola *Progettista*	2014-04-23	v1.0.3
Sistemate nomenclature dei diagrammi delle classi e dei package come segnalazioni in sede di revisione di qualifica	Martignago Jimmy *Progettista*	2014-04-30	v1.0.2
Restyling del documento	Luisetto Luca *Progettista*	2014-04-15	v1.0.1
Approvazione documento	Scapin Davide *Responsabile di Progetto*	2014-03-03	v1.0.0
Verifica documento prima dell'approvazione	Luisetto Luca *Verificatore*	2014-03-02	v0.2.0
Modifiche effettuate in base alle indicazioni del verificatore	Martignago Jimmy *Progettista*	2014-03-01	v0.1.1
Verifica documento prima dell'approvazione	Feltre Beatrice *Verificatore*	2014-02-27	v0.1.0
Inserimento sezione tracciamento e stesura sezione diagrammi di sequenza	Martignago Jimmy *Progettista*	2014-02-26	v0.0.7
Stesura sezione package qtmodel ed help	Bissacco Nicolò *Progettista*	2014-02-24	v0.0.6
Stesura sezione package util e suoi sottopackage	Martignago Jimmy *Progettista*	2014-02-23	v0.0.5

Stesura sezione package view::dialog e view::component	Adami Alberto *Progettista*	2014-02-21	v0.0.4
Stesura sezione package view::window	Adami Alberto *Progettista*	2014-02-20	v0.0.3
Stesura sezione package core e suoi sottopackage	Bissacco Nicolò *Progettista*	2014-02-18	v0.0.2
Inizio stesura del documento	Nicola Magnabosco *Progettista*	2014-02-15	v0.0.1

Indice

1 Introduzione **1**
1.1 Scopo del documento 1
1.2 Scopo del prodotto 1
1.3 Glossario 1
1.4 Riferimenti 1
1.4.1 Normativi 1
1.4.2 Informativi 1

2 Standard di progetto **2**
2.1 Standard di progettazione architetturale 2
2.2 Standard di documentazione del codice 2
2.3 Standard di denominazione delle entità e relazioni 2
2.4 Standard di programmazione 2
2.5 Strumenti di lavoro 2

3 Specifica componenti **3**
3.1 Romeo 3

4 Specifica componenti Romeo::Model **3**
4.1 Specifica componenti Model::Core 4
4.1.1 AlgCreator (class) 4
4.1.2 Analysis (class) 7
4.1.3 ISubject (interface) 14
4.1.4 ASubject (abstract) 15
4.1.5 ProxySubject (class) 18
4.1.6 RealSubject (abstract) 20
4.1.7 RealSubjectImage (class) 21
4.1.8 RealSubjectVideo (class) 22
4.1.9 GroupOfSubject (class) 23
4.1.10 Protocol (class) 26
4.1.11 Dataset (class) 30
4.1.12 FeatCreator (class) 33
4.1.13 InternalData (interface) 35
4.1.14 InternalData2D (class) 36
4.1.15 InternalData3D (class) 39
4.1.16 RomeoObject(interface) 42
4.1.17 RGBImage (abstract) 43
4.1.18 RGBImage2D (class) 45
4.1.19 RGBImage3D (class) 50
4.1.20 Video(abstract) 55
4.1.21 Video2D(class) 57
4.1.22 Video3D(class) 59
4.2 Specifica componenti Model::Core::Algorithm 61
4.2.1 AAlgorithm (abstract) 61
4.2.2 FuzzyCMeansAlgorithm (class) 66
4.2.3 KMeansAlgorithm (class) 69
4.3 Specifica componenti Model::Core::Feature 74
4.3.1 AFeature (abstract) 74
4.3.2 FirstOrderFeature (abstract) 77

4.3.3 SecondOrderFeature (abstract) . . . 81
4.3.4 DynamicFeature (abstract) . . . 86
4.3.5 MeanFeature (class) . . . 90
4.3.6 KurtosisFeature (class) . . . 92
4.3.7 SkewnessFeature (class) . . . 94
4.3.8 StandardDeviationFeature (class) . . . 96
4.3.9 ContrastFeature (class) . . . 98
4.3.10 CorrelationFeature (class) . . . 100
4.3.11 EnergyFeature (class) . . . 102
4.3.12 EntropyFeature (class) . . . 104
4.3.13 HomogeneityFeature (class) . . . 106
4.3.14 MaximumFeature (class) . . . 108
4.3.15 MeanDynamicFeature (class) . . . 110
4.3.16 MinimumFeature (class) . . . 112
4.3.17 AreaUnderCurveFeature (class) . . . 114
4.4 Specifica componenti Model::Util . . . 116
4.5 Specifica componenti Model::Util::Log . . . 116
4.5.1 Log (class) . . . 116
4.6 Specifica componenti Model::Util::ReaderModel . . . 119
4.6.1 Reader (interface) . . . 120
4.6.2 ImageReader (class) . . . 122
4.6.3 VideoReader (class) . . . 124
4.7 Specifica componenti Model::Util::ExporterModel . . . 125
4.7.1 Exporter (interface) . . . 126
4.7.2 Exporter2D (class) . . . 127
4.7.3 Exporter3D (class) . . . 129
4.8 Specifica componenti Model::Util::DAO . . . 131
4.8.1 ADatabase (Abstract) . . . 132
4.8.2 AlgorithmDAO(class) . . . 134
4.8.3 AnalysisDAO(class) . . . 137
4.8.4 DatasetDAO(class) . . . 141
4.8.5 FeatureDAO(class) . . . 143
4.8.6 GroupDAO(class) . . . 146
4.8.7 ProtocolDAO(class) . . . 149
4.8.8 SubjectDAO(class) . . . 152
4.9 Specifica componenti Romeo::Model::Help . . . 155
4.9.1 Assistant (class) . . . 155
4.10 Specifica componenti Model::QtModel . . . 157
4.10.1 TableModel(abstract) . . . 157
4.10.2 AnalysisProtocolsTableModel(class) . . . 160
4.10.3 AnalysisSubjectsTableModel(class) . . . 164
4.10.4 DatasetGroupTableModel(class) . . . 167
4.10.5 DatasetProtocolTableModel(class) . . . 170
4.10.6 DatasetsTableModel(class) . . . 173
4.10.7 GroupSubjectsTableModel(class) . . . 176
4.10.8 GroupTableModel(class) . . . 179
4.10.9 NewGroupTableModel(class) . . . 182
4.10.10 NewProtocolFeatureTableModel(class) . . . 185
4.10.11 ProtocolTableModel(class) . . . 189
4.10.12 ResultsTableModel(class) . . . 192

4.10.13 SubjectTableModel(class) 195

5 Specifica componenti Romeo::View 198
5.1 Specifica componenti View::Window 198
5.1.1 MainWindow (class) 199
5.1.2 APanel (abstract) 205
5.1.3 WelcomeView (class) 211
5.1.4 NewSubjectView (class) 216
5.1.5 NewGroupView (class) 223
5.1.6 NewProtocolView (class) 228
5.1.7 NewDatasetView (class) 235
5.1.8 SubjectsView (class) 241
5.1.9 GroupView (class) 245
5.1.10 ProtocolsView (class) 249
5.1.11 DatasetsView (class) 254
5.1.12 AnalysisView (class) 258
5.1.13 DetailedResult (class) 264
5.1.14 ResultsView (class) 268
5.2 Specifica componenti View::Dialog 272
5.2.1 Dialog (class) 273
5.2.2 FeatureParams (class) 275
5.2.3 AnalysisDialog (class) 280
5.3 Specifica componenti View::Component 285
5.3.1 ToolBar(class) 286
5.3.2 SelectDeselectWidget (class) 291
5.3.3 NavWidget (class) 292
5.3.4 ImageLabel (class) 298
5.3.5 ResultMessageWidget (class) 300

6 Specifica componenti Romeo::Controller 302
6.1 Romeo::Controller 302
6.1.1 AController (abstract) 302
6.1.2 AnalysisController (class) 304
6.1.3 ControllerManager (class) 310
6.1.4 DatasetsController (class) 312
6.1.5 DetailedResultController (class) 314
6.1.6 GroupsController (class) 317
6.1.7 MainWindowController (class) 319
6.1.8 NewDatasetController (class) 326
6.1.9 NewGroupController (class) 330
6.1.10 NewProtocolController (class) 333
6.1.11 NewSubjectController (class) 338
6.1.12 ProtocolsController (class) 341
6.1.13 ResultsController (class) 344
6.1.14 SubjectsController (class) 346
6.1.15 WelcomeController (class) 348

7 Diagrammi di sequenza 349
7.1 Creazione di un nuovo Subject 349
7.2 Creazione di un nuovo Protocol 351
7.3 Avvio di un'analisi 353

7.4 slotStartAnalysis 354

A Tracciamento 355
A.1 Tracciamento classi-requisiti 355
A.2 Tracciamento requisiti-classi 359
A.3 Tracciamento modulo-test 364
A.4 Codice Matlab 375
A.4.1 Standard Deviation Feature 375
A.4.2 Skewness Feature 375
A.4.3 Mean Feature 376
A.4.4 Kurtosis Feature 376
A.4.5 Contrast Feature 376
A.4.6 Correlation Feature 377
A.4.7 Energy Feature 378
A.4.8 Entropy Feature 378
A.4.9 Homogeneity Feature 379
A.4.10 Area Under the Curve Feature 379
A.4.11 Maximum Feature 379
A.4.12 Minimum Feature 380
A.4.13 Mean Dynamic Feature 380

Elenco delle figure

1 Componente Romeo . . . 3
2 Componente Romeo::Model . . . 3
3 Diagramma package *Romeo::Model::Core* . . . 4
4 Diagramma classe *AlgCreator* . . . 4
5 Diagramma classe *Analysis* . . . 7
6 Diagramma classe *ISubject* . . . 14
7 Diagramma classe *ASubject* . . . 15
8 Diagramma classe *ProxySubject* . . . 18
9 Diagramma classe *RealSubject* . . . 20
10 Diagramma classe *RealSubjectImage* . . . 21
11 Diagramma classe *RealSubjectVideo* . . . 22
12 Diagramma classe *GroupOfSubject* . . . 23
13 Diagramma classe *Protocol* . . . 26
14 Diagramma classe *Dataset* . . . 30
15 Diagramma classe *FeatCreator* . . . 33
16 Diagramma classe *InternalData* . . . 35
17 Diagramma classe *InternalData2D* . . . 36
18 Diagramma Classe *InternalData3D* . . . 39
19 Diagramma calsse *RomeoObject* . . . 42
20 Diagramma classe *RGBImage* . . . 43
21 Diagramma classe *RGBImage2D* . . . 45
22 Diagramma classe *RGBImage3D* . . . 50
23 Diagramma classe *Video* . . . 55
24 Diagramma classe *Video2D* . . . 57
25 Diagramma classe *Video3D* . . . 59
26 Componente Romeo::Model::Core::Algorithm . . . 61
27 Diagramma classe *AAlgorithm* . . . 61
28 Diagramma classe *FuzzyCMeansAlgorithm* . . . 66
29 Diagramma classe *KMeansAlgorithm* . . . 69
30 Diagramma package *Romeo::Model::Core::Feature* . . . 74
31 Diagramma classe *AFeature* . . . 74
32 Diagramma classe *FirstOrderFeature* . . . 77
33 Diagramma classe *SecondOrderFeature* . . . 81
34 Diagramma classe *DynamicFeature* . . . 86
35 Diagramma classe *MeanFeature* . . . 90
36 Diagramma classe *KurtosisFeature* . . . 92
37 Diagramma classe *SkewnessFeature* . . . 94
38 Diagramma classe *StandardDeviationFeature* . . . 96
39 Diagramma classe *ContrastFeature* . . . 98
40 Diagramma classe *CorrelationFeature* . . . 100
41 Diagramma classe *EnergyFeature* . . . 102
42 Diagramma classe *EntropyFeature* . . . 104
43 Diagramma classe *HomogeneityFeature* . . . 106
44 Diagramma classe *MaximumFeature* . . . 108
45 Diagramma classe *MeanDynamicFeature* . . . 110
46 Diagramma classe *MinimumFeature* . . . 112
47 Diagramma classe *AreaUnderCurveFeature* . . . 114
48 Diagramma package *Romeo::Model::Util* . . . 116

49 Diagramma package *Romeo::Model::Util::Log* . . . 116
50 Diagramma classe *Log* . . . 117
51 Diagramma package *Romeo::Model::Util::ReaderModel* . . . 119
52 Interfaccia Reader: metodi . . . 120
53 Classe ImageReader: attributi e metodi . . . 122
54 Classe VideoReader: attributi e metodi . . . 124
55 Diagramma package *Romeo::Model::Util::ExporterModel* . . . 125
56 Diagramma classe *Exporter* . . . 126
57 Diagramma classe *Exporter2D* . . . 127
58 Diagramma classe *Exporter3D* . . . 129
59 Diagramma package *Romeo::Model::Util::DAO* . . . 131
60 Diagramma classe *ADatabase* . . . 132
61 Diagramma classe *AlgorithmDAO* . . . 134
62 Diagramme classe *AnalysisDAO* . . . 137
63 Classe DatasetDAO: attributi e metodi . . . 141
64 Diagramma classe *FeatureDAO* . . . 143
65 Diagramma classe *GroupDAO* . . . 146
66 Diagramma classe *ProtocolDAO* . . . 149
67 Diagramma classe *SubjectDAO* . . . 152
68 Diagramma package *Romeo::Model::Help* . . . 155
69 Diagramma classe *Assistanti* . . . 155
70 Diagramma package *Romeo::Model::QtModel* . . . 157
71 Diagramma classe *TableModel* . . . 157
72 Diagramma classe *AnalysisProtocolsTableModel* . . . 160
73 Diagramma classe *tAnalysisSubjectsTableModel* . . . 164
74 Diagramma classe *DatasetGroupTableModel* . . . 167
75 Diagramma classe *DatasetProtocolTableModel* . . . 170
76 Diagramma classe *DatasetsTableModel* . . . 173
77 Diagramma classe *GroupSubjectsTableModel* . . . 176
78 Diagramma classe *GroupTableModel* . . . 179
79 Diagramma classe *NewGroupTableModel* . . . 182
80 Diagramme classe *NewProtocolFeatureTableModel* . . . 185
81 Diagramma classe *ProtocolTableModel* . . . 189
82 Diagramma classe *ResultsTableModel* . . . 192
83 Diagramma classe *SubjectTableModel* . . . 195
84 Componente Romeo::View . . . 198
85 Componente Romeo::View::Window . . . 198
86 Diagramma Classe MainWindow: attributi e metodi . . . 199
87 Diagramma Classe APanel: attributi e metodi . . . 205
88 Diagramma Classe WelcomeView: attributi e metodi . . . 211
89 Diagramma Classe NewSubjectView: attributi e metodi . . . 216
90 Diagramma Classe NewGroupView: attributi e metodi . . . 223
91 Diagramma Classe NewProtocolView: attributi e metodi . . . 228
92 Diagramma Classe NewDatasetView: attributi e metodi . . . 235
93 Diagramma Classe SubjectsView: attributi e metodi . . . 241
94 Diagramma Classe GroupView: attributi e metodi . . . 245
95 Diagramma Classe ProtocolsView: attributi e metodi . . . 249
96 Diagramma Classe DatasetsView: attributi e metodi . . . 254
97 Diagramma Classe AnalysisView: attributi e metodi . . . 258
98 Diagramma Classe DetailedResult: attributi e metodi . . . 264

99 Diagramma Classe ResultsView: attributi e metodi 268
100 Componente Romeo::View::Dialog 272
101 Diagramma Classe Dialog: attributi e metodi 273
102 Diagramma Classe FeatureParams: attributi e metodi 275
103 Diagramma Classe AnalysisDialog: attributi e metodi 280
104 Componente Romeo::View::Component 285
105 Diagramma Classe ToolBar: attributi e metodi 286
106 Diagramma Classe SelectDeselectWidget: attributi e metodi 291
107 Diagramma Classe NavWidget: attributi e metodi 292
108 Diagramma Classe ImageLabel: attributi e metodi 298
109 Diagramma Classe ResultMessageWidget: attributi e metodi 300
110 Diagramma package *Romeo::Controller* 302
111 Diagramma classe *AController* 302
112 Diagramma classe AnalysisController 304
113 Diagramma classe *ControllerManager* 310
114 Diagramma classe *DatasetsController* 312
115 Diagramma classe DetailedResultController 314
116 Diagramma classe *GroupsController* 317
117 Diagramma classe MainViewController 319
118 Diagramma classe NewDatasetController 326
119 Diagramma classe *NewGroupController* 330
120 Diagramma classe NewProtocolController 333
121 Diagramma classe *NewSubjectController* 338
122 Diagramma classe *ProtocolsController* 341
123 Diagramma classe ResultsController 344
124 Diagramma classe SubjectsController 346
125 Diagramma classe *WelcomeController* 348
126 Diagramma di sequenza: creazione Subject 350
127 Diagramma di sequenza: creazione Protocol 352
128 Diagramma di sequenza: esecuzione analisi 353
129 Diagramma di sequenza: slotStartAnalysis 354

1 Introduzione

1.1 Scopo del documento

Il seguente documento ha lo scopo di definire nel dettaglio la struttura del sistema Romeo, approfondendo quanto già riportato nel documento *Specifica Tecnica v2.0.0*. Tale documento fornisce una struttura dettagliata e completa che viene utilizzata dai *programmatori* per le attività di codifica.

1.2 Scopo del prodotto

Il prodotto che si intende realizzare, denominato Romeo, si propone di fornire un sistema software per applicare la cluster analysis$_{\mathbf{G}}$ ad immagini biomediche. Lo scopo principale è quello di offrire alla comunità scientifica internazionale uno strumento semplice, ma allo stesso tempo completo e flessibile per applicare gli algoritmi della cluster analysis$_{\mathbf{G}}$.

1.3 Glossario

Al fine di evitare ogni ambiguità e per permettere al lettore una migliore comprensione dei termini e acronimi utilizzati nei vari documenti formali, essi sono riportati nel *Glossario v3.0.0* che contiene una descrizione approfondita di tali termini e acronimi.
Ogni volta che compare un termine presente nel *Glossario*, esso è marcato con una "**G**" in pedice.

1.4 Riferimenti

1.4.1 Normativi

- **Specifica Tecnica:** *Specifica Tecnica v3.0.0*;
- **Analisi dei Requisiti:** *Analisi dei Requisiti v4.0.0*;
- **Norme di Progetto:** *Norme di Progetto v4.0.0.*

1.4.2 Informativi

- **Documentazione Qt$_{\mathbf{G}}$ per Signal$_{\mathbf{G}}$ e Slot$_{\mathbf{G}}$**
  http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html;
- **Documentazione ITK$_{\mathbf{G}}$**
  http://itk.org/Doxygen/html/;
- **Documentazione OpenCv**
  http://docs.opencv.org/;
- **Documentazione QtAssistant**
  http://qt-project.org/doc/qt-4.8/assistant-manual.html;
- **Glossario:** *Glossario v3.0.0.*

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard della progettazione architetturale sono definiti nella *Specifica Tecnica v2.0.0*.

2.2 Standard di documentazione del codice

Gli standard per la documentazione del codice sono definiti nelle *Norme di Progetto v3.0.0*.

2.3 Standard di denominazione delle entità e relazioni

Tutti gli elementi definiti, package$_G$, classi, metodi o attributi, devono avere una denominazione chiara e autoesplicativa. Qualora il nome risulti essere lungo è preferibile la chiarezza alla lunghezza. Le abbreviazione sono ammesse solo se:

- immediatamente comprensibili;
- non ambigue.

2.4 Standard di programmazione

Gli standard di programmazione sono definiti e descritti nelle *Norme di Progetto v3.0.0*.

2.5 Strumenti di lavoro

Gli strumenti da adottare e le procedure da seguire per utilizzarli correttamente durante la realizzazione del prodotto software sono definiti nelle *Norme di Progetto v3.0.0*.

3 Specifica componenti

3.1 Romeo

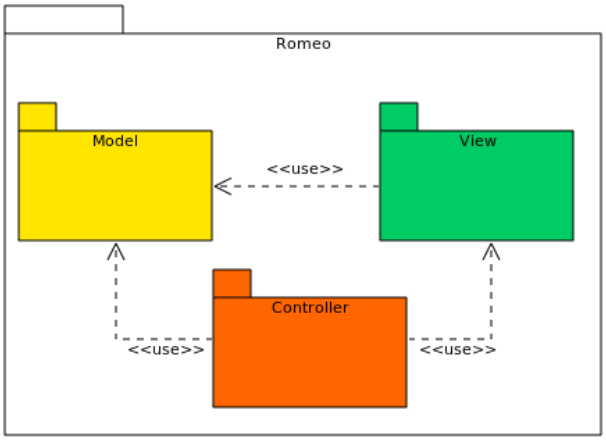

Figura 1: Componente Romeo

4 Specifica componenti Romeo::Model

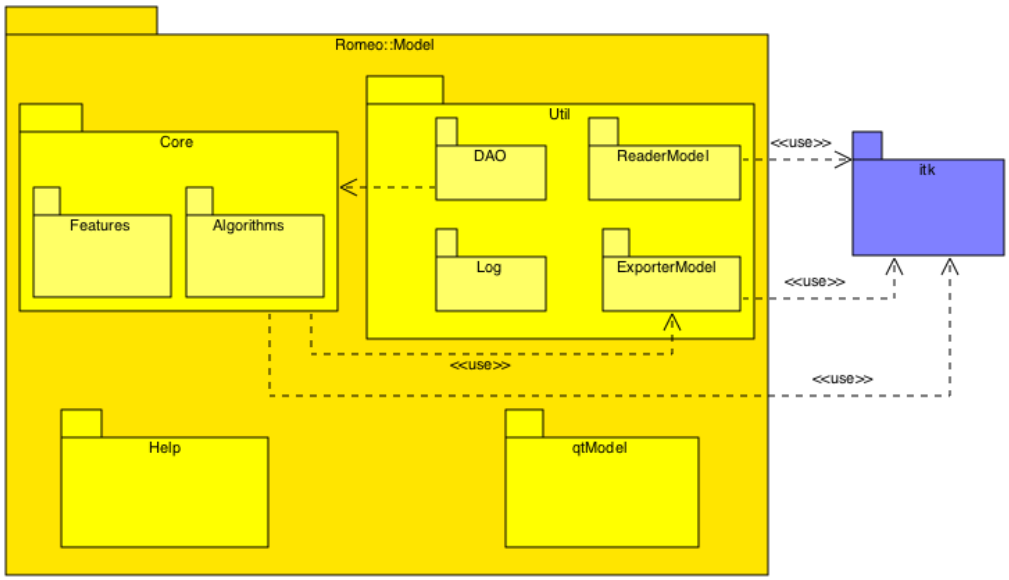

Figura 2: Componente Romeo::Model

Package$_{\mathbf{G}}$ per il componente Model dell'architettura MVC.

4.1 Specifica componenti Model::Core

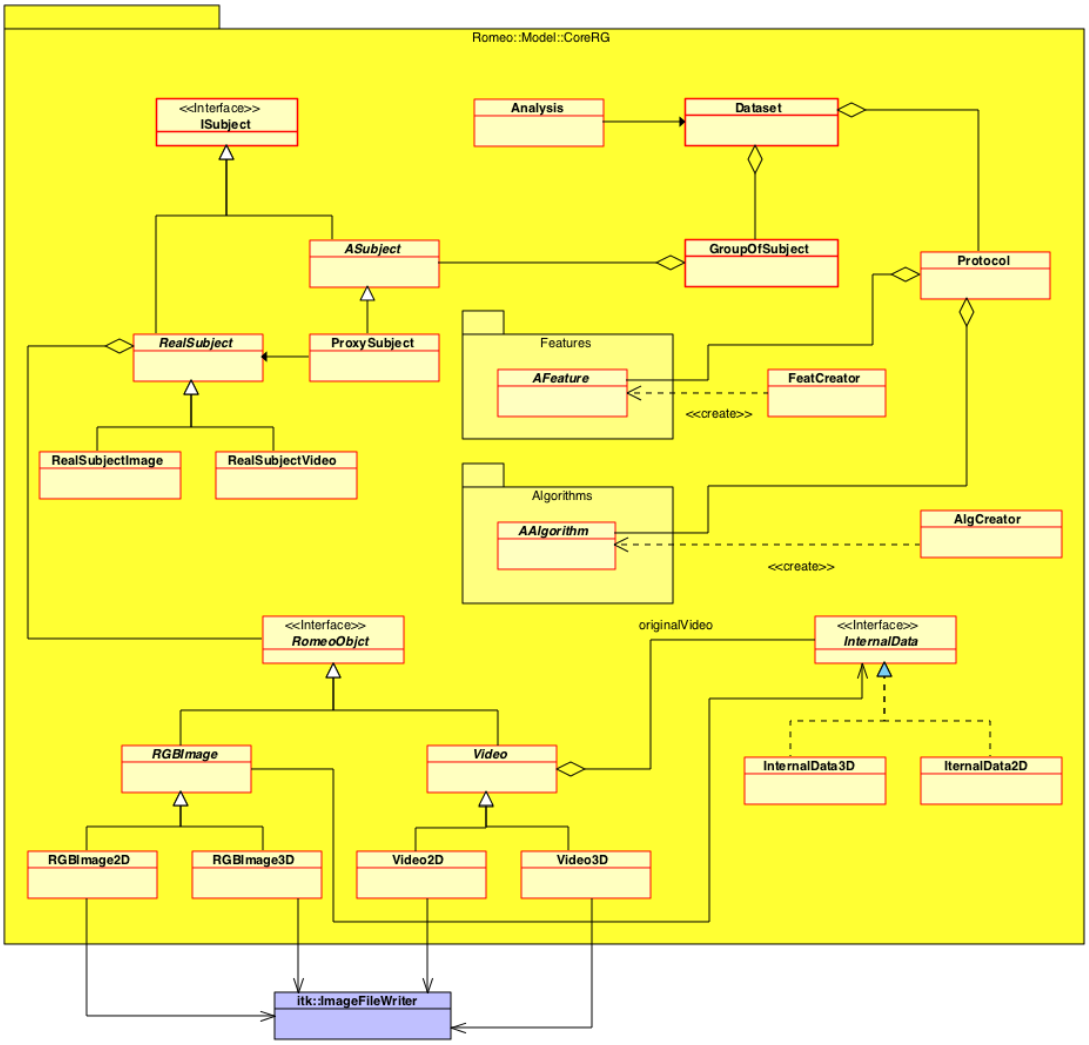

Figura 3: Diagramma package *Romeo::Model::Core*

Package$_{G}$ per il componente core.

4.1.1 AlgCreator (class)

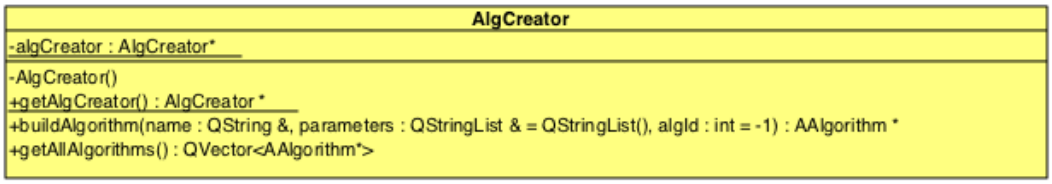

Figura 4: Diagramma classe *AlgCreator*

Descrizione

classe Factory avente la responsabilità di creare un oggetto di tipo *Romeo::Model::Core::Algorithms::AAlgorithm*, che rappresenta un' instanza dell' algoritmo da creare.
Rappresenta il componente Factory del design pattern$_{G}$ Factory.

Utilizzo

viene utilizzata in seguito alla ricezione di un signal$_G$ da parte dei controller che necessitano di utilizzare un oggetto di tipo *Romeo::Model::Core::Algorithms::AAlgorithm*.
A seconda dei parametri e nome dell' algoritmo di Cluster$_G$ passati, crea un oggetto rispetto ad un altro.

Attributi

- `- static algCreator: AlgCreator *`

  Descrizione: puntatore all'unica istanza della classe *AlgCreator* che verrà creata in modo *lazy* quando verrà richiesta la creazione dell'oggetto.

Metodi

- `- AlgCreator()`

  Descrizione: costruttore privato, come previsto dal design pattern$_G$ Singleton.

- `+ static getAlgCreator : AlgCreator *`

  Descrizione: metodo statico che ritorna il puntatore all'unica istanza della classe *AlgCreator*. Nel caso in cui l'istanza non esista ancora, essa verrà creata e successivamente ritornata.

  #### Note

  – Il metodo deve essere marcato come statico.

- `+ buildAlgorithm(name : const QString&, parameters : const QStringList& , algId : int) : AAlgorithm*`

  Descrizione: Metodo che ha il compito di costruire l'algoritmo specificato, con i relativi parametri.
  Per esempio se il parametro *name* è uguale a "KMeans" allora verrà creato un oggetto *KMeansAlgorithm* con i dati passati nei parametri.

  #### Argomenti

  – `name : const QString&`
  Stringa che rappresenta il nome dell'algoritmo di clustering$_G$ che si vuole creare;

  – `parameters : const QStringList&`
  Lista di stringhe che rappresenta i parametri da passare all'algoritmo di cluster$_G$ che si vuole creare;

  – `algId : int`
  Rappresenta l'id da associare all'algorimo che si vuole creare. Nel caso non venga specificato assumerà il valore di default di "-1" che indica che l'algorimo non è già presente nel database.

  #### Note

  – Il metodo deve essere marcato come costante.

- `+ getAllAlgorithms() : QVector<AAlgorithm *>`

Descrizione: metodo che ritorna tutte le possibili istanze di algoritmi di cluster$_{\mathbf{G}}$.

Note

– Il metodo deve essere marcato come costante.

4.1.2 Analysis (class)

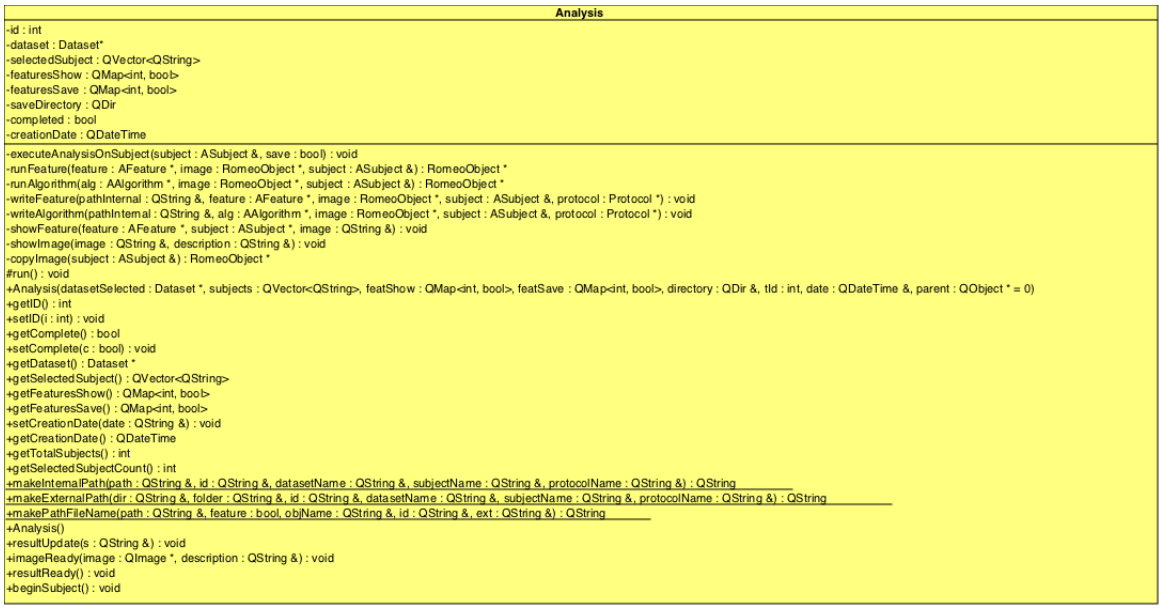

Figura 5: Diagramma classe *Analysis*

Descrizione

classe che rappresenta un' analisi con le relative proprietà: *Subject*$_G$ *da analizzare, Directory dei risultati, Feature*$_G$ *di cui salvare i risultati, Feature*$_G$ *di cui visualizzare i risultati e Data di creazione.*
Rappresenta un Thread.

Utilizzo

viene utilizzata dalla classe *AnalysisController*, alla ricezione di un signal$_G$ per l'avvio di una nuova analisi.

Eredita da:

- Qt::QThread.

Attributi

- `- id : int`

  Descrizione: rappresenta l'id associato all'analsi.

- `- dataset`

  Descrizione: puntatore all'oggetto *Dataset* associato all'analisi.

- `- selectedSubject : QVector<QString>`

  Descrizione: lista dei nomi dei Subject$_G$ selezionati dall'utente su qui l'utente vuole eseguire l'analisi.

- `- featuresShow : QMap<int, bool>`

Descrizione: mappa che rappresente le feature$_G$ che l'utente vuole visualizzare nel dialogo.
Ad ogni id delle feature è associato un `bool` che indica se il risultato della feature$_G$ deve essere visualizzata oppure no.

- `- featuresSave : QMap<int, bool>`

Descrizione: mappa che rappresente le feature$_G$ su cui l'utente vuole che siano esportati i risultati.
Ad ogni id delle feature è associato un `bool` che indica se il risultato della feature$_G$ deve essere esportato oppure no.

- `- saveDirectory : QDir`

Descrizione: cartella nel quale l'utente vuole che siano esportati i risultati.

- `- completed : bool`

Descrizione: indica se l'analisi è terminata oppure no.

- `- creationDate : QDateTime`

Descrizione: rappresenta la data in qui è stata avviata l'analisi.

Metodi

- `+ Analysis(datasetSelected : Dataset *, subjects : QVector<QString>, featShow: const QMap<int, bool>, featSave : const QMap<int, bool>, directory : const QDir&, tId : int, date : const QDateTime&, parent : QObject *);`

Descrizione: costruttore della classe.

Argomenti:

- `datasetSelected : Dataset *`
  Puntatore al *Dataset* associato all'analisi;
- `subjects : QVector<QString>`
  Lista dei nomi di Subject$_G$ su qui eseguire l'analisi;
- `featShow: const QMap<int, bool>`
  Mappa che associa ad un id di una feature$_G$ un `bool` che indica se la il risultato della feature$_G$ va visualizzato nel dialogo oppure no;
- `featSave : const QMap<int, bool>`
  Mappa che associa ad un id di uan feature$_G$ un `bool` che indica se il risultato della feature$_G$ va esportato oppure no;
- `directory : const QDir&`
  Directory nel quale salvare i risultati delle feature$_G$ e degli algoritmi di clustering$_G$;
- `tId : int`
  Id associato all'analisi. Di default vale "-1";
- `date : const QDateTime&`
  Data nel quale è stata creata l'analisi;
- `parent : QObject *`
  Parente dell'oggetto Analysis.

- `- executeAnalysisOnSubject(subject : ASubject&, save : bool) : void`

Descrizione: metodo che esegue l'intera analisi sul *Subject* passato.

Argomenti

- `subject : ASubject&`
  Subject su qui eseguire l'analisi;
- `save : bool`
  Booleano che indica se i risultati delle feature$_{\mathbf{G}}$ e degli algoritmi di clustering$_{\mathbf{G}}$ devono essere salvati o meno.

- `- runFeature(feature : AFeature *, image : RomeoObject *, subject : ASubject &) : RomeoObject *`

Descrizione: metodo che esegue una feature$_{\mathbf{G}}$ sul formato interno dell'immagine passato come parametro.
Ritorna un *RomeoObject* * rappresentante il formato interno dell'immagine di output.

Argomenti

- `feature : AFeature *`
  Feature da eseguire;
- `image : RomeoObject *`
  Formato interno su cui eseguire la feature$_{\mathbf{G}}$;
- `subject : ASubject &`
  Rappresenta il *Subject* su cui eseguire l'analisi.

- `- runAlgorithm(alg : AAlgorithm *, image : RomeoObject *, subject : ASubject &) : RomeoObject *`

Descrizione: metodo che esegue un algoritmo$_{\mathbf{G}}$ sul formato interno dell'immagine passato come parametro.
Ritorna un *RomeoObject* * rappresentante il formato interno dell'immagine di output.

Argomenti

- `alg : AAlgorithm *`
  Algoritmo da eseguire;
- `image : RomeoObject *`
  Formato interno su cui eseguire la feature$_{\mathbf{G}}$;
- `subject : ASubject &`
  Rappresenta il *Subject* su cui eseguire l'analisi.

- `- writeFeature(pathInternal : const QString&, feature : AFeature*, image : RomeoObject *,subject : ASubject &, protocol : Protocol *) : void`

Descrizione: metodo che esporta il risultato della feature$_{\mathbf{G}}$ nel filesystem.

Argomenti

- `pathInternal : const QString&`
  Path nel quale salvare il risultato della feature$_{\mathbf{G}}$;
- `feature : AFeature*`
  Rappresenta la feature$_{\mathbf{G}}$ che è appena stata eseguita;
- `image : RomeoObject *`
  Rappresenta il formato interno dell'imamgine che va esportata;
- `subject : ASubject &`
  Rappresenta il Subject$_{\mathbf{G}}$ sul quale è stata eseguita la feature$_{\mathbf{G}}$;
- `protocol : Protocol *`
  Rappresenta il Protocol$_{\mathbf{G}}$ al quale appartiene la Feature$_{\mathbf{G}}$ appena eseguita.

- `- writeAlgorithm(pathInternal : const QString&, alg : AAlgorithm*, image : RomeoObject *, subject : ASubject &, protocol : Protocol *) : void`

Descrizione: metodo che esporta il risultato dell'algoritmo di clustering$_{\mathbf{G}}$ nel filesystem.

Argomenti

- `pathInternal : const QString&`
  Path nel quale salvare il risultato dell'algoritmo$_{\mathbf{G}}$;
- `alg : AAlgorithm*`
  Rappresenta l'algoritmo di clustering$_{\mathbf{G}}$ che è appena stato eseguito;
- `image : RomeoObject *`
  Rappresenta il formato interno dell'imamgine che va esportata;
- `subject : ASubject &`
  Rappresenta il Subject$_{\mathbf{G}}$ sul quale è stato eseguito l'algorimo di clustering$_{\mathbf{G}}$;
- `protocol : Protocol *`
  Rappresenta il Protocol$_{\mathbf{G}}$ al quale appartiene l'algoritmo di clustering$_{\mathbf{G}}$ appena eseguito.

- `- showFeature(feature : AFeature *, subject : ASubject *, image : const QString &) : void`

Descrizione: metodo che si occupa di creare un oggetto `QImage` sul risultato di una feature$_{\mathbf{G}}$.

Argomenti

- `feature : AFeature *`
  Rappresenta la feature$_{\mathbf{G}}$ che è stata eseguita;
- `subject : ASubject *`
  Rappresenta il Subject$_{\mathbf{G}}$ su cui è stata eseguita la feature$_{\mathbf{G}}$;
- `image : const QString &`
  Percorso nel quale è presente il risultato della feature$_{\mathbf{G}}$ appena eseguita.

- `- showImage(image : const QString &, type : const QString &, description : const QString &) : void`

Descrizione: metodo che emette il signal$_{G}$ `imageReady` quando un algoritmo di clustering$_{G}$ o una feature$_{G}$ è terminata.

Argomenti

- `image : const QString &`
  Rappreenta il percorso nel quale si trova l'immagine risultato;
- `type : const QString &`
  Rappresenta il tipo di immagine;
- `description : const QString &)`
  Rappresenta la descrizione dell'immagine.

- `+ run() : void`

Descrizione: avvia il thread.

Note

- Il metodo deve essere marcato come virtuale.

- `+ getID() :int`

Descrizione: metodo che ritorna l'id associato all'istanza.

Note

- Il metodo deve essere marcato costante.

- `+ setID(i : int) : void`

Descrizione: cambia l'id associato all'oggetto *Analysis*.

Argomenti

- `i : int`
  Il nuovo id da associare.

- `+ getComplete() : bool`

Descrizione: ritorna un `bool` che indica se l'analisi è stata terminata o meno.

Note

- Il metodo deve essere marcato come costante.

- `+ setComplete(c : bool) : void`

Descrizione: metodo che cambia lo stato di completamento dell'analisi.

Argomenti

- `c : bool`
  Rappresenta il nuovo stato di completamento dell'analsi.

- `+ getDataset() : Dataset*`

Descrizione: metodo che ritorna l'oggetto *Dataset* associato all'analisi.

Note

- Il metodo deve essere marcato come costante.

- `+ getSelectedSubject() : QVector<QString>`

Descrizione: metodo che ritorna la lista dei Subject$_G$ su cui eseguire l'analisi.

Note

- Il metodo deve essre marcato come costante.

- `+ getFeaturesShow() : QMap<int, bool>`

Descrizione: metodo che ritorna una `QMap` che associa all'id di una feature un `bool` che indica se il risultato della feature$_G$ va visualizzati nel dialogo oppure no.

Note

- Il metodo deve essere marcato come costante.

- `+ getFeaturesSave() :QMap<int, bool>`

Descrizione: metodo che ritorna una `QMap` che associa all'id di una feature un `bool` che indica se il risultato della feature$_G$ va esportato oppure no.

Note

- Il metodo deve essere marcato come costante.

- `+ setCreationDate(date : const QString &) : void`

Descrizione: metodo che cambia la data di creazione di un oggetto *Analysis*.

Argomenti

- `date : const QString &`
  Nuova data di creazione.

- `+ getCreationDate() : QDateTime`

Descrizione: metodo che ritorna la data di creazione di un oggetto *Analysis*.

Note

- Il metodo va marcato come costante.

- `+ getTotalSubjects() :int`

Descrizione: metodo che ritorna il numero di Subject$_{\mathbf{G}}$ presenti nel Gruppo di Subject associati al Dataset$_{\mathbf{G}}$ dell'analisi.

Note

- Il metodo deve essere marcato costante.

- `+ getSelectedSubjectCount() : int`

Descrizione: metodo che ritorna il numero di Subject$_{\mathbf{G}}$ su cui eseguire l'analisi.

Note

- Il metodo deve essere marcato costante.

- `+ resultUpdate(s : const QString &) : void (signal)`

Descrizione: signal$_{\mathbf{G}}$ emesso quando va aggiornata la descrizione dell'oggetto *AnalysisDialog*.

Argomenti

- `s : const QString &`
  Nuovo testo da visualizzare nel dialogo.

- `+ imageReady(image : QImage *, description : const QString &) : void (signal)`

Descrizione: signal$_{\mathbf{G}}$ emesso quando una nuova immagine è disponibile.

Argomenti

- `image : QImage *`
  Nuova immagine disponibile;
- `description : const QString &`
  Descrizione associata all'immagine.

- `+ resultReady() : void (signal)`

Descrizione: signal$_{\mathbf{G}}$ emesso quando l'esecuzione dell'analisi è terminata.

- `+beginSubject() : void (signal)`

Descrizione: signal$_{\mathbf{G}}$ emesso quando l'analisi viene effettuata su un nuovo Subject$_{\mathbf{G}}$.

4.1.3 ISubject (interface)

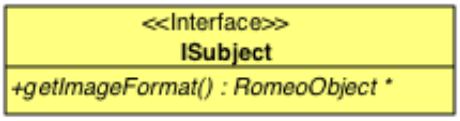

Figura 6: Diagramma classe *ISubject*

Descrizione

Definisce l'interfaccia comune per le classi *ASubject* e *RealSubjec*, fornisce un metodo per ottenere il formato interno per l'immagine del Subject$_{\mathbf{G}}$.
Rappresenta il componente Subject del design pattern$_{\mathbf{G}}$ Proxy.

Utilizzo

Viene utilizzata quando le componenti di Romeo$_{\mathbf{G}}$ necessitano di riferirsi a un Subject$_{\mathbf{G}}$ per ottenere il formato interno ad esso associato.

Metodi

- `+ getImageFormat() : RomeoObject *`

  Descrizione: contratto che restituisce un puntatore al formato interno dell'immagine di un Subject$_{\mathbf{G}}$

  Note

  – Il metodo deve essere marcato come virtuale.

4.1.4 ASubject (abstract)

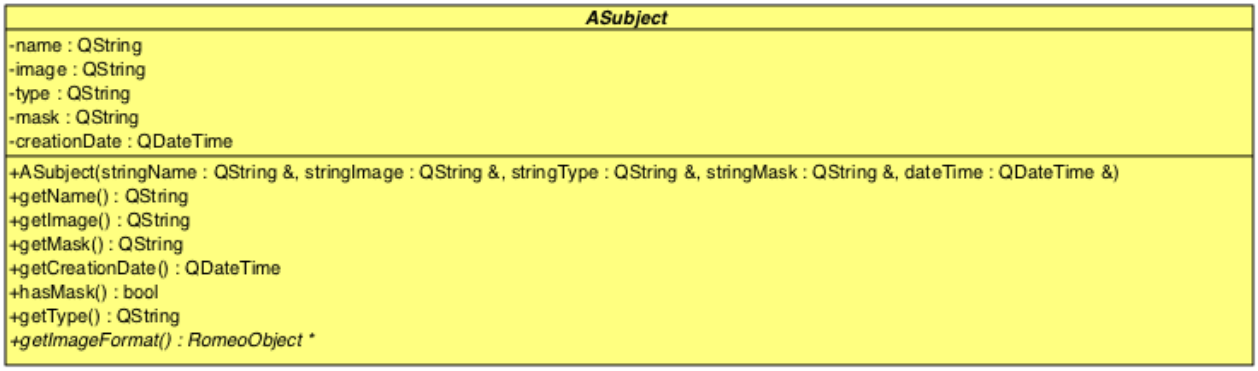

Figura 7: Diagramma classe *ASubject*

Descrizione

Classe astratta che rappresenta un generico Subject$_{\mathbf{G}}$ con le relative proprietà *Nome, Tipo del Subject$_{\mathbf{G}}$, Immagine, Maschera e Data di creazione.*

Utilizzo

Viene utilizzata in seguito alla ricezione di un signal$_{\mathbf{G}}$ da parte dei controller che necessitano di riferirsi ad uno o più oggetti di tipo *ASubject.*
Inoltre viene utilizzata dalle classi DAO e dalla classe *GroupOfSubject.*

Eredita da:

- Romeo::Model::Core::ISubject: non implementa ancora i contratti di questa classe.

Attributi

- `- name : QString`

  Descrizione: nome del Subject$_{\mathbf{G}}$.
- `- image : QString`

  Descrizione: stringa rappresentante il percorso assoluto dell'immagine del Subject$_{\mathbf{G}}$.
- `- type : QString`

  Descrizione: stringa rappresentante il tipo di un Subject$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t).
- `- mask : QString`

  Descrizione stringa rappresentante il percorso della maschera del Subject$_{\mathbf{G}}$.
- `- creationDate : QDateTime`

  Descrizione: data di creazione del Subject$_{\mathbf{G}}$

Metodi

- `+ ASubject(stringName : const QString &, stringImage : const QString &, stringType : const QString &, stringMask : const QString &, dateTime : const QDateTime &)`

  Descrizione: costruttore della classe.

  Argomenti

  - `stringName : const QString &`
    Nome del Subject$_{\mathbf{G}}$.
  - `stringImage : const QString &`
    Percorso dell'immagine del Subject$_{\mathbf{G}}$;
  - `stringType : const QString &`
    Stringa che rappresenta il tipo del Subject$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t).
  - `stringMask : const QString &`
    Stringa rappresentante il percorso della maschera del Subject$_{\mathbf{G}}$.
  - `dateTime : const QDateTime &`
    Data di creazione del Subject$_{\mathbf{G}}$.

- `+ getName() : QString`

  Descrizione: metodo che ritorna una stringa, contenente il nome del Subject$_{\mathbf{G}}$.

  Note

  - Il metodo deve essere marcato costante.

- `+ getImage() : QString`

  Descrizione: metodo che ritorna una stringa, contenete il percorso assoluto dell'immagine associata al Subject$_{\mathbf{G}}$.

  Note

  - Il metodo deve essere marcato costante.

- `+ getMask() : QString`

  Descrizione: metodo che ritorna una stringa, contente il percorso assoluto della (eventuale) maschera associata al Subject$_{\mathbf{G}}$.

  Note

  - Il metodo deve essere marcato costante.

- `+ getCreationDate() : QDateTime`

  Descrizione: metodo che ritorna la data di creazione del Subject$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato costante.

- `+ hasMask() : bool`

  Descrizione: metodo che controlla se il Subject$_{\mathbf{G}}$ contiene una maschera$_{\mathbf{G}}$ oppure no.
  Ritorna `true` se il Subject$_{\mathbf{G}}$ possiede una maschera, `false` viceversa.

  Note

  - Il metodo deve essere marcato costante.

- `+ getType() : QString`

  Descrizione Metodo che ritorna una stringa, contente il tipo del Subject$_{\mathbf{G}}$ (2D, 2-t, 3D, 3D-t).

  Note

  - Il metodo deve essere marcato const.

4.1.5 ProxySubject (class)

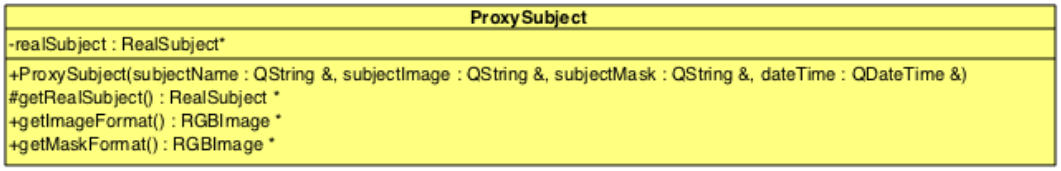

Figura 8: Diagramma classe *ProxySubject*

Descrizione

Classe che gestisce l'accesso ad un *RealSubject*, mantiene un riferimento che consente al proxy di accedere all'oggetto rappresentato *RealSubject*.
Fornisce la stessa interfaaccia della classe *ASubject*, consentendo di utilizzare un oggetto *ProxySubject* quando è richiesto un oggetto *ASubject*.
Rappresenta il componente Proxy del design pattern$_{\mathbf{G}}$ Proxy.

Utilizzo

Viene utilizzando quando è necessario creare un Subject$_{\mathbf{G}}$ all'interno di Romeo$_{\mathbf{G}}$.

Eredita da:

- Romeo::Model::Core::ISubject: implementa i contratti di questa classe.

Attributi

- `- realSubject : RealSubject *`

  Descrizione: puntatore ad un oggetto di tipo *RealSubject* che il Proxy sta rappresentando.

Metodi

- `+ ProxySubject(subjectName : const QString&, subjectImage : const QString&, maskName : const QString&, dateTime : const QDateTime&)`

  Descrizione: costruttore della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `subjectName : const QString&`
    Nome del Subject$_{\mathbf{G}}$. Viene passato al costruttore della superclasse;
  - `subjectImage : const QString&`
    Immagine del Subject$_{\mathbf{G}}$. Viene passato al costruttore della superclasse;
  - `maskName : const QString&`
    Maschera$_{\mathbf{G}}$ del Subject$_{\mathbf{G}}$. Viene passato al costruttore della superclasse;
  - `dateTime : QDateTime &`
    Data di creazione del Subject$_{\mathbf{G}}$. Viene passato al costruttore della superclasse.

- `# getRealSubject() : RealSubject *`

Descrizione: restituisce il puntatore al *RealSubject* rappresentato. Nel caso non esista viene creato e successivmente ritornato.

- `+ getImageFormat() : RomeoObject *`

Descrizione: Implementa il contratto fornito dalla superclasse. Si limita a chiamare il metodo `getImageFormat()` sull'oggetto *RealSubject*.

Note

– Il metodo deve essere marcato virtuale.

4.1.6 RealSubject (abstract)

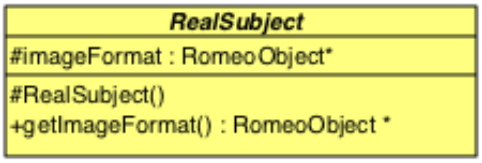

Figura 9: Diagramma classe *RealSubject*

Descrizione

classe astratta che caratterizza l'oggetto rappresentato dal *ProxySubject*, rappresenta un oggetto reale utilizzato dalla classe *ProxySubject*.
Contiene la proprietà *imageFormat*.
Le sottoclassi rappresentano i componenti RealSubject del design pattern$_{\mathbf{G}}$ Proxy.

Utilizzo

viene utilizzato dalla classe *ProxySubject*, la quale contiene un riferimento al *RealSubject* che sta gestendo.

Eredita da:

- Romeo::Model::Core::ISubject: implementa i contratti di questa classe.

Attributi

- `# imageFormat : RomeoObject *`

  Descrizione: puntatore al formato interno utilizzato per rappresentare l'immagine associata al Subject$_{\mathbf{G}}$.

Metodi

- `# RealSubject()`

  Descrizione: costruttore della classe. Il costruttore è stato reso protetto in quanto la classe è ancora astratta.

4.1.7 RealSubjectImage (class)

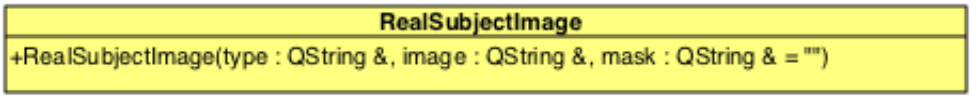

Figura 10: Diagramma classe *RealSubjectImage*

Descrizione
Classe concreta che rappresenta un Subject$_{\mathbf{G}}$ nel formato bidimensionale (2D, 3D).

Utilizzo
Viene utilizzata dalla classe *ProxySubject* per ottenere l'oggetto di tipo *RomeoObject* che rappresenta l'immagine, quando il *ProxySubject* sta rappresentando un Subject$_{\mathbf{G}}$ 2D o 3D.

Eredita da:

- Romeo::Model::Core::RealSubject.

Metodi

- `+ RealSubjectImage(image : const QString &, mask : const QString &)`

  Descrizione: costruttore della classe. Richiama il costruttore della supeclasse. Rappresenta il componente RealSubject del design pattern$_{\mathbf{G}}$ Proxy.

  Argomenti

  - `image : const QString &`
    Stringa che rappresenta il path nel quale è presente l'immagine del Subject$_{\mathbf{G}}$;
  - `mask : const QString &)`
    Stringa che rappresenta il path nel quale è presente la maschera del Subject$_{\mathbf{G}}$. Nel caso in qui il Subject$_{\mathbf{G}}$ non abbia alcuna maschera l'argomento `mask` assumerà il valore di default ".

4.1.8 RealSubjectVideo (class)

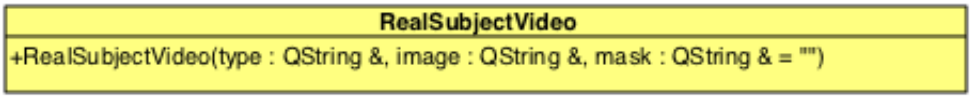

Figura 11: Diagramma classe *RealSubjectVideo*

Descrizione

Classe concreta che rappresenta un Subject$_{\mathbf{G}}$ di tipo 2D-t o 3D-t.
Rappresenta il componente RealSubject del design pattern$_{\mathbf{G}}$ proxy.

Utilizzo

Viene utilizzata dalla classe *ProxySubject* per ottenere l'oggetto di tipo *RomeoObject* che rappresenta l'immagine, quando il *ProxySubject* sta rappresentando un Subject$_{\mathbf{G}}$ 2D-t o 3D-t.

Eredita da:

- Romeo::Model::Core::RealSubject.

Metodi

- `+ RealSubjectVideo(image : const QString &, mask : const QString &)`

  Descrizione: costruttore della classe. Richiama il costruttore della supeclasse. Rappresenta il componente RealSubject del design pattern$_{\mathbf{G}}$ Proxy.

  Argomenti

  - `image : const QString &`
    Stringa che rappresenta il path nel quale è presente l'immagine del Subject$_{\mathbf{G}}$;
  - `mask : const QString &)`
    Stringa che rappresenta il path nel quale è presente la maschera del Subject$_{\mathbf{G}}$. Nel caso in qui il Subject$_{\mathbf{G}}$ non abbia alcuna maschera l'argomento `mask` assumerà il valore di default ".

4.1.9 GroupOfSubject (class)

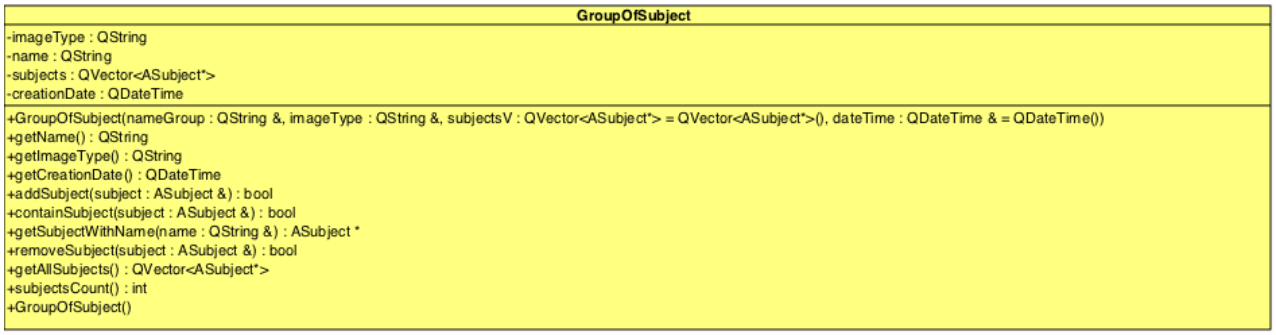

Figura 12: Diagramma classe *GroupOfSubject*

Descrizione

Classe che rappresenta un gruppo di Subject$_{\mathbf{G}}$ con le relative propietà: *Nome del gruppo, tipo del gruppo, data di creazione e la lista di Subject$_{\mathbf{G}}$ presenti nel gruppo.*

Utilizzo

Viene utilizzata in seguito alla ricezione di un signal$_{\mathbf{G}}$ da parte dei controller che necessitano di riferirsi ad uno o più gruppi di Subject$_{\mathbf{G}}$
Inoltre viene utilizzata dalla classe *Dataset*, che contiene un riferimento al GroupOfSubject$_{\mathbf{G}}$ associato e dalle classi DAO.

Attributi

- `- imageType : QString`

  Descrizione: stringa rappresentante il tipo del gruppo.

- `- name : QString`

  Descrizione: nome del gruppo.

- `- subjects : QVector<ASubject*>`

  Descrizione: vettore di puntatori ad *ASubject* contenente i Subject$_{\mathbf{G}}$ facenti parte del gruppo.

- `- creationDate : QDateTime`

  Descrizione: data di creazione del gruppo.

Metodi

- `+ GroupOfSubject(nameGroup : const QString&, imageType : const QString&, subjectsV : QVector<ASubject*> , dateTime : const QDateTime&)`

  Descrizione: costruttore della classe *GroupOfSubjcect*.

Argomenti

- `nameGroup : const QString&`
  Nome del Gruppo di Subject$_{\mathbf{G}}$;
- `imageType : const QString&`
  Tipo del Gruppo di Subject$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t);
- `subjectsV : QVector<ASubject*>`
  Subject$_{\mathbf{G}}$ presenti nel gruppo;
- `dateTime : const QDateTime&`
  Data di creazione del Subject$_{\mathbf{G}}$.

- `+ getName() : QString`

Descrizione: ritorna una stringa contenente il nome del gruppo.

Note

- Il metodo deve essere marcato come costante.

- `+ getImageType() : QString`

Descrizione: ritorna una stringa contenente il percorso del gruppo.

Note

- Deve essere marcato costante.

- `+ getCreationDate() : QDateTime`

Descrizione: ritorna la data di craezione del gruppo di Subjcect.

Note

- Deve essere marcato come costante.

- `+ addSubject( subject : const ASubject& ) : void`

Descrizione: aggiunge un Subject$_{\mathbf{G}}$ al vettore dei Subject$_{\mathbf{G}}$ del gruppo.

Argomenti

- `subject : const ASubject &`
  Subject$_{\mathbf{G}}$ da inserire.

- `+ containsSubject( subject : const ASubject& ) : boolean`

Descrizione: ritorna un booleano per indicare se il gruppo contiene o meno un determinato Subject$_{\mathbf{G}}$.

Argomenti

- `subject : const ASubject &`
  Subject$_G$ da cercare.

- `+ getSubjectWithName(name : const QString &) : ASubject *`

Descrizione: metodo che ritorna un puntatore al Subject$_G$ avente il nome richiesto.

Argomenti

- `name : const QString &`
  Nome del Subject$_G$ da cercare.

Note

- Il metodo deve essre marcato costante

- `+ removeSubject( subject : const ASubject& ) : boolean`

Descrizione: rimuove un Subject$_G$ dal vettore dei Subject$_G$ del gruppo. Ritorna un booleano per indicare se l'operazione è andata a buon fine o meno.

Argomenti

- `subject : const ASubject &`
  Subject$_G$ da rimuovere.

- `+ getAllSubject() : QVector<ASubject*>`

Descrizione: ritorna un vettore contenente i puntatori a tutti i Subject$_G$ del gruppo.

Note

- Il metodo deve essere marcato come costante.

- `+ subjectsCount() : int`

Descrizione: metodo che ritorna il number di subject nel gruppo.

Note

- Il metodo deve essere marcato costante.

4.1.10 Protocol (class)

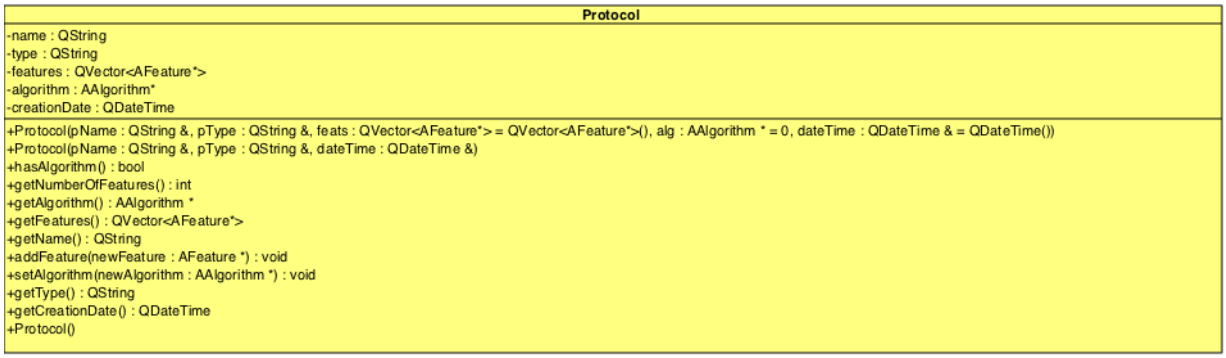

Figura 13: Diagramma classe *Protocol*

Descrizione

Classe che rappresenta un Protocol$_G$ con le relative propietà: *Nome, Tipo, Data di creazione, Lista di feature$_G$ e Algoritmo di cluster$_G$*.

Utilizzo

viene utilizzata in seguito alla ricezione di un signal$_G$ da parte dei controller che necessitano di riferirsi ad uno o più Protocol$_G$.
Inoltre viene utilizzata dalla classe *Dataset*, che contiene un riferimento ai Protocol$_G$ associati, e dalle classi DAO.

Attributi

- `- name : QString`

  Descrizione: nome del Protocol$_G$.

- `-type : QString`

  Descrizione: il tipo del Protocol$_G$ (2D, 2D-t, 3D, 3D-t).

- `- features : QVector<AFeature*>`

  Descrizione: vettore di puntatori polimorfi ad oggetti *AFeature* contenente le feature$_G$ del Protocol$_G$.

- `- algorithm : AAlgorithm*`

  Descrizione: puntatore polimorfo all'algoritmo di clustering$_G$. Nel caso in cui non ci sia alcun algoritmi, `algorithm` varrà 0.

- `- creationDate : QDateTime`

  Descrizione: data di creazione del Protocol$_G$.

Metodi

- `+ Protocol(pName : const QString&, pType : const QString& , feats : QVector<AFeature*>, alg : AAlgorithm *, dateTime : const QDateTime& )`

  Descrizione: costruttore della classe *Protocol*.

  Argomenti

  - `pName : const QString&`
    Nome del Protocol$_{\mathbf{G}}$;
  - `pType : const QString&`
    Tipo del Protocol$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t);
  - `feats : QVector<AFeature*>`
    Elenco di feature$_{\mathbf{G}}$ presenti nel Protocol$_{\mathbf{G}}$;
  - `alg : AAlgorithm *`
    Algoritmo di clustering$_{\mathbf{G}}$ presente nel Protocol$_{\mathbf{G}}$;
  - `dateTime : const QDateTime&`
    Data di creazione del Protocol$_{\mathbf{G}}$.

- `+ Protocol(pName : const QString&, pType : const QString&, dateTime : const QDateTime&)`

  Descrizione: costuttore utilizzato per creare un Protocol$_{\mathbf{G}}$ vuoto.

  Argomenti

  - `pName : const QString&`
    Nome del Protocol$_{\mathbf{G}}$;
  - `pType : const QString&`
    Tipo del Protocol$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t).

- `+ hasAlgorithm() : boolean`

  Descrizione: metodo che controlla se il Protocol$_{\mathbf{G}}$ ha un algoritmo di clustering$_{\mathbf{G}}$ o meno.

  Note

  - Il metodo deve essere marcato come costante.

- `+ getNumberOfFeatures(): int`

  Descrizione: metodo che ritorna il numero di feature$_{\mathbf{G}}$ presenti nel Protocol$_{\mathbf{G}}$.

  Note

  - Il metodo deve essere marcato come costante.

- `+ getAlgorithm() : AAlgorithm*`

Descrizione: ritorna il puntatore all'algoritmo di clustering$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$. S il Protocol$_{\mathbf{G}}$ non contiene alcun algoritmo di clsutering$_{\mathbf{G}}$ viene ritornato 0.

Note

- Il metodo deve essere marcato come costante.

- `+ getFeatures() : QVector<AFeature*>`

Descrizione: ritorna un vettore di puntatori alle feature$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getName() : QString`

Descrizione: ritorna una stringa contenente il nome del Protocol$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ addFeature(newFeature : AFeature *) : void`

Descrizione: aggiunge una nuova feature$_{\mathbf{G}}$ al Protocol$_{\mathbf{G}}$.

Argomenti

- `newFeature : AFeature *`
  Puntatore alla feature$_{\mathbf{G}}$ da aggiungere al Protocol$_{\mathbf{G}}$.

- `+ setAlgorithm(newAlgorithm : AAlgorithm *) : void`

Descrizione: assegna un nuovo algoritmo di clustering$_{\mathbf{G}}$ al Protocol$_{\mathbf{G}}$.

Argomenti

- `newAlgorithm : AAlgorithm *`
  Puntatore al nuovo algoritmo di clustering$_{\mathbf{G}}$ da assegnare al Protocol$_{\mathbf{G}}$.

- `+ getType() : QString`

Descrizione: ritorna una stringa contenente il tipo del Protocol$_{\mathbf{G}}$ (2D, 2D-t, 3D, 3D-t).

Note

- Deve essere marcato come costante.

- `+ getCreationDate() :QDateTime`

Descrizione: metodo che ritorna la data di creazione del Protool$_{\mathbf{G}}$.

Note

– Il metodo deve essere marcato come costante.

4.1.11 Dataset (class)

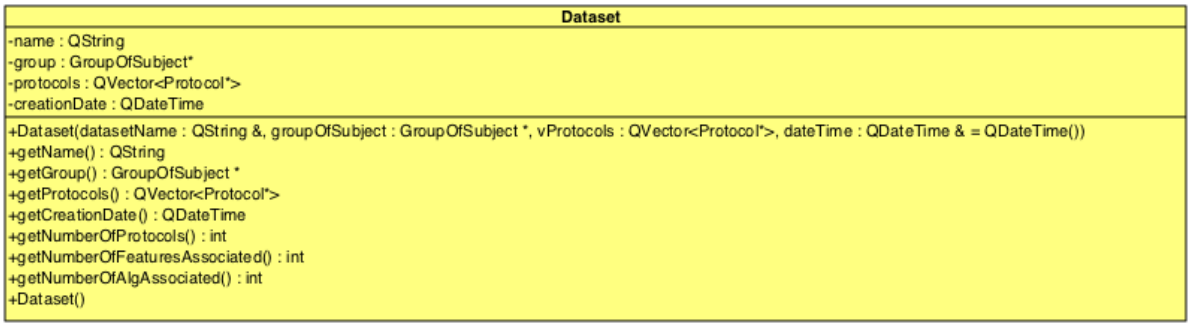

Figura 14: Diagramma classe *Dataset*

Descrizione

Classe che rappresenta un Dataset$_{\mathbf{G}}$ con le relative proprietà: *Nome*, *Tipo*, *Data di creazione*, *Gruppo di Subject*$_{\boldsymbol{G}}$ e *Protocol*$_{\boldsymbol{G}}$.

Utilizzo

Viene utilizzata in seguito alla ricezione di un signal$_{\mathbf{G}}$ da parte dei controller che necessitano di riferirsi ad uno o più Dataset$_{\mathbf{G}}$.
Inoltre viene utilizzata dalla classe *Analysis*, che contiene un riferimento al Dataset$_{\mathbf{G}}$ associato all'analisi, e dalle classi DAO.

Attributi

- `- name : QString`

  Descrizione: nome del Dataset$_{\mathbf{G}}$.

- `- group : GroupOfSubject *`

  Descrizione: puntatore al gruppo di Subject$_{\mathbf{G}}$ associato al Dataset$_{\mathbf{G}}$.

- `- protocol : QVector<Protocol*>`

  Descrizione: vettore di puntatori a oggetti di tipo *Protocol*, contenente i vari Protocol$_{\mathbf{G}}$ presenti nel Dataset$_{\mathbf{G}}$.

- `-creationDate : QDateTime`

  Descrizione: data di creazione del Dataset$_{\mathbf{G}}$.

Metodi

- `+ Dataset(datasetName : const QString&, groupOfSubject : GroupOfSubject *, vProtocols : QVector<Protocol*> , dateTime : const QDateTime&)`

  Descrizione: costruttore della classe *Dataset*.

Argomenti

- `datasetName : const QString&`
  Nome del Dataset$_{\mathbf{G}}$;
- `groupOfSubject : GroupOfSubject *`
  Gruppo di Subject$_{\mathbf{G}}$ associato al Dataset$_{\mathbf{G}}$;
- `vProtocols : QVector<Protocol*>`
  Vettore di puntatori ad oggetti *Protocol* presenti nel Dataset$_{\mathbf{G}}$;
- `dateTime : const QDateTime&`
  Data di creazione del Dataset$_{\mathbf{G}}$.

- `+ getName() : QString`

Descrizione: ritorna una stringa contenente il nome del Dataset$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getGroup() : GroupOfSubject *`

Descrizione: ritorna un puntatore al *GroupOfSubject* del Dataseet$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getProtocols() : QVector<Protocol*>`

Descrizione: metodo che ritorna un vettore contenente i *Protocol* presenti nell'oggetto *Dataset*.

Note

- Il metodo deve essere marcato come costante.

- `+ getCreationDate() : QDateTime`

Descrizione: metodo che ritorna la data di creazione del Dataset$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfProtocols() : int`

Descrizione: metodo che ritorna il numero di Protocol$_{\mathbf{G}}$ nel Dataset$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfFeaturesAssociated() : int`

Descrizione: metodo che ritorna il numero totale di $\text{feature}_{\mathbf{G}}$ presenti nel $\text{Dataset}_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfAlgAssociated() : int`

Descrizione: metodo che ritorna il numero totale di $\text{algoritmi di clustering}_{\mathbf{G}}$ presenti nel $\text{Dataset}_{\mathbf{G}}$.

Note

- il metodo deve essere marcato come costante.

4.1.12 FeatCreator (class)

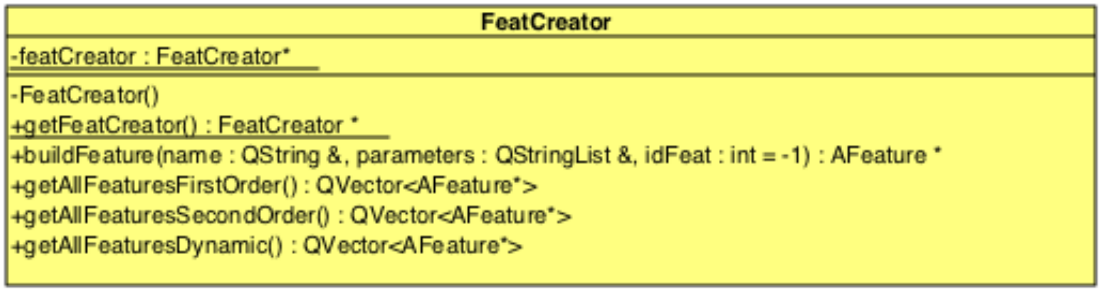

Figura 15: Diagramma classe *FeatCreator*

Descrizione

Classe Factory avente la responsibilità di creare un oggetto della classe Romeo::Model::Core::Features::AFeature, che rappresenta un'istanza della feature$_G$ da creare.
Rappresenta il componente Factory del design pattern$_G$ Factory.
A seconda dei parametri e nome della Feature$_G$ passati, crea un oggetto rispetto ad un altro.

Utilizzo

Viene utilizzata in seguito alla ricezione di un signal$_G$ da parte dei controller che necessitano di utilizzare un oggetto di tipo *Romeo::Model::Core::Algorithms::AFearure*.
A seconda dei parametri e nome della feature$_G$ passati, crea un oggetto rispetto ad un altro.

Attributi

- `- static featCreator: FeatCreator *`

  Descrizione: puntatore all'unica istanza della classe *FeatCreator* che verrà creata in modo *lazy* quando verrà richiesta la creazione dell'oggetto.

Metodi

- `+ FeatCreator()`

  Descrizione: costruttore della classe *FeatCreator*.

- `+ static getFeatCreator() : FeatCreator *`

  Descrizione: metodo statico che ritorna il puntatore all'unica istanza della classe *FeatCreator*. Nel caso in cui l'istanza non esista ancora, essa verrà creata e successivamente ritornata.

  Note

  - Il metodo deve essere marcato come statico.

- `+ buildFeature(name : const QString &, parameters : QStringList & , idFeat : int) : AFeature*`

Descrizione: Metodo che ha il compito di costruire la feature$_G$ specificato, con i relativi parametri.
Per esempio se il parametro *name* è uguale a "Mean" allora verrà creato un oggetto *MeanFeature* con i dati passati nei parametri.

Argomenti

- `name : const QString&`
  Stringa che rappresenta il nome della feature$_G$ che si vuole creare;
- `parameters : const QStringList&`
  Lista di stringhe che rappresenta i parametri da passare alla feature$_G$ che si vuole creare;
- `idFeat : int`
  Rappresenta l'id da associare alla feature$_G$ che si vuole creare. Nel caso non venga specificato assumerà il valore di default di "-1" che indica che la feature$_G$ non è già presente nel database.

Note

- Il metodo deve essere marcato come costante.

- `+ getAllFeatureFirstOrder() : QVector<AFeature *>`

Descrizione: metodo che ha il compito di ritornare tutte le feature$_G$ del primo ordine esistenti.

Note

- Il metodo deve essere marcato come costante.

- `+ getAllFeatureSecondOrder() : QVector<AFeature *>`

Descrizione: metodo che che ha il compito di ritornare tutte le feature$_G$ del secondo ordine esistenti.

Note

- Il metodo deve essere marcato come costante.

- `+ getAllFirsDynamicFeature() : QVector<AFeature *>`

Descrizione: metodo che ha il compito di ritornare tutte le feature$_G$ del dinamiche esistenti.

Note

- Il metodo deve essere marcato come costante.

4.1.13 InternalData (interface)

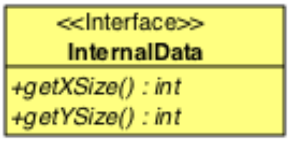

Figura 16: Diagramma classe *InternalData*

Descrizione Definisce l'interfaccia per accedere alle informazioni sulle dimensioni di un'immagine.
Rappresenta il componente Target del design pattern$_{G}$ Adapter.

Utilizzo
Viene utilizzata all'interno delle feature, ogni qualvolta ci si aspetta un' immagine generica.

Metodi

- `+ getXSize() : int`

  Descrizione: contratto che ritorna la dimensione orizzantale di un'immagine generica.

  Note

  – Il metodo va marcato come costante.

- `+ getYSize() : int`

  Descrizione: contratto che ritorna la dimensione verticale di un'immagine generica.

  Note

  – Il metodo va marcato come costante.

4.1.14 InternalData2D (class)

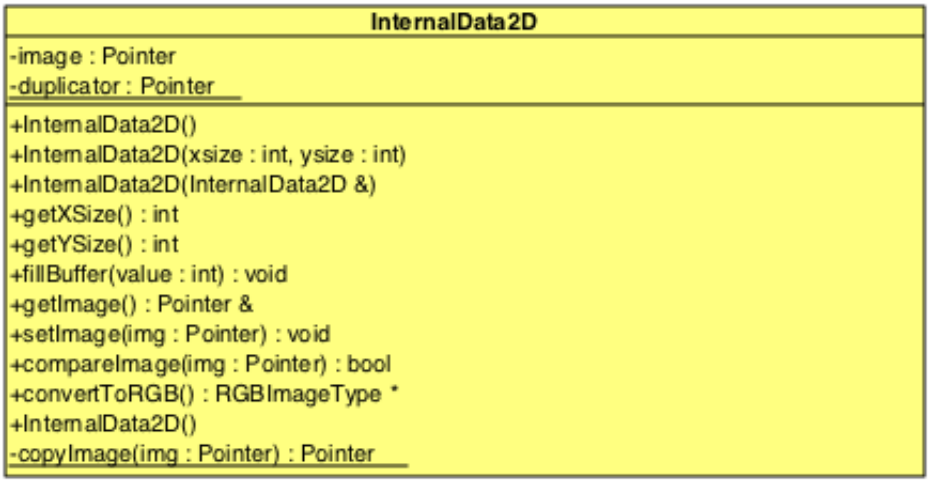

Figura 17: Diagramma classe *InternalData2D*

Descrizione

Classe che implementa i contratti definiti da InternalData e rappresenta il formato interno bidimensionale sul quale operare. Implementa il design pattern Adapter e adatta la classe itk::Image della libreria ITK$_{\mathbf{G}}$. Rappresenta la componente Adapter dell'omonimo design pattern$_{\mathbf{G}}$.

Utilizzo

Viene utlizzata nelle feature$_{\mathbf{G}}$ come parametro di input da elaborare e rappresenta i signoli canali nelle immagini in formato RGB.

Eredita da:

- Romeo::Model::Core::InternalData: implementa i contratti di questa classe.

Attributi

- `- image : ImageType2D::Pointer`

  Descrizione: puntatore all'immagine bidimensionale (Tipo ITK$_{\mathbf{G}}$).

- `- static duplicator : DuplicatorType::Pointer`

  Desrizione: "duplicatore" delle immagini *InternalData2D*.

Metodi

- `+ InternalData2D()`

  Descrizione: costruttore di *default* della lcasse *InternalData*.

- `+ InternalData2D(xSize : int, ySize : int)`

Descrizione: costruttore a due argomenti. Costruisce un immagine di dimensioni *xSize x ySize*.

Argomenti

- `xSize : int`
  Dimensione orizzontale dell'immagine.
- `ySize : int`
  Dimensione verticale dell'immagine.

- `+ InternalData2D(image : const InternalData &)`

Descrizione: costruttore di copia *profondo*.

Argomenti

- `image : const InternalData &`
  Oggetto *InternalData2D* di cui si vuole fare la copia *profonda*.

- `+ operator=(image : const InternalData2D &) : InternalData2D&`

Descrizione: operatore di assegnazione *profondo*.

Argomenti

- `image : const InternalData &`
  Oggetto *InternalData2D* di cui si vuole fare la copia *profonda*.

- `+ getXSize() : int`

Descrizione: implementa il contratto che ritorna la dimensione orizzontale di un'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ getYSize() : int`

Descrizione: implementa il contratto che ritorna la dimensione verticale di un'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ fillBuffer(value : int) : void`

Descrizione: metodo che "riempie" tutti i piexel del colore passato come argomenti.

Argomenti

- `value : int`
  Rappresenta il colore dell'immagine (0-255).

- `+ getImage() : ImageType::Pointer&`

Descrizione: ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine rappresentata.

- `+ setImage(img : ImageType::Pointer) : void`

Descrizione: metodo che cambia l'immagine rappresentata dall'oggetto *InternalData2D*.

Argomenti

- `img : ImageType::Pointer`
  Nuova immagine da rappresentare.

- `+ compareImage(img : ImageType::Pointer) : boolean`

Descrizione: metodo che controlla se due immagini sono uguali.

Argomenti

- `img : ImageType::Pointer`
  Immagine da confrontare.

- `+ convertToRGB() : RGBImageType *`

Descrizione: metodo che converte l'oggetto *InternalData2D* in un oggetto di tipo *RGBImageType*.

Argomenti

- Il metodo deve essere marcato costante.

- `- static copyImage(image : ImageType2D::Pointer) : ImageType2D::Pointer`

Descrizione: copia un immagine restituendone la copia.

Argomenti

- `image : ImageType2D::Pointer`
  Puntatore di tipo ITK$_{\mathbf{G}}$ all'immagine da copiare (tipo ITK$_{\mathbf{G}}$).

Note

- Il metodo deve essere marcato come statico.

4.1.15 InternalData3D (class)

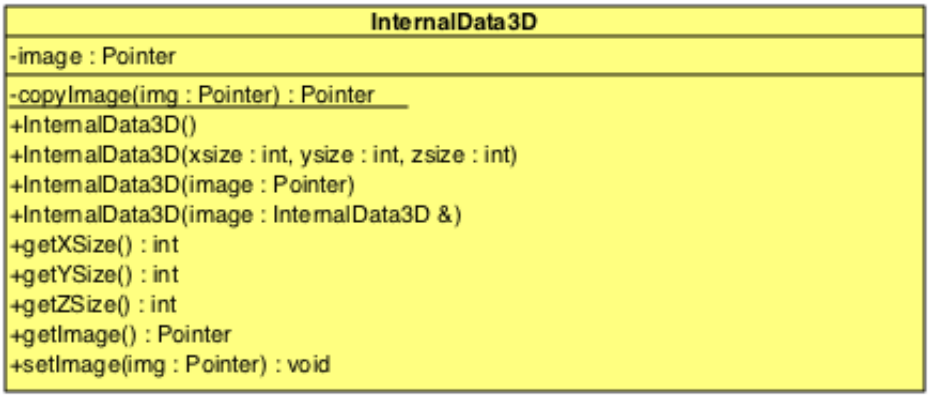

Figura 18: Diagramma Classe *InternalData3D*

Descrizione

Classe che rappresenta il formato interno per un'immagine di tipo 3D.
Classe che "adatta" la classe itk::image fornita dalla libreria esterna ITK$_{\mathbf{G}}$.
Rappresenta il componente Adapter del design pattern$_{\mathbf{G}}$ Adapter.

Utilizzo

Viene utlizzata nelle feature$_{\mathbf{G}}$ come parametro di input da elaborare e rappresenta i signoli canali nelle immagini in formato RGB.

Eredita da:

- InternalData: implementa contratti di questa classe.

Attributi

- `- image : ImageType3D::Pointer`

  Descrizione: puntatore all'immagine tridimensionale (Tipo ITK$_{\mathbf{G}}$).

Metodi

- `+ InternalData3D()`

  Descrizione: costruttore di *default* della classe *InternalData3D*.

- `+ InternalData3D(xSize : int, ySize : int, zSize : int)`

  Descrizione: costruttore a tre argomenti. Costruisce un immagine tridimensionale *xSize x ySize x zSize*.

Argomenti

- `xSize : int`
  Dimensione orizzontale;
- `ySize : int`
  Dimensione verticale;
- `zSize : int`
  Terza dimensione.

- `+ InternalData3D(image : RGBImageType::Pointer )`

Descrizione: costruttore che costriuisce un oggetto *InternalData3D*.

Argomenti

- `image : RGBImageType::Pointer`
  Puntatore all'immagine da creare.

- `InternalData3D(image : const InternalData3D &)`

Descrizione: costruttore di copia *profondo*.

Argomenti

- `image : const InternalData3D &`
  Immagine di cui si vuole effettuare la copia.

- `+ getXSize() : int`

Descrizione: implementa il contratto che ritorna la dimensione orizzontale di un'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ getYSize() : int`

Descrizione: implementa il contratto che ritorna la dimensione verticale di un'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ getZSize() : int`

Descrizione: implementa il contratto che ritorna la terza dimensione di un'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ getImage() : ImageType::Pointer&`

Descrizione: ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine rappresentata.

- `+ setImage(img : ImageType::Pointer) : void`

Descrizione: metodo che cambia l'immagine rappresentata dall'oggetto *InternalData2D*.

Argomenti

- `img : ImageType::Pointer`
  Nuova immagine da rappresentare.

- `- static copyImage(image : ImageType2D::Pointer) : ImageType2D::Pointer`

Descrizione: copia un immagine restituendone la copia.

Argomenti

- `image : ImageType2D::Pointer`
  Puntatore di tipo ITK$_{\mathbf{G}}$ all'immagine da copiare (tipo ITK$_{\mathbf{G}}$).

Note

- Il metodo deve essere marcato come statico.

4.1.16 RomeoObject(interface)

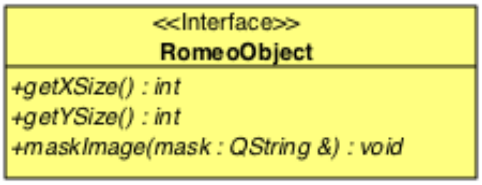

Figura 19: Diagramma calsse *RomeoObject*

Descrizione
Intefaccia che rappresenta un generico dato da analizzare in Romeo$_{G}$.

Utilizzo
Viene utilizzato nei metodi delle feature e algoritmi per lavorare su un generico dato.

Metodi

- `+ getXSize() : int`

  Descrione: contratto che ritorna la dimensione x del dato.

  Note

  – Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

  Descrione: contratto che ritorna la dimensione y del dato.

  Note

  – Il metodo deve essere marcato come costante.

- `+ maskImage(mask : const QString &) : void`

  Descrizione: contratto che applica la maschera ad un dato.

  Argomenti

  – `mask : const QString &`
  Percorso della maschera$_{G}$ da applicare al dato.

4.1.17 RGBImage (abstract)

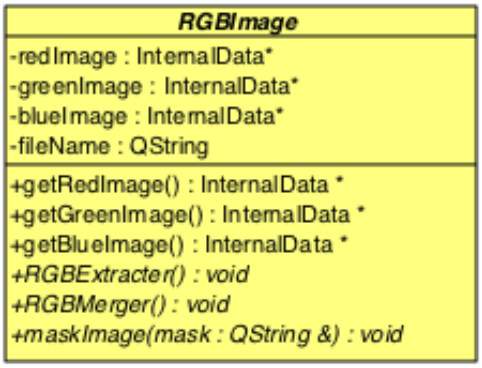

Figura 20: Diagramma classe *RGBImage*

Descrizione

Classe astratta che rappresenta un'immagine RGB, composta da tre livelli di colore *red, green* e *blue*. Definisce dei contratti per l'accesso ai tre livelli di colore, per la loro scomposizione e fusione.

Utilizzo

Viene utilizzata quando una feature$_G$ deve essere applicata ad un immagine e non ad un video.
Inoltre viene utilizzata per salvare le informazioni riguardanti il colore dell'immagine, che altrimenti sarebbe in scala di grigi.

Attributi

- `- fileName : QString`

  Descrizione: nome dell'immagine.

- `- redImage : InternalDat a*` Puntatore polimorfo al formato interno rappresentante il livello di rosso di un'immagine.
- `- greenImage : InternalData*`

  Descrizione: puntatore polimorfo al formato interno rappresentante il livello di verde di un'immagine.

- `- blueImage : InternalData*`

  Descrizione: puntatore polimorfo al formato interno rappresentante il livello di blu di un'immagine.

Metodi

- `+ getRedImage() : InternalData *`

Descrizione: metodo che ritorna il puntatore al formato interno rappresentante il livello di rosso di un'immagine.

- `+ getGreenImage() : InternalData *`

Descrizione: metodo che ritorna il puntatore al formato interno rappresentante il livello di verde di un'immagine.

- `+ getBlueImage() : InternalData *`

Descrizione; metodo che ritorna il puntatore al formato interno rappresentante il livello di blu di un'immagine.

- `+ RGBExtracter() : void`

Descrizione: contratto che estrae da un'immagine RGB tre formati interni rappresentanti i tre diversi livelli di un'immagine.

- `+ RGBMerger() : void`

Descrizione: contratto che ricompone un'immagine RGB ricostruendola dai tre formati interni rappresentanti i tre diversi livelli di un'immagine.

4.1.18 RGBImage2D (class)

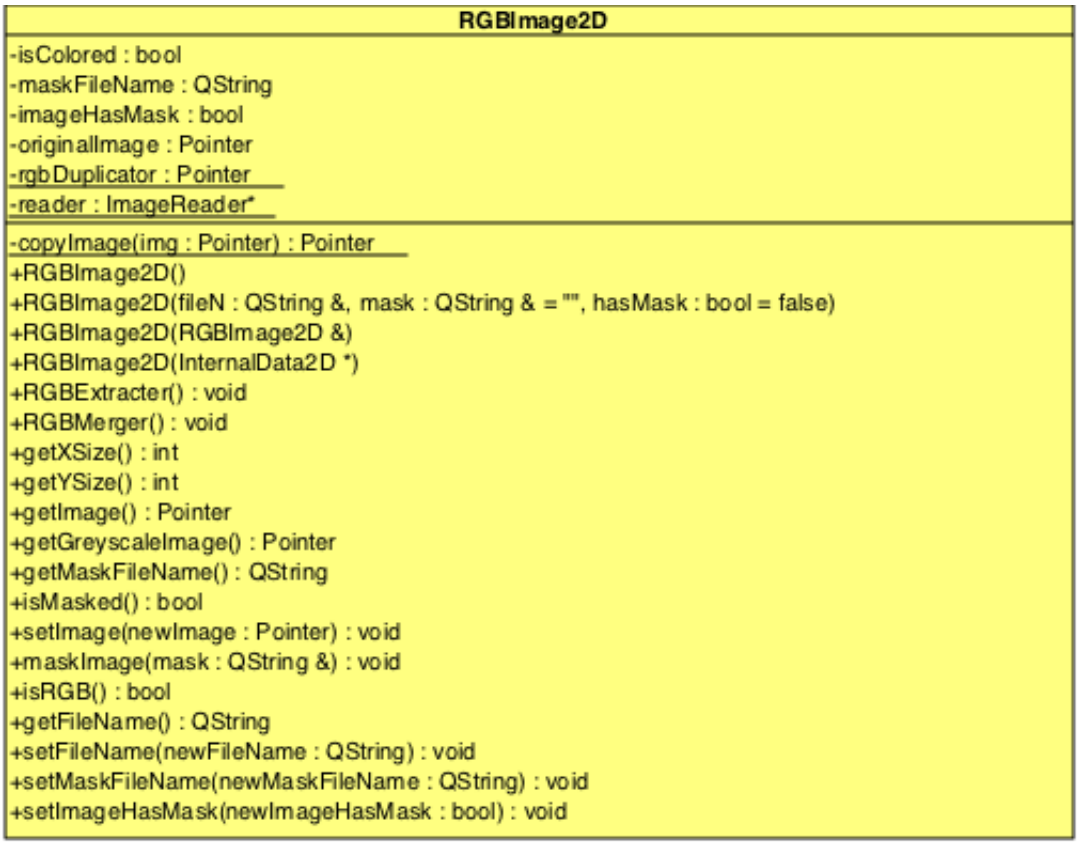

Figura 21: Diagramma classe *RGBImage2D*

Descrizione

Classe che rappresenta un immagine bidimensionale, a colori in Romeo$_{\mathbf{G}}$
Classe che "adatta" la classe itk::image fornita da itk.
Rappresenta il componente Adapter del design pattern$_{\mathbf{G}}$ Adapter.

Utilizzo

Viene utilizzata dalle feature$_{\mathbf{G}}$ che elaborano immagini bidimensionali.
Inoltre racchiude in se le informazioni sul colore delle immagini importate

Eredita da:

- Romeo::Model::Core::RGBImage.

Attributi

- `- originalImage : RGBImageType::Pointer`

  Descrizione: puntatore all'immagine.

- `-isColored : boolean`

Descrizione: booleano che indica se l'immagine è colorata o a scala di grigi.

- `- maskFileName : QString`

Descrizione: percorseo della maschera$_{\mathbf{G}}$ dell'immagine.

- `- imageHasMask : boolean`

Descrizione: booelano che indica se l'immagine ha una maschera o no.

- `-static rgbDuplicator : DuplicatorType::Pointer`

Descrizione: duplicatore per un immagine di tipo *RGBImage2D*.

- `- static  reader : ImageReader*`

Descrizione: reader pr un'immagine di tipo *RGBImage2D*.

Metodi

- `+ RGBImage2D()`
  Costruttore di *default* della classe *RGBImage2D*.
- `+ RGBImage2D(fileName : const QString &,  mask ; const QString&, hasMask : boolean)`

Descrizione: costruttore a tre argomenti della classe *RGBImage2D*.

Argomenti

- `fileName : QString &`
  Percorso dell'immagine da caricare. Verrà passato al *ReaderImage* che caricherà l'immagine.

- `+ RGBImage2D(image : const RGBImage2D &)`

Descrizione: costruttore di copia *profondo*.

Argomenti:

- `image : const RGBImage2D &)`
  Immagine di cui si vuole fare la copia.

- `+ RGBImage2D(image : InternalData2D*)`

Descrizione: costruisce un'immagine *RGBImage2D* a partire da un oggetto *InternalData2D*.

Argomenti

- `image : InternalData2D*`
  Oggetto *InternalData2D* da cui si vuole ricavare l'oggetto *RGBImage2D*.

- `+ RGBExtracter() : void`

Descrizione: metodo che estrae da un'immagine RGB tre formati interni rappresentanti i tre diversi livelli di un'immagine.

- `+ RGBMerger() : void`

Descrizione: metodo che ricompone un'immagine RGB ricostruendola dai tre formati interni rappresentanti i tre diversi livelli di un'immagine.

- `+ getXSize() : int`

Descrizione: ritorna la dimensione orizzontale dell'immagine.

Note

- Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

Descrizione: ritorna la dimensione verticale dell'immagine.

Note

- Il metodo deve essere marcato come costante.

- `+ getImage() : RGBImageType::Pointer`

Descrizione: metodo che ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine.

- `+ getGreyscaleImage() : ImageType::Pointer`

Descrizione: metodo che ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine rappresentata come scala di grigi.

- `+ getMaskFileName() : QString`

Descrizione: metodo che ritorna il percorso dell'eventuale maschera$_{\mathbf{G}}$ dell'immagine.

Note

- Il metodo deve essere marcato costante.

- `+ isMasked() : boolean`

Descrizione: metodo che ritorna un booelano indicante se l'immagine ha una maschera o meno.

Note

- Il metodo deve essere marcato costante.

- `+ setImage(newImage : RGBImageType::Pointer ) : void`

Descrizione: metodo che cambia l'immagine rappresentata.

Argomeni

- `newImage : RGBImageType::Pointer`
  Puntatore ITK$_{\mathbf{G}}$ alla nuova immagine.

- `+ maskImage(mask : const QString &) : void`

Descrizione: metodo che applica la maschera ad un immagine *RGBImage2D*.

Argomenti

- `mask : const QString &`
  Percorso della maschera$_{\mathbf{G}}$ da applicare all'oggetto *RGBImage2D*.

- `+ isRGB() : boolean`

Descrizione: metodo che ritorna un booleano indicante se l'immagine è RGB o no.

Argomenti

- Il metodo deve essere marcato costante.

Note

- Il metodo deve essere marcato come costante.

- `+ getFileName() : QString`

Descrizione: metodo che ritorna il nome del file rappresentato.

Note

- Il metodo deve essere marcato come costante.

- `+ setFileName(newFileName : QString ) : void`

Descrizione: metodo che cambia il file rappresentato.

Argomenti

- `newFileName : QString`
  Nuovo file da rappresentato.

- `+ setMaskFileName(newMaskFileName : QString) : void`

Descrizione: metodo che cambia la maschera$_G$ dell'immagine rappresentata.

Argomenti

- `newMaskFileName : QString`
  Nuova maschera.

- `+setImageHasMask(newImageHasMask : bool ) : void`

Descrizione: assegna all'attributo `imageHasMask` un nuovo valore.

Argomenti

- `newImageHasMask : bool`
  Il nuovo valore.

4.1.19 RGBImage3D (class)

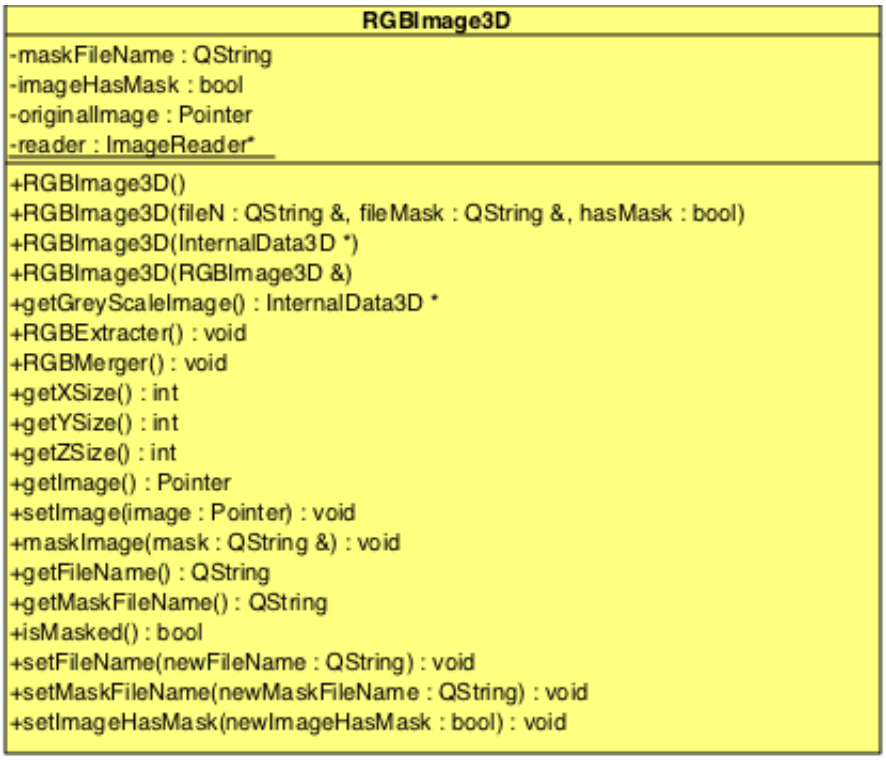

Figura 22: Diagramma classe *RGBImage3D*

Descrizione

Classe che concretizza RGBImage e rappresenta un'immagine RGB tridimensionale, composta da tre livelli di colore *red, green* e *blue*. Implementa i contratti per l'accesso ai tre livelli di colore, per la loro scomposizione e fusione. Utilizza il template itk::Image della libreria ITK$_{\mathbf{G}}$ per rappresentare l'immagine.

Utilizzo

Viene utilizzata dalle sottoclassi di AFeature e di AAlgorithm per applicare le varie feature$_{\mathbf{G}}$ e algoritmi di clustering$_{\mathbf{G}}$.

Eredita da:

- Romeo::Model::Core::RGBImage.

Attributi

- `- originalImage : RGBImageType::Pointer`

  Descrizione: puntatore all'immagine.

- `- maskFileName : QString`

  Descrizione: percorseo della maschera$_{\mathbf{G}}$ dell'immagine.

- `- imageHasMask : boolean`

Descrizione: booelano che indica se l'immagine ha una maschera o no.

- `- static reader : ImageReader*`

Descrizione: reader pr un'immagine di tipo *RGBImage3D*.

Metodi

- `+ RGBImage3D()`
  Costruttore di *default* della classe *RGBImage3D*.
- `+ RGBImage3D(fileName : const QString &, mask ; const QString&, hasMask : boolean)`

Descrizione: costruttore a tre argomenti della classe *RGBImage3D*.

Argomenti

- `fileName : QString &`
  Percorso dell'immagine da caricare. Verrà passato al *ReaderImage* che caricherà l'immagine.

- `+ RGBImage3D(image : const RGBImage3D &)`

Descrizione: costruttore di copia *profondo*.

Argomenti:

- `image : const RGBImag3D &)`
  Immagine di cui si vuole fare la copia.

- `+ RGBImage3D(image : InternalData3D*)`

Descrizione: costruisce un'immagine *RGBImage3D* a partire da un oggetto *InternalData3D*.

Argomenti

- `image : InternalData3D*`
  Oggetto *InternalDat3D* da cui si vuole ricavare l'oggetto *RGBImage3D*.

- `+ RGBExtracter() : void`

Descrizione: metodo che estrae da un'immagine RGB tre formati interni rappresentanti i tre diversi livelli di un'immagine.

- `+ RGBMerger() : void`

Descrizione: metodo che ricompone un'immagine RGB ricostruendola dai tre formati interni rappresentanti i tre diversi livelli di un'immagine.

- `+ getXSize() : int`

Descrizione: ritorna la dimensione orizzontale dell'immagine.

Note

– Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

Descrizione: ritorna la dimensione verticale dell'immagine.

Note

– Il metodo deve essere marcato come costante.

- `+ getZSize() : int`

Descrizione: ritorna la terza dimensione dell'immagine.

Note

– Il metodo deve essere marcato come costante.

- `+  getImage() : RGBImageType::Pointer`

Descrizione: metodo che ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine.

- `+ getGreyscaleImage() : ImageType::Pointer`

Descrizione: metodo che ritorna un puntatore ITK$_{\mathbf{G}}$ all'immagine rappresentata come scala di grigi.

- `+ getMaskFileName() : QString`

Descrizione: metodo che ritorna il percorso dell'eventuale maschera$_{\mathbf{G}}$ dell'immagine.

Note

– Il metodo deve essere marcato costante.

- `+ isMasked() : boolean`

Descrizione: metodo che ritorna un booelano indicante se l'immagine ha una maschera o meno.

Note

– Il metodo deve essere marcato costante.

- `+ setImage(newImage : RGBImageType::Pointer ) : void`

Descrizione: metodo che cambia l'immagine rappresentata.

Argomeni

- `newImage : RGBImageType::Pointer`
  Puntatore ITK$_{\mathbf{G}}$ alla nuova immagine.

- `+ maskImage(mask : const QString &) : void`

Descrizione: metodo che applica la maschera ad un immagine *RGBImage3D*.

Argomenti

- `mask : const QString &`
  Percorso della maschera$_{\mathbf{G}}$ da applicare all'oggetto *RGBImage3D*.

- `+ isRGB() : boolean`

Descrizione: metodo che ritorna un booleano indicante se l'immagine è RGB o no.

Argomenti

Note

- Il metodo deve essere marcato come costante.

- `+ getFileName() : QString`

Descrizione: metodo che ritorna il nome del file rappresentato.

Note

- Il metodo deve essere marcato come costante.

- `+ setFileName(newFileName : QString ) : void`

Descrizione: metodo che cambia il file rappresentato.

Argomenti

- `newFileName : QString`
  Nuovo file da rappresentato.

- `+ setMaskFileName(newMaskFileName : QString) : void`

Descrizione: metodo che cambia la maschera$_{\mathbf{G}}$ dell'immagine rappresentata.

Argomenti

- `newMaskFileName : QString`
  Nuova maschera.

- `+setImageHasMask(newImageHasMask : bool ) : void`

Descrizione: assegna all'attributo `imageHasMask` un nuovo valore.

Argomenti

- `newImageHasMask : bool`
  Il nuovo valore.

4.1.20 Video(abstract)

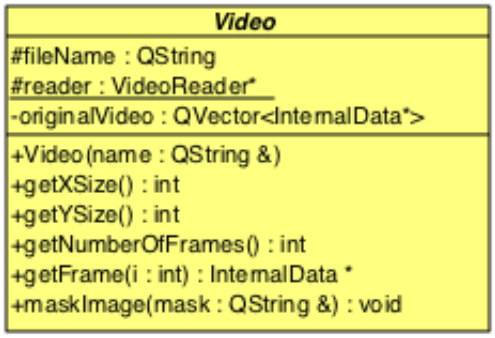

Figura 23: Diagramma classe *Video*

Descrizione

Classe astratta che rappresenta la classe base di tutti i video in Romeo$_{\mathbf{G}}$.
È composta da un vettore di oggetti di tipo *InternalData*.

Utilizzo

viene utilizzata dalle Feature$_{\mathbf{G}}$ dinamiche quando ci si aspetta un video generico.

Attributi

- `# fileName : QString`

  Descrizione: percorso del video.

- `# static reader : VideoReader *`

  Descrizione: puntatore al reader dei video.

- `-originalVideo : QVector<InternalData *>`

  Descrizione: vettore di frames del video.

Metodi

- `+ Video(name : const QStirng &)`

  Descrizione: costruttore della classe.

  Argomenti

  - `name : const QStirng &`
    Percorso del video.

- `+ getXSize() : int`

Descrizione: metodo che ritorna la larghezza del frame video.

Note

- Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

Descrizione: metodo che ritorna l'altezza del frame video.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfFrames() : int`

Descrizione: metodo che ritorna il numero di frame presenti nel video.

Note

- Il metodo deve essere marcato come costante.

- `+ getFrame(i : int) : InternalData *`

Descrizione: metodo che ritorna il frame i-esimo del video.

Argomenti

- `i : int`
  Indice del frame da ottenere.

- `+ maskImage(mask : QString &) : void`

Descrizione: metodo che aggiunge una maschera ad un video.

Argomenti

- `mask : QString &`
  Percorso della maschera nel filesystem.

4.1.21 Video2D(class)

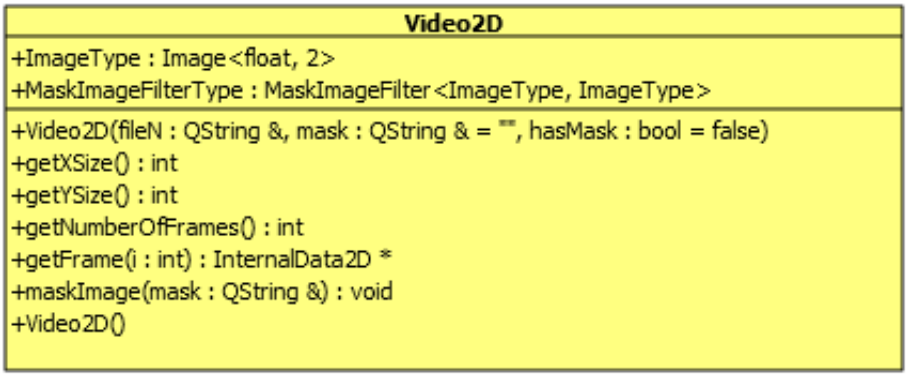

Figura 24: Diagramma classe *Video2D*

Descrizione

Classe che rappresenta un video bidimensionale in Romeo$_{\mathbf{G}}$.
È composta da un vettore di oggetti di tipo *InternalData2D*.

Utilizzo

viene utilizzata dalle Feature$_{\mathbf{G}}$ dinamiche quando ci si aspetta un video bidimensionale.

Metodi

- `+ Video2D(fileN : const QString& fileN, mask : const QString & , hasMask : bool)`

  Descrizione: costruttore della classe.

  Argomenti

  - `fileN : const QStirng &`
    Percorso del video nel filesystem.
  - `mask : const QStirng &`
    Percorso della maschera del video nel filesystem.
  - `hasMask : bool`
    true se il video possiede una maschera.

- `+ getXSize() : int`

  Descrizione: metodo che ritorna la larghezza del frame video.

  Note

  - Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

  Descrizione: metodo che ritorna l'altezza del frame video.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfFrames() : int`

Descrizione: metodo che ritorna il numero di frame presenti nel video.

Note

- Il metodo deve essere marcato come costante.

- `+ getFrame(i : int) : InternalData2D *`

Descrizione: metodo che ritorna il frame i-esimo del video.

Argomenti

- `i : int`
  Indice del frame da ottenere.

- `+ maskImage(mask : QString &) : void`

Descrizione: metodo che aggiunge una maschera ad un video.

Argomenti

- `mask : QString &`
  Percorso della maschera nel filesystem.

4.1.22 Video3D(class)

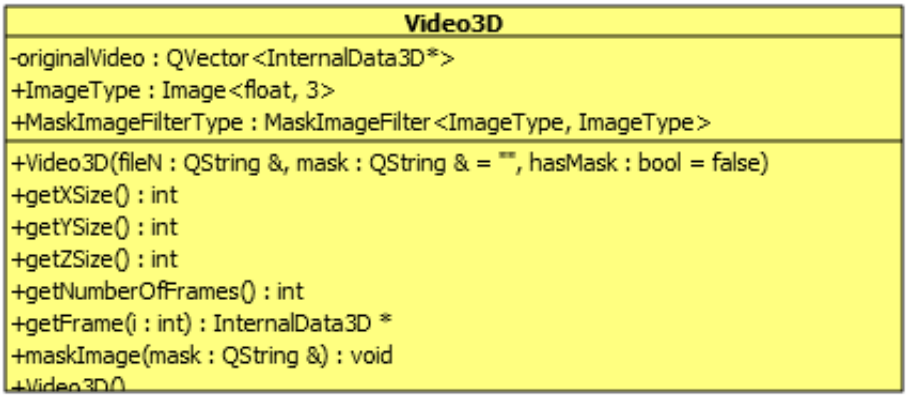

Figura 25: Diagramma classe *Video3D*

Descrizione

Classe che rappresenta un video tridimensionale in Romeo$_{\mathbf{G}}$.
È composta da un vettore di oggetti di tipo *InternalData2D*.

Utilizzo

Viene utilizzata dalle Feature$_{\mathbf{G}}$ dinamiche quando ci si aspetta un video tridimensionale.

Metodi

- `+ Video2D(fileN : const QString& fileN, mask : const QString & , hasMask : bool)`

  Descrizione: costruttore della classe.

  Argomenti

  - `fileN : const QStirng &`
    Percorso del video nel filesystem.
  - `mask : const QStirng &`
    Percorso della maschera del video nel filesystem.
  - `hasMask : bool`
    true se il video possiede una maschera.

- `+ getXSize() : int`

  Descrizione: metodo che ritorna la larghezza del frame video.

  Note

  - Il metodo deve essere marcato come costante.

- `+ getYSize() : int`

Descrizione: metodo che ritorna l'altezza del frame video.

Note

- Il metodo deve essere marcato come costante.

- `+ getZSize() : int`

Descrizione: metodo che ritorna la profondità del frame video.

Note

- Il metodo deve essere marcato come costante.

- `+ getNumberOfFrames() : int`

Descrizione: metodo che ritorna il numero di frame presenti nel video.

Note

- Il metodo deve essere marcato come costante.

- `+ getFrame(i : int) : InternalData3D *`

Descrizione: metodo che ritorna il frame i-esimo del video.

Argomenti

- `i : int`
  Indice del frame da ottenere.

- `+ maskImage(mask : QString &) : void`

Descrizione: metodo che aggiunge una maschera ad un video.

Argomenti

- `mask : QString &`
  Percorso della maschera nel filesystem.

4.2 Specifica componenti Model::Core::Algorithm

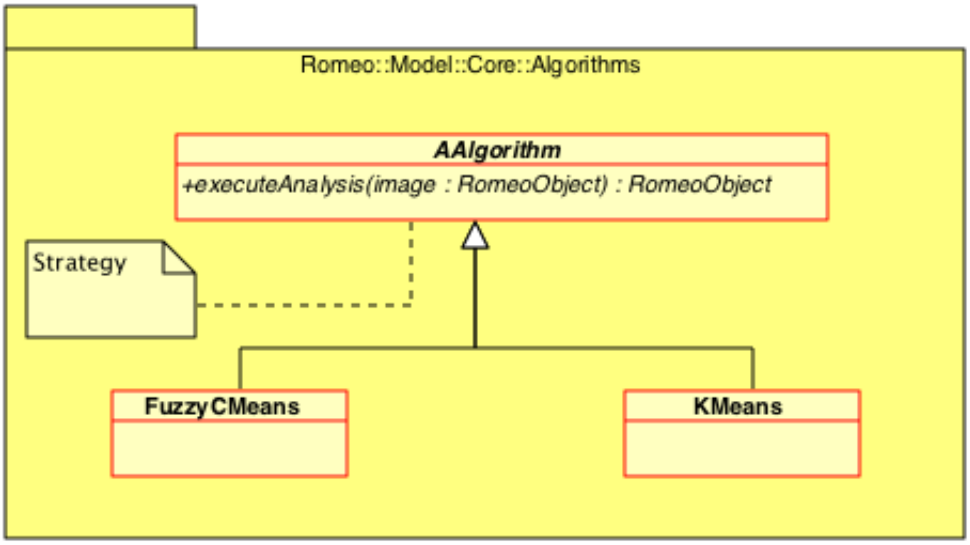

Figura 26: Componente Romeo::Model::Core::Algorithm

Package$_{\mathbf{G}}$ contenente le classi per gli algoritmi di clustering$_{\mathbf{G}}$.

4.2.1 AAlgorithm (abstract)

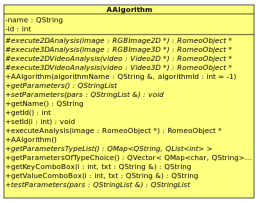

Figura 27: Diagramma classe *AAlgorithm*

Descrizione

Classe astratta che rappresenta un generico algoritmo di clustering$_{\mathbf{G}}$, secondo il design pattern Strategy. Definisce dei contratti per l'esecuzione degli algoritmi, che dovranno essere implementati dalle sue sottoclassi.

Utilizzo

Fornisce i metodi per l'esecuzione di un algoritmo di clustering su un immagine bidimensionale o tridimensionale.

Attributi

- `- name : QString`

  Descrizione: nome dell'algoritmo.

- `- id : int`

Descrizione: codice identificativo dell'algoritmo.

Metodi

- `# execute2DAnalysis(image :RGBImage2D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine bidimensionale.

  #### Argomenti

  - `image : RGBImage2D*`
    immagine bidimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale puro.

- `# execute3DAnalysis(image :RGBImage3D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine tridimensionale.

  #### Argomenti

  - `image : RGBImage3D*`
    immagine tridimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale puro.

- `# execute2DVideoAnalysis(video :Video2D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un video bidimensionale.

  #### Argomenti

  - `video ; Video2D*`
    video bidimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale puro.

- `# execute3DVideoAnalysis(video : Video3D*) : RomeoObject*`

Descrizione: metodo che esegue l'algoritmo di clustering su un video tridimensionale.

Argomenti

- `image : RGBImage2D*`
  video tridimensionale su cui eseguire l'algoritmo di clustering.

Note

- il metodo deve essere marcato virtuale puro.

- `+ AAlgorithm(algorithmName : const QString& ,algorithmId : int)`

Descrizione: costruttore della classe.

Argomenti

- `algorithmName : const QString&`
  nome dell'algoritmo.
- `algorithmId : int`
  codice identificativo dell'algoritmo.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato cost.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri dell'algoritmo di clustering$_{\mathbf{G}}$..

Note

- il metodo deve essere marcato virtuale.

- `+ getName() : QString`

Descrizione: metodo che ritorna il nome dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato cost.

- `+ getId() :int`

Descrizione: metodo che ritorna il codice identificativo dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato cost.

- `+ setId(i : int) : void`

Descrizione: inserisce il codice identificativo dell'algoritmo di clustering$_{\mathbf{G}}$.

Argomenti

- `i : int`
  codice identificativo dell'algoritmo di clustering$_{\mathbf{G}}$..

- `+ executeAnalysis(image : RomeoObject* ) : RomeoObject*`

Descrizione: esegue l'algoritmo di clustering$_{\mathbf{G}}$ su un generico dato.

Argomenti

- `image : RomeoObject*`
  generico dato su cui applicare l'algoritmo di clustering$_{\mathbf{G}}$..

- `+ AAlgorithm()`

Descrizione: costruttore di default.

- `+ getParametersTypeList() : QMap<QString, QList<int> >`

Descrizione: metodo che ritorna la lista dei parametri di tipo dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale puro.
- il metodo deve essere marcato costante.

- `+ getParametersOfTypeChoice() : QVector< QMap<char, QString> >`

Descrizione: metodo che ritorna la lista dei parametri di scelta dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ getKeyComboBox(i : int , txt : const QString & ) : QString`

Descrizione: metodo che ritorna la lista delle chaivi combobox.

Argomenti

- `i : int`
  indice combobox.
- `txt : const QString &`
  valore combobox.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ getValueComboBox(i : int , txt : const QString & ) : QString`

Descrizione: metodo che ritorna la lista del valori del combobox.

Argomenti

- `i : int`
  indice combobox.
- `txt : const QString &`
  valore combobox.

Note

- il metodo deve essere marcato virtuale .
- il metodo deve essere marcato costante.

- `+ testParameters(pars : const QStringList &) : QStringList`

Descrizione: metodo che ritorna la lista il test dei parametri.

Argomenti

- `pars : const QStringList &`
  valore dei parametri.

Note

- il metodo deve essere marcato virtuale puro .
- il metodo deve essere marcato costante.

4.2.2 FuzzyCMeansAlgorithm (class)

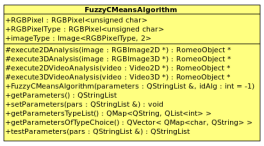

Figura 28: Diagramma classe *FuzzyCMeansAlgorithm*

Descrizione
Classe che implementa l'algoritmo di clustering$_{\mathbf{G}}$ *FuzzyCMeans*.

Utilizzo
Viene utilizzata durante un'analisi per applicare l'algoritmo a un Dataset$_{\mathbf{G}}$.

Eredita da:

- AAlgorithm.

Metodi

- `# execute2DAnalysis(image :RGBImage2D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine bidimensionale.

  Argomenti

  – `image : RGBImage2D*`
  immagine bidimensionale su cui eseguire l'algoritmo di clustering.

  Note

  – il metodo deve essere marcato virtuale.

- `# execute3DAnalysis(image :RGBImage3D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine tridimensionale.

  Argomenti

  – `image : RGBImage3D*`
  immagine tridimensionale su cui eseguire l'algoritmo di clustering.

  Note

  – il metodo deve essere marcato virtuale.

- `# execute2DVideoAnalysis(video :Video2D*) : RomeoObject*`

Descrizione: metodo che esegue l'algoritmo di clustering su un video bidimensionale.

Argomenti

- `video ; Video2D*`
  video bidimensionale su cui eseguire l'algoritmo di clustering.

Note

- il metodo deve essere marcato virtuale.

- `# execute3DVideoAnalysis(video : Video3D*) : RomeoObject*`

Descrizione: metodo che esegue l'algoritmo di clustering su un video tridimensionale.

Argomenti

- `image : RGBImage2D*`
  video tridimensionale su cui eseguire l'algoritmo di clustering.

Note

- il metodo deve essere marcato virtuale.

- `+ FuzzyCMeansAlgorithm(parameters : StringList & ,idAlg : int)`

Descrizione: costruttore della classe.

Argomenti

- `parameters: StringList &`
  parametri dell'algoritmo.
- `idAlg : int`
  codice identificativo dell'algoritmo.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato cost.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri dell'algoritmo di clustering$_{\mathbf{G}}$..

Note

- il metodo deve essere marcato virtuale.

- `+ getParametersTypeList() : QMap<QString, QList<int> >`

Descrizione: metodo che ritorna la lista dei parametri di tipo dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ getParametersOfTypeChoice() : QVector< QMap<char, QString> >`

Descrizione: metodo che ritorna la lista dei parametri di scelta dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ testParameters(pars : const QStringList &) : QStringList`

Descrizione: metodo che ritorna la lista il test dei parametri.

Argomenti

- `pars : const QStringList &`
  valore dei parametri.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

4.2.3 KMeansAlgorithm (class)

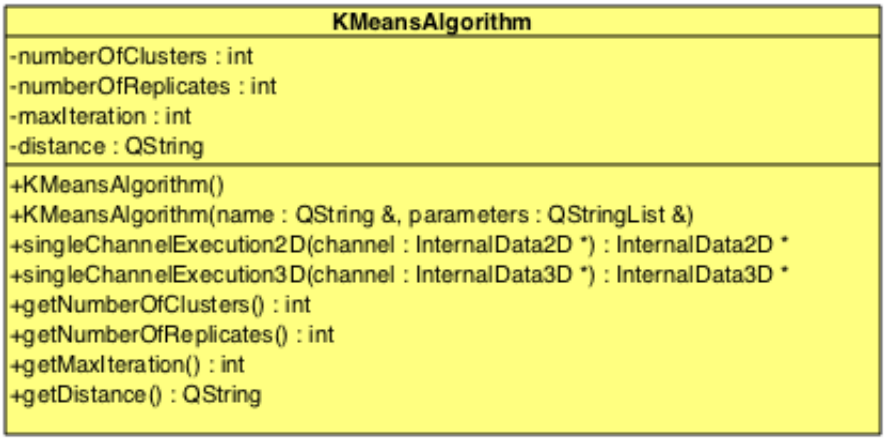

Figura 29: Diagramma classe *KMeansAlgorithm*

Descrizione
Classe che implementa l'algoritmo di clustering$_{\mathbf{G}}$ *KMeans*.

Utilizzo
Viene utilizzata durante un'analisi per applicare l'algoritmo a un Dataset$_{\mathbf{G}}$.

Eredita da:

- AAlgorithm.

Attributi

- `- numberOfCluster : int`

  Descrizione: Numero dei cluster per l'applicazione dell'algoritmo.

- `- numberOfReplicates : int`

  Descrizione: Specifica il numero di volte che l'algoritmo deve essere applicato.

- `- maxIteration : int`

  Descrizione: Limite massimo di iterazioni per l'algoritmo.

- `- distance : QString`

  Descrizione: Tipo di distanza per l'applicazione dell'algoritmo.

Metodi

- `# execute2DAnalysis(image :RGBImage2D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine bidimensionale.

  #### Argomenti

  - `image : RGBImage2D*`
    immagine bidimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale.

- `# execute3DAnalysis(image :RGBImage3D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un immagine tridimensionale.

  #### Argomenti

  - `image : RGBImage3D*`
    immagine tridimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale.

- `# execute2DVideoAnalysis(video :Video2D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un video bidimensionale.

  #### Argomenti

  - `video ; Video2D*`
    video bidimensionale su cui eseguire l'algoritmo di clustering.

  #### Note

  - il metodo deve essere marcato virtuale.

- `# execute3DVideoAnalysis(video : Video3D*) : RomeoObject*`

  Descrizione: metodo che esegue l'algoritmo di clustering su un video tridimensionale.

Argomenti

- `image : RGBImage2D*`
  video tridimensionale su cui eseguire l'algoritmo di clustering.

Note

- il metodo deve essere marcato virtuale.

- `+ MeansAlgorithm(idAlg : int)`

Descrizione: costruttore della classe a un parametro.

Argomenti

- `idAlg : int`
  codice identificativo dell'algoritmo.

- `+ MeansAlgorithm(parameters : StringList & ,idAlg : int)`

Descrizione: costruttore della classe.

Argomenti

- `parameters: StringList &`
  parametri dell'algoritmo.
- `idAlg : int`
  codice identificativo dell'algoritmo.

- `+ getNumberOfClusters() : int`

Descrizione: metodo che ritorna il numero di cluster dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato cost.

- `+ getNumberOfReplicates() : int`

Descrizione: metodo che ritorna il numero di replicates dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato cost.

- `+ getMaxIteration() : int`

Descrizione: metodo che ritorna il numero massimo di iterazioni dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato cost.

- `+ getDistance() : char`

Descrizione: metodo che ritorna il tipo di distanza dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato cost.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato cost.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri dell'algoritmo di clustering$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri dell'algoritmo di clustering$_{\mathbf{G}}$..

Note

- il metodo deve essere marcato virtuale.

- `+ getParametersTypeList() : QMap<QString, QList<int> >`

Descrizione: metodo che ritorna la lista dei parametri di tipo dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ getParametersOfTypeChoice() : QVector< QMap<char, QString> >`

Descrizione: metodo che ritorna la lista dei parametri di scelta dell'algoritmo di clustering$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

- `+ testParameters(pars : const QStringList &) : QStringList`

Descrizione: metodo che ritorna la lista il test dei parametri.

Argomenti

- `pars : const QStringList &`
  valore dei parametri.

Note

- il metodo deve essere marcato virtuale.
- il metodo deve essere marcato costante.

4.3 Specifica componenti Model::Core::Feature

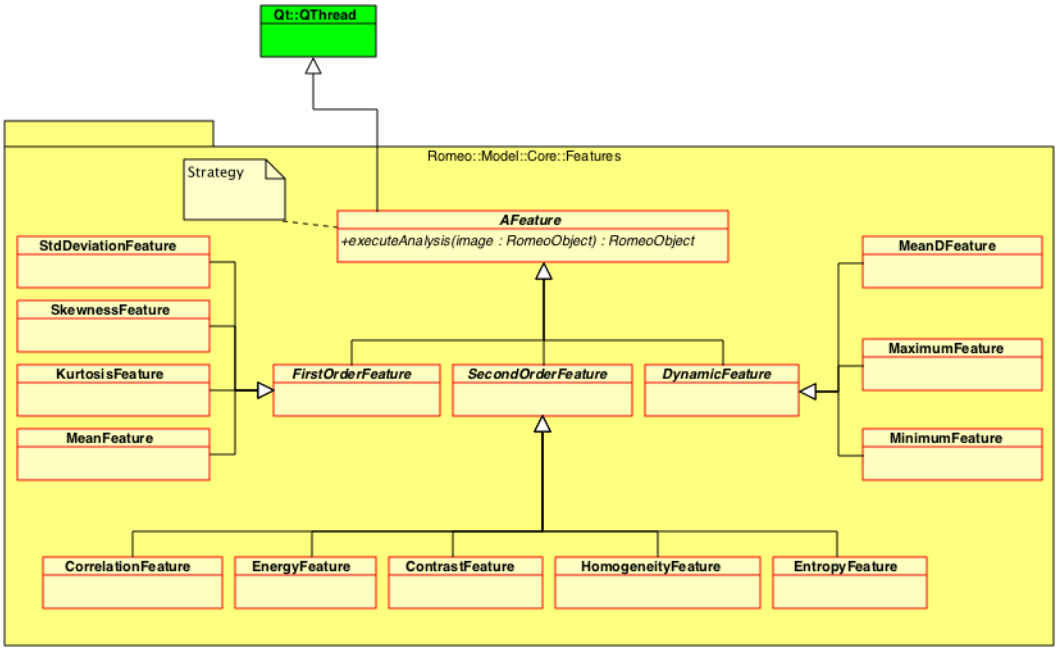

Figura 30: Diagramma package *Romeo::Model::Core::Feature*

Package$_{\mathbf{G}}$ contenente le classi per le feature$_{\mathbf{G}}$.

4.3.1 AFeature (abstract)

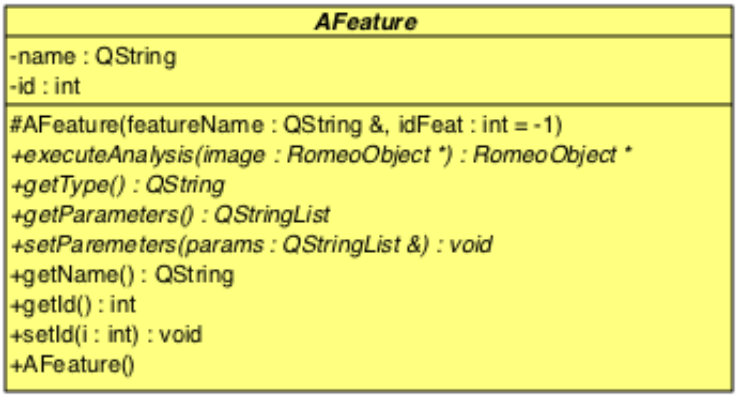

Figura 31: Diagramma classe *AFeature*

Descrizione

Classe astratta che rappresenta una generica feature$_{\mathbf{G}}$. Definisce dei contratti per l'esecuzione degli algoritmi, che dovranno essere implementati dalle sue sottoclassi. Rappresenta la componente Strategy dell'omonimo design pattern$_{\mathbf{G}}$.

Utilizzo

Fornisce i metodi per l'esecuzione di una feature$_{\mathbf{G}}$ su un immagine bidimensionale o tridimensionale.

Eredita da

- Qt::QThread

Attributi

- `- name : QString`

  Descrizione: stringa contenente il nome della feature$_{G}$.

- `- id : int`

  Descrizione: intero contentente il codice identificativo della feature$_{G}$.

Metodi

- `# AFeature(featureName : const QString & , id : int)`

  Descrizione: costruttore della classe AFeature.

  Argomenti

  - `name : const QString &`
    nome della feature$_{G}$.
  - `id : int`
    codice identificativo della feature$_{G}$.

- `+ executeAnalysis(image : RomeoObject *) : RomeoObject *`

  Descrizione: metodo puro che esegue la feature su un dato e ritorna il dato di output.

  Argomenti

  - `image : RomeoObject *`
    dato in ingresso alla feature.

  Note

  - il metodo deve essere marcato virtuale puro.

- `+ getType() : QString`

  Descrizione: contratto per restituire il tipo della feature$_{G}$.

  Note

  - deve essere marcato costante.

- `+ getParameters() : QStringList`

  Descrizione: contratto per restituisce una lista di stringe, contenente la lista degli attributi.

Note

- deve essere marcato costante.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri della feature$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri della feature.

Note

- il metodo deve essere marcato virtuale.

- `+ getName() : QString`

Descrizione: restituisce il nome della feature$_{\mathbf{G}}$.

Note

- deve essere marcato costante.

- `+ getId() : int`

Descrizione: restituisce il codice identificativo della feature$_{\mathbf{G}}$.

Note

- deve essere marcato costante.

- `+ setId(i : int) : void`

Descrizione: inserisce il codice identificativo della feature.

Argomenti

- `params : const QStringList &`
  codice identificativo della feature.

4.3.2 FirstOrderFeature (abstract)

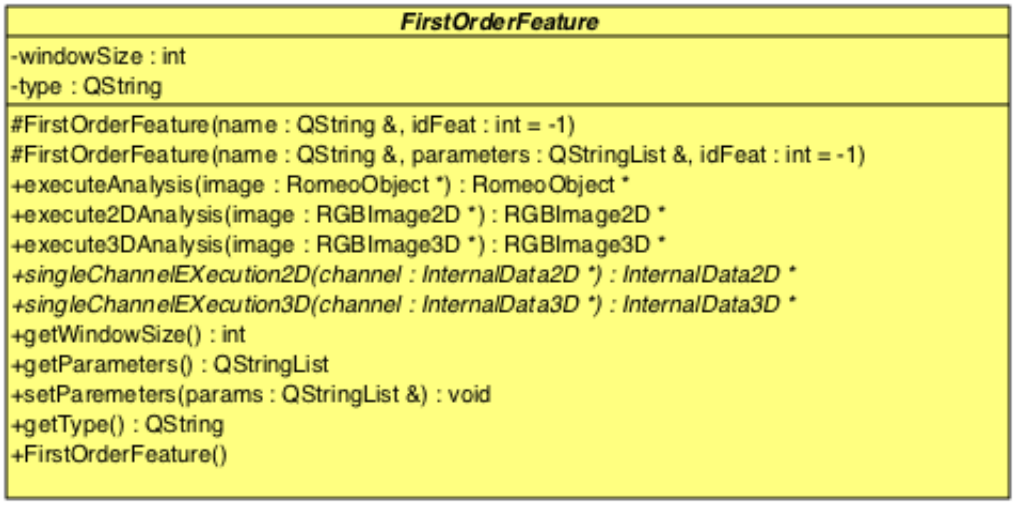

Figura 32: Diagramma classe *FirstOrderFeature*

Descrizione

Classe astratta che rappresenta una generica feature$_G$ del primo ordine. Deriva da AFeature e rimane astratta perché non implementa i contratti di quest'ultima. Le sue sottoclassi rappresenteranno le componenti ConcreteStrategy del design pattern$_G$ Strategy.

Utilizzo

Viene utilizzata durante un'analisi recuperare le informazioni sulla dimensione della finestra, la lista dei parametri e il tipo della feature$_G$.

Eredita da:

- AFeature.

Attributi

- `- windowSize : int`

  Descrizione: dimensione della finestra sulla quale effettuare le operazioni.

- `- type : QString`

  Descrizione: specifica il tipo della feature$_G$.

Metodi

- `# FirstOrderFeature(name : QString & , idFeat : int)`

  Descrizione: costruttore della classe FirstOrderFeature.

Argomenti

- `name : QString &`
  Nome della feature. Viene passato al costruttore della superclasse.
- `id : int`
  codice identificativo della feature$_{\mathbf{G}}$.

- `# FirstOrderFeature(name : QString &, parameters : QStringList & , id : int)`

Descrizione: Costruttore a tre parametri della classe. Richiama il costruttore della superclasse.

Argomenti

- `name : QString &`
  Nome della feature. Viene passato al costruttore della superclasse.
- `parameters : QStringList &`
  Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
- `id : int`
  codice identificativo della feature$_{\mathbf{G}}$.

- `+ executeAnalysis(image : RomeoObject *) : RomeoObject *`

Descrizione: metodo puro che esegue la feature su un dato e ritorna il dato di output.

Argomenti

- `image : RomeoObject *`
  dato in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ execute2DAnalysis(image : RGBImage2D *) : RGBImage2D *`

Descrizione: metodo puro che esegue la feature su un immagine 2D e ritorna un immagine 2D processata.

Argomenti

- `image : RGBImage2D *`
  immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ execute3DAnalysis(image : RGBImage3D *) : RGBImage3D *`

Descrizione: metodo puro che esegue la feature su un immagine 3D e ritorna un immagine 3D processata.

Argomenti

- `image : RGBImage3D *`
  immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ getWindowSize() : int`

Descrizione: metodo che ritorna la dimensione della finestra .

Note

- il metodo deve essere marcato costante.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri della feature$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri della feature$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri della feature.

Note

- il metodo deve essere marcato virtuale.

- `+ getType() : QString`

Descrizione: metodo che ritorna il tipo della feature$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato costante.
- il metodo deve essere marcato virtuale.

4.3.3 SecondOrderFeature (abstract)

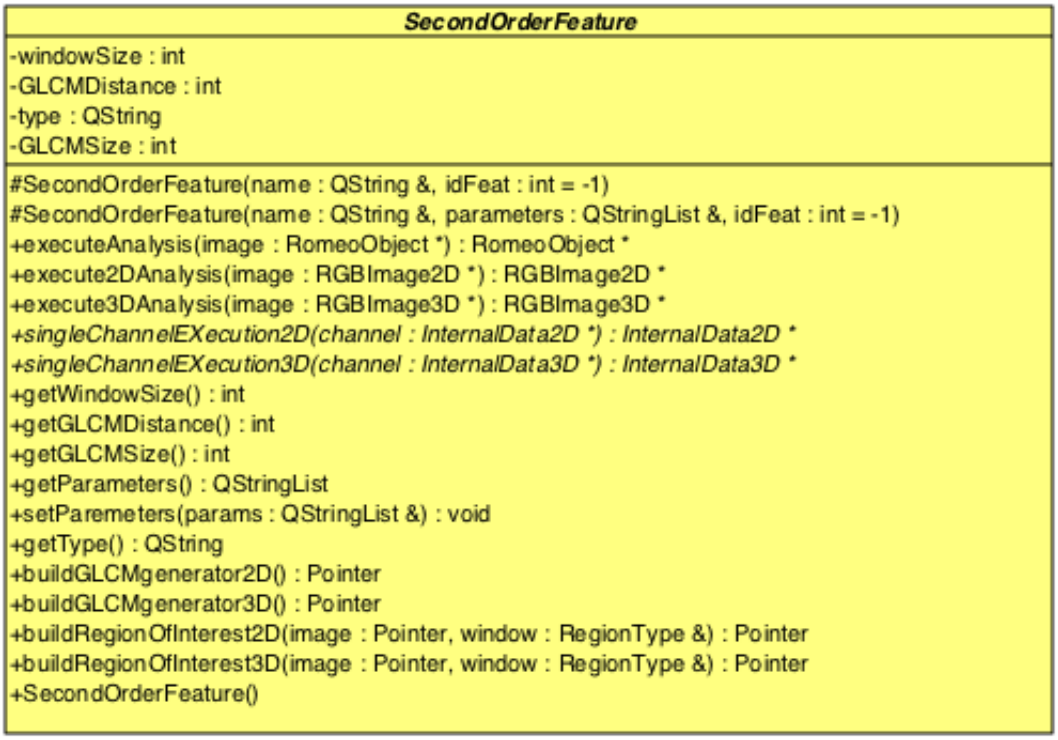

Figura 33: Diagramma classe *SecondOrderFeature*

Descrizione

Classe astratta che rappresenta una generica feature$_{\mathbf{G}}$ del secondo ordine. Deriva da AFeature e rimane astratta perché non implementa i contratti di quest'ultima. Le sue sottoclassi rappresenteranno le componenti ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.

Utilizzo

Viene utilizzata durante un'analisi recuperare le informazioni della feature$_{\mathbf{G}}$, la lista dei parametri e il tipo della feature$_{\mathbf{G}}$.

Eredita da:

- AFeature.

Attributi

- `- windowSize : int`

  Descrizione: dimensione della finestra sulla quale effettuare le operazioni.

- `- type : QString`

  Descrizione: specifica il tipo della feature$_{\mathbf{G}}$.

- `- GLCMSize : int`

Descrizione: specifica la dimensione della matrice GLCM.

- `- GLCMDistance : int`

  Descrizione: specifica la distanza della matrice GLCM.

Metodi

- `# SecondOrderFeature(name : QString & , idFeat : int)`

  Descrizione: costruttore della classe SecondOrderFeature.

  #### Argomenti

  - `name : QString &`
    Nome della feature. Viene passato al costruttore della superclasse.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# SecondOrderFeature(name : QString &, parameters : QStringList & , id : int)`

  Descrizione: Costruttore a tre parametri della classe. Richiama il costruttore della superclasse.

  #### Argomenti

  - `name : QString &`
    Nome della feature. Viene passato al costruttore della superclasse.
  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ executeAnalysis(image : RomeoObject *) : RomeoObject *`

  Descrizione: metodo puro che esegue la feature su un dato e ritorna il dato di output.

  #### Argomenti

  - `image : RomeoObject *`
    dato in ingresso alla feature.

  #### Note

  - il metodo deve essere marcato virtuale.

- `+ execute2DAnalysis(image : RGBImage2D *) : RGBImage2D *`

  Descrizione: metodo puro che esegue la feature su un immagine 2D e ritorna un immagine 2D processata.

Argomenti

- `image : RGBImage2D *`
  immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ execute3DAnalysis(image : RGBImage3D *) : RGBImage3D *`

Descrizione: metodo puro che esegue la feature su un immagine 3D e ritorna un immagine 3D processata.

Argomenti

- `image : RGBImage3D *`
  immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ getWindowSize() : int`

Descrizione: metodo che ritorna la dimensione della finestra .

Note

- il metodo deve essere marcato costante.

- `+ getGLCMDistance() : int`

Descrizione: metodo che ritorna la distanza della matrice GLCM .

Note

- il metodo deve essere marcato costante.

- `+ getGLCMSize() : int`

Descrizione: metodo che ritorna la dimensione della matrice GLCM .

Note

- il metodo deve essere marcato costante.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri della feature$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri della feature$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri della feature.

Note

- il metodo deve essere marcato virtuale.

- `+ getType() : QString`

Descrizione: metodo che ritorna il tipo della feature$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato costante.
- il metodo deve essere marcato virtuale.

- `+ buildGLCMgenerator2D() : InternalData2D::ImageToGlcmType::Pointer`

Descrizione: costruisce l'oggetto che si occupa del calcolo della GLCM per le immagini 2D.

- `+ buildGLCMgenerator3D() : InternalData3D::ImageToGlcmType::Pointer`

Descrizione: costruisce l'oggetto che si occupa del calcolo della GLCM per le immagini 3D.

- `+ buildRegionOfInterest2D(image : InternalData2D::ImageType::Pointer , window : InternalData2D::ImageType::RegionType &) : InternalData2D::roiType::Pointer`

  Descrizione: costruisce la regione di iterazione per le immagini 2D.

  Argomenti

  - `image : InternalData2D::ImageType::Pointer`
    immagine in input.

  - `window : InternalData2D::ImageType::RegionType &`
    regione dell'immagine da scorrere.

- `+ buildRegionOfInterest3D(image : InternalData3D::ImageType::Pointer , window : InternalData3D::ImageType::RegionType &) : InternalData3D::roiType::Pointer`

  Descrizione: costruisce la regione di iterazione per le immagini 3D.

  Argomenti

  - `image : InternalData3D::ImageType::Pointer`
    immagine in input.

  - `window : InternalData3D::ImageType::RegionType &`
    regione dell'immagine da scorrere.

4.3.4 DynamicFeature (abstract)

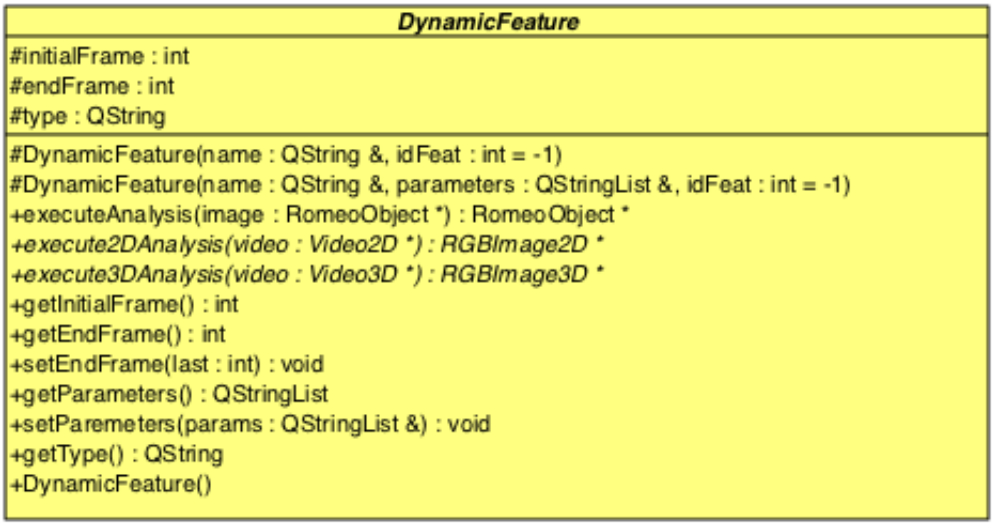

Figura 34: Diagramma classe *DynamicFeature*

Descrizione

Classe astratta che rappresenta una generica feature$_G$ dinamica. Deriva da AFeature e rimane astratta perché non implementa i contratti di quest'ultima. Le sue sottoclassi rappresenteranno le componenti ConcreteStrategy del design pattern$_G$ Strategy.

Utilizzo

Viene utilizzata durante un'analisi recuperare la lista dei parametri e il tipo della feature$_G$.

Eredita da:

- AFeature.

Attributi

- `- type : QString`

  Descrizione: specifica il tipo della feature$_G$.
- `- initialFrame : int`

  Descrizione: specifica il frame d'inizio del video.
- `- endFrame : int`

  Descrizione: specifica l'ultimo frame del video.

Metodi

- `# DynamicFeature(name : QString & , idFeat : int)`

  Descrizione: costruttore della classe SecondOrderFeature.

Argomenti

- `name : QString &`
  Nome della feature. Viene passato al costruttore della superclasse.
- `id : int`
  codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(name : QString &, parameters : QStringList & , id : int)`

Descrizione: Costruttore a tre parametri della classe. Richiama il costruttore della superclasse.

Argomenti

- `name : QString &`
  Nome della feature. Viene passato al costruttore della superclasse.
- `parameters : QStringList &`
  Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
- `id : int`
  codice identificativo della feature$_{\mathbf{G}}$.

- `+ executeAnalysis(image : RomeoObject *) : RomeoObject *`

Descrizione: metodo puro che esegue la feature su un dato e ritorna il dato di output.

Argomenti

- `image : RomeoObject *`
  dato in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ execute2DAnalysis(image : RGBImage2D *) : RGBImage2D *`

Descrizione: metodo puro che esegue la feature su un immagine 2D e ritorna un immagine 2D processata.

Argomenti

- `image : RGBImage2D *`
  immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ execute3DAnalysis(image : RGBImage3D *) : RGBImage3D *`

Descrizione: metodo puro che esegue la feature su un immagine 3D e ritorna un immagine 3D processata.

Argomenti

- `image : RGBImage3D *`
  immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale.

- `+ getInitialFrame() : int`

Descrizione: metodo che ritorna l'indice del frame iniziale.

Note

- il metodo deve essere marcato costante.

- `+ getEndFrame() : int`

Descrizione: metodo che ritorna il frame finale.

Note

- il metodo deve essere marcato costante.

- `+ setEndFrame(last : int) : void`

Descrizione: metodo che inserisce l'indice dell'ultimo frame del video da considerare.

Argomenti

- `last : int`
  indice dell'ultimo frame del video da considerare.

- `+ getParameters() : QStringList`

Descrizione: metodo che ritorna la lista dei parametri della feature$_{\mathbf{G}}$.

Note

- il metodo deve essere marcato virtuale.

- `+ setParameters(params : const QStringList &) : void`

Descrizione: inserisce i parametri della feature$_{\mathbf{G}}$.

Argomenti

- `params : const QStringList &`
  lista di parametri della feature.

Note

- il metodo deve essere marcato virtuale.

- `+ getType() : QString`

  Descrizione: metodo che ritorna il tipo della feature$_{\mathbf{G}}$.

  Note

  - il metodo deve essere marcato costante.
  - il metodo deve essere marcato virtuale.

4.3.5 MeanFeature (class)

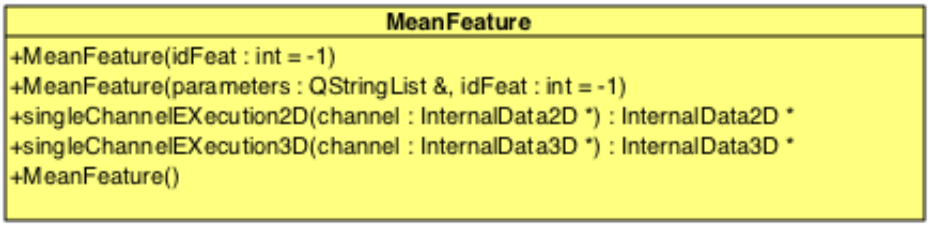

Figura 35: Diagramma classe *MeanFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Mean*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- FirstOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe MeanFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.6 KurtosisFeature (class)

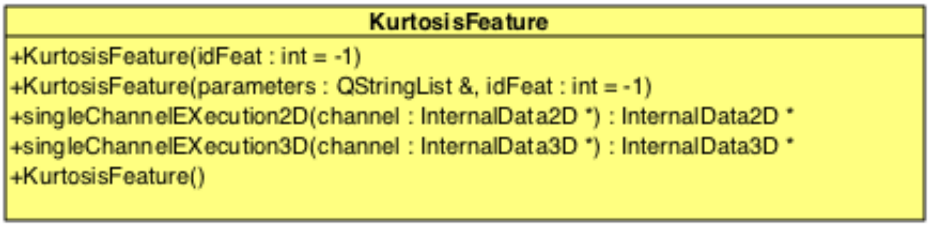

Figura 36: Diagramma classe *KurtosisFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Kurtosis*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- FirstOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe MeanFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.7 SkewnessFeature (class)

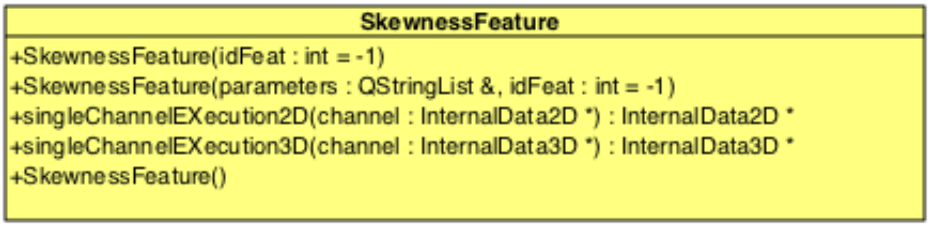

Figura 37: Diagramma classe *SkewnessFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Skewness*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- FirstOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe SkewnessFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.8 StandardDeviationFeature (class)

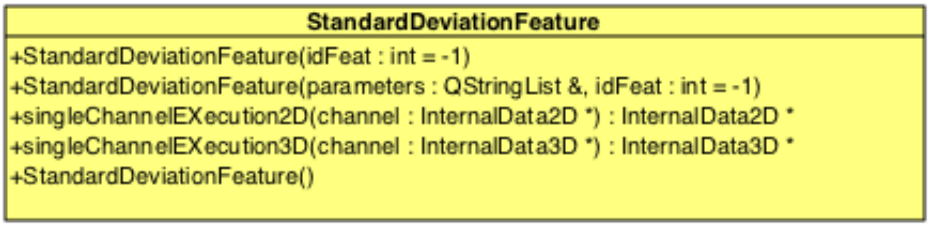

Figura 38: Diagramma classe *StandardDeviationFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Standard Deviation*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- FirstOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe StandardDeviationFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.9 ContrastFeature (class)

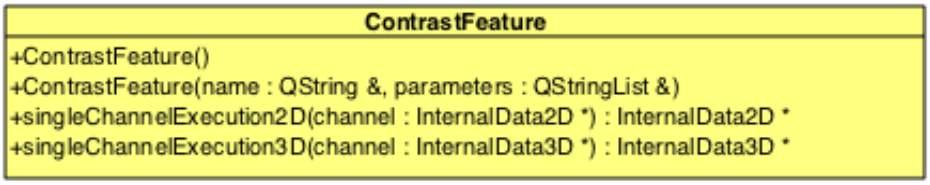

Figura 39: Diagramma classe *ContrastFeature*

Descrizione

Classe che implementa la feature$_G$ *Contrast*. Rappresenta la componente ConcreteStrategy del design pattern$_G$ Strategy.
Per l'implementazione della feature$_G$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_G$ a un Dataset$_G$.

Eredita da:

- SecondOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe ContrastFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_G$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_G$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_G$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.10 CorrelationFeature (class)

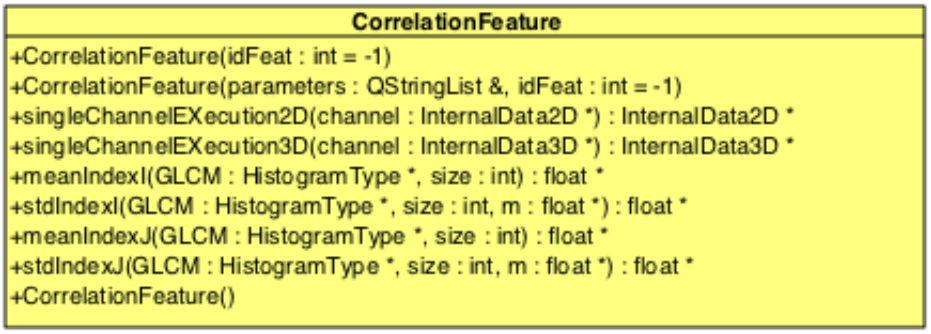

Figura 40: Diagramma classe *CorrelationFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Correlation*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- SecondOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe CorrelationFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.11 EnergyFeature (class)

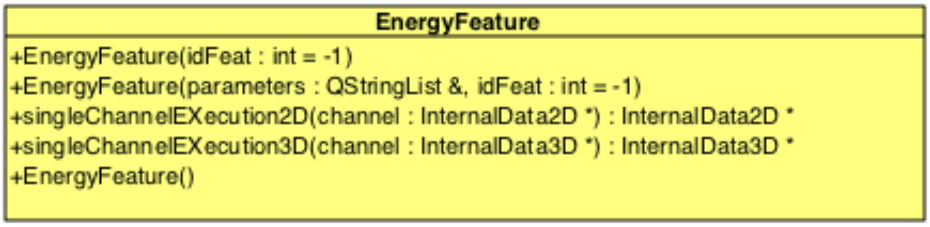

Figura 41: Diagramma classe *EnergyFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Energy*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- SecondOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe EnergyFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.12 EntropyFeature (class)

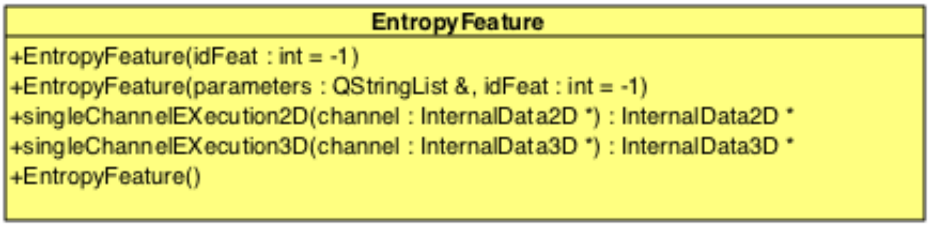

Figura 42: Diagramma classe *EntropyFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Entropy*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- SecondOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe EntropyFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.13 HomogeneityFeature (class)

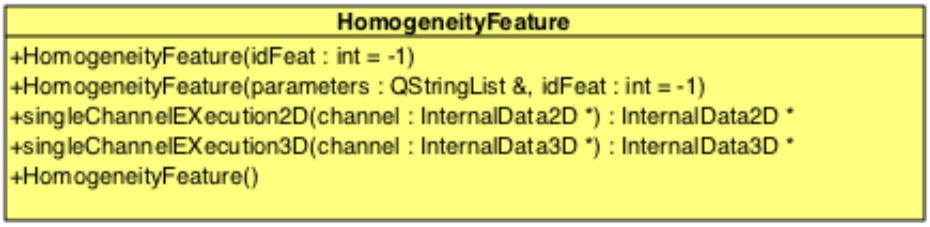

Figura 43: Diagramma classe *HomogeneityFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ *Homogeneity*. Rappresenta la componente Concrete-Strategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- SecondOrderFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe HomogeneityFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.14 MaximumFeature (class)

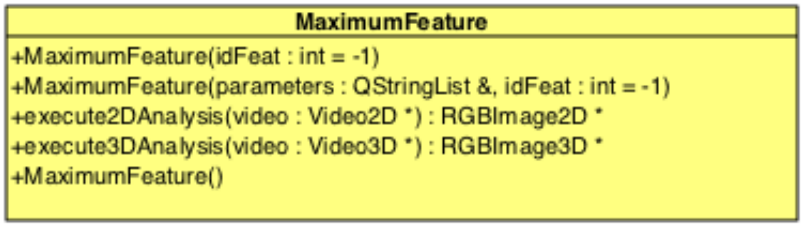

Figura 44: Diagramma classe *MaximumFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ dinamica *Maximum*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- DynamicFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe MaximumFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.15 MeanDynamicFeature (class)

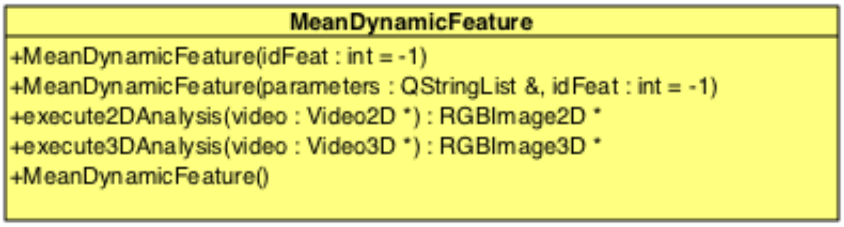

Figura 45: Diagramma classe *MeanDynamicFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ dinamica *DynamicMean*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- DynamicFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe MeanDynamicFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.16 MinimumFeature (class)

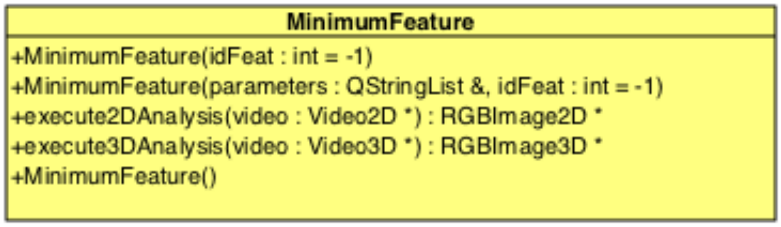

Figura 46: Diagramma classe *MinimumFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ dinamica *Minimum*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- DynamicFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe MinimumFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.3.17 AreaUnderCurveFeature (class)

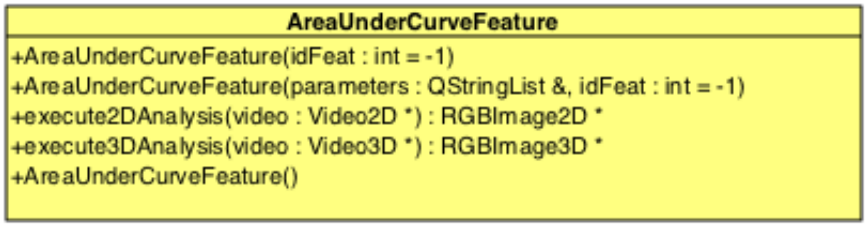

Figura 47: Diagramma classe *AreaUnderCurveFeature*

Descrizione

Classe che implementa la feature$_{\mathbf{G}}$ dinamica *Area Under the Curve*. Rappresenta la componente ConcreteStrategy del design pattern$_{\mathbf{G}}$ Strategy.
Per l'implementazione della feature$_{\mathbf{G}}$ il programmatore dovrà seguire l'appendice A.4.

Utilizzo

Viene utilizzata durante un'analisi per applicare la feature$_{\mathbf{G}}$ a un Dataset$_{\mathbf{G}}$.

Eredita da:

- DynamicFeature.

Metodi

- `# DynamicFeature(idFeat : int)`

  Descrizione: costruttore della classe AreaUnderCurveFeature.

  Argomenti

  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `# DynamicFeature(parameters : QStringList & , id : int)`

  Descrizione: Costruttore a due parametri della classe. Richiama il costruttore della superclasse.

  Argomenti

  - `parameters : QStringList &`
    Lista contenente i parametri della feature$_{\mathbf{G}}$. Verrà effettuato il *parsing* della lista, dalla quale verrà estratta la dimensione della finestra.
  - `id : int`
    codice identificativo della feature$_{\mathbf{G}}$.

- `+ singleChannelEXecution2D(channel : InternalData2D *) : InternalData2D *`

  Descrizione: metodo puro che esegue la feature su un canale di un immagine 2D.

Argomenti

- `channel : InternalData2D *`
  canale di un immagine 2D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

- `+ singleChannelEXecution3D(channel : InternalData3D *) : InternalData3D *`

Descrizione: metodo puro che esegue la feature su un canale di un immagine 3D.

Argomenti

- `channel : InternalData3D *`
  canale di un immagine 3D in ingresso alla feature.

Note

- il metodo deve essere marcato virtuale puro.

4.4 Specifica componenti Model::Util

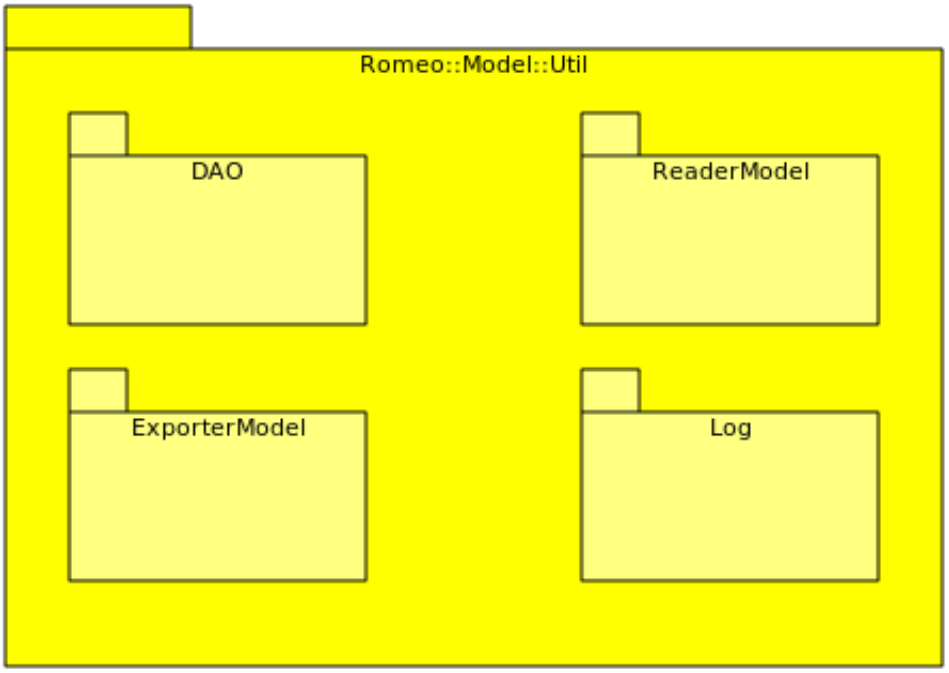

Figura 48: Diagramma package *Romeo::Model::Util*

Componente che contiene le classi di supporto per alcune operazioni del core.

4.5 Specifica componenti Model::Util::Log

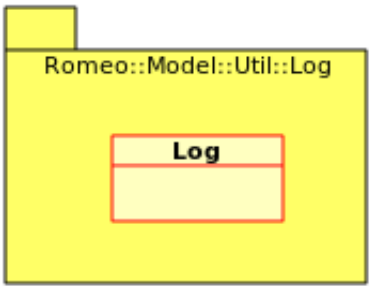

Figura 49: Diagramma package *Romeo::Model::Util::Log*

Package$_{\mathbf{G}}$ che contiene la classe log.

4.5.1 Log (class)

Descrizione

Classe utilizzata per la creazione di file di log contenenti i dettagli delle operazioni effettuate in Romeo.

Utilizzo

La classe è utilizzata ad ogni operazioni effettuata dall'utente e ad ogni operazione eseguita sul DAO, essa genera differenti file di log, a seconda delle operazioni che memorizza.

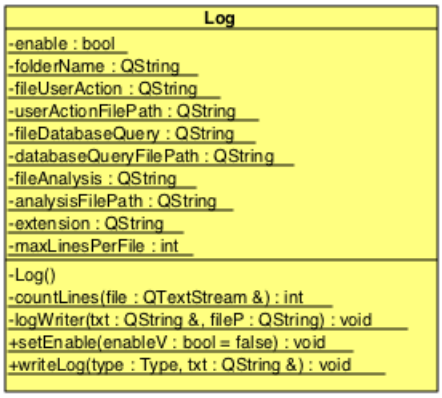

Figura 50: Diagramma classe *Log*

Attributi

- `- folderName : static QString`

  Descrizione: stringa che identifica il nome della cartella in cui creare i file di log.

- `- enable : static boolean`

  Descrizione: attributo che indica se è abilitata o meno la scrittura del log.

- `- fileUserAction : static QString`

  Descrizione: indica il nome del file di log contenente le informazioni riguardo alle operazioni utente.

- `- userActionFilePath : static QString`

  Desrizine: indica il path del file di log, compreso il nome del file, definito dall'attributo fileUserAction.

- `- fileDatabaseQuery : static QString`

  Descrizione: indica il nome del file di log contenente le informazioni riguardanti le operazioni effettuate sul database.

- `- databaseQueryFilePath : static QString`

  Descrizione: indica il path del file di log, compreso il nome del file, definito dall'attributo fileDatabaseQuery.

- `- fileAnalysis : static QString`

  Descrizione: indica il nome del file di log contenente le informazioni riguardanti le operazioni effettuate durante un'analisi.

- `- extension : static QString`

  Descrizione: indica l'estensione dei file di log.

- `- analysisFilePath : static QString`

Descrizione: indica il path del file di log, compreso il nome del file, definito dall'attributo fileAnalysis.

- `- maxLinesPerFile : static QString`

Descrizione: indica il numero massimo di linee di testo contenute in un file di log. Una volta superato tale valore, il contenuto del file corrente viene copiato in un altro, cosi da poterlo svuotare.

Metodi

- `- Log()`

Descrizione: costruttore privato.

- `+static void setEnable(enableV : boolean)`

Descrizione: metodo che cambia lo stato del Log. Lo abilità o lo disabilità.

Argomenti

- `enableV : boolean`
  Abilita o disabilita il Log.

Note

- Il metodo deve essere marcato statico.

- `+ static void writeLog(type : Type, txt : const QString&)`

Decrizione: metodo che scrive testo su un file.

Argomenti

- `type : Type`
  Tipo di log che si vuole generare;
- `txt : const QString&`
  Testo che va scritto nel log.

Note

- Il metodo deve essere marcato statico.

- `- static countLines(file : QTextStream &): int`

Descrizione: metodo che conta le linee presenti nel parametro file.

Argomenti

- `file : QTextStream &`
  File di cui si vogliono testare le linee.

Note

- Il metodo deve essere marcato statico.

- `- static logWriter(txt : const QString&, fileP : QString) : void`

Descrizione: metodo che scrive testo su un file.

Argomenti

- `txt : const QString&`
  Testo da scrivere nel file;
- `fileP : QString`
  Percorso del file.

Note

- Il metodo deve essere marcato statico.

4.6 Specifica componenti Model::Util::ReaderModel

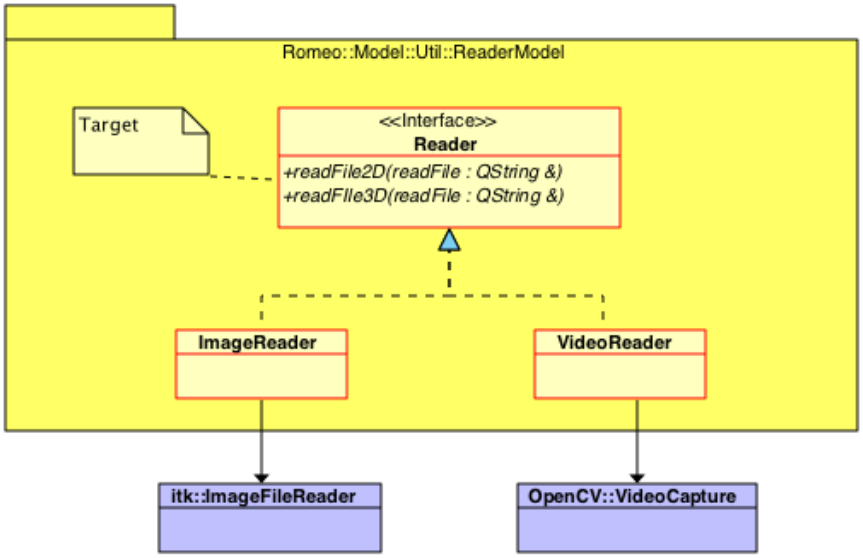

Figura 51: Diagramma package *Romeo::Model::Util::ReaderModel*

Componente che contiene le classi utilizzate per leggere i vari formati di immagini su cui opera Romeo.

4.6.1 Reader (interface)

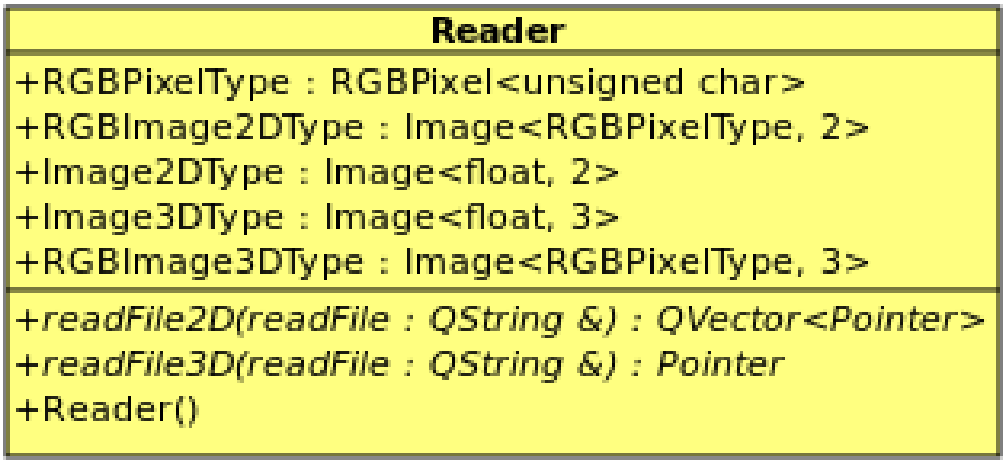

Figura 52: Interfaccia Reader: metodi

Descrizione

L'interfaccia rappresenta un generico oggetto reader, essa fornisce il contratto ai reader concreti.

Utilizzo

L'interfaccia fornisce dei contratti puri che saranno implementati dalle sottoclassi.

Metodi

`+ readFile2D(readFile : QString &): RGBImage2DType::Pointer`

Metodo virtuale puro che ha come contratto la lettura di un file 2D. Il metodo deve essere ridefinito dalle classi che ereditano da questa, definendo la lettura di un file 2D.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale puro;
- questo metodo deve essere ridefinito.

`+ readFil32D(readFile : QString &): RGBImage3DType::Pointer`

Metodo virtuale puro che ha come contratto la lettura di un file 3D. Il metodo deve essere ridefinito dalle classi che ereditano da questa, definendo la lettura di un file 3D.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale puro;
- questo metodo deve essere ridefinito.

4.6.2 ImageReader (class)

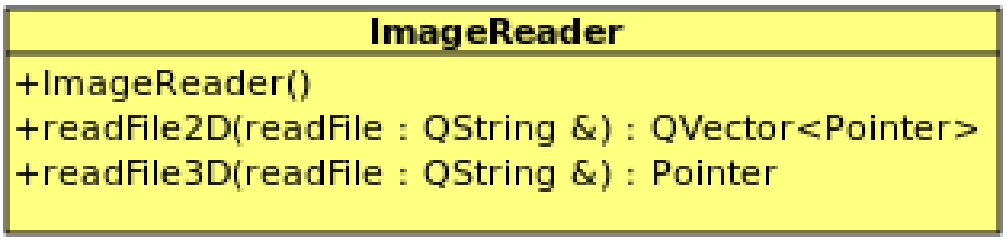

Figura 53: Classe ImageReader: attributi e metodi

Descrizione

Classe che rappresenta l'oggetto incaricato di caricare file di tipo 2D e 3D non time-dipendent.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse.

Classi ereditate

- Reader.

Attributi

`- itk::ImageFileReader reader`

Oggetto del toolkit itk che identifica un reader per la lettura di file di tipo immagine dal file system.

Metodi

`+ ImageReader()`

Costruttore pubblico della classe ImageReader.

`+ readFile2D(readFile : QString &): RGBImage2DType::Pointer`

Metodo che implementa il contratto fornito dalla classe astratta Reader.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

`+ readFile3D(readFile : QString &): RGBImage3DType::Pointer`

Metodo che implementa il contratto fornito dalla classe astratta Reader.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

4.6.3 VideoReader (class)

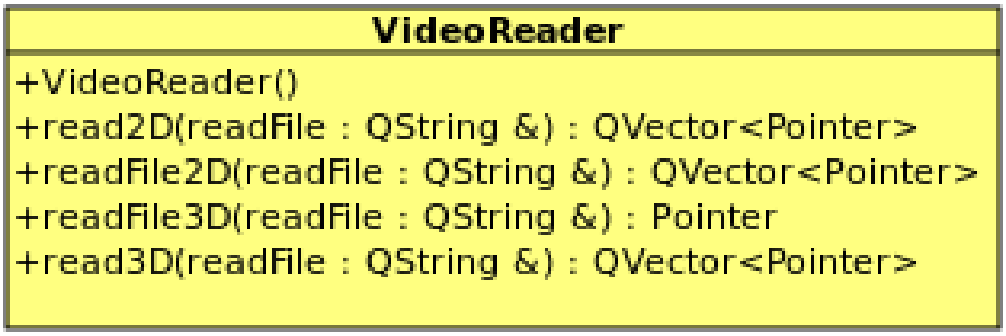

Figura 54: Classe VideoReader: attributi e metodi

Descrizione
Classe che rappresenta l'oggetto incaricato di caricare file di tipo 2D e 3D time-dipendent.

Utilizzo
La classe implementerà i metodi virtuali puri della superclasse Reader.

Classi ereditate

- Reader.

Attributi

`- itk::VideoFileReader reader`

Oggetto del toolkit itk che identifica un reader per la lettura di file di tipo video dal file system.

Metodi

`+ VideoReader()`

Costruttore pubblico della classe VideoReader.

`+ readFile2D(readFile : QString &): RGBImage2DType::Pointer`

Metodo che implementa il contratto fornito dalla classe astratta Reader.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

`+ readFile3D(readFile : QString &): RGBImage3DType::Pointer`

Metodo che implementa il contratto fornito dalla classe astratta Reader.

Argomenti

- `readFile : QString &`
  Riferimento al nome del file che il metodo deve leggere.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

4.7 Specifica componenti Model::Util::ExporterModel

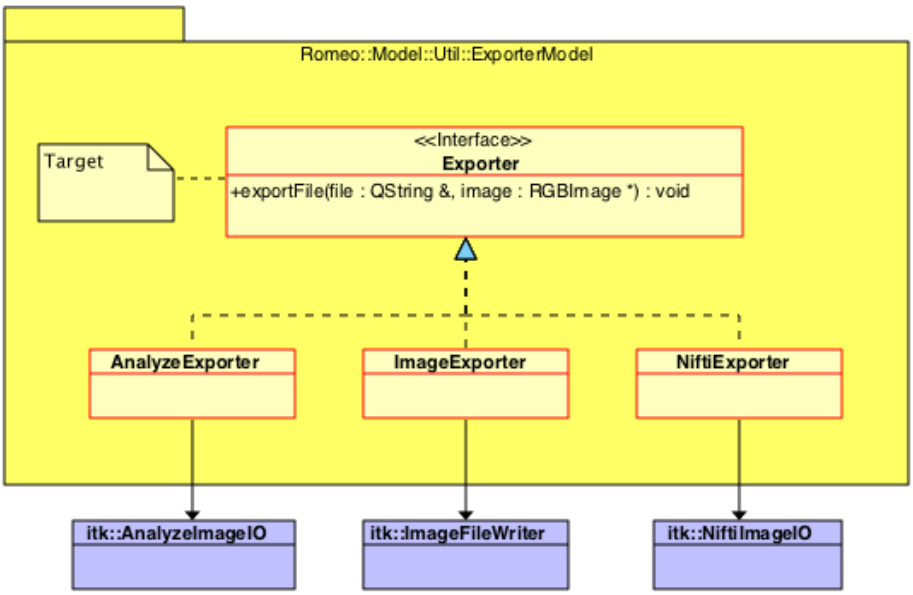

Figura 55: Diagramma package *Romeo::Model::Util::ExporterModel*

Package$_{\mathbf{G}}$ che contiene le classi utilizzate per esportare in vari formati i risultati delle analisi effettuati da Romeo.

4.7.1 Exporter (interface)

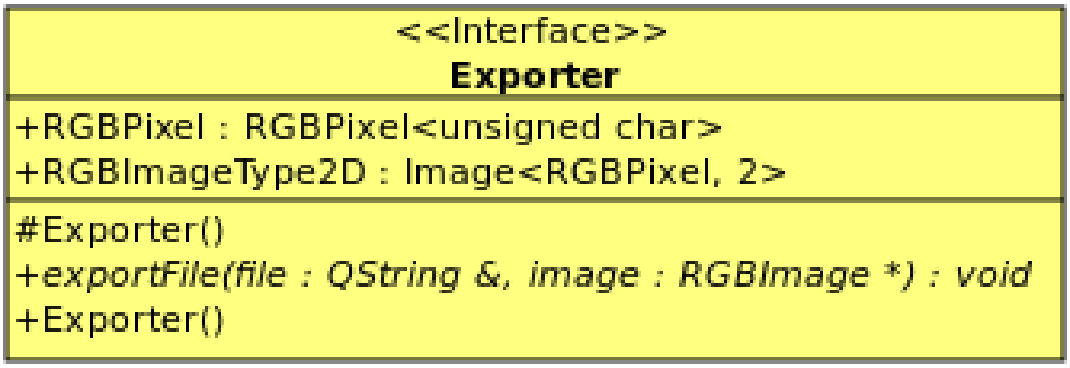

Figura 56: Diagramma classe *Exporter*

Descrizione
L'interfaccia rappresenta un generico oggetto exporter, essa fornisce il contratto alle classi exporter concrete.

Utilizzo
L'interfaccia fornisce dei contratti puri che saranno implementati dalle sottoclassi.

Metodi

- `+ exportImage(file : QString &, image: RGBImage*)`

  Descrizione: Metodo virtuale puro che ha come contratto la scrittura di un oggetto di tipo RGBImage su file system con il nome contenuto nel parametro file.

  Argomenti

  - `file : QString &`
    Riferimento al nome che il metodo deve dare al file che scriverà nel file system.
  - `image : RGBImage*`
    Puntatore all'oggetto che rappresenta l'immagine che il metodo deve scrivere su file system

  Note

  - questo metodo deve essere marcato virtuale puro;
  - questo metodo deve essere ridefinito.

4.7.2 Exporter2D (class)

Figura 57: Diagramma classe *Exporter2D*

Descrizione

Classe che rappresenta l'oggetto incaricato di scrivere sul disco le immagini 2D elaborate in Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse.

Classi ereditate

- Exporter.

Attributi

- `- itk::ImageFileWriter writer`

  Descrizione: Oggetto del toolkit itk che identifica un writer per la scrittura di file di tipo immagine 2D nel file system.

Metodi

- `+ Export2D()`

  Descrizione: Costruttore pubblico della classe Export2D.

- `+ ExportFileconst (file: QString&, image: RGBImage*): void RGBImage2DType::Pointer`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta Exporter. Esso scrive su disco l'immagine rappresentata dal parametro image di tipo puntatore ad RGBImage nel percorso indicato dal parametro file di tipo riferimento a QString.

Argomenti

- `file : QString &`
  Riferimento al nome che l'immagine scritta sul disco dovrà avere.
- `image : RGBImage*`
  Puntatore all'oggetto rappresentante l'immagine che dovrà essere scritta su disco.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo deve essere marcato costante;
- questo metodo è stato ridefinito.

4.7.3 Exporter3D (class)

Figura 58: Diagramma classe *Exporter3D*

Descrizione
Classe che rappresenta l'oggetto incaricato di scrivere sul disco le immagini 3D elaborate in Romeo.

Utilizzo
La classe implementerà i metodi virtuali puri della superclasse.

Classi ereditate

- exporter.

Attributi

- `- itk::ImageFileWriter writer`

  Descrizione: Oggetto del toolkit itk che identifica un writer per la scrittura di file di tipo immagine 3D nel file system.

Metodi

- `+ Export3D()`

  Descrizione: Costruttore pubblico della classe Export3D.

- `+ ExportFileconst (file: QString&, image: RGBImage*): void RGBImage2DType::Pointer`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta Exporter. Esso scrive su disco l'immagine rappresentata dal parametro image di tipo puntatore ad RGBImage nel percorso indicato dal parametro file di tipo riferimento a QString.

Argomenti

- `file : QString &`
  Riferimento al nome che l'immagine scritta sul disco dovrà avere.
- `image : RGBImage*`
  Puntatore all'oggetto rappresentante l'immagine che dovrà essere scritta su disco.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo deve essere marcato costante;
- questo metodo è stato ridefinito.

4.8 Specifica componenti Model::Util::DAO

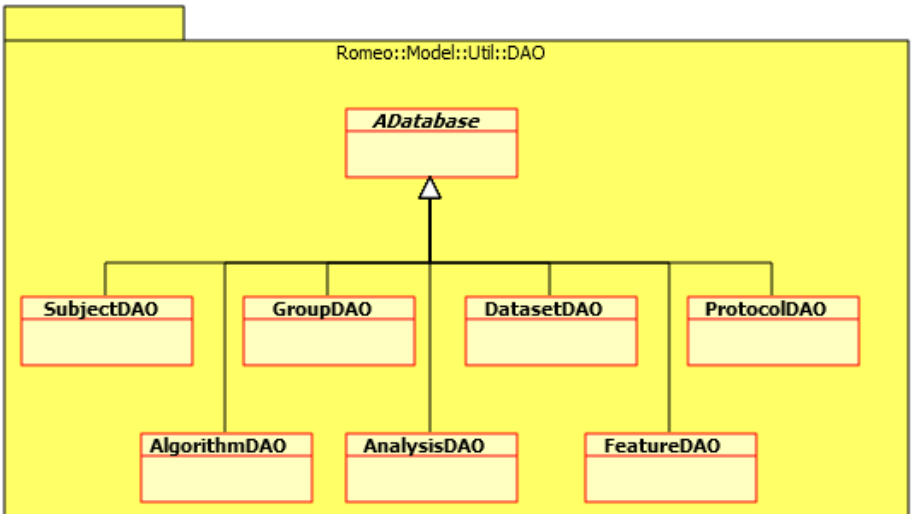

Figura 59: Diagramma package *Romeo::Model::Util::DAO*

Package$_{\mathbf{G}}$ che contiene le classi per l'interazione di Romeocon il Database.

4.8.1 ADatabase (Abstract)

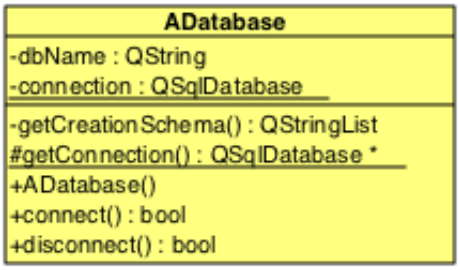

Figura 60: Diagramma classe *ADatabase*

Descrizione

Classe astratta che fornisce i metodi di connessione e disconnessione al database e dei metodi per notificare eventuali errori dovuti al collegamento con componenti esterne, nel nostro caso, il database. Tale classe viene solamente estesa dalle classi che opereranno sulle tabelle del database.

Utilizzo

La classe viene creata alla creazione di una sua sottoclasse.

Eredita da:

- Qt::QSqlDatabase.

Attributi

- `- static connection : QSqlDatabase`

  Descrizione: attributo statico che identifica la connessione ad un database.

- `-dbName : QString`

  Descrizione: nome del database a qui si è connesso.

Metodi

- `-getCreationSchema() : QStringList`

  Descrizione: metodo che ritorna lo schema SQL del dtabase.

- `# static getConnection() : QSqlDatabase *`

  Descrizione: metodo che si occupa di creare la connessione al database.

- `+ ADatabase()`

Descrizione: costruttore delle classe. Esso controlla se l'attributo `connection` contiene già una connesione, in caso negativo la crea.

- `+ connect() : boolean`

Descrizione: metodo pubblico che effettua la connessione al database e ritorna se questa è stata effettuata con successo.

- `+ disconnect() : boolean`

Descrizione: metodo che effettua la disconnessione dal database e ritorna se questa è stata effettuata con successo.

4.8.2 AlgorithmDAO(class)

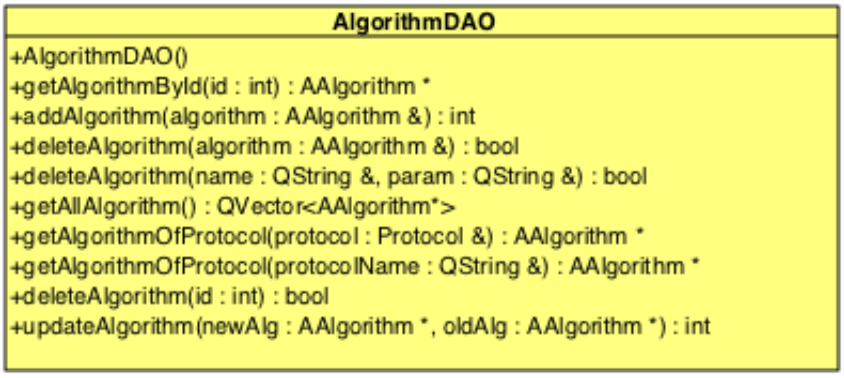

Figura 61: Diagramma classe *AlgorithmDAO*

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella Algorithm del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti gli algoritmi creati dall'utente o utilizzati daRomeo.

Classi ereditate

- Romeo::Model::Util::DAOADatabase.

Metodi

- `+ AlgorithmDAO()`

  Descrizione: costruttore della classe *AlgorithmDAO*, richiama il costruttore della superclasse.

- `+ getAlgorithmById(id : const int): AAlgorithm*`

  Descrizione: metodo pubblico che ricerca all'interno del database un algoritmo avente l'id indicato nel parametro. Esso ritorna un puntatore a tale algoritmo di tipo *AAlgorithm*.

  Argomenti

  - `id : const int`
    Indica l'id dell'algoritmo da cercare.

- `+ addAlgorithm(algorithm : const AAlgorithm*): QVariant`

Descrizione: metodo pubblico che aggiunge nel database un nuovo algoritmo passato come parametro. Esso ritorna l'ultimo id creato nella tabella Algorithm del database.

Argomenti

- `algorithm : AAlgorithm*`
  Puntatore all'oggetto di tipo *AAlgorithm* da inserire nel database.

- `+ deleteAlgorithm(algorithm : AAlgorithm*) : boolean`

Descrizione: metodo pubblico che rimuove dal database un algoritmo passato come parametro.

Argomenti

- `algorithm : AAlgorithm*`
  Puntatore all'oggetto di tipo *AAlgorithm* da eliminare dal database.

- `+ deleteAlgorithm(name : const QString &, param: const QString &) : boolean`

Descrizione: metodo pubblico che rimuove dal database un algoritmo avente nome e parametri passati come parametro.

Argomenti

- `name : const QString &`
  Riferimento alla stringa avente il nome dell'algoritmo da eliminare.
- `param : const QString &`
  Riferimento alla stringa avente tutti i parametri dell'algoritmo che si vuole eliminare.

- `+ getAllAlgorithm(): QVector<AAlgorithm *>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *AAlgorithm* contenente tutti gli algoritmi presenti nel database.

- `+ getAlgorithmOfProtocol(protocol : const Protocol &): AAlgorithm *`

Descrizione: metodo pubblico che ritorna un puntatore all'oggetto AAlgorithm, che rappresenta l'algoritmo che fa parte del protocol passato come parametro.

Argomenti

- `protocol : const Protocol &`
  Riferimento al *Protocol* del quale si vuole ricavare l'algoritmo.

- `+ getAlgorithmOfProtocol(protocolName : const QString &): AAlgorithm*`

Descrizione: metodo pubblico che ritorna un puntatore all'oggetto *AAlgorithm*, che rappresenta l'algoritmo membro del protocol$_{\mathbf{G}}$ avente il nome passato come parametro.

Argomenti:

- `protocolName : const QString &`
  Riferimento al nome del Protocol$_{\mathbf{G}}$ del quale si vuole ricavare l'algoritmo.

- `+ deleteAlgorithm(id : int) : boolean`

Descrizone: metodo che elimina l'algoritmo dal database.

Argomenti

- `id : int`
  Id dell'algoritmo che si vuole eliminare.

- `+ updateAlgorithm(newAlg : AAlgorithm *, oldAlg : AAlgorithm *) : int`

Descrizione: metodo che sostituisce un algoritmo$_{\mathbf{G}}$. Ritorna l'id del nuovo algoritmo.

Argomenti

- `newAlg : AAlgorithm *`
  Nuovo algoritmo;
- `oldAlg : AAlgorithm *`
  Vecchio algoritmo da modificare.

4.8.3 AnalysisDAO(class)

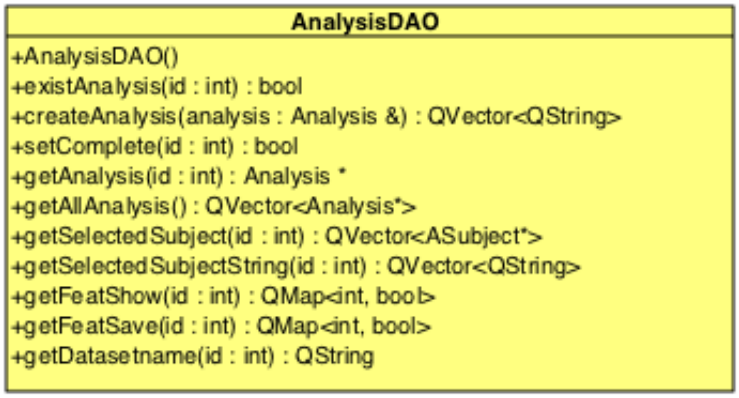

Figura 62: Diagramme classe *AnalysisDAO*

Descrizione
Classe che rappresenta l'oggetto incaricato di operare con la tabella Analysis del database.

Utilizzo
La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti le analisi effettuate dall'utente.

Classi ereditate

- Romeo::Model::Util::DAO::ADatabase.

Metodi

- `+ AnalysisDAO()`

  Descrizione: costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

- `+ existAnalysis(id : int) : boolean`

  Descrizione: metodo che controlla se esiste giù un analsi avente l'id passato.

  Argomenti

  - `id : int`
    Rappresenta l'id dell'oggetto *Analysis* da cercare.

- `+ createAnalysis(analysis : const Analysis &): boolean`

  Descrizione: metodo pubblico che aggiunge nel database informazioni riguardanti l'analisi passata come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `analysis : const Analysis &`
  Riferimento all'oggetto di tipo *Analysis*, le cui informazioni vengono aggiunte al database.

- `+ setComplete(id : int) : boolean`

Descrizione: metodo che assegna il valore di completato all'oggetto *Analysis* avente l'id passato come argomento.

Argomenti

- `id : int`
  Id dell'analisi a cui assegnare il valore complete.

- `+ getAnalysis(id : int) : Analysis *`

Descrizione: metodo che ritorna un puntatore ad un oggetto *Analysis*, avente come id il valore passato.

Argomenti

- `id : int`
  Id dell'analisi che si vuole ottenere.

Note

- Il metodo deve essere marcato come costante.

- `+ getAllAnalysis() : QVector<Analysis*>`

Descrizione: metodo che ritorna un vettore contenente tutte le analisi effettuate dall'utente.

Note

- Il metodo deve essere marcato come costante.

- `+ getSelectedSubject(id : int) : QVector<ASubject* >`

Descrizione: metodo che ritorna un vettore contenente i $\text{Subject}_{\mathbf{G}}$ selezionati nell'analisi avente l'id dato.

Argomenti

- `id : int`
  Id dell'analisi di cui si vogliono conoscere i $\text{Subject}_{\mathbf{G}}$ selezionati.

Note

- Il metodo deve essere marcato come costante.

- `+ getSelectedSubjectString(id : int) : QVector<QString>`

Descrizione: metodo che ritorna l'elenco dei nomi dei Subject$_G$ selezionati dall'utente nell'analisi avente l'id passato.

Argomenti

- `id : int`
  Id dell'analisi di cui si vogliono conoscere i nomi del Subject$_G$ selezionati.

Note

- Il metodo deve essere marcato come costante.

- `+ getFeatShow(id : int) : QMap<int, bool>`

Descrizione: metodo che ritorna una mappa contenente per ogni feature un booleano che rappresenta se la feature$_G$ è stata mostrata o meno.

Argomeni

- `id : int`
  Id dell'analisi di cui si vogliono ottenere le feature$_G$ mostrate.

Note

- Il metodo deve essere marcato come costante.

- `+getFeatSave(id : int) : QMap<int, bool>`

Descrizione: metodo che ritorna una mappa contenente per ogni feature$_G$ un booleano che rappresenta se la feature$_G$ è stata salvata o meno.

Argomeni

- `id : int`
  Id dell'analisi di cui si vogliono ottenere le feature$_G$ esportate.

Note

- Il metodo deve essere marcato come costante.

- `+ getDatasetname(id : int) : QString`

Descrizone: metodo che ritorna il nome del Dataset$_G$ associato all'analisi avente l'id passato.

Argomenti

- `id : int`
  Id dell'analisi di cui si vole ottenere il Dataset$_{\mathbf{G}}$.

4.8.4 DatasetDAO(class)

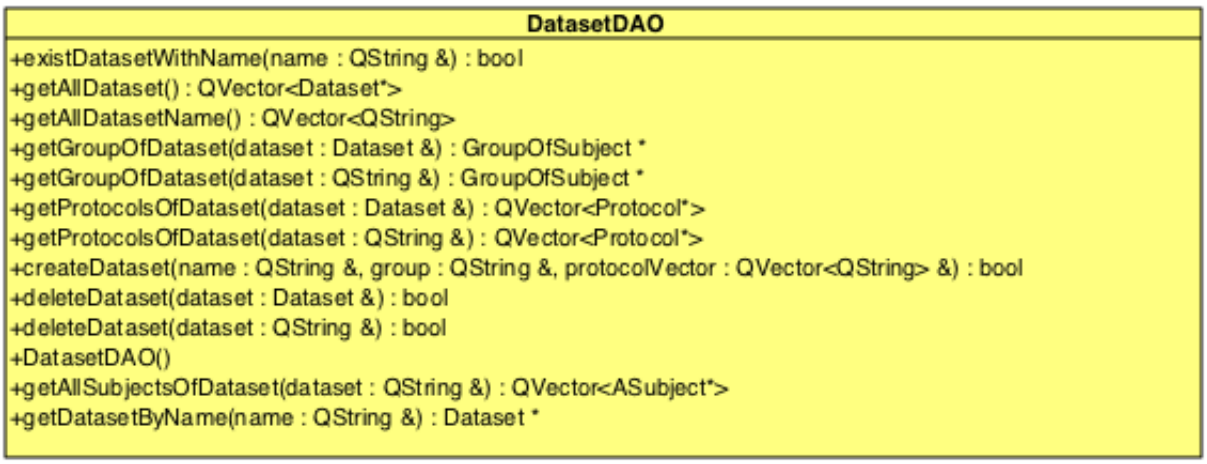

Figura 63: Classe DatasetDAO: attributi e metodi

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella Dataset$_G$ del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Dataset$_G$ creati dall'utente o utilizzati da Romeo.

Classi ereditate

- ADatabase.

Metodi

`+ DatasetDAO()`

Costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

`+ existSubjectWithName(name : const QString &)`

Metodo pubblico che controlla se all'interno del database è già presente un Dataset$_G$ avente il nome passato come parametro.

Argomenti

- `name : const QString &`
  Riferimento al nome del Dataset$_G$ del quale si vuole verificare l'esistenza.

`+ getAllDataset(): QVector<Dataset*>`

Metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo Dataset, contenente tutti i Dataset$_G$ presenti nel database.

`+ getAllDatasetName(): QVector<QString>`

Metodo pubblico che ritorna un QVector di QString, contenente il nome tutti i Dataset$_G$ presenti nel database.

`+ getGroupOfDataset(dataset : const Dataset &): QVector<GroupOfSubject*>`

Metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo GroupOfSubject, contenente tutti i gruppi di Subject$_{\mathbf{G}}$ membri del Dataset$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `dataset : const Dataset &`
  Riferimento al Dataset$_{\mathbf{G}}$ del quale si vogliono ricavare i gruppi membri.

`+ getProtocolOfDataset(dataset : const Dataset &): QVector<Protocol*>`

Metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo Protocol, contenente tutti i Protocol$_{\mathbf{G}}$ membri del Dataset$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `dataset : const Dataset &`
  Riferimento al Dataset$_{\mathbf{G}}$ del quale si vogliono ricavare i Protocol$_{\mathbf{G}}$ membri.

`+ createDataset(dataset : const Dataset &)`

Metodo pubblico che aggiunge nel database un nuovo Dataset$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `dataset : const Dataset &`
  Riferimento al Dataset$_{\mathbf{G}}$ che si vuole inserire nel database.

`+ deleteDataset(dataset : const Dataset &)`

Metodo pubblico che elimina dal database il Dataset$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `dataset : const Dataset &`
  Riferimento al Dataset$_{\mathbf{G}}$ che si vuole eliminare dal database.

`+ addProtocol(dataset : const Dataset &, protocol : const Protocol &)`

Metodo pubblico che aggiunge un Protocol$_{\mathbf{G}}$ ad un Dataset$_{\mathbf{G}}$ già esistente all'interno del database.

Argomenti

- `dataset : const Dataset &`
  Riferimento al Dataset$_{\mathbf{G}}$ al quale si vuole aggiungere un Protocol$_{\mathbf{G}}$.
- `protocol : const Protocol &`
  Riferimento al Protocol$_{\mathbf{G}}$ da aggiungere.

4.8.5 FeatureDAO(class)

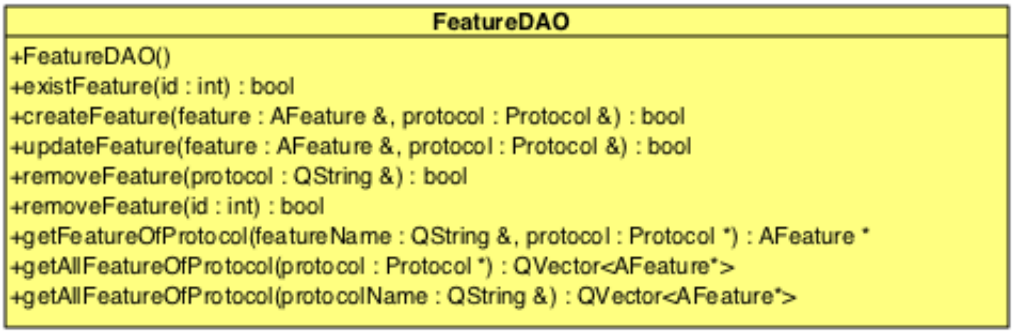

Figura 64: Diagramma classe *FeatureDAO*

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella Feature$_{\mathbf{G}}$ del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti le feature$_{\mathbf{G}}$ create dall'utente o utilizzate da Romeo.

Classi ereditate

- Romeo::Model::Util::DAO::ADatabase.

Metodi

- `+ FeatureDAO()`

  Descrizione: costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

- `+ existFeature(id : int) : boolean`

  Desrizione: metodo che controlla se esiste già un algoritmo con l'id passato.

  Argomenti

  - `id : int`
    L'id della feature da centrare.

  Note

  - Il metodo deve essere marcato costante.

- `+ createFeature(feature : AFeature*, protocol: Protocol*): boolean`

Descrizione: metodo pubblico che aggiunge nel database una nuova Feature$_{\mathbf{G}}$ passata come parametro e simultaneamente la associa al Protocol$_{\mathbf{G}}$ passato come parametro. Ritorna true se l'inserimento avviene correttamente.

Argomenti

- `feature : AFeature*`
  Puntatore all'oggetto *AFeature* che deve essere aggiunto nel database;
- `protocol: Protocol*`
  Puntatore all'oggetto *Protocol* a cui deve essere aggiunta la feature$_{\mathbf{G}}$.

- `+ updateFeature(feature : AFeature &, protocol : Protocol &) : boolean`

Descrizione: metodo che aggiorna una feature$_{\mathbf{G}}$ presente nel database.

Argomenti

- `feature : AFeature &`
  Rappresenta la feature$_{\mathbf{G}}$ da aggiornare;
- `protocol : Protocol &`
  Rappresenta il Protocol$_{\mathbf{G}}$ associato alla feature$_{\mathbf{G}}$.

- `+ removeFeature(protocol : const QString &): boolean`

Descrizione: metodo pubblico che rimuove nel database tutte le Feature$_{\mathbf{G}}$ membre del Protocol$_{\mathbf{G}}$ avente il nome passato come parametro. Ritorna true se l'eliminazione va a buon fine.

Argomenti

- `protocol : const QString &`
  Nome del Protocol di cui si vogliono eliminare le feature$_{\mathbf{G}}$.

- `+removeFeature(id : const int) : boolean`

Descrizione: metodo pubblico che rimuove dal database la Feature$_{\mathbf{G}}$ avente l'id passato. Ritorna `true` se l'eleiminazione va a buon fine.

Argomenti

- `id : const int)`
  Rappresenta l'id della feature$_{\mathbf{G}}$ che si vuole eliminare.

- `+ getFeatureOfProtocol(featureName : const QString &, protocol : Protocol *) : AFeature *`

Descrizione: metodo pubblico che ritorna un puntatore ad un oggetto di tipo *AFeature* che rappresenta la feature$_{\mathbf{G}}$ avente il nome passato come parametro e facente parte del Protocol$_{\mathbf{G}}$.

Argomenti

- `featureName : const QString &`
  Riferimento al nome della feature$_{\mathbf{G}}$ che si vuole cercare;
- `protocol : const Protocol*`
  Puntatore ad un oggetto di tipo Protocol$_{\mathbf{G}}$ nel quale si vuole cercare una Feature$_{\mathbf{G}}$ avente il nome passato come parametro.

- `+ getAllFeatureOfProtocol(protocol : Protocol*): QVector<AFeature*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *AFeature*, esso contiene tutte le feature$_{\mathbf{G}}$ facenti parte del Protocol$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `protocol : const Protocol*`
  Puntatore ad un oggetto di tipo Protocol$_{\mathbf{G}}$, identifica il Protocol$_{\mathbf{G}}$ di cui si vogliono ottenere le feature$_{\mathbf{G}}$.

- `+ getAllFeatureOfProtocol(protocolName : const QString &): QVector<AFeature*>`

Descrizione: metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo AFeature, esso contiene tutte le feature$_{\mathbf{G}}$ facenti parte del Protocol$_{\mathbf{G}}$ avente il nome uguale alla QString passata come parametro.

Argomenti

- `protocolName : const QString &`
  Riferimento all'oggetto di tipo QString che rappresenta il nome del Protocol$_{\mathbf{G}}$ del quale si vogliono ottenere le feature$_{\mathbf{G}}$.

4.8.6 GroupDAO(class)

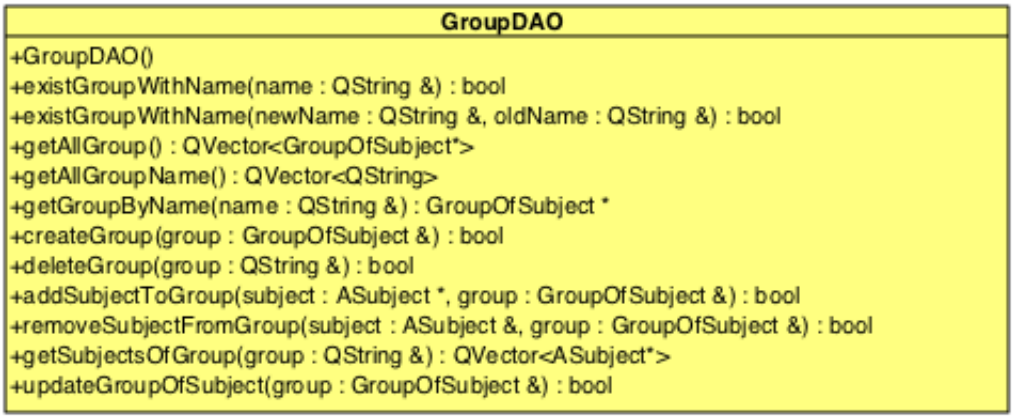

Figura 65: Diagramma classe *GroupDAO*

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella GroupOfSubject del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti i gruppi di $\text{Subject}_{\mathbf{G}}$ creati dall'utente o utilizzati da Romeo.

Classi ereditate

- Romeo::Model::Util::DAO::ADatabase.

Metodi

- `+ GroupDAO()`

  Descrizione: costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

- `+ existGroupWithName(name : const QString &): boolean`

  Descrizione: metodo pubblico che controlla se all'interno del database è presente un gruppo di $\text{Subject}_{\mathbf{G}}$ aventi il nome passato come parametro. In caso affermativo il metodo ritorna true, altrimenti false.

  Argomenti

  - `name : const QString &`
    Riferimento ad un oggetto *QString* che rappresenta il nome del Gruppo di $\text{Subject}_{\mathbf{G}}$ del quale si vuole controllare la presenza nel database.

- `+ getAllGroup(): QVector<GroupOfSubjec*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *GroupOfSubject*, i quali rappresentano tutti i gruppi di subject presenti nel database.

- `+ getAllGroup(): QVector<QString>`

Descrizone: metodo pubblico che ritorna un *QVector* di oggetti di tipo *QString*, i quali rappresentano il nome di ogni gruppo di subject$_{\mathbf{G}}$ presente nel database.

- `+ getGroupByName(name : const QString &): GroupOfSubject*`

Descrizione: metodo pubblico che ritorna un puntatore ad un oggetto *GroupOfSubject*, il quale rappresenta il gruppo di Subject$_{\mathbf{G}}$ avente il nome passato come parametro al metodo, recuperato dal database.

Argomenti

- `name : const QString &`
  Riferimento ad un oggetto di tipo *QString*, il quale rappresenta il nome del gruppo di Subject$_{\mathbf{G}}$ che si vuole recuperare dal database.

- `+ createGroup(group :const GroupOfSubject &): boolean`

Descrizione: metodo pubblico che aggiunge nel database un nuovo gruppo di Subject$_{\mathbf{G}}$ passato come parametro. Ritorna true se l'operazione è andata a buon fine, false altrimenti.

Argomenti

- `group : const GroupOfSubject &`
  Riferimento al gruppo di Subject$_{\mathbf{G}}$ che si vuole inserire nel database

- `+ deleteGroup(group : const QString &): boolean`

Descrizione: metodo pubblico che elimina dal database un gruppo di Subject$_{\mathbf{G}}$ avente il nome uguale alla QString passata come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `group : const QString &`
  Riferimento ad un oggetto di tipo *QString* il quale rappresenta il nome del gruppo di Subject$_{\mathbf{G}}$ da eliminare.

- `+ addSubjectToGroup(subject : ASubject*, group : const GroupOfSubject &): boolean`

Descrizione: metodo pubblico che aggiunge un Subject$_{\mathbf{G}}$, il quale viene passato come parametro, ad un gruppo di Subject$_{\mathbf{G}}$, anch'esso passato come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `subject : ASubject*`
  Puntatore ad un oggetto di tipo *ASubject*, il quale rappresenta il Subject$_{\mathbf{G}}$ da aggiungere al gruppo;
- `group : const GroupOfSubject &`
  Riferimento ad un oggetto di tipo *GroupOfSubject*, il quale rappresenta il gruppo a cui aggiungere il Subject$_{\mathbf{G}}$.

- `+ removeSubjectFromGroup(subject : ASubject*, group : const GroupOfSubject &): boolean`

Descrizione: metodo pubblico che rimuove un Subject$_{\mathbf{G}}$, il quale viene passato come parametro, da un gruppo di Subject$_{\mathbf{G}}$, anch'esso passato come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `subject : const ASubject*`
  Puntatore ad un oggetto di tipo *ASubject*, il quale rappresenta il Subject$_{\mathbf{G}}$ da rimuovere dal gruppo;
- `group : const GroupOfSubject &`
  Riferimento ad un oggetto di tipo *GroupOfSubject*, il quale rappresenta il gruppo da cui rimuovere il Subject$_{\mathbf{G}}$.

- `+ getSubjectOfGroup(group : const QString &): QVector<ASubject*>`

Descrizione: metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo *ASubject*, i quali rappresentano tutti i Subject$_{\mathbf{G}}$ che sono membri del gruppo di Subject$_{\mathbf{G}}$ passato come parametro.

Argomenti

- `group : const QString &`
  Riferimento ad un oggetto di tipo QString, il quale rappresenta il nome del gruppo di cui ottenere i membri.

- `+ updateGroupOfSubject(GroupOfSubject & : group) : boolean`

Descrizione: metodo che esegue l'aggiornamento di un gruppo di subject$_{\mathbf{G}}$. Il metodo ritorna un booleano che indica se l'operazione è andata a buon fine.

Argomenti

- `GroupOfSubject & : group)`
  Il gruppo di subject$_{\mathbf{G}}$ da aggiornare.

4.8.7 ProtocolDAO(class)

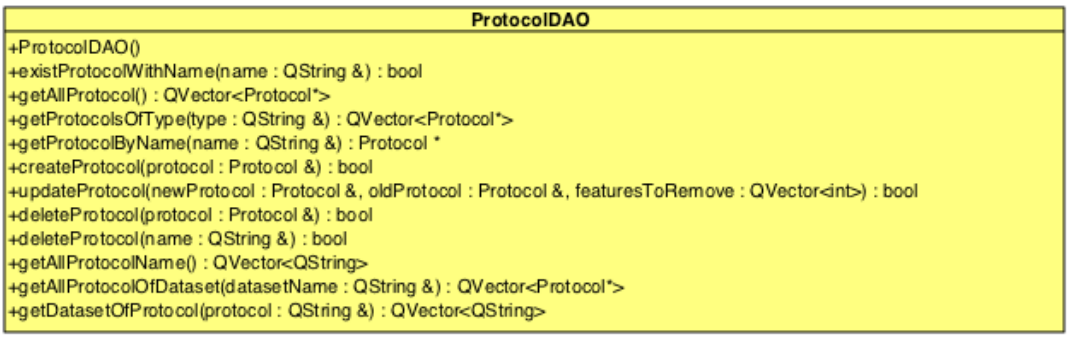

Figura 66: Diagramma classe *ProtocolDAO*

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella Protocol$_G$ del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Protocol$_G$ creati dall'utente o utilizzati da Romeo.

Classi ereditate

- Romeo::Model::Util::DAO::ADatabase.

Metodi

- `+ ProtocolDAO()`

  Descrizione: costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

- `+ existProtocolWithName(name : const QString &): boolean`

  Descrizione: metodo pubblico che controlla se all'interno del database è presente un Protocol$_G$ avente il nome passato come parametro. In caso affermativo il metodo ritorna true, altrimenti false.

  Argomenti

  - `name : const QString &`
    Riferimento ad un oggetto *QString* che rappresenta il nome del Protocol$_G$ del quale si vuole controllare la presenza nel database.

- `+ getAllProtocol(): QVector<Protocol*>`

  Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo Protocol$_G$, i quali rappresentano tutti i Protocol$_G$ presenti nel database.

- `+ getProtocolsOfType(type : QString &) : QVector<Protocol*>`

Descrizione: metodo che ritorna un vettore di oggetti *Protocol*, contenente tutti i Protocol$_{\mathbf{G}}$ di un determinato tipo (2D, 2D-T, 3D, 3D-t).

Argomenti

- `type : QString &`
  Tipo dei Protocol$_{\mathbf{G}}$ che si vogliono ottenere.

- `+ getProtocolByName(name : const QString &): Protocol*`

Descrizione: metodo pubblico che ritorna un puntatore ad un oggetto di tipo Protocol$_{\mathbf{G}}$, il quale rappresenta il Protocol$_{\mathbf{G}}$ avente il nome uguale alla QString passata come parametro, recuperato dal database.

Argomenti

- `name : const QString &`
  Riferimento ad un oggetto di tipo *QString* che rappresenta il nome del Protocol$_{\mathbf{G}}$ da cercare nel database.

- `+ createProtocol(protocol : const Protocol &): boolean`

Descrizione: metodo pubblico che consente l'inserimento nel database, del Protocol$_{\mathbf{G}}$ passato come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `protocol : const Protocol &`
  Riferimento ad un oggetto di tipo Protocol$_{\mathbf{G}}$, il quale rappresenta il Protocol$_{\mathbf{G}}$ da inserire nel database.

- `+ updateProtocol(newProtocol : Protocol &, oldProtocol : Protocol &, featuresToRemove : QVector<int>) : boolean`

Descrizione: metodo che aggiorna un Protocol$_{\mathbf{G}}$.

Argomenti

- `newProtocol : Protocol &`
  Protocol$_{\mathbf{G}}$ con le nuove informazioni;
- `oldProtocol : Protocol &`
  Vecchio Protocol$_{\mathbf{G}}$ da aggiornare;
- `featuresToRemove : QVector<int>`
  Lista di feature$_{\mathbf{G}}$ da rimuovere.

- `+ deleteProtocol(protocol : const Protocol &): boolean`

Descrizione: metodo pubblico che consente l'eliminazione dal database, del Protocol$_{\mathbf{G}}$ passato come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti:

- `protocol : const Protocol &`
  Riferimento ad un oggetto di tipo Protocol$_{\mathbf{G}}$, il quale rappresenta il Protocol$_{\mathbf{G}}$ da rimuovere dal database.

- `+ getAllProtocolName() : QVector<QString>`

Descrizione: metodo pubblico che ritorna un vettore di oggetti *QString* contenente l'elenco dei nomi dei vari Protocol$_{\mathbf{G}}$ esistenti.

- `+ getAllProtocolOfDataset(datasetName : const QString &): QVector<Protocol*>`

Descrizione: metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo Protocol$_{\mathbf{G}}$, i quali rappresentano tutti i Protocol$_{\mathbf{G}}$ che sono membri del Dataset$_{\mathbf{G}}$ avente il nome uguale al parametro passato.

Argomenti

- `datasetName : const QString &`
  Riferimento ad un oggetto di tipo *QString*, il quale rappresenta il nome del Dataset$_{\mathbf{G}}$ del quale ottenere i Protocol$_{\mathbf{G}}$.

- `+ getDatasetOfProtocol(protocol : const QString &) : QVector<QString>`

Descrizione: metodo pubblico che ritorna un QVector di puntatori ad oggetti di tipo QString contentente i nomi dei dataset associati col Protocol$_{\mathbf{G}}$ passato.

Argomenti

- `protocol : const QString &`
  Nome del protocol$_{\mathbf{G}}$ di cui si vogliono conoscere i dataset$_{\mathbf{G}}$ associati.

4.8.8 SubjectDAO(class)

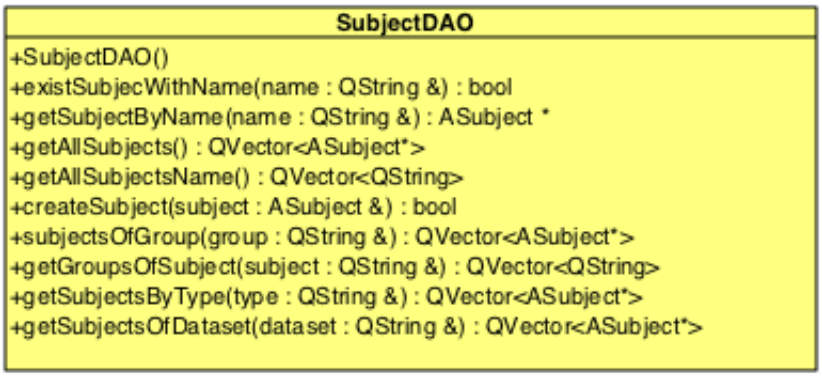

Figura 67: Diagramma classe *SubjectDAO*

Descrizione

Classe che rappresenta l'oggetto incaricato di operare con la tabella Subject$_G$ del database.

Utilizzo

La classe verrà utilizzata dal core quando dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Subject$_G$ creati dall'utente o utilizzati da Romeo.

Classi ereditate

- Romeo::Model::Util::DAO.

Metodi

- `+SubjectDAO()`

  Descrizione: costruttore pubblico che richiama il costruttore della superclasse in modo da stabilire una connessione con il database.

- `+ existSubjectlWithName(name : const QString &): boolean`

  Descrizione: metodo pubblico che controlla se all'interno del database è presente un Subject$_G$ avente il nome passato come parametro. In caso affermativo il metodo ritorna true, altrimenti false.

  Argomenti:

  - `name : const QString &`
    Riferimento ad un oggetto *QString* che rappresenta il nome del Subject$_G$ del quale si vuole controllare la presenza nel database.

  Note

  - Il metodo deve esseere marcato come costante.

- `+ getSubjectByName(name : const QString &): ASubject*`

Descrizione: metodo pubblico che ritorna un puntatore ad un oggetto *ASubject*, il quale rappresenta il Subject$_{\mathbf{G}}$ avente il nome passato come parametro al metodo, recuperato dal database.

Argomenti

- `name : const QString &`
  Riferimento ad un oggetto di tipo QString, il quale rappresenta il nome del Subject$_{\mathbf{G}}$ che si vuole recuperare dal database.

Note

- Il metodo deve esseere marcato come costante.

- `+ getAllSubject(): QVector<ASubject*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *ASubject*, i quali rappresentano tutti i Subject$_{\mathbf{G}}$ presenti nel database.

Note

- Il metodo deve esseere marcato come costante.

- `+ getAllSubjectName(): QVector<QString>`

Descrizione: metodo pubblico che ritorna un *QVector* di oggetti di tipo *QString*, i quali rappresentano il nome di ogni Subject$_{\mathbf{G}}$ presente nel database.

Note

- Il metodo deve esseere marcato come costante.

- `+ createSubject(subject : const ASubject &): boolean`

Descrizione: metodo pubblico che aggiunge nel database un nuovo Subject$_{\mathbf{G}}$ passato come parametro. Esso ritorna true se l'operazione va a buon fine, false altrimenti.

Argomenti

- `subject : const ASubject &`
  Riferimento al Subject$_{\mathbf{G}}$ che si vuole inserire nel database.

- `+ subjectsOfGroup(group : const QString &): QVector<ASubject*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *ASubject*, i quali rappresentano i Subject$_{\mathbf{G}}$ che fanno parte del gruppo di Subject$_{\mathbf{G}}$ avente il nome uguale alla *QString* passata come parametro.

Argomenti

- `group : const QString &`
  Riferimento ad un oggetto di tipo *QString* che rappresenta il nome del gruppo del quale si vogliono ricavare i Subject$_{\mathbf{G}}$

Note

- Il metodo deve esseere marcato come costante.

- `+ getGroupOfSubject(subject : const QString &): QVector<GroupOfSubject*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *GroupOfSubject*, i quali rappresentano i gruppi di $\text{Subject}_{\mathbf{G}}$ di cui è membro il $\text{Subject}_{\mathbf{G}}$ avente il nome uguale alla stringa passata come parametro.

Argomenti

- `subject : const QString &`
  Riferimento ad un oggetto di tipo QString che rappresenta il nome del $\text{Subject}_{\mathbf{G}}$ del quale si vogliono ricavare i gruppi a cui esso appartiene.

Note

- Il metodo deve esseere marcato come costante.

- `+ getSubjectsByType(type : const QString &): QVector<ASubject*>`

Descrizione: metodo pubblico che ritorna un *QVector* di puntatori ad oggetti di tipo *ASubject*, i quali rappresentano i $\text{Subject}_{\mathbf{G}}$ aventi immagini del tipo passato come parametro.

Argomenti

- `type : const QString &`
  Riferimento ad un oggetto di tipo *QString*, il quale rappresenta il tipo dei $\text{Subject}_{\mathbf{G}}$ che si vogliono recuperare dal database.

Note

- Il metodo deve esseere marcato come costante.

- `+ getSubjectsOfDataset(dataset : const QString &) : QVector<ASubject*>`

Descrizione: metodo che ritorna un vettore di puntatori a oggetti *ASubject*, contenente la lista di $\text{Subject}_{\mathbf{G}}$ associati ad un determinato $\text{dataset}_{\mathbf{G}}$.

Argomenti

- `dataset : const QString &`
  Nome del $\text{dataset}_{\mathbf{G}}$ di cui si vogliono conoscere i $\text{subject}_{\mathbf{G}}$ associati.

Note

- Il metodo deve esseere marcato come costante.

4.9 Specifica componenti Romeo::Model::Help

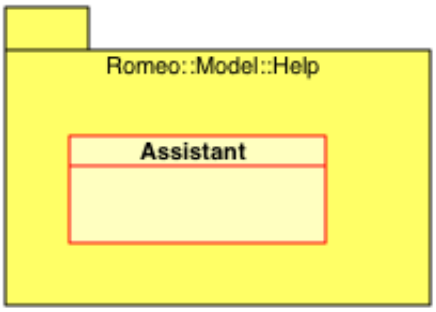

Figura 68: Diagramma package *Romeo::Model::Help*

Package$_{\mathbf{G}}$ che contiene la classe che si occupa della gestione logica della guida utente.

4.9.1 Assistant (class)

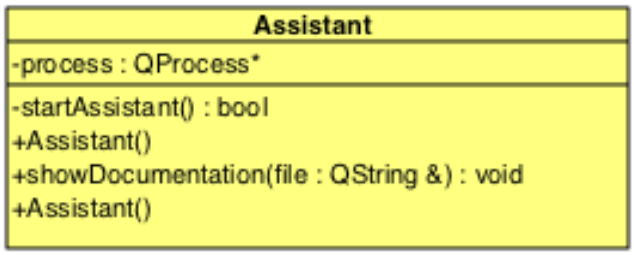

Figura 69: Diagramma classe *Assistanti*

Descrizione:
Classe che rappresenta il processo Qt Assistant, utilizzato per la gestione della guida utente.

Utilizzo
Viene utilizzata per gestire la richiesta dell'utente, che vuole visualizzare una determinata pagina all'interno della guida utente.

Eredita da:

- Qt::QProcess

Attributi

- `- process : QProcess *`

  Descrizione: puntatore al processo esterno Qt Assistant.

Metodi

- `+ Assistant()`

Descrizione: costruttore per un oggetto di tipo *Assistant*.

- `- startAssistant() : boolean`

Descrizione: metodo che controlla se il processo esterno associato a Qt Assistant è già attivo, in caso negativo si preoccupa di farlo partire.
Ritorna un booleano per indicare se il processo è attivo o no.

- `+ showDocumentation(file : const QString &) : void`

Descrizione: metodo che si occupa di visualizzare nell'help di Qt Assistant la pagina `HTML` passata come argomento.

Argomenti

– `file : const QString &`
Rappresenta il path della pagina `HTML` da visualizzare nell'help di Qt Assistant.

4.10 Specifica componenti Model::QtModel

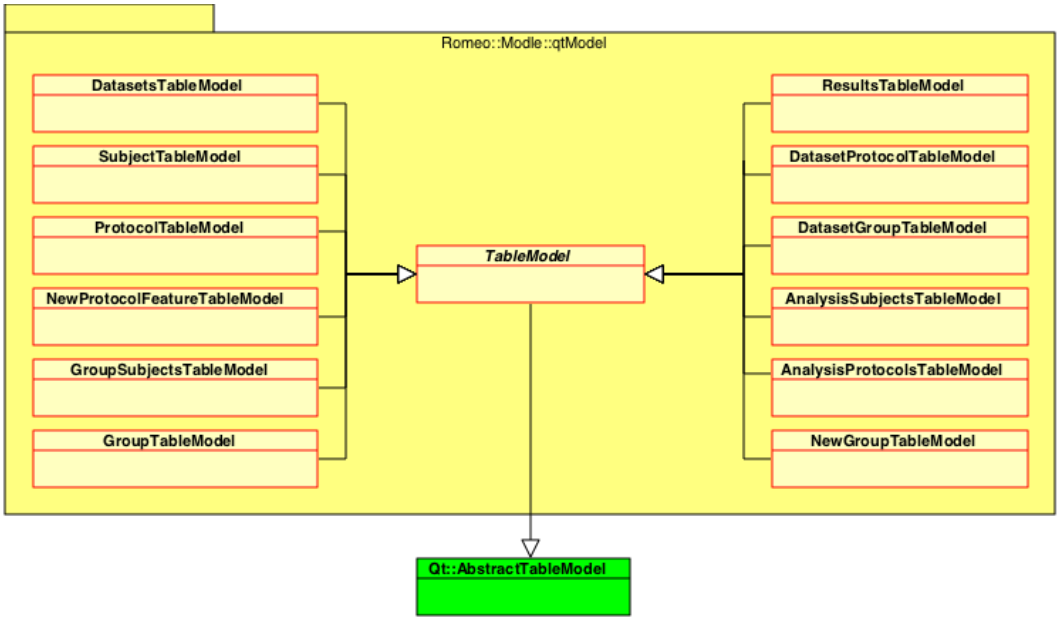

Figura 70: Diagramma package *Romeo::Model::QtModel*

Package$_{\mathbf{G}}$ per il componente Model dell'architettura Model/View di Qt$_{\mathbf{G}}$.

4.10.1 TableModel(abstract)

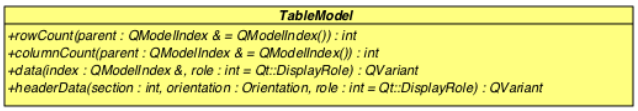

Figura 71: Diagramma classe *TableModel*

Descrizione

Classe astratta che eredita da Qt::QAbstractTableModel.

Utilizzo

La classe viene ereditata da tutte le classi che rappresentano i model delle tabelle visualizzate nelle view.

Classi ereditate:

- Qt::QAbstractTableModel.

Metodi

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale puro che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
  Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante puro e deve essere ridefinito dalle sottoclassi.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale puro che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante puro e deve essere ridefinito dalle sottoclassi.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale puro che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante puro e deve essere ridefinito dalle sottoclassi.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale puro che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante puro e deve essere ridefinito dalle sottoclassi.

4.10.2 AnalysisProtocolsTableModel(class)

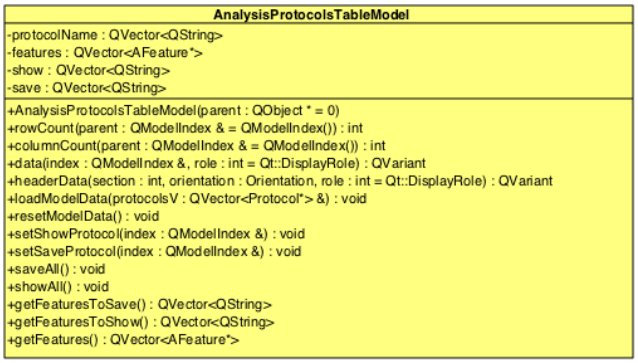

Figura 72: Diagramma classe *AnalysisProtocolsTableModel*

Descrizione

Classe che rappresenta il model della tabella delle feature$_{G}$ presente nella vista di avvio analisi.

Utilizzo

La classe viene utilizzata come model dalla tabella delle feature$_{G}$ della vista di avvio analisi.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- protocolName : QVector<QString>`

  Descrizione: È il vettore dei nomi dei protocol$_{G}$ presenti nel dataset$_{G}$.

- `- features : QVector<AFeature*>`

  Descrizione: È il vettore delle feature$_{G}$ che compone il model.

- `- show : QVector<QString>`

  Descrizione: È il vettore di si e di no per le feature$_{G}$ da mostrare durante l'analisi.

- `- save : QVector<QString>`

  Descrizione: È il vettore di si e di no per le feature$_{G}$ da esportare durante l'analisi.

Metodi

- `+ AnalysisProtocolsTableModel(parent : QObject*)`

Descrizione: costruttore della classe.

Argomenti

- `parent : QObject*`
  Parente dell'oggetto AnalysisProtocolsTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
Il metodo viene invocato automaticamente da $Qt_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna. Il metodo viene invocato automaticamente da $Qt_{\mathbf{G}}$ nella generazione del Model. Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da $Qt_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione $Qt_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da $\text{Qt}_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione $\text{Qt}_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ loadModelData(protocolsV : const QVector<Protocol*>&) : void`

Descrizione: Carica i dati del model in base al nome del $\text{dataset}_{\mathbf{G}}$ passato come parametro al metodo.

Argomenti

- `protocolsV : const QVector<Protocol*>&`
  Vettore dei $\text{protocol}_{\mathbf{G}}$ che andranno caricati nel model.

- `+ resetModelData() : void`

Descrizione: Elimina i dati nel model portandolo allo stato vuoto.

- `+ setShowProtocol(index : const QModelIndex &) : void`

Descrizione: Imposta la $\text{feature}_{\mathbf{G}}$ puntata dall'indice come da visualizzare o no durante l'analisi.

Argomenti

- `index : const QModelIndex &`
  Indice selezionato della tabella delle $\text{feature}_{\mathbf{G}}$.

- `+ setSaveProtocol(index : const QModelIndex &) : void`

Descrizione: Imposta la feature$_{\mathbf{G}}$ puntata dall'indice come da esportare o no durante l'analisi.

Argomenti

- `index : const QModelIndex &`
  Indice selezionato della tabella delle feature$_{\mathbf{G}}$.

- `+ saveAll() : void`

Descrizione: Imposta tutte le feature$_{\mathbf{G}}$ come da esportare durante l'analisi.

- `+ showAll() : void`

Descrizione: Imposta tutte le feature$_{\mathbf{G}}$ come da visualizzare durante l'analisi.

- `+ getFeaturesToSave() : QVector<QString>`

Descrizione: Il metodo ritorna il vettore delle feature$_{\mathbf{G}}$ da esportare durante l'analisi.

Note

- Il metodo è costante.

- `+ getFeaturesToShow() : QVector<QString>`

Descrizione: Il metodo ritorna il vettore delle feature$_{\mathbf{G}}$ da visualizzare durante l'analisi.

Note

- Il metodo è costante.

- `+ getFeatures() : QVector<AFeature*>`

Descrizione: Il metodo ritorna il vettore delle feature$_{\mathbf{G}}$ del model.

Note

- Il metodo è costante.

4.10.3 AnalysisSubjectsTableModel(class)

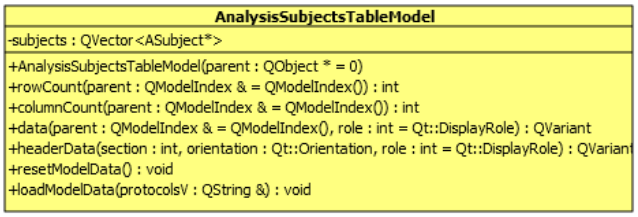

Figura 73: Diagramma classe *tAnalysisSubjectsTableModel*

Descrizione

Classe che rappresenta il model della tabella dei subject$_{G}$ presente nella vista di avvio analisi.

Utilizzo

La classe viene utilizzata come model dalla tabella dei subject$_{G}$ della vista di avvio analisi.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- subjects : QVector<ASubject*>`

  Descrizione: È il vettore dei subject$_{G}$ che compone il model.

Metodi

- `+ AnalysisSubjectsTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto AnalysisSubjectsTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{G}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da $\mathrm{Qt}_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione $\mathrm{Qt}_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ loadModelData(dataset : const QString&) : void`

Descrizione: Carica i dati del model in base al nome del $\mathrm{dataset}_{\mathbf{G}}$ passato come parametro al metodo.

Argomenti

- `dataset : const QString&`
  Nome del $\mathrm{dataset}_{\mathbf{G}}$.

- `+ resetModelData() : void`

Descrizione: Elimina i dati nel model portandolo allo stato vuoto.

4.10.4 DatasetGroupTableModel(class)

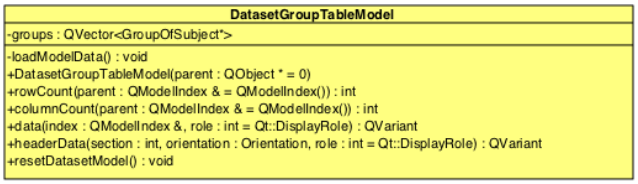

Figura 74: Diagramma classe *DatasetGroupTableModel*

Descrizione

Classe che rappresenta il model della tabella dei gruppi di subject$_{\mathbf{G}}$ presente nella vista di creazione del dataset$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei gruppi di subject$_{\mathbf{G}}$ della vista di creazione del dataset$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- groups : QVector<GroupOfSubject*>`

  Descrizione: È il vettore dei gruppi di subject$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ DatasetGroupTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto DatasetGroupTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ resetDatasetModel() : void`

Descrizione: Elimina i dati nel model portandolo allo stato vuoto.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

4.10.5 DatasetProtocolTableModel(class)

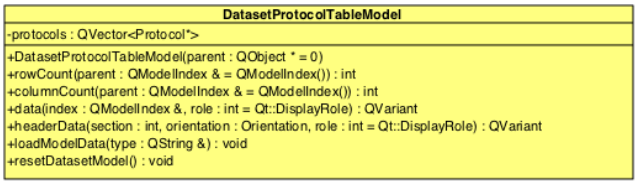

Figura 75: Diagramma classe *DatasetProtocolTableModel*

Descrizione

Classe che rappresenta il model della tabella dei protocol$_{\mathbf{G}}$ presente nella vista di creazione dei dataset$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei protocol$_{\mathbf{G}}$ della vista di creazione dei dataset$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- protocols : QVector<Protocol*>`

  Descrizione: È il vettore dei protocol$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ DatasetProtocolTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto DatasetProtocolTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ loadModelData(type : const QString&) : void`

Descrizione: Carica i dati del model in base al tipo passato come parametro al metodo.

Argomenti

- `type : const QString&`
  Tipo del protocol$_{\mathbf{G}}$.

- `+ resetDatasetModel() : void`

Descrizione: Elimina i dati nel model portandolo allo stato vuoto.

4.10.6 DatasetsTableModel(class)

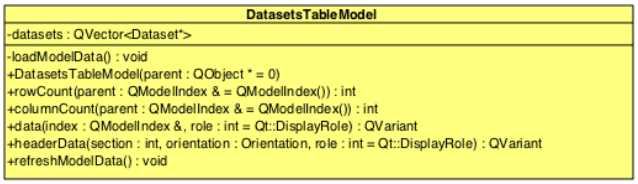

Figura 76: Diagramma classe *DatasetsTableModel*

Descrizione

Classe che rappresenta il model della tabella dei dataset$_{\mathbf{G}}$ presente nella vista di visualizzazione dei dataset$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei dataset$_{\mathbf{G}}$ della vista di visualizzazione dei dataset$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- datasets : QVector<Dataset*>`

  Descrizione: È il vettore dei dataset$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ DatasetsTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto DatasetsTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt_{G} nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt_{G} nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt_{G}.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ refreshModelData() : void`

Descrizione: Ricarica i dati del model.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

4.10.7 GroupSubjectsTableModel(class)

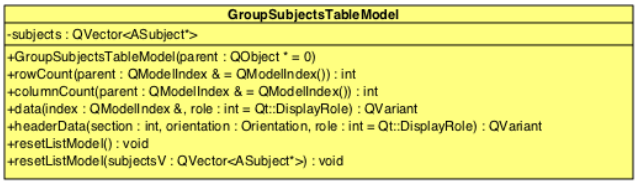

Figura 77: Diagramma classe *GroupSubjectsTableModel*

Descrizione

Classe che rappresenta il model della tabella dei subject$_G$ presente nella vista di visualizzazione dei gruppi di subject$_G$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei subject$_G$ della vista di visualizzazione dei gruppi di subject$_G$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- subjects : QVector<ASubject*>`

  Descrizione: È il vettore dei subject$_G$ che compone il model.

Metodi

- `+ GroupSubjectsTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto GroupSubjectsTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_G$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ resetListModel(subjectsV : QVector<ASubject*>) : void`

Descrizione: Carica i dati del model con i dati passati come parametro al metodo.

Argomenti

- `subjectsV : QVector<ASubject*>`
  Vettore di subject$_{\mathbf{G}}$.

- `+ resetListModel() : void`

Descrizione: Ricarica i dati nel model.

4.10.8 GroupTableModel(class)

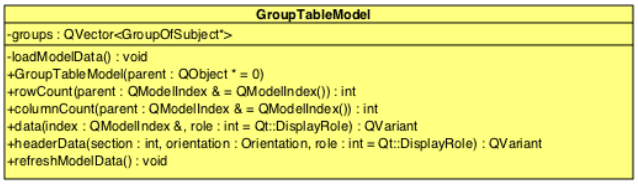

Figura 78: Diagramma classe *GroupTableModel*

Descrizione

Classe che rappresenta il model della tabella dei gruppi di subject$_{\mathbf{G}}$ presente nella vista di visualizzazione dei gruppi di subject$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei gruppi di subject$_{\mathbf{G}}$ della vista di visualizzazione dei gruppi di subject$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- groups : QVector<GroupOfSubject*>`

  Descrizione: È il vettore dei gruppi di subject$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ GroupTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto GroupTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ refreshModelData() : void`

Descrizione: Ricarica i dati del model.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

4.10.9 NewGroupTableModel(class)

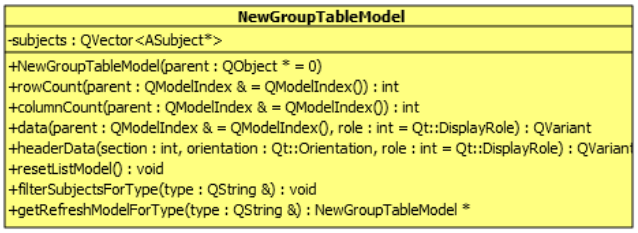

Figura 79: Diagramma classe *NewGroupTableModel*

Descrizione

Classe che rappresenta il model della tabella dei subject$_G$ presente nella vista di creazione dei gruppi di subject$_G$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei subject$_G$ della vista di creazione dei gruppi di subject$_G$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- subjects : QVector<ASubject*>`

  Descrizione: È il vettore dei subject$_G$ che compone il model.

Metodi

- `+ NewGroupTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto NewGroupTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_G$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ filterSubjectsForType(type : const QString&) : void`

Descrizione: Carica i dati dei subject$_{\mathbf{G}}$ nel model in base al tipo passato come parametro al metodo.

Argomenti

- `type : const QString&`
  Tipo del subject$_{\mathbf{G}}$.

- `+ resetListModel() : void`

Descrizione: Elimina i dati nel model portandolo allo stato vuoto.

- `+ getRefreshModelForType(type : const QString&) : NewGroupTableModel*`

Descrizione: Carica i dati dei subject$_{\mathbf{G}}$ nel model in base al tipo passato come parametro al metodo e ritorna il puntatore al model.

Argomenti

- `type : const QString&`
  Tipo del subject$_{\mathbf{G}}$.

4.10.10 NewProtocolFeatureTableModel(class)

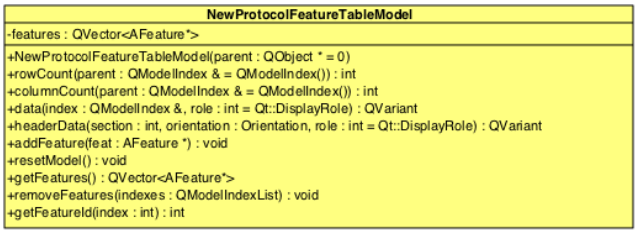

Figura 80: Diagramme classe *NewProtocolFeatureTableModel*

Descrizione

Classe che rappresenta il model della tabella delle feature$_{\mathbf{G}}$ presente nella vista di creazione dei protocol$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella delle feature$_{\mathbf{G}}$ della vista di creazione dei protocol$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- features : QVector<AFeature*>`

  Descrizione: È il vettore delle feature$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ NewProtocolFeatureTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto NewProtocolFeatureTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt_G nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt_G.

Note

- Il metodo è costante.

- `+ addFeature(feat : AFeature*) : void`

Descrizione: Il metodo aggiunge la $feature_G$ al model.

Argomenti

- `feat : AFeature*`
  È la $feature_G$ da aggiungere.

- `+ resetModel() : void`

Descrizione: Il metodo svuota il model.

- `+ getFeatures() : QVector<AFeature*>`

Descrizione: Il metodo ritorna il vettore di tutte le $feature_G$ del model.

Note

- Il metodo è costante.

- `+ removeFeatures(indexes : QModelIndexList) : void`

Descrizione: Il metodo rimuove le $feature_G$ puntate dagli indici dal model.

Argomenti

- `indexes : QModelIndexList`
  È la lista degli indici della tabella.

- `+ getFeatureId(index : int) : void`

Descrizione: Il metodo ritorna l'ID della feature$_G$ nella posizione index del vettore del model.

Argomenti

- `index : int`
  È la posizione del vettore.

Note

- Il metodo è costante.

4.10.11 ProtocolTableModel(class)

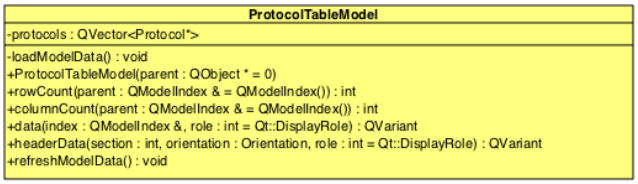

Figura 81: Diagramma classe *ProtocolTableModel*

Descrizione

Classe che rappresenta il model della tabella dei protocol$_G$ presente nella vista di visualizzazione dei protocol$_G$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei protocol$_G$ della vista di visualizzazione dei protocol$_G$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- protocols : QVector<Protocol*>`

  Descrizione: È il vettore dei protocol$_G$ che compone il model.

Metodi

- `+ ProtocolTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto ProtocolTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_G$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ refreshModelData() : void`

Descrizione: Ricarica i dati del model.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

4.10.12 ResultsTableModel(class)

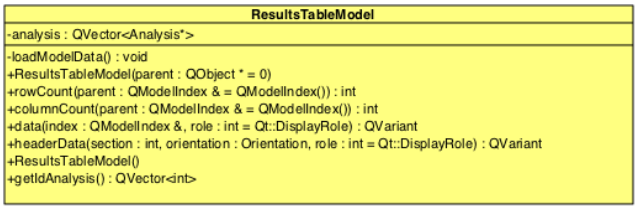

Figura 82: Diagramma classe *ResultsTableModel*

Descrizione

Classe che rappresenta il model della tabella delle analisi effettuate presente nella vista di visualizzazione delle analisi.

Utilizzo

La classe viene utilizzata come model dalla tabella delle analisi effettuate della vista di visualizzazione analisi.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- analysis : QVector<Analysis*>`

  Descrizione: È il vettore di analisi che compone il model.

Metodi

- `+ ResultsTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto ResultsTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ getIdAnalysis() : QVector<int>`

Descrizione: Ritorna gli ID delle analisi presenti nel model.

Note

- Il metodo è costante.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

4.10.13 SubjectTableModel(class)

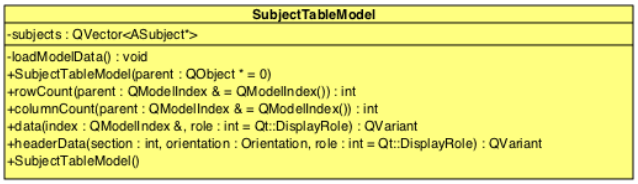

Figura 83: Diagramma classe *SubjectTableModel*

Descrizione

Classe che rappresenta il model della tabella dei subject$_{\mathbf{G}}$ presente nella vista di visualizzazione dei subject$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata come model dalla tabella dei subject$_{\mathbf{G}}$ della vista di visualizzazione subject$_{\mathbf{G}}$.

Classi ereditate:

- Romeo::Model::QtModel::TableModel.

Attributi

- `- subjects : QVector<ASubject*>`

  Descrizione: È il vettore di subject$_{\mathbf{G}}$ che compone il model.

Metodi

- `+ SubjectTableModel(parent : QObject*)`

  Descrizione: costruttore della classe.

  Argomenti

  - `parent : QObject*`
    Parente dell'oggetto SubjectTableModel.

- `+ rowCount(parent : const QModelIndex &) : int`

  Descrizione: metodo virtuale che ha come contratto il conteggio di numero di righe e lo ritorna.
  Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ columnCount(parent : const QModelIndex &) : int`

Descrizione: metodo virtuale che ha come contratto il conteggio di numero di colonne e lo ritorna.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresenta l'indice attuale del model.

Note

- Il metodo è costante.

- `+ data(index : const QModelIndex &, role: int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento dei dati nella riga della tabella, ritorna il testo da inserire nelle singole celle
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `parent : const QModelIndex &`
  Rappresena l'indice attuale del model;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `+ headerData(section : int, orientation : Qt::Orientation, role : int) : QVariant`

Descrizione: metodo virtuale che ha come contratto l'inserimento delle intestazioni della tabella, ritorna il testo da inserire in ogni intestazione.
Il metodo viene invocato automaticamente da Qt$_{\mathbf{G}}$ nella generazione del Model.
Il metodo deve essere ridefinito nelle classi che ereditano da questa.

Argomenti

- `section : int`
  Rappresenta l'indice della colonna o riga in base all'orientamento;
- `orientation : Qt::Orientation`
  Rappresenta l'orientamento della tabella;
- `role : int`
  Regola di visualizzazione Qt$_{\mathbf{G}}$.

Note

- Il metodo è costante.

- `- loadModelData() : void`

Descrizione: Carica i dati del model .

5 Specifica componenti Romeo::View

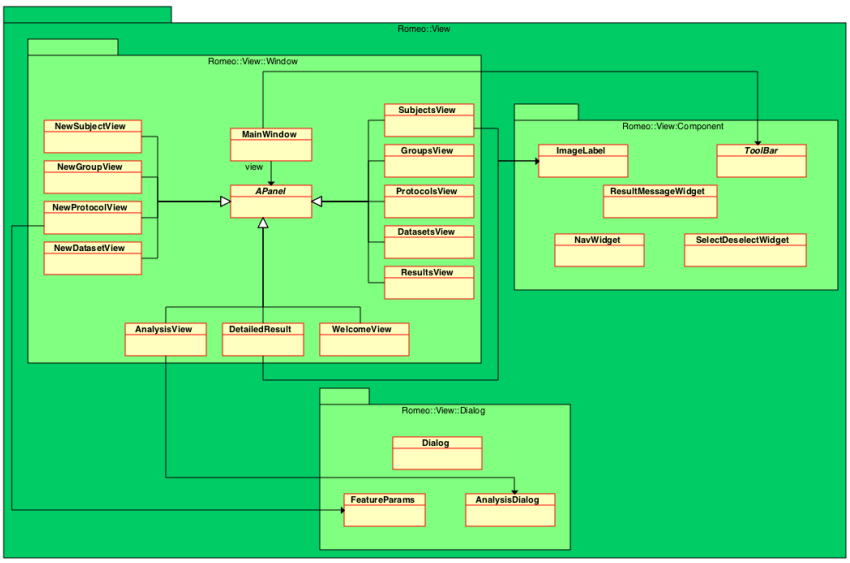

Figura 84: Componente Romeo::View

Package$_{G}$ per il componente View dell'architettura MVC.

5.1 Specifica componenti View::Window

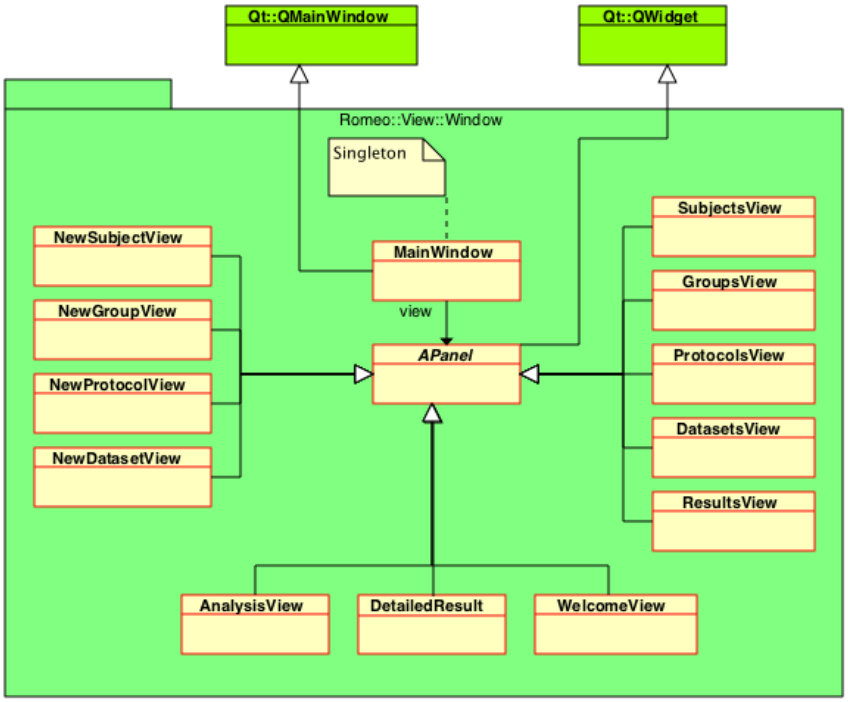

Figura 85: Componente Romeo::View::Window

Componente che contiene tutti i tipi di finestre disponibili nel sistema con cui l'utente potrà interagire per accedere alle funzionalità di Romeo.

5.1.1 MainWindow (class)

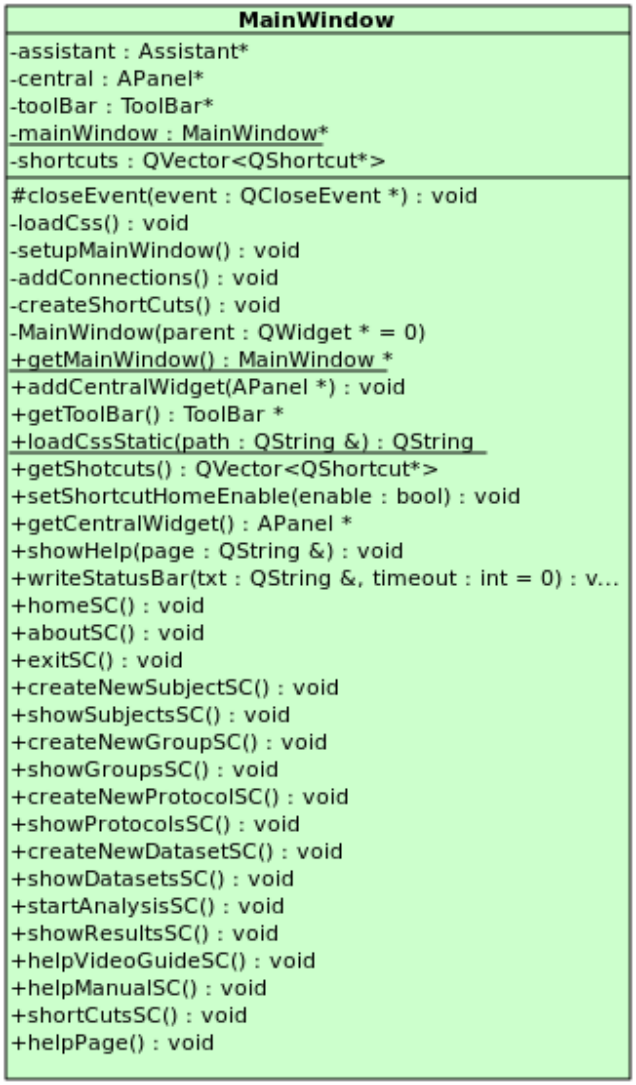

Figura 86: Diagramma Classe MainWindow: attributi e metodi

Descrizione

Classe che rappresenta la sola ed unica finestra attiva presente in Romeo. Dovendo essere l'unica istanza all'interno dell'applicativo, è stata implementata tramite design pattern$_G$ Singleton.

Utilizzo

La classe viene utilizzata per poter interagire con il core dell'applicazione mettendo a disposizione le varie funzionalità.

Classi ereditate

- Qt::QMainWindow.

Attributi

- `- central: Window::APanel*`

  Descrizione: Puntatore al widget che verrà impostato come centralWidget di MainWindow.

- `- shortcuts : QVector<QShortCut*>`

  Descrizione: contiene la lista di shortcut disponibili per Romeo.

- `- toolBar : Component::ToolBar*`

  Descrizione: Puntatore alla barra che darà la possibilità all'utente di filtrare e ordinare i dati.

- `- static mainWindow MainWindow*`

  Descrizione: Puntatore al campo dati statico che rappresenta l'unica istanza presente per la classe stessa, MainWindow che verrà istanziata in modo *lazy* quando verrà richiesta la creazione dell'oggetto.

Metodi

- `- MainWindow(parent : QWidget*=0)`

  Descrizione: Costruttore privato (come prevede il design pattern$_{G}$ Singleton) per la classe MainWindow.

  #### Argomenti

  - `parent : QWidget*=0`
    Puntatore al QWidget padre di MainWindow.

- `-loadCss():void` Metodo che carica il file contentente il css di default per MainWindow.

  #### Note

  - deve essere marcato costante.

- `-setupMainWindow(): void`

  Descrizione: Metodo che imposta la dimensione della finestra e la menuBar per la MainWindow.

- `-addConnections(): void`

  Descrizione: Metodo che si occupa di creare tutte le connessioni necessarie tra i vari oggetti che inviano un signal$_{G}$ e quelli che implementano lo slot$_{G}$ alla ricezione del signal$_{G}$.

Note:

- deve essere marcato costante.

- `-createShortCuts(): void`

Descrizione: Metodo che si occupa di creare gli shortcut disponibili.

- `+ static getMainWindow():MainWindow*`

Descrizione: Metodo statico che ritorna il puntatore all'unica istanza della classe MainWindow. Se l'istanza non esiste ancora, essa verrà creata e poi verrà ritornata.

Note

- deve essere un metodo statico.

- `+addCentralWidget(central : APanel*):void`

Descrizione: Metodo che imposta l'oggetto puntato da central come central widget di MainWindow; central deve puntare a un oggetto sottotipo di APanel.

Argomenti

- `central : APanel*`
  rappresenta il widget da settare come centralWidget di MainWindow.

- `+ getToolBar() : QToolBar*`

Descrizione: Metodo che ritorna un puntatore alla toolBar.

Note

- il metodo deve essere marcato come costante.

- `+ loadCssStatic(path : const QString&): QString`

Descrizione: Metodo che apre e legge il file, il cui nome è contenuto in path, che contiene le impostazioni stilistiche per l'applicativo Romeo e ritorna il contenuto del file

Note

- deve essere marcato statico;
- deve essere marcato costante.

Argomenti:

- `path : const QString\&`
  Stringa contenente il path del file da aprire e leggere.

- `+ getShortCuts() : QVector<QShortCut*>`

Descrizione: Metodo che ritorna un vettore contenente tutti gli shorcut per l'applicativo Romeo.

Note:

- il metodo deve essere marcato costante;

- `+ setShortcutHomeEnable(enable: bool) : void`

Descrizione: Metodo che ha il compito di impostare lo shortcut per la pagina iniziale al valore definito dal parametro.

Argomenti:

- `enable: bool`
  rappresenta il valore a cui deve essere impostata la proprietà dello shortcut per la pagina iniziale: true se è abilitato, false altrimenti.

- `+ getCentralWidget() : APanel*`

Descrizione: Metodo che ritorna il puntatore al widget centrale della finestra principale.

Note:

- il metodo deve essere marcato costante;

- `+ showHelp(page: const QString\&) : void`

Descrizione: Metodo che ha il compito di visualizzare la guida interattiva sull'utilizzo dell'applicativo Romeo.

Argomenti:

- `page: const QString\&`
  Rappresenta la pagina dove la guida è chiamata.

Note:

- il metodo deve essere marcato costante;

- `+ homeSC():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando lo shortcut *home* è attivato.

- `+ aboutSC():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando lo shortcut *about* è attivato.

- `+ exitSC():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando lo shortcut *exit* è attivato.

- `+ createNewSubjectSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *create new Subject* è attivato.

- `+ createNewGroupSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *create new Group* è attivato.

- `+ createNewProtocolSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *create new Protocol* è attivato.

- `+ createNewDatasetSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *create new Dataset* è attivato.

- `+ showSubjectsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *manage Subjects* è attivato.

- `+ showGroupsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *manage Groups* è attivato.

- `+ showProtocolsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *manage Protocols* è attivato.

- `+ showDatasetsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *manage Datasets* è attivato.

- `+ showResultsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *show Results* è attivato.

- `+ StartAnalysisSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *start analysis* è attivato.

- `+ helpVideoGuideSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *video guide* è attivato.

- `+ helpManualSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *interactive guide* è attivato.

- `+ shortCutsSC():void` (signal)

Descrizione: Signal$_G$ emesso quando lo shortcut *show shortcats* è attivato.

- `+ helpPageSC():void` (signal)

Descrizione: Signal$_G$ emesso viene premuto il pulsante per l'help.

- `+ writeStatusBar(txt : const QString\&, timeout:int=0):void` (slot)

Descrizione: Slot$_{\mathbf{G}}$ che ha il compito di visualizzare la barra di stato attivato alla ricezione di un signal$_{\mathbf{G}}$.

Argomenti:

- `txt : const QString\&`
  Stringa contenente il testo da visualizzare;
- `timeout: int=0`
  intero che rappresenta il tempo per il quale il messaggio (txt) debba essere visualizzato: di default 0.

5.1.2 APanel (abstract)

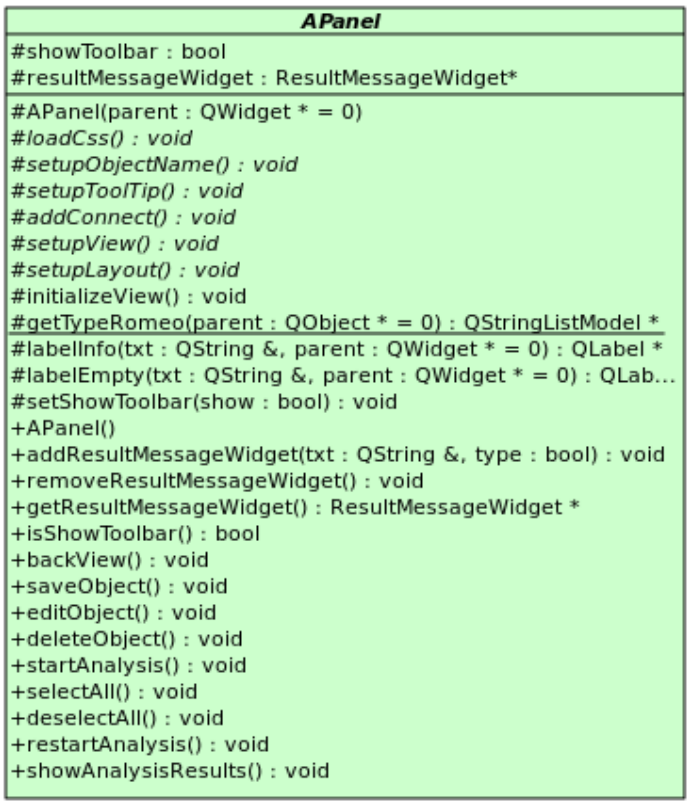

Figura 87: Diagramma Classe APanel: attributi e metodi

Descrizione

Classe che rappresenta il widget che verrà poi impostato come centralWidget di MainWindow.

Utilizzo

La classe astratta fornisce dei contratti puri che verranno implementati dalle sottoclassi, e offre dei metodi disponibili alle sottoclassi.

Classi ereditate

- Qt::QWidget.

Attributi

- `- showToolbar: bool`

  Descrizione: Rappresenta un flag che vale true se la tool bar è visibile nella view, false atrimenti.

- `- resultMessageWidget: ResultMessageWidget*`

  Descrizione: Puntatore alla componente che mostra all'utente il risultato dell'operazione eseguita sulla view (se è andata a buon fine o meno).

Metodi

- `# APanel(parent : QWidget*=0)`

  Descrizione: Costruttore protetto per la classe APanel.

  Argomenti

  - `parent : QWidget*=0`
    Puntatore al QWidget padre di APanel.

- `#setupLayout(): void`

  Descrizione: Metodo virtuale puro che ha come contratto la costruzione del layout del widget. Il metodo deve essere ridefinito dalle classi che ereditano da questa, definendo la costruzione del proprio layout.

  Note:

  - questo metodo deve essere marcato virtuale puro;
  - questo metodo deve essere ridefinito.

- `#loadCss(): void`

  Descrizione: Metodo virtuale puro che ha come contratto il caricamento del file contentente il css per il widget. Il metodo deve essere ridefinito dalle classi che ereditano da questa, definendo il proprio css, qualora ne sia provvisto, altrimenti avrà corpo vuoto.

  Note:

  - questo metodo deve essere marcato virtuale puro;
  - questo metodo deve essere ridefinito.

- `#setupObjectName():void`

  Descrizione: Metodo virtuale puro con contratto quello di settare il nome ad ogni oggetto, contenuto nel widget. Il metodo deve essere ridefinito dalle classi che ereditano da questa, per gli oggetti contenuti nella classe.

  Note:

  - questo metodo deve essere marcato virtuale puro;
  - questo metodo deve essere ridefinito.

- `#setupToolTip():void`

  Descrizione: Metodo virtuale puro che ha come contratto quello di settare i Tooltip$_G$ agli oggetti della finestra. Il metodo deve essere ridefinito dalle classi che ereditano da questa impostando per ogni oggetto i ToolTip necessari per mostrare all'utente che sta utilizzando Romeo i suggerimenti e spiegare a cosa serve quell'oggetto su cui si è posizionato con il cursore.

Note:

- questo metodo deve essere marcato virtuale puro;
- questo metodo deve essere ridefinito.

- `#addConnect(): void`

Descrizione: Metodo virtuale puro che ha come contratto quello di fissare tutte le istruzioni *connect*. Il metodo deve essere ridefinito dalle classi che ereditano da questa inserendo le connect necessarie al widget.

Note:

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale puro;
- questo metodo deve essere ridefinito.

- `#setupView(): void`

Descrizione: Metodo virtuale puro che ha come contratto quello di settare il layout del widget. Il metodo deve essere ridefinito dalle classi che ereditano da questa impostando il proprio layout.

Note:

- questo metodo deve essere marcato virtuale puro;
- questo metodo è stato ridefinito.

- `# initializeView(): void`

Descrizione: Metodo che invoca i metodi astratti precedentemente descritti, ridefiniti nella sottoclasse che eredita da APanel.

- `# static getTypeRomeo(parent:QObject*=0):QStringListModel*`

Descrizione: Metodo che ritorna una lista di stringhe usate dalla combobox per selezionare il tipo dell'immagine.

Note:

- il metodo deve essere marcato statico.

- `# labelInfo(txt: const QString\&, parent:QWidget* =0): QLabel*`

Descrizione: Metodo che ha il compito di creare una label informativa da mostrare all'utente vicino ai campi di input; ritorna un puntatore alla label appena creata.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente;
- `parent : QObject*`
  Puntatore al parent dell'oggetto label in creazione.

Note:

- il metodo deve essere marcato costante.

- `# labelEmpty(txt: const QString\&, parent:QWidget* =0): QLabel*`

Descrizione: Metodo che ha il compito di creare una label per mostrare all'utente una lista vuota; ritorna un puntatore alla label appena creata.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente;
- `parent : QObject*`
  Puntatore al parent dell'oggetto label in creazione.

Note:

- il metodo deve essere marcato costante.

- `# setShowToolBar(show : bool): void*`

Descrizione: Metodo che ha il compito di impostare se la toolbare è visibile o meno nella view in base al valore passato come parametro.

Argomenti:

- `show: bool`
  Rappresenta il valore a cui impostare il flag che dice se la toolbar è visibile (true) o meno.

- `+ addResultMessageWidget(txt: const QString\&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+ removeResultMessageWidget():void`

Descrizione: Metodo che ha il compito di rimuovere il messaggio di risultato che l'utente sta visualizzando.

- `+ isShowToolbar():bool`

Descrizione: Metodo che il flag che dice se la toolbar è visibile o meno; ritorna true se la toolbar è visibile nella view, ritorna false se è nascosta.

Note:

– il metodo deve essere marcato costante.

`+ backView():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide di ritornare alla vista precedente, ovvero quando viene premuto il pulsante back presente nelle viste che mettono a disposizione questa funzionalità.

`+ saveObject():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide di salvare le modifiche fatte nella vista corrente, ovvero quando viene premuto il pulsante save presente nelle viste che mettono a disposizione questa funzionalità.

`+ editObject():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide di modificare qualcosa nella vista corrente, ovvero quando viene premuto il pulsante edit presente nelle viste che mettono a disposizione questa funzionalità.

`+ deleteObject():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide di eliminare l'oggetto selezionato ovvero quando viene premuto il pulsante delete presente nelle viste che mettono a disposizione questa funzionalità.

`+ startAnalysis():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide avviare un'analisi, ovvero quando viene premuto il pulsante start Analysis presente nelle viste che mettono a disposizione questa funzionalità.

`+ selectAll():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide selezionare tutti gil elementi presenti in una lista o tabella. Emesso dalle viste che mettono a disposizione questa funzionalità.

`+ deselectAll():void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente decide deselezionare tutti gil elementi selezionati in una lista o tabella. Emesso dalle viste che mettono a disposizione questa funzionalità.

- `+ restartAnalysis():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente decide di far ripartire l'analisi, ovvero quando viene premuto il pulsante di restart analysis.

- `+ showAnalysisResults():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente decide di mostrare i risultati delle analisi, ovvero quando viene premuto il pulsante di visualizzazione dei risultati.

5.1.3 WelcomeView (class)

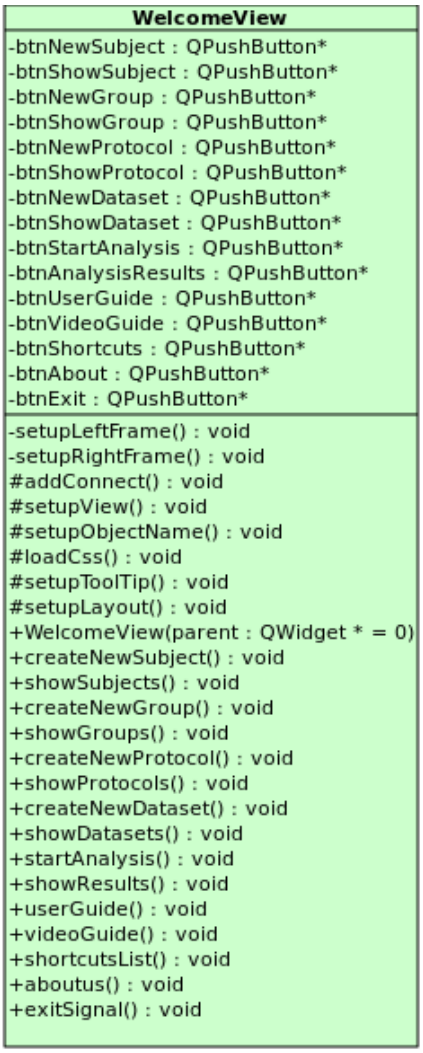

Figura 88: Diagramma Classe WelcomeView: attributi e metodi

Descrizione

Classe che rappresenta il widget della pagina iniziale, dove l'utente può scegliere le diverse operazioni da eseguire.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre metterà a disposizione i pulsanti che daranno all'utente la possibilità di interagire con l'applicazione svolgendo l'operazione desiderata.

Classi ereditate

- Window::APanel.

Attributi

- `-newSubjectButton:QPushButton*`

Descrizione: pulsante per la creazione di un nuovo Subject$_{\mathbf{G}}$. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la creazione di un nuovo Subject$_{\mathbf{G}}$.

- `-showSubjectButton:QPushButton*`

  Descrizione: pulsante per la visualizzazione dei Subject$_{\mathbf{G}}$ presenti nell'applicativo. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la visualizzazione dei Subject$_{\mathbf{G}}$.

- `-newGroupButton:QPushButton*`

  Descrizione: pulsante per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$.

- `-showGroupButton:QPushButton*`

  Descrizione: pulsante per la visualizzazione dei gruppi di Subject$_{\mathbf{G}}$ presenti nell'applicativo e la possibilità di modificarne il contenuto, aggiungendo o togliendo Subject$_{\mathbf{G}}$ dal gruppo selezionato. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la visualizzazione dei Subject$_{\mathbf{G}}$.

- `-newProtocolButton:QPushButton*`

  Descrizione: pulsante per la creazione di un nuovo Protocol$_{\mathbf{G}}$. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la creazione di un nuovo Protocol$_{\mathbf{G}}$.

- `-showGroupButton:QPushButton*`

  Descrizione: pulsante per la visualizzazione dei Protocol$_{\mathbf{G}}$ presenti nell'applicativo e la possibilità di eliminare quelli selezionati. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la visualizzazione dei Subject$_{\mathbf{G}}$.

- `-newGroupButton:QPushButton*`

  Descrizione: pulsante per la creazione di un nuovo Dataset$_{\mathbf{G}}$. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la creazione di un nuovo Dataset$_{\mathbf{G}}$.

- `-showDatasetButton:QPushButton*`

  Descrizione: pulsante per la visualizzazione dei Dataset$_{\mathbf{G}}$ presenti nell'applicativo. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la visualizzazione dei Dataset$_{\mathbf{G}}$.

- `-showDatasetButton:QPushButton*`

  Descrizione: pulsante per iniziare una nuova analisi. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la creazione di una nuova analisi e la sua successiva esecuzione.

- `-analysisResultsButton:QPushButton*`

Descrizione: pulsante per la visualizzazione delle analisi effettuate con Romeo e i risultati prodotti da esse. Alla pressione del pulsante verrà emesso un signal$_{\mathbf{G}}$ e verrà mostrata l'interfaccia per la visualizzazione delle analisi presenti nell'applicativo.

Metodi

- `+ WelcomeView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe WelcomeView.

  Argomenti:

  - `parent : QWidget*=0`
    Puntatore al QWidget padre di WelcomeView.

- `- setupLeftFrame(): void`

  Descrizione: Metodo che ha il compito di costruire la parte sinistra del widget iniziale.

- `-setupRightFrame(): void`

  Descrizione: Metodo che ha il compito di costruire la parte destra del widget iniziale.

- `#loadCss(): void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `# setupObjectName():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `# setupToolTip(): void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+ aboutus():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *about us*.

- `+ exitSignal():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *exit*.

- `+ createNewSubject():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *new Subject*.

- `+ createNewGroup():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *new Gruop*.

- `+ createNewProtocol():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *new Protocol*.

- `+ createNewDataset():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *new Dataset*.

- `+ showSubjects():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante *show Subjects*.

- `+ showGroups():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *show Groups*.

- `+ showProtocols():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *show Protocols*.

- `+ showDatasets():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *show Dataset*.

- `+ showResults():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *show Results*.

- `+ StartAnalysis():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *start analysis*.

- `+ VideoGuide():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *video guide*.

- `+ userGuide():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *interactive guide*.

- `+ shortcutsList():void` (signal)

Descrizione: Signal$_G$ emesso quando viene premuto il pulsante *show shortcuts*.

5.1.4 NewSubjectView (class)

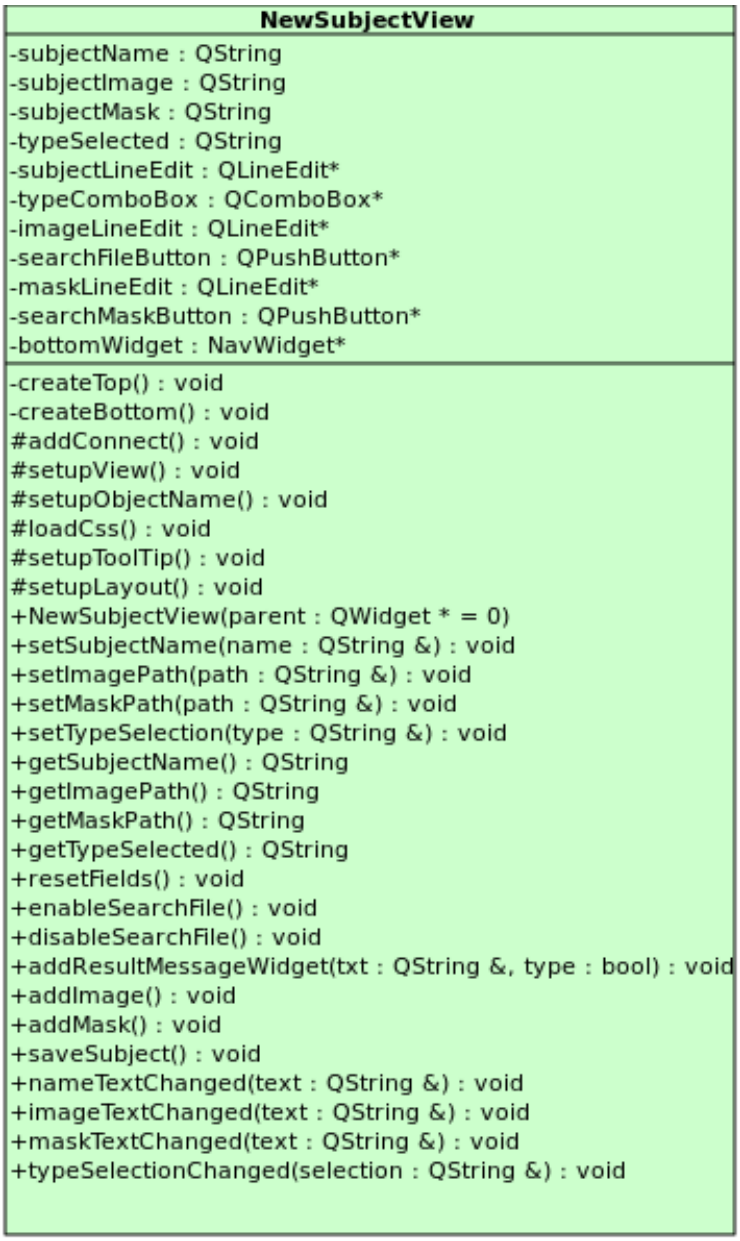

Figura 89: Diagramma Classe NewSubjectView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la creazione di un nuovo Subject$_{\mathbf{G}}$.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di inserire il nome del Subject$_{\mathbf{G}}$, selezionare il file corrispondente all'immagine da analizzare, opzionalmente potrà scegliere il file per la maschera e infine salvare le operazioni fatte, o ritornare alla finestra iniziale.

Classi ereditate

- Window::APanel.

Attributi

- `-subjectName: QString`

  Descrizione: Stringa che rappresenta il nome del Subject$_{\mathbf{G}}$.

  `-subjectImage: QString`

  Descrizione: Stringa che rappresenta il percorso dell'immagine da associare al Subject$_{\mathbf{G}}$.

  `-subjectMask: QString`

  Descrizione: Stringa che rappresenta il percorso della maschera da associare all'immagine del Subject$_{\mathbf{G}}$.

  `-typeSelected: QString`

  Descrizione: Stringa che rappresenta il tipo di immagine associata al Subject$_{\mathbf{G}}$ in creazione.

  `-searchFileButton: QPushButton*`

  Descrizione: pulsante che quando verrà premuto, emetterà un signal$_{\mathbf{G}}$ e verrà aperta una finestra di dialogo per navigare il file system e selezionare il file contenente l'immagine da caricare.

  `-searchMaskButton: QPushButton*`

  Descrizione: pulsante che quando verrà premuto, emetterà un signal$_{\mathbf{G}}$ e verrà aperta una finestra di dialogo per navigare il file system e selezionare il file contenente la maschera da caricare.

  `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti necessari al salvataggio all'annullamento delle operazioni e tornare indietro.

Metodi

- `+ NewSubjectView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe NewSubjectView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di NewSubjectView.

- `#setupLayout():void` Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `-createTop():void`

Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente la form per l'inserimento dei dati necessari per la creazione di un nuovo Subject$_{\mathbf{G}}$.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, il pulsante per il salvataggio e l'accesso alla guida.

- `#loadCss():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metododeve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+ addResultMessageWidget(txt: const QString\&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+setSubjectName(name: QString\&):void`

Descrizione: Metodo che modifica il nome associato al $\text{Subject}_{\mathbf{G}}$ da creare.

Argomenti:

- `name : const QString\&`
  riferimento costante alla stringa contentente il nuovo nome per il $\text{Subject}_{\mathbf{G}}$.

Note:

- questo metodo verrà invocato dal controller che si occupa di questa vista.

- `+ setImagePath(path: QString\&):void`

Descrizione: Metodo che modifica il path dell'immagine associata al $\text{Subject}_{\mathbf{G}}$ da creare.

Argomenti:

- `path : const QString\&`
  riferimento costante alla stringa contentente il nuovo path dell'immagine per il Subject$_G$.

Note

- questo metodo verrà invocato dal controller che si occupa di questa vista.

- `+setMaskPath(path: QString\&): void`

Descrizione: Metodo che modifica il path della maschera associata al Subject$_G$ da creare.

Argomenti:

- `path : const QString\&`
  riferimento costante alla stringa contentente il nuovo path della maschera per il Subject$_G$.

Note

- questo metodo verrà invocato dal controller che si occupa di questa vista.

- `+setTypeSelection(type: const QString\&):void`

Descrizione: Metodo che modifica il tipo dell'immagine associata al Subject$_G$ in creazione.

Argomenti:

- `type : const QString\&`
  riferimento costante alla stringa contentente il nuovo tipo dell'immagine associata al Subject$_G$.

Note

- questo metodo verrà invocato dal controller che si occupa di questa vista.

- `+ resetFields(): void`

Descrizione: Metodo che imposta ai valori di default tutti i campi della view.

- `+ enableSearchFile(): void`

Descrizione: Metodo che imposta ad abilitata la ricerca nel fileSystem.

- `+ disableSearchFile(): void`

Descrizione: Metodo che imposta ad disabilitata la ricerca nel fileSystem.

- `+ getSubjectName(): QString`

Descrizione: Metodo che restituisce un QString contenente il nome del Subject$_{\mathbf{G}}$.

Note:

- questo metodo deve essere marcato costante.

- `+ getImagePath(): QString`

Descrizione: Metodo che restituisce un QString contenente il path dell'immagine associata al Subject$_{\mathbf{G}}$.

Note:

- questo metodo deve essere marcato costante.

- `+ getMaskPath(): QString`

Descrizione: Metodo che restituisce un QString contenente il path della maschera associata al Subject$_{\mathbf{G}}$

Note

- questo metodo deve essere marcato costante.

- `+ getTypeSelected(): QString`

Descrizione: Metodo che restituisce un QString contenente il tipo selezionato.

Note

- questo metodo deve essere marcato costante.

- `+ resetFields(): void`

Descrizione: Metodo che resetta la form, eliminando il contenuto di ogni campo mettendo le impostazioni di default.

- `+ addImage():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente decide di cercare nel file system l'immagine da associare al Subject$_{\mathbf{G}}$ ovvero quando viene premuto il pulsante searchFileButton.

- `+ addMask():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente decide di cercare nel file system la maschera da associare al Subject$_{\mathbf{G}}$ ovvero quando viene premuto il pulsante searchMaskButton.

- `+ saveSubject():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente decide di salvare il Subject$_{\mathbf{G}}$ appena creato, ovvero quando viene premuto il pulsante save contentuto in bottomWidget.

- `+ nameTextChanded(text:QString\&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente modifica il nome contenuto nella linea di testo per l'inseriemnto del nome del Subject$_{\mathbf{G}}$.

Argomenti:

- `text: QString\&`
  rappresenta il nuovo nome da assegnare al Subject$_{\mathbf{G}}$.

- `+ imageTextChanded(text:QString\&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente modifica il file dell'immagine da associare al Subject$_{\mathbf{G}}$.

Argomenti:

- `text: QString\&`
  rappresenta il path della nuova immagine da associare al Subject$_{\mathbf{G}}$.

- `+ maskTextChanded(text:QString\&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ eemesso quando l'utente modifica il file dell'immagine da associare al Subject$_{\mathbf{G}}$.

Argomenti:

- `text: QString\&`
  rappresenta il path della nuova maschera da associare al Subject$_{\mathbf{G}}$.

5.1.5 NewGroupView (class)

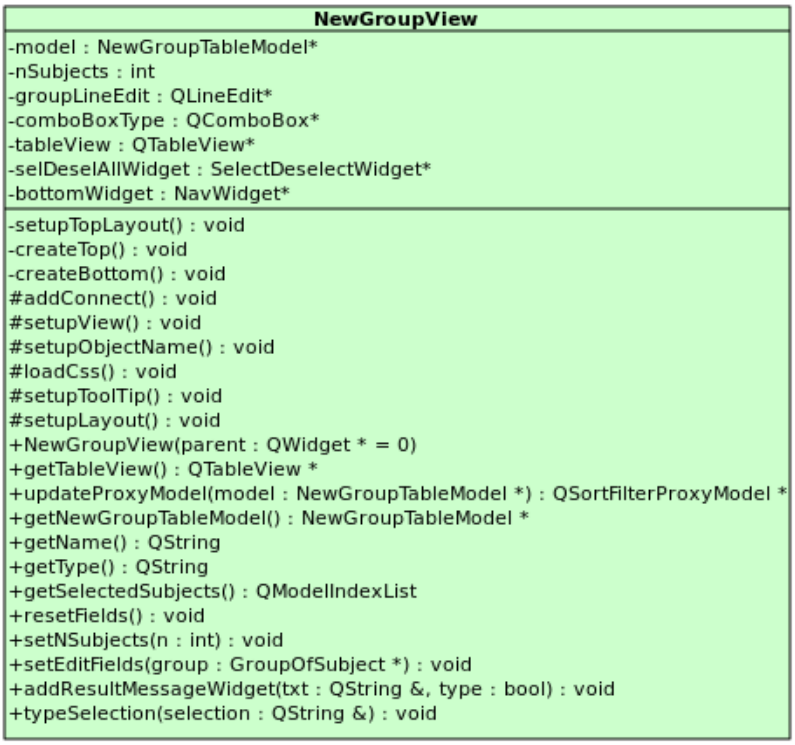

Figura 90: Diagramma Classe NewGroupView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di:

- inserire il nome del nuovo gruppo di Subject$_{\mathbf{G}}$;
- selezionare il tipo dell'immagine che dovranno avere i Subject$_{\mathbf{G}}$ da aggiungere al gruppo;
- selezionare, dalla lista dei Subject$_{\mathbf{G}}$con associata un'immagine del tipo precedentemente scelto, i Subject$_{\mathbf{G}}$ di interesse;
- salvare le operazioni fatte, o ritornare alla finestra iniziale.

Classi ereditate

- Window::APanel.

Attributi

- `-model: NewGroupTableModel*`

  Descrizione: Modello contenente i Subject$_{\mathbf{G}}$ contenuti nel gruppo

- `-tableView: QTabelView*`

Descrizione: Contiene la lista dei Subject$_{\mathbf{G}}$ presenti in *model*.

- `-nSubject:int`

  Descrizione: Indica il numero di Subject$_{\mathbf{G}}$ presenti nel sistema con associata un'immagine del tipo indicato dal campo *comboBoxType*.

- `-groupLineEdit: QLineEdit*`

  Descrizione: Linea di testo, contenente il nome del gruppo in creazione.

- `-comboBoxType: QComboBox*`

  Descrizione: Contiene l'elenco dei tipi di immagine che Romeoè in grado di elaborare; serve per selezionare poi i Subject$_{\mathbf{G}}$ che hanno il tipo di immagine indicato da questo campo dati.

Metodi

- `+ NewGroupView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe NewGroupView.

  #### Argomenti

  – `parent: QWidget*=0`
  Puntatore al QWidget padre di NewGrouView.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare il layout in alto del widget, aggiungendo gli oggetti che lo compongono, e dando delle impostazioni grafiche.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente la form per l'inserimento dei dati necessari per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagina iniziale, il pulsante per il salvataggio e l'accesso alla guida.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#loadCss():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+ addResultMessageWidget(txt: const QString&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `#addConnect():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato costante;
  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupView():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `+ getTableView():QTableView*`

  Descrizione: Metodo che ritorna la tabella contenente tutti i $\text{Subject}_{\mathbf{G}}$ contenuti che gruppo.

- `+ updateProxyModel(): QSortFilterProxyModel*`

  Descrizione: Metodo che aggiorna il model della tabella dei $\text{Subject}_{\mathbf{G}}$ cliccando gli header della tabella stessa.

- `+ getNewGroupTableModel(): NewGroupTableModel*`

  Descrizione: Metodo che ritorna il campo dati *model*.

  Note

  - questo metodo deve essere marcato costante.

- `+ getName(): QString`

  Descrizione: Metodo ch ritorna il nome del gruppo in creazione.

  Note:

  - questo metodo deve essere marcato costante.

- `+ getType(): QString`

  Descrizione: Metodo che ritorna il tipo di immagine che devono avere i $\text{Subject}_{\mathbf{G}}$ da inserire nel gruppo.

Note

– questo metodo deve essere marcato costante.

- `+ getSelectedSubjects(): QModelIndexList*`

Descrizione: Metodo che ritorna la lista dei Subject$_G$selezionati.

Note

– questo metodo deve essere marcato costante.

- `+ resetFields(): void`

Descrizione: Metodo che imposta tutti gli object contenuti dentro alla finestra al loro valore di default.

- `+ setNSubjects(n: int):void`

Descrizione: Metodo che imposta il campo dati *nSubject*.

Argomenti:

– `n: int`
numero diSubject$_G$ presenti per il tipo selezionato, da assegnare al campo dati *nSubject*.

- `+ setEditFields (group: GroupsOfSubject*):void`

Descrizione: Metodo che imposta i campi con i valori.

Argomenti:

– `group: GroupOfSubject*`
Rappresenta il puntatore al gruppo da editare.

- `+ typeSelection(selection: QString&):void` (signal)

Descrizione: Signal$_G$ emesso quando l'utente ha selezionato il tipo dalla combo-box.

Argomenti:

– `selection: QString&`
Rappresenta la stringa selezionata dall'utente.

5.1.6 NewProtocolView (class)

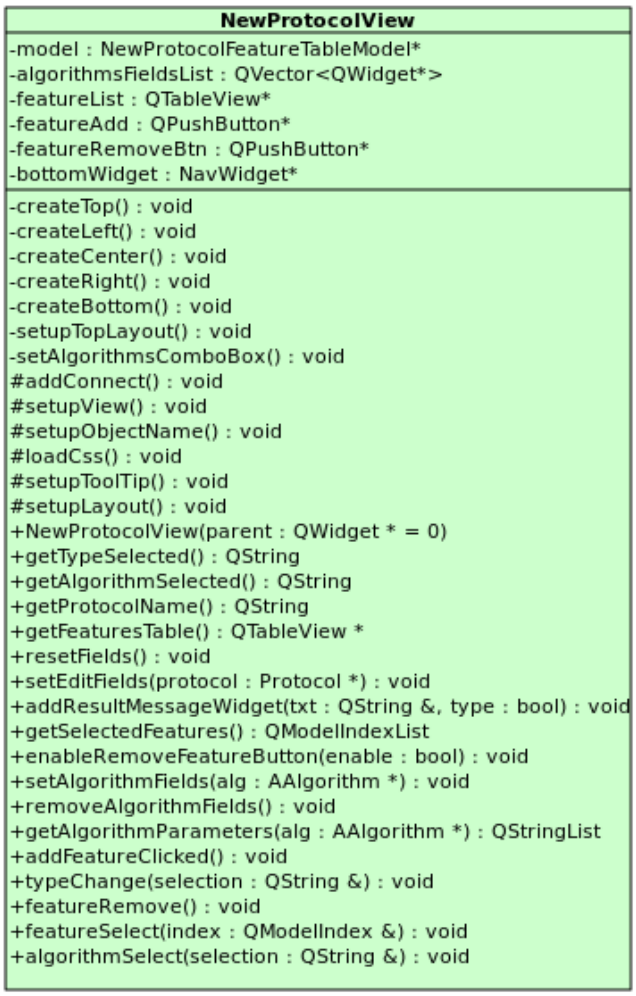

Figura 91: Diagramma Classe NewProtocolView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la creazione di un nuovo Protocol$_{\mathbf{G}}$.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di inserire il nome del Protocol$_{\mathbf{G}}$, selezionare il tipo di immagine a cui il protocol$_{\mathbf{G}}$ verrà applicato, potrà selezionare i Feature Extractors$_{\mathbf{G}}$ di interesse, settando i parametri richiesti lasciando opzionalmente quelli presenti di default, e/o selezionare uno e uno solo algoritmo di clastering$_{\mathbf{G}}$ comportandosi allo stesso modo per i Feature Extractors$_{\mathbf{G}}$. Può poi proseguire al salvataggio del Protocol$_{\mathbf{G}}$ creato.

Classi ereditate

- Window::APanel.

Attributi

- `-featureAdd: QPushButton*`

Descrizione: Pulsante che permette di aggiungere una nuova feature, da inserire nel Protocol in creazione.

- `-model: NewProtocolFeatureTabelModel*`

  Descrizione: Modello che contiene la lista delle Feature fino a questo momento inserite nel Protocol$_G$ in creazione.

- `- algorithmFieldsList : QVector<QWidget*>`

  Descrizione: Vettore che contiene i puntatori ai campi degll'algoritmo.

- `-featureList: QTabelView*`

  Descrizione: contiene la lista delle Feature$_G$ contenute dentro a *model*.

- `-algorithmCombobox: QComboBox*`

  Descrizione: contiene la lista degli algoritmi di clustering$_G$ presenti nell'applicativo Romeo.

- `-nameLineEdit: QLineEdit*`

  Descrizione: Linea di testa utilizzata per inserire il nome del Protocol$_G$ che si sta creando.

- `-typeComboBox: QComboBox*`

  Descrizione: contiene la lista dei tipi di immagine a cui il Protocol$_G$ verrà applicato. Una volta selezionato un tipo, verranno rese disponibili solo le Feature$_G$ applicabili al tipo di immagine scelto.

Metodi

- `+ NewProtocolView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe NewSubjectView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di NewSubjectView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `-createTop():void`

Descrizione: Metodo che ha il compito di costruire la parte in alto del widget.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, il pulsante per il salvataggio e l'accesso alla guida.

- `#loadCss():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metododeve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+getTypeSelected():QString`

Descrizione: Metodo che ritorna il tipo a cui verrà applicato il protocol$_{G}$.

Note

- questo metodo deve essere marcato costante.

- `+getAlgorithmSelected():QString`

Descrizione: Metodo che ritorna il nome dell'algoritmo selezionato.

Note

- questo metodo deve essere marcato costante.

- `+getProtocolName():QString`

Descrizione: Metodo che ritorna il nome da associare al nuovo Protocol$_{G}$.

Note

- questo metodo deve essere marcato costante.

- `+getFeaturesTable():QString`

Descrizione: Metodo che ritorna la tabella contenente le features impostate.

Note

- questo metodo deve essere marcato costante.

- `+resetFields(): void`

Descrizione: Metodo che ripulisce i campi e li setta alle opzioni di predefinite.

- `-createLeft(): void`

Descrizione: Metodo che crea il layout per la parte sinistra della finestra contenente il nome da dare al nuovo protocol$_{G}$ e il tipo di immagine a cui il protocol$_{G}$ verrà applicato.

- `-createRight(): void`

Descrizione: Metodo che crea il layout per la parte destra della finestra contenente le opzioni per l'inserimento dell'algoritmo di clustering$_{G}$.

- `-createCenter(): void`

Descrizione: Metodo che crea il layout per la parte centrale della finestra contenente le opzioni per l'inserimento delle feature$_G$.

- `-setupTopLayout(): void`

Descrizione: Metodo che imposta il layout della parte alta della finestra contenente le parti: destra, sinistra e centrale create dai metodi precedentemente illustrati.

- `+ setEditableFields(protocol: Protocol*):void`

Descrizione: Metodo che ha il compito di impostare i campi della view con i valori.

Argomenti:

- `protocol: Protocol*`
  Rappresenta il Protocol$_G$ da editare.

- `+ addResultMessageWidget(txt: const QString\&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+ getSelectedFeatures(): QModelIndexList`

Descrizione: Metodo che ha il compito di ritornare al Feature$_G$ selezionata; ritorna una lista di indici di tabella.

Note:

- il metodo deve essere marcato costante.

- `+ enableRemoveFeatureButton(enable: const bool): void`

Descrizione: Metodo che ha il compito di impostare la proprietà del pulsante abilitato o disabilitato, a seconda del valore del parametro passato.

Argomenti:

- `enable: const bool`
  rappresenta il valore a cui impostare la proprietà del pulsante: true per abilitare il pulsante di rimozione della Feature$_G$ false altrimenti.

- `+ setAlgorithmFields(alg : AAlgorithm *) :void`

Descrizione: Metodo che ha il compito di impostare i campi dei parametri dell'algoritmo.

Argomenti:

- `alg: AAlgorithm*`
  rappresenta l'algoritmo di cui impostare i campi dei parametri.

- `+ removeAlgorithmFields() :void`

Descrizione: Metodo che ha il compito di rimuovere i campi dei parametri dell'algoritmo.

- `+ setAlgorithmFields(alg : AAlgorithm *) :void`

Descrizione: Metodo che ha il compito di ritornare una lista contenente i valori dei parametri.

Argomenti:

- `alg: AAlgorithm*`
  rappresenta l'algoritmo di cui cui ritornare il valore dei parametri.

Note:

- il metodo deve essere marcato come costante.

- `+ addFeatureClicked():void` (signal)

Descrizione: $\text{Signal}_{\mathbf{G}}$ emesso quando l'utente decide di aggiungere una nuova $\text{Feature}_{\mathbf{G}}$ da calcolare al $\text{Protocol}_{\mathbf{G}}$.

- `+ typeChange(index:int):void` (signal)

Descrizione: $\text{Signal}_{\mathbf{G}}$ emesso quando l'utente ha selezionato il tipo dalla combobox, servirà poi per poter mostrare all'utente solo le $\text{Feature}_{\mathbf{G}}$ disponibili per il tipo di immagine selezionato.

Argomenti:

- `index: int`
  rappresenta l'indice della voce selezionata dal menu a tendina.

- `+ featureRemove():void` (signal)

Descrizione: $\text{Signal}_{\mathbf{G}}$ emesso quando l'utente preme il pulsante per la rimozione di una $\text{Feature}_{\mathbf{G}}$.

- `+ featureSelect(index:const QModelIndex \&):void` (signal)

Descrizione: $\text{Signal}_{\mathbf{G}}$ emesso quando l'utente seleziona un elemento dalla tabella.

Argomenti:

- `index: const QModelIndex\&`
  rappresenta l'indice della tabella.

- `+ AlgorithmSelect(selection:const QString \&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un algoritmo.

Argomenti:

- `selection: const QString\&`
  rappresenta l'indice della tabella.

5.1.7 NewDatasetView (class)

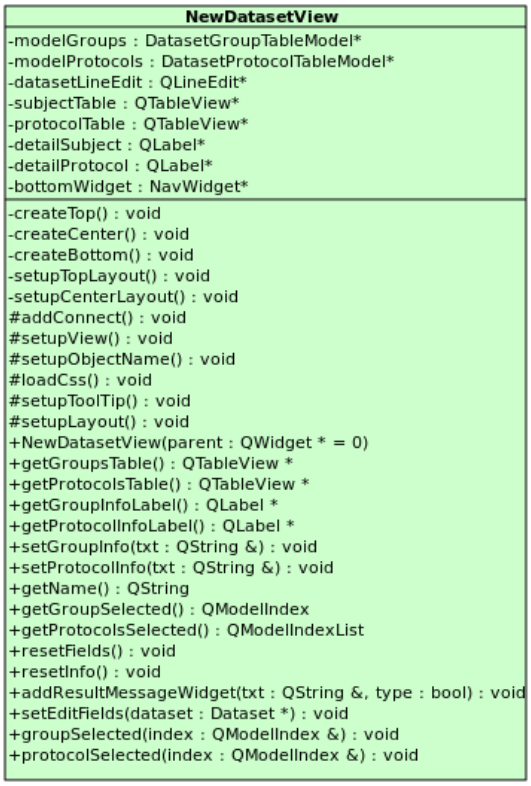

Figura 92: Diagramma Classe NewDatasetView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la creazione di un nuovo Dataset$_{\mathbf{G}}$.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di inserire un gruppo di Subject$_{\mathbf{G}}$ precedentemente creato, selezionare uno o più protocol$_{\mathbf{G}}$ da eseguire per quel gruppo di Subject$_{\mathbf{G}}$ e infine salvarlo.

Classi ereditate

- Window::APanel.

Attributi

- `-datasetLineEdit: QLineEdit*`

  Descrizione: Linea di testo che conterrà il nome del Dataset$_{\mathbf{G}}$ che l'utente sta andando a creare.

- `-modelGroups: DatasetGroupTableModel*`

  Descrizione: Modello per la tabella dei Subject$_{\mathbf{G}}$.

- `-modelProtocols: DatasetProtocolTableModel*`

Descrizione: Modello che contiene tutti i Protocol$_G$ che

- `-detailSubject: QLabel*`

  Descrizione: Contiene tutte le informazioni relative al Subject$_G$ selezionato.

- `-ProtocolTable: QTableView*`

  Descrizione: Contiene la lista dei Protocol$_G$ contenuti dentro a *modelProtocols*

- `-detailProtocol: QTableView*`

  Descrizione: Contiene tutte le informazioni relative al Protocol$_G$ selezionato.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti necessari al salvataggio all'annullamento delle operazioni e tornare indietro.

Metodi

- `+ NewDatasetView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe NewDatasetView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di NewDatasetView.

- `#setupToolTip():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente la form per l'inserimento dei dati necessari per la creazione di un nuovo Dataset$_{\mathbf{G}}$.

- `-createCenter():void`

  Descrizione: Metodo che ha il compito di costruire la parte centrale del widget per la vista che si occupa della creazione di un nuovo Dataset$_{\mathbf{G}}$.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, il pulsante per il salvataggio e l'accesso alla guida.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `-setupCenterLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte centrale del layout della finestra.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  – questo metododeve essere marcato virtuale;
  – questo metodo è stato ridefinito.

- `#setupObjectName():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  – questo deve essere marcato virtuale;
  – questo metodo è stato ridefinito.

- `#addConnect():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  – questo metodo deve essere marcato costante;
  – questo metodo deve essere marcato virtuale;
  – questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+getGroupTable(): QTableView*`

  Descrizione: Metodo che ritorna la lista dei gruppi di $\text{Subject}_{\mathbf{G}}$ disponibili.

  Note:

  - questo metodo deve essere marcato costante.

- `+getProtocolsTable(): QTableView*`

  Descrizione: Metodo che ritorna la lista dei $\text{Protocol}_{\mathbf{G}}$ disponibili.

  Note:

  - questo metodo deve essere marcato costante.

- `+getName(): QString`

  Descrizione: Metodo che ritorna il nome del $\text{Dataset}_{\mathbf{G}}$.

  Note:

  - questo metodo deve essere marcato costante.

- `+getGroupSelected(): QModelIndex*`

  Descrizione: Metodo che ritorna un indice di tabella del gruppo selezionato.

  Note:

  - questo metodo deve essere marcato costante.

- `+getProtocolsSelected(): QModelIndex*`

  Descrizione: Metodo che ritorna un indice di tabella del $\text{Protocol}_{\mathbf{G}}$selezionato.

  Note:

  - questo metodo deve essere marcato costante.

- `+ resetInfo(): void`

  Descrizione: Metodo che reimposta il box delle informazioni.

- `+ resetFields(): void`

Descrizione: Metodo che reimposta tutti i campi della view.

- `+getGroupInfoLabel(): QLabel*`

  Descrizione: Metodo che ritorna una label contenenente le informazioni relative al gruppo selezionato.

  Note:

  – questo metodo deve essere marcato costante.

- `+getProtocolInfoLabel(): QLabel*`

  Descrizione: Metodo che ritorna una label contenenente le informazioni relative al Protocol$_G$ selezionato.

  Note:

  – questo metodo deve essere marcato costante.

- `+setProtocolInfo(txt: QString\&): void`

  Descrizione: Metodo che imposta le informazioni del protocol selezionato.

  Argomenti:

  – `txt: QString\&`
  txt contiene il testo che verrà settato per il capo dati *detailProtocol*.

- `+setGroupInfo(txt: QString\&): void`

  Descrizione: Metodo che imposta le informazioni del gruppo selezionato.

  Argomenti:

  – `txt: QString\&`
  txt contiene il testo che verrà settato per il capo dati *detailsSubject*.

- `+setEditFields(dataset: Dataset*): void`

  Descrizione: Metodo che imposta i campi con i dati.

  Argomenti:

  – `dataset: Dataset*`
  Rappresenta il puntatore al Dataset$_G$ da editare.

- `+ addResultMessageWidget(txt: const QString\&, type: bool):void`

  Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+groupSelected(index:QModelIndex\&):void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un gruppo dalla tabella identificata dal campo dati *subjectTable*.

- `+groupSelected(index:QModelIndex\&):void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un Protocol$_{\mathbf{G}}$ dalla tabella identificata dal campo dati *protocolTable*.

5.1.8 SubjectsView (class)

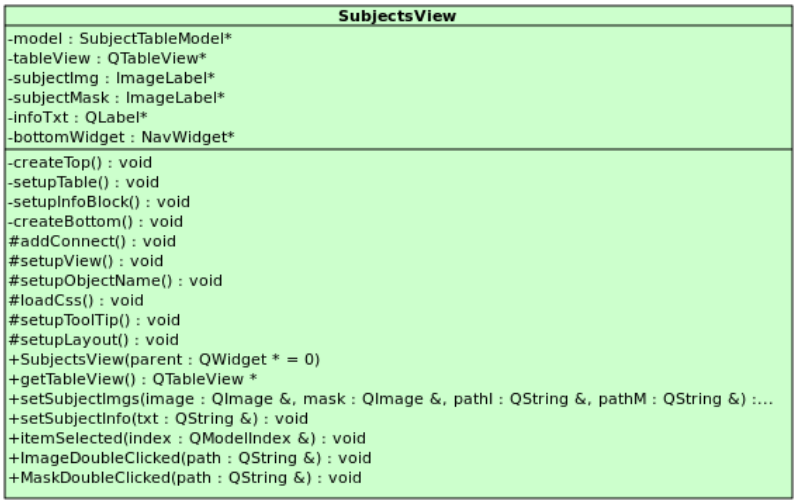

Figura 93: Diagramma Classe SubjectsView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutti i Subject$_{\mathbf{G}}$ presenti all'interno di Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutti i Subject$_{\mathbf{G}}$ fino a quel momento creati e memorizzati dentro a Romeo. Selezionando un Subject$_{\mathbf{G}}$ sarà possibile visualizzare le informazioni relative a quel Subject$_{\mathbf{G}}$.

Classi ereditate

- Window::APanel.

Attributi

- `-model: SubjectTableModel *`
  Modello che contiene i Subject$_{\mathbf{G}}$presenti nel sistema.
- `-tableView: QTabelView*`
  Contiene la lista dei Subject$_{\mathbf{G}}$contenuti nel campo dati *model*.
- `-subjectImg: QLabel*`
  Contiene l'immagine relativa al subject$_{\mathbf{G}}$ selezionato.
- `-subjectMask: QLabel*`
  Contiene la maschera relativa al subject$_{\mathbf{G}}$ selezionato.
- `-infoTxt: QLabel*`
  Contiene le informazioni relative al Subject$_{\mathbf{G}}$ selezionato.
- `bottomWidget:NavWidget*`
  Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro e per la visualizzazione della guida interattiva.

Metodi

- `+ SubjectsView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe SubjectsView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di SubjectsView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei Subject$_{\mathbf{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTable():void`

  Descrizione: Metodo che ha il compito di impostare la tabella che visualizzerà l'elenco dei Subject$_{\mathbf{G}}$ presenti nel sistema.

- `-setupInfoBlock():void`

  Descrizione: Metodo che ha il compito di impostare il layout riguardante la visualizzazione delle informazioni.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metododeve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupObjectName():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `-getTableView():QTabelView*` Metodo che ritorna il puntatore campo dati *tableView*.

Note:

- questo metodo deve essere marcato costante.

- `-setSubjectImg(imgage:QImage&):void`

Descrizione: Metodo che imposta l'immagine del $\text{Subject}_{\mathbf{G}}$ nella parte delle informazioni relative al $\text{Subject}_{\mathbf{G}}$.

Argomenti:

- `image: QImage&`
  contiene l'immagine che verrà impostata nella parte delle informazioni.

- `-setSubjectMask(imgage:QImage&):void`

Descrizione: Metodo che imposta la maschera del Subject$_{\mathbf{G}}$ nella parte delle informazioni relative al Subject$_{\mathbf{G}}$.

Argomenti:

- `image: QImage&`
  contiene la maschera che verrà impostata nella parte delle informazioni.

- `-setSubjectInfo(txt:QString&):void`

Descrizione: Metodo che imposta tutte le informazioni del Subject$_{\mathbf{G}}$selezionato nella parte deidicata alle informazioni.

Argomenti:

- `txt: QString&`
  stringa contenente le nuove informazioni da visualizzare.

- `+itemSelected(index:QModelIndex&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un Subject$_{\mathbf{G}}$ dalla tabella contenente tutti i Subject$_{\mathbf{G}}$ presenti.

Attributi:

- `index: QModelIndex&`
  rappresenta l'indice della tabella selezionato.

- `+ imageDoubleClicked(path:const QString&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona con un doppio click sull'immagine del Subject$_{\mathbf{G}}$.

Argomenti:

- `path: QString&`
  stringa contenente il path relativo all'immagine.

- `+ maskDoubleClicked(path:const QString&):void`(signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona con un doppio click la maschera associata all'immagine del Subject$_{\mathbf{G}}$.

Argomenti:

- `path: QString&`
  stringa contenente il path relativo all'immagine.

5.1.9 GroupView (class)

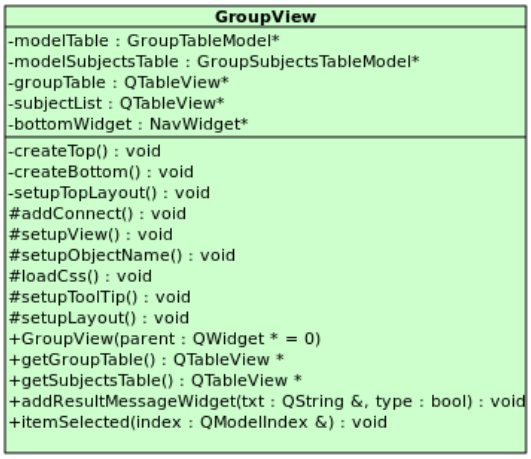

Figura 94: Diagramma Classe GroupView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutti i gruppi di Subject$_{\mathbf{G}}$ presenti all'interno di Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutti i gruppi di Subject$_{\mathbf{G}}$ fino a quel momento creati e memorizzati dentro a Romeo. Selezionando un gruppo, sarà possibile visualizzare le informazioni relative a quel gruppo, come per esempio i Subject$_{\mathbf{G}}$ contenuti al suo interno.

Classi ereditate

- Window::APanel.

Attributi

- `-modelTable: GroupTableModel *`

  Descrizione: Modello che contiene i gruppi diSubject$_{\mathbf{G}}$ presenti nel sistema fino a quel momento.

- `-modeSubjectslTable: GroupSubjectsTableModel *`

  Descrizione: Modello che contiene i Subject$_{\mathbf{G}}$ presenti in un determinato gruppo di Subject$_{\mathbf{G}}$ fino a quel momento presenti nel sistema.

- `-groupTable: QTabelView*`

  Descrizione: Contiene la lista dei gruppi di Subject$_{\mathbf{G}}$ contenuti nel campo dati *modelTable*.

- `-subjectList: QTabelView*`

Descrizione: Contiene la lista dei Subject$_{G}$ contenuti nel campo dati *modelSubjectsTable*.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro, la visualizzazione della guida interattiva, la possibilità di editare un gruppo e salvare successivamente le modifiche.

Metodi

- `+ GroupView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe GroupView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di GroupView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei Subject$_{G}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metododeve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+getGroupTable():QTabelView*`

Descrizione: Metodo che ritorna il puntatore al campo dati *groupTable*.

Note

- questo metodo deve essere marcato costante.

- `+getSubjectsTable():QTabelView*`

Descrizione: Metodo che ritorna il puntatore al campo dati *modelSubjectTable*.

Note

- questo metodo deve essere marcato costante.

- `+ addResultMessageWidget(txt: const QString&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+itemSelected(index:QModelIndex&): void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un elemento da una tabella, che sia quella contentente la lista di gruppi di Subject$_{\mathbf{G}}$ o quella con la lista dei Subject$_{\mathbf{G}}$ per un determinato gruppo.

Attributi:

- `index: QModelIndex&`
  rappresenta l'indice della tabella selezionato.

5.1.10 ProtocolsView (class)

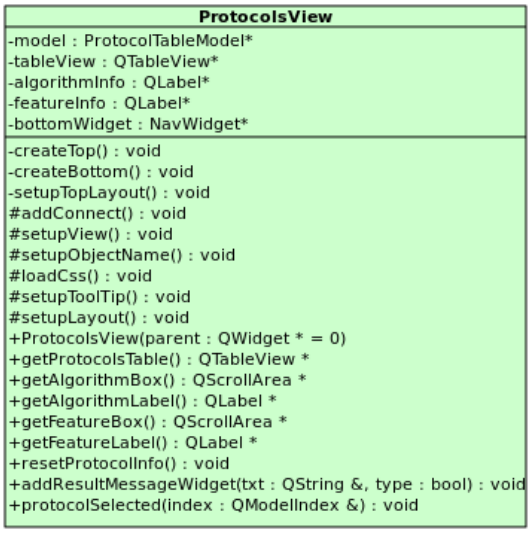

Figura 95: Diagramma Classe ProtocolsView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutti i Protocol$_G$ presenti all'interno di Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutti i Protocol$_G$ fino a quel momento creati e memorizzati dentro a Romeo. Selezionando un Protocol$_G$, sarà possibile visualizzare le informazioni relative a quel Protocol$_G$: feature$_G$ presenti con il valore dei parametri (qualora fossero presenti), e algoritmo di clustering$_G$ con valore dei parametri.

Classi ereditate

- Window::APanel.

Attributi

- `-model: protocolTableModel *`

  Descrizione: Modello che contiene i Protocol$_G$ presenti nel sistema fino a quel momento.

- `-tableView: QTabelView*`

  Descrizione: Contiene la lista dei Protocol$_G$ contenuti nel campo dati *model*.

- `-algorithmInfo: QLabel*`

  Descrizione: Contiene le informazioni relative all'algoritmo di clustering$_G$ del Protocol$_G$ selezionato, qualora fosse presente.

- `-featureInfo: QLabel*`

Descrizione: Contiene le informazioni relative alle Feature$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$ selezionato qualora fosse presente almeno una Feature$_{\mathbf{G}}$.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro, la visualizzazione della guida interattiva, la possibilità di eliminare un Protocol$_{\mathbf{G}}$ e confermare l'operazione.

Metodi

- `+ ProtocolsView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe ProtocolsView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ProtocolsView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei Subject$_{\mathbf{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metododeve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+getProtoclsTable():QTabelView*`

Descrizione: Metodo che ritorna il puntatore al campo dati *protocolsTable*.

Note

- questo metodo deve essere marcato costante.

- `+getAlgorithmLabel():QLabel*`

Descrizione: Metodo che ritorna la label contenente le informazioni riguardanti l'algoritmo di clustering$_{\mathbf{G}}$ presente nel Protocol$_{\mathbf{G}}$.

Note

- questo metodo deve essere marcato costante.

- `+getFeatureLabel():QLabel*`

Descrizione: Metodo che ritorna la label contenente le informazioni riguardanti le $\text{Feature}_{\mathbf{G}}$ presenti nel $\text{Protocol}_{\mathbf{G}}$.

Note

- questo metodo deve essere marcato costante.

- `+getAlgorithmBox():QScrollArea*`

Descrizione: Metodo che ritorna il puntatore al box contentente le informazioni dell'algoritmo.

Note

- questo metodo deve essere marcato costante.

- `+getFeaturesBox():QScrollArea*`

Descrizione: Metodo che ritorna il puntatore al box contentente le informazioni dei $\text{Protocol}_{\mathbf{G}}$.

Note

- questo metodo deve essere marcato costante.

- `+ addResultMessageWidget(txt: const QString&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+ resetProtocolInfo(): void`

Descrizione: Metodo che reimposta il box contenente le informazioni.

Note

- questo metodo deve essere marcato costante.

- `+protocolSelected(index:QModelIndex&):void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona un elemento da una tabella, che sia quella contentente la lista di gruppi di Subject$_{\mathbf{G}}$ o quella con la lista dei Subject$_{\mathbf{G}}$ per un determinato gruppo.

5.1.11 DatasetsView (class)

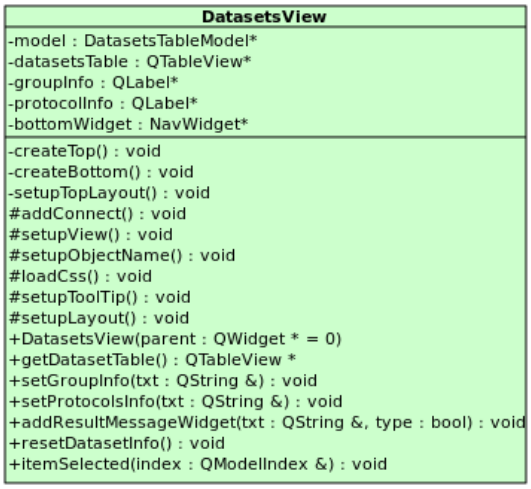

Figura 96: Diagramma Classe DatasetsView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutti i Dataset$_{\mathbf{G}}$ presenti all'interno di Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutti i Dataset$_{\mathbf{G}}$ fino a quel momento creati e memorizzati dentro a Romeo. Selezionando un Dataset$_{\mathbf{G}}$, sarà possibile visualizzare le informazioni relative a quel Dataset$_{\mathbf{G}}$: gruppo di Subject$_{\mathbf{G}}$ contenuto e Protocol$_{\mathbf{G}}$ presenti (uno o più).

Classi ereditate

- Window::APanel.

Attributi

- `-model: DatasetsTableModel*`

  Descrizione: Rappresenta il modello per la tabella dei Dataset$_{\mathbf{G}}$ presenti nell'applicativo Romeo.

- `-DatasetsView: QTabelView*`

  Descrizione: Contiene la lista dei Dataset$_{\mathbf{G}}$ contenuti nel campo dati *model*.

- `-algorithmInfo: QLabel*`

  Descrizione: Contiene le informazioni relative all'algoritmo di clustering$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$ selezionato, qualora fosse presente.

- `-featureInfo: QLabel*`

Descrizione: Contiene le informazioni relative alle Feature$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$ selezionato qualora fosse presente almeno una Feature$_{\mathbf{G}}$.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro, la visualizzazione della guida interattiva, la possibilità di eliminare un Dataset$_{\mathbf{G}}$ e confermare successivamente l'operazione.

Metodi

- `+ DatasetsView(parent : QWidget*=0)`
  Costruttore per la classe DatasesView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di DatasetsView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei Subject$_{\mathbf{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metododeve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+ resetDatasetInfo(): void`

Descrizione: Metodo che reimposta il box contenente le informazioni riguardanti il Dataset$_{\mathbf{G}}$.

Note

- questo metodo deve essere marcato costante.

- `+ getDatasetTab():QTableView*`

Descrizione: Metodo che ha il compito di ritornare la tabella contenente il Dataset$_{\mathbf{G}}$.

Note:

- il metodo deve essere marcato costante.

- `+ setProtocolsInfo(txt: const QString&): void`

  Descrizione: Metodo che ha il compito di impostare le informazioni relative al Protocol$_G$.

  Attributi:

  - `txt:const QString&`
    Rappresenta il testo da mostrare.

- `+ setGroupInfo(txt: const QString&): void`

  Descrizione: Metodo che ha il compito di impostare le informazioni relative al gruppo di Subject$_G$.

  Attributi:

  - `txt:const QString&`
    Rappresenta il testo da mostrare.

- `+ addResultMessageWidget(txt: const QString&, type: bool):void`

  Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

  Argomenti:

  - `txt: const QString&`
    Rappresenta il testo da mostrare all'utente all'interno del widget;
  - `type : bool`
    Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

  Note:

  - il metodo deve essere marcato virtuale.

- `+itemSelected(index:QModelIndex&):void` (signal)

  Descrizione: Signal$_G$ emesso quando l'utente seleziona un elemento da una tabella, che sia quella contentente la lista di gruppi di Subject$_G$ o quella con la lista dei Subject$_G$ per un determinato gruppo.

  Attributi:

  - `index: QModelIndex&`
    rappresenta l'indice della tabella selezionato.

5.1.12 AnalysisView (class)

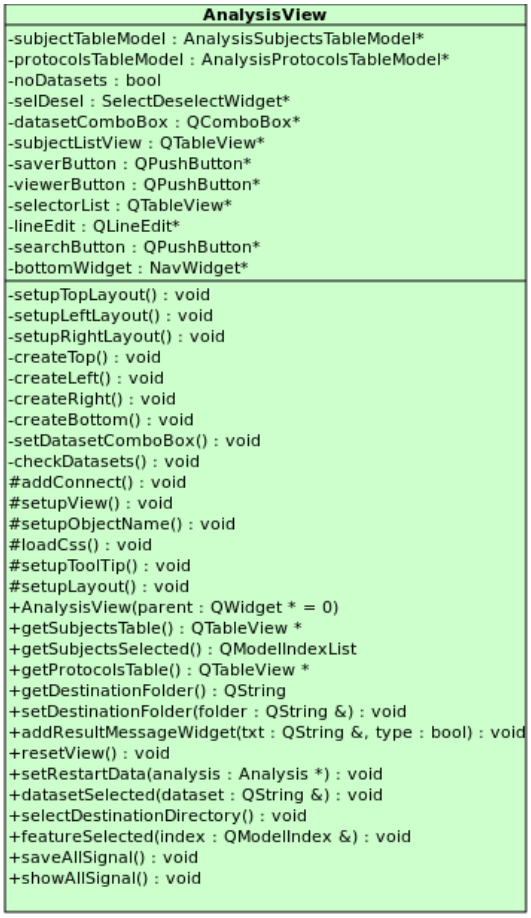

Figura 97: Diagramma Classe AnalysisView: attributi e metodi

Descrizione

Classe che rappresenta il widget per avviare un'analisi su un Dataset$_{\mathbf{G}}$.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di selezionare un Dataset$_{\mathbf{G}}$ sul quale effettuare un'analisi,selezionare quali Subject$_{\mathbf{G}}$ analizzare, scegliere quali risultati salvare per le Feature$_{\mathbf{G}}$ (per l'algoritmo, di default viene sia visualizzato, sia esportato), selezionare il path di dove salvare i risultati e infine far partire l'analisi.

Classi ereditate

- Window::APanel.

Attributi

- `-subjectsTableModel: AnalysisSubjectsTableModel*`

  Descrizione: Rappresenta il model per la tabella contenente i Subject$_{\mathbf{G}}$.
- `-subjectsTableModel: AnalysisSubjectsTableModel*`

Descrizione: Rappresenta il model per la tabella contenente i Protocol$_{\mathbf{G}}$.

- `-noDatasets: bool`

  Descrizione: Rappresenta il flag che vale true se non sono presenti Dataset$_{\mathbf{G}}$ all'interno dell'applicativo Romeo.

- `- subjectsListView: QTableView*`

  Descrizione: Rappresenta la tabella contenente la lista di Subject$_{\mathbf{G}}$.

- `-selDesel: SelectDeselectWidget*`

  Descrizione: Rappresenta il widget per selezionare/deselezionare tutti gli oggetti della lista.

- `-searchButton:QPushButton*`

  Descrizione: Rappresenta il pulsante per la ricerca della cartella dove salvare i risultati dell'analisi.

- `-saverButton:QPushButton*`

  Descrizione: Rappresenta il pulsante per il salvataggio dei risultati delle Feature Extractor$_{\mathbf{G}}$.

- `-viewerButton:QPushButton*`

  Descrizione: Rappresenta il pulsante per la visualizzazione dei risultati delle Feature Extractor$_{\mathbf{G}}$ dopo averli calcolati.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro, la visualizzazione della guida interattiva, la possibilità di eliminare un Dataset$_{\mathbf{G}}$ e confermare successivamente l'operazione.

Metodi

- `+ DatasetsView(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe DatasesView.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di DatasetsView.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note:

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `-createTop():void`

Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei Subject$_{\mathbf{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-setDatasetComboBox():void`

Descrizione: Metodo che ha il compito di costruire la comboBox contenente i Dataset$_{\mathbf{G}}$ presenti all'interno dell'applicativo Romeo.

- `-checkDatasets():void`

Descrizione: Metodo che testa se sono presenti Dataset$_{\mathbf{G}}$ da mostrare.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+getSubjectsTable():QTabelView*`

Descrizione: Metodo che ritorna il puntatore al campo dati *subjectTableModel*.

Note

- questo metodo deve essere marcato costante.

- `+getProtocolsTable():QTabelView*`

Descrizione: Metodo che ritorna il puntatore al campo dati *protocolTableModel*.

Note

- questo metodo deve essere marcato costante.

- `+getSubjectsSelected():QModelIndexList`

Descrizione: Metodo che ritorna i $\text{Subject}_{\mathbf{G}}$ selezionati nella tabella; ritorna una lista di indici.

Note

- questo metodo deve essere marcato costante.

- `+getDestinationFolder(): QString`

Descrizione: Metodo che ritorna la cartella di destinazione selezionata.

Note

- questo metodo deve essere marcato costante.

- `+setDestinationFolder(folder : const QString&): void`

Descrizione: Metodo che ritorna la cartella di destinazione selezionata.

Argomenti:

- `folder:const QString&`
  Rappresenta il path della cartella di destinazione.

- `+ addResultMessageWidget(txt: const QString&, type: bool):void`

Descrizione: Metodo che ha il compito di aggiungere il messaggio di risultato per avvenuta operazione o meno a seconda del valore del secondo parametro passato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da mostrare all'utente all'interno del widget;
- `type : bool`
  Vale true se il messaggio è un messaggio di successo, false se il messaggio è di errore.

Note:

- il metodo deve essere marcato virtuale.

- `+resetView(): void`

Descrizione: Metodo che reimposta tutti i campi della view.

- `+setRestartData(analysis: Analysis*): void`

Descrizione: Metodo che imposta la view con i dati riguardanti l'analisi da far ripartire

Argomenti:

- `analysis: Analysis*`
  Rappresenta l'analisi da far ripartire.

- `+featureSelected(index:const QModelIndex&) :void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona una Feature$_{\mathbf{G}}$.

Attributi:

- `index: const QModelIndex&`
  rappresenta l'indice della Feature$_{\mathbf{G}}$ selezionato.

- `+datasetSelected(dataset:const QString&) : void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona una Feature$_{\mathbf{G}}$.

Attributi:

- `dataset: const QString&`
  rappresenta il nome del Dataset$_{\mathbf{G}}$ selezionato.

- `+selectDestinationDirectory():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente preme il pulsante per la ricerca della cartella in cui salvare i risultati.

- `+saveAllSignal() :void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente preme il pulsante per salvare o non salvare tutto.

- `+showAllSignal() :void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente preme il pulsante per visualizzare o non visualizzare tutto.

5.1.13 DetailedResult (class)

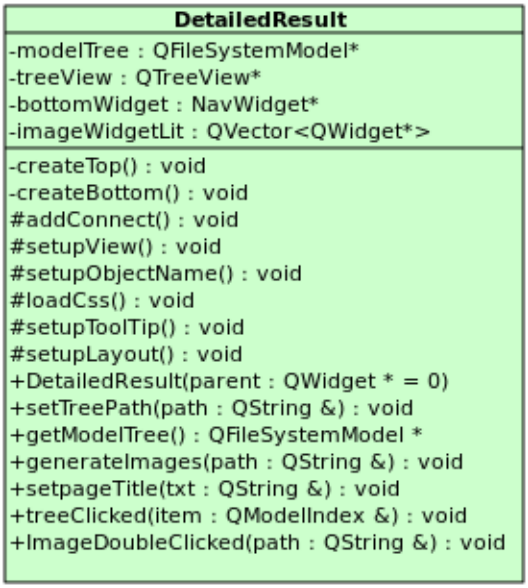

Figura 98: Diagramma Classe DetailedResult: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutti i gruppi di Subject$_{\mathbf{G}}$ presenti all'interno di Romeo.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutti i gruppi di Subject$_{\mathbf{G}}$ fino a quel momento creati e memorizzati dentro a Romeo. Selezionando un gruppo, sarà possibile visualizzare le informazioni relative a quel gruppo, come per esempio i Subject$_{\mathbf{G}}$ contenuti al suo interno.

Classi ereditate

- Window::APanel.

Attributi

- `-treeView: QTreeView*`

  Descrizione: Rappresenta l'albero dei risultati di un'analisi. L'albero è ordinabile per l'elenco dei Subject$_{\mathbf{G}}$ su cui è stata effettuata l'analisi oppure per Protocol$_{\mathbf{G}}$ presenti nel Dataset$_{\mathbf{G}}$ su cui è stata eseguita l'analisi.

- `-modelTree: QFileSystemModel*`

  Descrizione: Rappresenta il modello per l'albero dei risultati delle analisi.

- `-imageWidgetLit: QVector<QWidget*>`

  Descrizione: Rappresenta il vettore contenenente i widget delle immagini.

- `bottomWidget:NavWidget*`

Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro,e la visualizzazione della guida interattiva.

Metodi

- `+ DetailedResult(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe DetailedResult.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di DetailedResult.

- `#setupLayout():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note:

  - questo metodo deve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `-createTop():void`

  Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei $\text{Subject}_{\text{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

  Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

  Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

  Note

  - questo metododeve essere marcato virtuale;
  - questo metodo è stato ridefinito.

- `#setupObjectName():void`

  Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+setTreePath(path: const QString&): void`

Descrizione: Metodo che imposta il path dell'albero da visualizzare

Argomenti:

- `path: const QString&`
  Rappresenta il path dell'albero.

- `+generateImages(path: const QString&): void`

Descrizione: Metodo che genera le immagini raggiungibili dal valore contenuto nel parametro

Argomenti:

- `path: const QString&`
  Rappresenta il path delle immagini da visualizzare.

- `+setpageTitle(txt: const QString&): void`

Descrizione: Metodo che imposta il titolo nella parte alta della pagina.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da impostare come titolo.

- `+getModelTree(): QFileSystemModel*`

Descrizione: Metodo che ritorna il campo dati *modelTree*.

- `+ treeClicked(item:const QModelIndex&) : void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente seleziona un elemento dall'albero.

Attributi:

- `item: const QModelIndex&`
  Rappresenta l'indice nell'albero selezionato.

- `+ imageDoubleClicked(path:const QString&) : void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente fa un doppio click sull'immagine visualizzata.

Attributi:

- `path: const QString&`
  Rappresenta il path relativo all'immagine.

5.1.14 ResultsView (class)

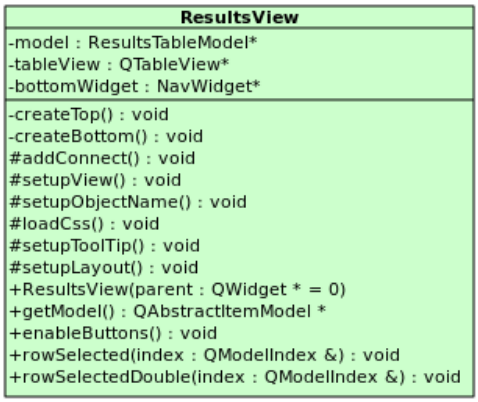

Figura 99: Diagramma Classe ResultsView: attributi e metodi

Descrizione

Classe che rappresenta il widget per la visualizzazione di tutte le analisi effettuate contenute dentro a Romeo.Contiene sia le analisi complete, effettuate con successo sia quelle interrotte che quelle incomplete.

Utilizzo

La classe implementerà i metodi virtuali puri della superclasse inoltre darà la possibilità all'utente di visualizzare l'elenco di tutte le analisi effettuate, e di aprire una nuova finestra per vedere nel dettaglio i risultati di una specifica analisi.

Classi ereditate

- Window::APanel.

Attributi

- `-model: ResultsTableModel*`

  Descrizione: Rappresenta il modello per la tabella dei risultati delle analisi effettuate.

- `-tableView: QTableView*`

  Descrizione: Contiene la lista di tutte le analisi effettuate anche se non complete (non eseguite su tutti i $\text{Subject}_{\mathbf{G}}$del $\text{Dataset}_{\mathbf{G}}$) e interrotte.

- `bottomWidget:NavWidget*`

  Descrizione: Puntatore al widget che rappresenta la parte bassa della finestra contentente i pulsanti per tornare indietro,e la visualizzazione della guida interattiva.

Metodi

- `+ ResultsView(parent : QWidget*=0)`

Descrizione: Costruttore per la classe ResultsView.

Argomenti:

- `parent: QWidget*=0`
  Puntatore al QWidget padre di ResultsView.

- `#setupLayout():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `-createTop():void`

Descrizione: Metodo che ha il compito di costruire la parte in alto del widget contenente l'elenco dei $\text{Subject}_{\text{G}}$e a lato lo spazio per visualizzare le informazioni.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per ritornare alla pagine iniziale, e per l'accesso alla guida.

- `-setupTopLayout():void`

Descrizione: Metodo che ha il compito di impostare la parte alta del layout della finestra.

- `#loadCss():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metododeve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupObjectName():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupToolTip():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#addConnect():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato costante;
- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `#setupView():void`

Descrizione: Metodo che implementa il contratto fornito dalla classe astratta APanel.

Note

- questo metodo deve essere marcato virtuale;
- questo metodo è stato ridefinito.

- `+ enableButtons():void`

Descrizione: Metodo che ha il compito di abilitare i pulsanti presenti nella view.

- `+getModelTree(): QFileSystemModel*`

Descrizione: Metodo che ritorna il campo dati *model*.

Note:

- il metodo deve essere marcato come costante;

- `+ rowSelected(index :const QModelIndex&) : void` (signal)

Descrizione: Signal$_{G}$ emesso quando l'utente seleziona un elemento dalla tabella.

Attributi:

- `index: const QModelIndex&`
  Rappresenta l'indice della tabella selezionato.

- `+ rowSelected(index :const QModelIndex&) : void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente fa il doppio click su un elemento dalla tabella.

Attributi:

- `index: const QModelIndex&`
  Rappresenta l'indice della tabella selezionato.

5.2 Specifica componenti View::Dialog

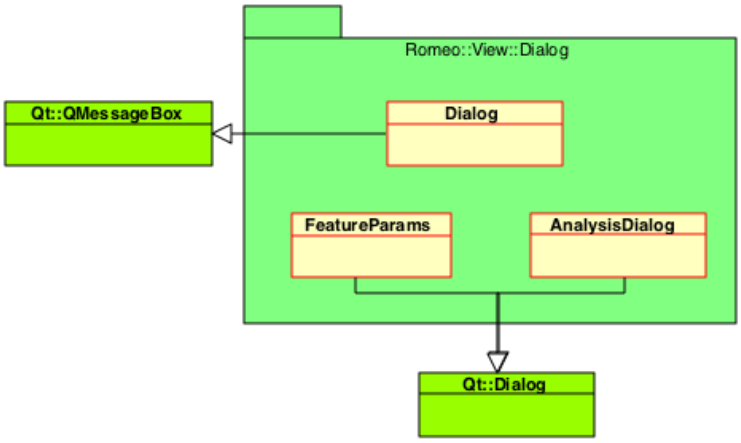

Figura 100: Componente Romeo::View::Dialog

Componente che contiene tutte le classi che rappresentano le finestre di dialogo con cui l'utente potrà interagire

5.2.1 Dialog (class)

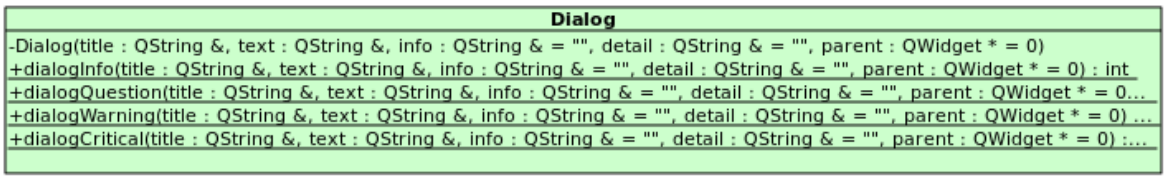

Figura 101: Diagramma Classe Dialog: attributi e metodi

Descrizione

Classe che rappresenta le varie finestre di dialogo che mandano all'utente un messaggio, e che attende la pressione di un pulsante da parte dell'utente per proseguire. Possono essere messaggi che se accettati, interrompono la regolare operazione che era in esecuzione prima dell'apertura del messaggio di dialogo.

Utilizzo

La classe viene utilizzata per dare messaggi all'utente.

Classi ereditate

- Qt::QMessageBox.

Metodi

- `- Dialog(title:QString\&, text:QString\&, info:QString\&, detail:QString\&, parent:QWidget`

  Descrizione Costruttore privato per la classe Dialog. Viene invocato dai metodi statici che mette a disposizione la classe in esame.

  Argomenti

  - `title: QString\\&`
    title viene settato come titolo della finestra di dialogo;
  - `text: QString\\&`
    text viene utilizzato per impostare il testo della finestra di dialogo;
  - `info: QString\&`
    info viene utilizzato per impostare le informazioni della finestra di dialogo;
  - `detail: QString\&`
    detail viene utilizzato per impostare i dettagli che la finestra di dialogo fornisce all'utente;
  - `parent: QWidget*=0`
    Puntatore al QWidget padre di Dialog.

- `+ static dialogInfo(title:QString\&, text:QString\&, info:QString\&, detail:QString\&, parent:QWidget * = 0): int`

  Descrizione: Metodo statico che invoca la creazione di una finestra di dialogo che da un messaggio informativo. Ritorna il numero del pulsante che l'utente ha premuto.

Note:

- Questo metodo deve essere marcato statico.
- Gli argomenti del metodo, vengono passati al costruttore della classe *Dialog*.

- `+ static dialogCritical(title:QString\&, text:QString\&, info:QString\&, detail:QString\&, parent:QWidget * = 0): int`

Descrizione: Metodo statico che invoca la creazione di una finestra di dialogo che da un messaggio di criticità. Ritorna il numero del pulsante che l'utente ha premuto.

Note:

- Questo metodo deve essere marcato statico.
- Gli argomenti del metodo, vengono passati al costruttore della classe *Dialog*.

- `+ static dialogQuestion(title:QString\&, text:QString\&, info:QString\&, detail:QString\&, parent:QWidget * = 0): int`

Descrizione: Metodo statico che invoca la creazione di una finestra di dialogo che richiede una risposta da parte dell'utente. Ritorna il numero del pulsante che l'utente ha premuto.

Note:

- Questo metodo deve essere marcato statico.
- Gli argomenti del metodo, vengono passati al costruttore della classe *Dialog*.

- `+ static dialogCritical(title:QString\&, text:QString\&, info:QString\&, detail:QString\&, parent:QWidget * = 0): int`

Descrizione: Metodo statico che invoca la creazione di una finestra di dialogo che da un messaggio di warning. Ritorna il numero del pulsante che l'utente ha premuto.

Note:

- Questo metodo deve essere marcato statico.
- Gli argomenti del metodo, vengono passati al costruttore della classe *Dialog*.

5.2.2 FeatureParams (class)

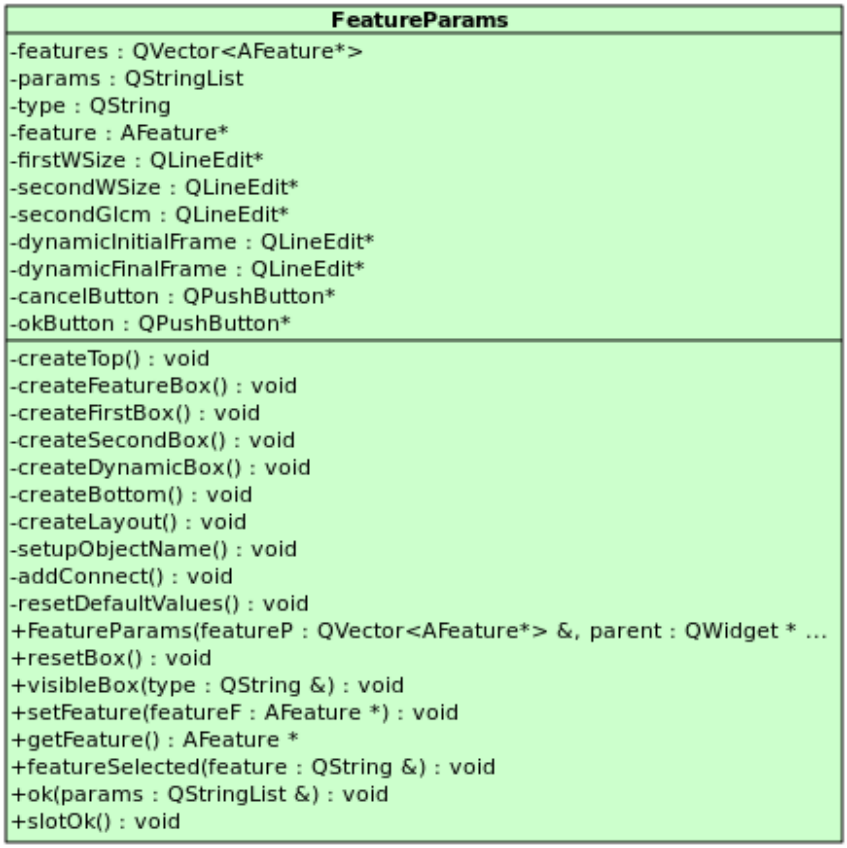

Figura 102: Diagramma Classe FeatureParams: attributi e metodi

Descrizione

Classe che rappresenta la finestra di dialogo che permette all'utente di impostare i valori dei parametri per una determinata Feature$_{\mathbf{G}}$.

Utilizzo

La classe viene utilizzata dalla finestra *NewProtocolView* alla pressione del pulsante che permette l'inserimento di una nuova Feature$_{\mathbf{G}}$ nel Protocol$_{\mathbf{G}}$in creazione. Dà la possibilità di impostare i valori dei parametri che richiede la Feature$_{\mathbf{G}}$, dando comunque un valore di default per ogni parametro.

Classi ereditate

- Qt::QDialog.

Attributi

- `-features: QVector<AFeature*>`

  Descrizione Contiene la lista di tutte le Feature$_{\mathbf{G}}$ rese disponibili da Romeo.

- `- params:QStringList*`

Descrizione Lista contentente tutti i parametri richiesti dalla Feature$_G$ che si vuole aggiungere.

- `- type:QString`

Descrizione Stringa che contiene il tipo della Feature$_G$ che si sta creando.

- `- feature: AFeature*`

Descrizione: Puntatore polimorfo alla Feature$_G$ selezionata.

- `- featuresComboBox: ComboBox*`

Descrizione: Visualizza le Feature$_G$ contenute nel campo dati *features*.

- `- firstWSize: QLineEdit*`

Descrizione: Linea di testo per impostare la dimensione della finestra per la Feature$_G$ di primo ordine.

- `- secondWSize: QLineEdit*`

Descrizione: Linea di testo per impostare la dimensione della finestra per la Feature$_G$ di secondo ordine.

- `- secondGlcm: QLineEdit*`

Descrizione: Linea di testo per impostare il valore della GLCM$_G$.

- `- dynamicWSize: QLineEdit*`

Descrizione: Linea di testo per impostare la dimensione della finestra per la Feature$_G$ di tipo dinamico.

- `- dynamicInitialFrame: QLineEdit*`

Descrizione: Linea di testo per impostare il frame iniziale per la Feature$_G$ dinamica.

- `- dynamicInitialFrame: QLineEdit*`

Descrizione: Linea di testo per impostare il frame finale per la Feature$_G$ dinamica.

- `- cancelButton: QPushButton*`

Descrizione: pulsante che permette all'utente di eliminare le modifiche fatte e non ancora salvate.

- `- okButton: QPushButton*`

Descrizione: pulsante che permette all'utente di confermare i dati inseriti e di conseguenza salvarli all'interno dell'applicativo Romeo.

Metodi

- `- FeatureParams(featureP: QVector<AFeature*>\&, parent:QWidget*=0)`

Descrizione: Costruttore per la classe.

Argomenti

- `featureP: QVector<AFeature*>\&`
  Vector di puntatori ad oggetti di tipo *AFeature*. Utilizzato per inizializzare il campo dati *features*;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di FeatureParams.

- `-setupLayout(): void`

Descrizione: Metodo che ha il compito di impostare il layout del widget.

- `-setupObjectName():void`

Descrizione: Metodo che ha il compito di impostare il nome di ogni oggetto, contenuto nel widget.

- `-addConnect(): void`

Descrizione: Metodo che ha il compito di fissare tutte le istruzioni *connect* degli oggetti che emetteranno un signal$_{\mathbf{G}}$ verso il rispettivo controller.

- `-setupView(): void`

Descrizione: Metodo che ha il compito di impostare il layout del widget.

- `-createTop():void`

Descrizione: Metodo che ha il compito di creare la parte in alto della finestra di dialogo contenente i campi per l'inserimento dei valori dei parametri per la Feature$_{\mathbf{G}}$ selezionata.

- `-createLayout():void`

Descrizione: Metodo che ha il compito di creare il layout per la finestra di dialogo.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente il pulsante per il salvataggio dei valori inseriti oppure per poter reimpostare la form.

- `-createFeatureBox():void`

Descrizione: Metodo che ha il compito di costruire il box contenente i campi per l'inserimento delle informazioni necessarie.

- `-createFirstBox():void`

Descrizione: Metodo che ha il compito di costruire il box contenente i campi per l'inserimento delle informazioni necessarie, relative alle Feature$_{\mathbf{G}}$ di primo ordine.

- `-createFirstBox():void`

Descrizione: Metodo che ha il compito di costruire il box contenente i campi per l'inserimento delle informazioni necessarie, relative alle Feature$_G$ di secondo ordine.

- `-createFirstBox():void`

Descrizione: Metodo che ha il compito di costruire il box contenente i campi per l'inserimento delle informazioni necessarie, relative alle Feature$_G$ dinamiche.

- `-resetDefaultValues():void`

Descrizione: Metodo che ha il compito di impostare i campi di input ai valori di default.

- `+ visibleBox(type:QString\&): void`

Descrizione: Metodo che imposta la finestra di dialogo mostrando i campi corretti per l'inserimento dei valori dei parametri in base al valore della stringa passata come parametro al metodo.

Argomenti:

  - `type: QString\&`
    rappresenta il tipo di Feature$_G$ ovvero se di primo, secondo ordine o dinamica.

- `+ resetBox(): void`

Descrizione: Metodo che reimposta la finestra di dialogo, mostrando solamente il menu di selezione con le Feature disponibili.

- `+ visibleBox(type:QString\&): void`

Descrizione: Metodo che imposta la finestra di dialogo mostrando i campi corretti per l'inserimento dei valori dei parametri in base al valore della stringa passata come parametro al metodo.

Argomenti

  - `type: QString\&`
    rappresenta il tipo di Feature$_G$ ovvero se di primo, secondo ordine o dinamica.

- `+ setFeature(featureF:AFeature*): void`

Descrizione: Metodo che imposta il campo dati *feature*.

Argomenti

  - `featureF: AFeature*`
    puntatore polimorfo alla Feature$_G$ selezionata il cui valore verrà dato al campo dati *feature*.

- `+ getFeature(): AFeature*`

Descrizione: Metodo che ritorna un puntatore polimorfo alla Feature$_{\mathbf{G}}$ selezionata.

- `+ featureSelected(featureF:QString\&):void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona una voce dal menu di selezione.

  Argomenti:

  - `featureF: QString\&`
    identifica la selezione fatta dall'utente.

- `+ ok(params:QStringList):void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando dallo slot$_{\mathbf{G}}$ *slotOk* quando i parametri inseriti hanno un valore corretto.

  Argomenti

  - `params: QStringList\&`
    Lista contentente i valori di tutti i parametri settati dall'utente.

- `+ slotOk():void` (slot)

  Descrizione: Slot$_{\mathbf{G}}$ che riceve il signal$_{\mathbf{G}}$ del click sul pulsante *ok* della finestra di dialgo. Ha il compito di controllare se i valori inseriti per i parametri sono corretti e in caso, emette il signal$_{\mathbf{G}}$ *ok*.

5.2.3 AnalysisDialog (class)

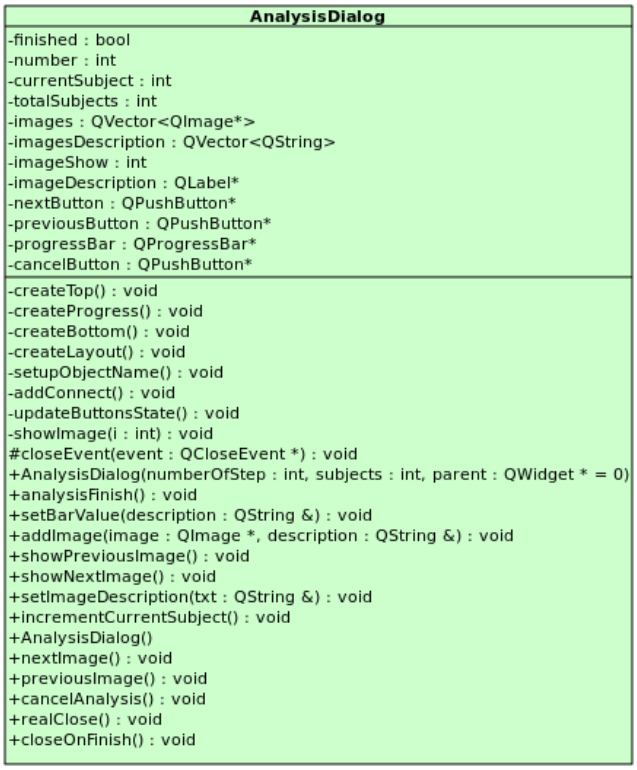

Figura 103: Diagramma Classe AnalysisDialog: attributi e metodi

Descrizione

Classe che rappresenta la finestra di dialogo che viene visualizzata dall'utente durante l'esecuzione dell'analisi.

Utilizzo

La classe viene creata quando l'utente ha scelto il dataset$_{\mathbf{G}}$ su cui eseguire l'analisi e le opzioni per la visualizzazione ed esportazione delle Feature$_{\mathbf{G}}$. Mostra all'utente la barra di avanzamento dell'analisi; ad ogni Feature$_{\mathbf{G}}$ se precedentemente selezionato, verrà mostrato il risultato ottenuto dall'applicazione della Feature$_{\mathbf{G}}$ all'immagine associata al Subject$_{\mathbf{G}}$. Mette inoltre a disposizione la possibilità di interrompere l'analisi, di proseguire senza più mostrare i risultati delle Feature$_{\mathbf{G}}$ oppure di proseguire con l'analisi con le impostazioni scelte.

Classi ereditate

- Qt::QDialog.

Attributi

- `-nextButton: QPushButton*`

  Descrizione: Pulsante che permette di proseguire l'analisi con le impostazioni scelte prima di inziare l'analisi.

- `-previousButton:QPushButton*`

Descrizione: Pulsante che permette di visualizzare l'immagine precedente a quella che si sta visualizzando al momento.

- `-progressBar:QProgressBar*`

  Descrizione: Identifica la barra di avanzamento che informa l'utente sul tempo mancante alla terminazione dell'analisi.

- `-cancelButton: QPushButton*`

  Descrizione: Pulsante che permette l'interruzione dell'analisi in corso.

- `-totalSubject: int`

  Descrizione: Rappresenta il numero totale di $\text{Subject}_{\mathbf{G}}$ presenti all'interno del $\text{Dataset}_{\mathbf{G}}$ di cui si vuole far partire l'analisi.

- `-number: int`

  Descrizione: Rappresenta il numero di $\text{Subject}_{\mathbf{G}}$ su cui si vuole far partire l'analisi.

- `-currentSubject: int`

  Descrizione: Rappresenta il numero del $\text{Subject}_{\mathbf{G}}$ attualmente processato.

- `-finished: bool`

  Descrizione: Rappresenta un flag che vale true se e solo se l'analisi è stata completata.

- `-imageShow: int`

  Descrizione: Rappresenta il numero di immagine che è visualizzata.

- `-images: QVector<QImage*>`

  Descrizione: Rappresenta la lista di immagini risultati dall'analisi.

- `-imagesDescription: QVector<QString>`

  Descrizione: Rappresenta la lista delle informazioni delle immagini presenti all'interno di *images*.

Metodi

- `- FeatureParams(featureP: QVector<AFeature*>\&, parent:QWidget*=0)`

  Descrizione: Costruttore per la classe.

Argomenti:

- `featureP: QVector<AFeature*>\&`
  Vector di puntatori ad oggetti di tipo *AFeature*. Utilizzato per inizializzare il campo dati *features*;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di FeatureParams.

- `-updateButtonState(): void`

Descrizione: Metodo che ha il compito di aggiornare lo stato dei pulsanti sotto all'immagine che l'utente sta visualizzando.

- `-setupObjectName():void`

Descrizione: Metodo che ha il compito di impostare il nome di ogni oggetto, contenuto nel widget.

- `-showImage(i : int):void` Metodo che ha il compito di visualizzare l'immagine che si trova all'indice rappresentato dal parametro passato al metodo.

Argomenti:

- `i: int`
  rappresenta il numero dell'immagine da far visualizzare nella view.

- `-addConnect(): void`

Descrizione: Metodo che ha il compito di fissare tutte le istruzioni *connect* degli oggetti che emetteranno un signal$_{\mathbf{G}}$ verso il rispettivo controller.

- `-createProgress(): void`

Descrizione: Metodo che ha il compito creare il layout contentente la barra di avanzamento dell'analisi.

- `-createTop():void`

Descrizione: Metodo che ha il compito di creare la parte in alto della finestra di dialogo contenente la visualizzazione dell'anteprima dei risultati intermedi e la barra di avanzamento.

- `-createLayout():void`

Descrizione: Metodo che ha il compito di creare il layout per la finestra di dialogo.

- `-createButtom():void`

Descrizione: Metodo che ha il compito di costruire la parte in basso del widget contenente i pulsanti per l'interazione con l'utente.

- `#closeEvent(QCloseEvent* event):void`

Descrizione: Metodo virtuale che ridefinisce il comportamento del metodo presente nella classe *Qt::QDialog*. Il metodo viene invocato quando Qt$_{\mathbf{G}}$ riceve una richiesta di chiusura di una finestra di più alto livello.

Argomenti:

- `event: QCloseEvent*`
  Rappresenta l'evento con il quale il metodo verrà invocato; nel nostro caso riguarda un evento di chiusura della finestra.

- `+analysisFinish():void`

Descrizione: Metodo che ha il compito di aggiornare la view quando l'analisi è stata completata.

- `+setBarValue(description: const QString\&):void`

Descrizione: Metodo che ha il compito di incrementare la barra di avanzamento e imposta il testo informativo.

Argomenti:

- `description: const QString\&`
  Rappresenta il testo informativo da far visualizzare all'utente nella barra di avanzamento.

- `+addImage(image: QImage* ,description: const QString\&):void`

Descrizione: Metodo che ha il compito riceve l'immagine da mostrare nella finestra di dialogo.

Argomenti:

- `image: QImage*`
  Rappresenta l'immagine da visualizzare nella finestra di dialogo
- `description: const QString\&`
  Rappresenta la descrizione dell'immagine passata come parametro.

- `+showPreviousImage():void`

Descrizione: Metodo che ha il compito di visualizzare nella finestra di dialogo l'immagine precedente a quella attuale, se questa esiste.

- `+showNextImage():void`

Descrizione: Metodo che ha il compito di visualizzare nella finestra di dialogo l'immagine successiva a quella attuale, se questa esiste.

- `+setImageDescription(txt: const QString\&):void`

Descrizione: Metodo che ha il compito di impostare la descrizione per la corrente immagine che si sta visualizzando.

Argomenti:

- `txt: const QString\&`
  Rappresenta il testo informativo relativa all'immagine corrente.

- `+incrementCurrentSubject():void`

Descrizione: Metodo che ha il compito di incrementare il numero attuale di Subject$_{\mathbf{G}}$ presenti nella finestra di dialogo.

- `+nextImage():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona il pulsante next dalla finestra di dialogo.

- `+previousImage():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona il pulsante previous dalla finestra di dialogo.

- `+cancelAnalysis():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona il pulsante cancel dalla finestra di dialogo.

- `+realClose():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'utente seleziona il pulsante yes dalla conferma di voler interrompere l'analisi.

- `+nextImage():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando l'analisi è stata completata.

5.3 Specifica componenti View::Component

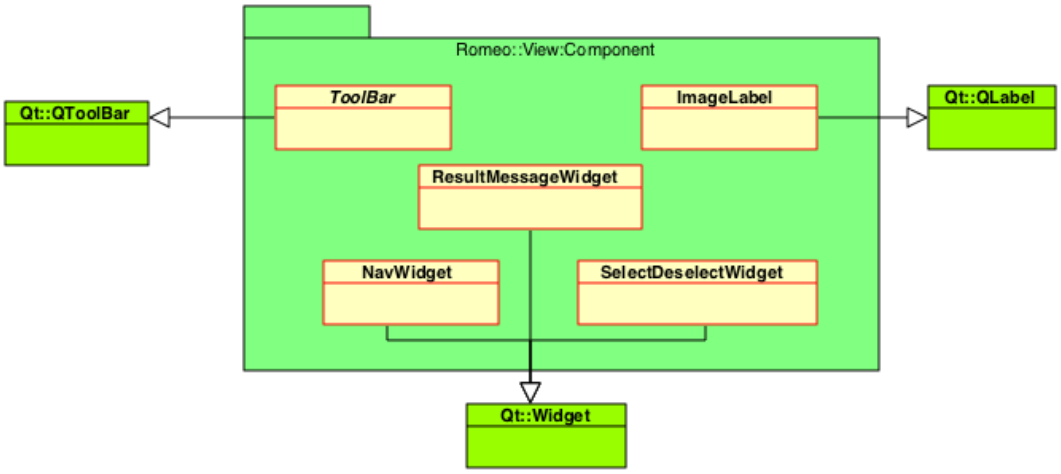

Figura 104: Componente Romeo::View::Component

5.3.1 ToolBar(class)

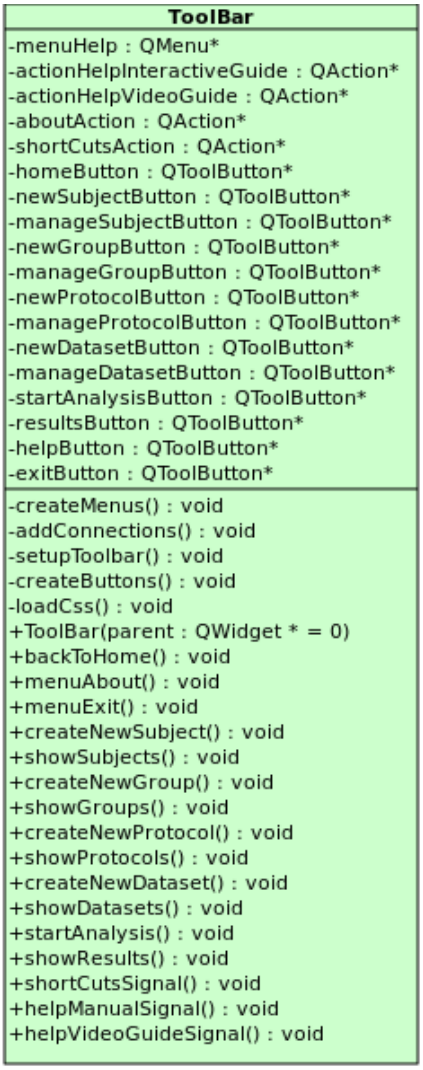

Figura 105: Diagramma Classe ToolBar: attributi e metodi

Descrizione

Classe che rappresenta la componente contenuta nelle finestre per poter filtrare i contenuti delle tabelle presentei nella vista in base a diversi parametri secondo le preferenze dell'utente.

Utilizzo

La classe viene utilizzata per dare all'utente la possibilità di visualizzare i dati tabellari ordinandoli secondo una certa caratteristica scelta dall'utente nel momento in cui si trova sulla finestra che espone una tabella.

Classi ereditate

- Qt::QWidget.

Attributi

- `-menuHelp:QMenu*`

Descrizione: rappresenta il menu per l'help.

- `- actionHelpInteractiveGuide:QAction*`

Descrizione: Rappresenta l'azione l'help interattivo.

- `- actionHelpVideoGuide:QAction*`

Descrizione: Rappresenta l'azione per la video guida, elemento del menu help.

- `- aboutAction:QAction*`

Descrizione: Rappresenta l'azione per l'elemento about del menu help.

- `- shortCutsAction:QAction*`

Descrizione: Rappresenta l'azione per i shortcuts, elemento del menu help.

- `-homeButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la pagina iniziale dell'applicativo Romeo.

- `-newSubjectButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la creazione di un nuovo Subject$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-newGroupButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-newProtocolButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la creazione di un nuovo Protocol$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-newDatasetButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la creazione di un nuovo Dataset$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-manageSubjectButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la gestione dei Subject$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-manageGroupButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la gestione dei gruppi di Subject$_{\mathbf{G}}$ dell'applicativo Romeo.

- `-manageProtocolButton: QToolButton*`

Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la gestione dei Protocol$_{G}$ dell'applicativo Romeo.

- `-manageDatasetButton: QToolButton*`

  Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la gestione dei Dataset$_{G}$ dell'applicativo Romeo.

- `-startAnalysisButton: QToolButton*`

  Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per creare e avviare una nuova analisi dell'applicativo Romeo.

- `-resultsButton: QToolButton*`

  Descrizione: Rappresenta il pulsante della toolbar per raggiungere la view per la visualizzazione dei risultati delle analisi effettuate dell'applicativo Romeo.

- `-helpButton: QToolButton*`

  Descrizione: Rappresenta il pulsante della toolbar per ottenere maggiori aiuti nell'utilizzo dell'applicativo Romeo.

- `-exitButton: QToolButton*`

  Descrizione: Rappresenta il pulsante della toolbar per chiudere l'applicativo Romeo.

Metodi

- `+ ToolBar(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe ToolBar.

  #### Argomenti

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ToolBar.

- `-setupToolBar():void`

  Descrizione: Metodo che ha il compito di impostare le configurazioni del menu e creare i componenti della la toolbar.

- `-createMenus():void`

  Descrizione: Metodo che ha il compito creare il menu di help.

- `-addConnections():void`

  Descrizione: Metodo che ha il compito di impostare le connessioni tra i pulsanti e i Signal$_{G}$.

- `-createButtons():void`

Descrizione: Metodo che ha il compito di creare i pulsanti per la toolbar.

- `-loadCss():void`

Descrizione: Metodo che ha il compito di impostare lo stile per la toolbar.

- `+backToHome():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante home dalla toolbar.

- `+menuAbout():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuta l'azione *about*.

- `+menuExit():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante exit dalla toolbar.

- `+createNewSubject():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la creazione di un nuovo Subject$_{\mathbf{G}}$ dalla toolbar.

- `+createNewGroup():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$ dalla toolbar.

- `+createNewProtocol():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la creazione di un nuovo Protocol$_{\mathbf{G}}$ dalla toolbar.

- `+createNewDataset():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la creazione di un nuovo Dataset$_{\mathbf{G}}$ dalla toolbar.

- `+showSubjects():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la gestione dei Subject$_{\mathbf{G}}$ dalla toolbar.

- `+showGroups():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la gestione dei gruppi di Subject$_{\mathbf{G}}$ dalla toolbar.

- `+showProtocols():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la gestione dei Protocol$_{\mathbf{G}}$ dalla toolbar.

- `+showDatasets():void` (signal)

Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la gestione dei Dataset$_{\mathbf{G}}$ dalla toolbar.

- `+showResults():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la visualizzazione dei risultati dalla toolbar.

- `+startAnalysis():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la creazione e l'avvio di un'analisi dalla toolbar.

- `+shortCutsSignal():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuto il pulsante per la visualizzazione degli shortcut disponibili dalla toolbar.

- `+helpManualSignal():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuta l'azione per la visualizzazione del manuale di utilizzo dalla toolbar.

- `+helpVideoGuideSignal():void` (signal)

  Descrizione: Signal$_{\mathbf{G}}$ emesso quando viene premuta l'azione per la visualizzazione della video guida.

5.3.2 SelectDeselectWidget (class)

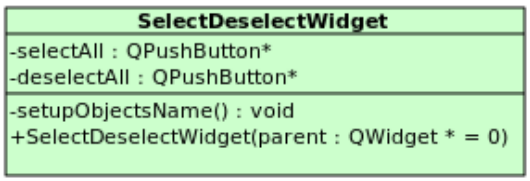

Figura 106: Diagramma Classe SelectDeselectWidget: attributi e metodi

Descrizione

Classe che rappresenta la componente che permette, dove previsto, di velocizzare le operazioni dell'utente qualora voglia selezionare (o deselezionare) l'intero insieme dei dati.

Utilizzo

La classe viene utilizzata dalle viste che presentano un elenco di dati, che l'utente può selezionare per poi compiere determinate azioni sugli elementi scelti. È molto utile quando l'utente ha da selezionare (rispettivamente, deselezionare) un numero molto alto di voci.

Classi ereditate

- Qt::QWidget.

Attributi

- `-selectAll: QPushButton*`

  Descrizione: Pulsante che permette la selezione contemporanea di tutte le voci della tabella.

- `-deselectAll: QPushButton*`

  Descrizione: Pulsante che permette la deselezione contemporanea di tutte le voci della tabella.

Metodi

- `+ SelectDeselectWidget(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe SelectDeselectWidget.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ToolBar.

- `-setupObjectName():void`

  Descrizione: Metodo che ha il compito di impostare i nomi degli oggetti contenuti all'interno del widget *SelectDeselectAll*.

5.3.3 NavWidget (class)

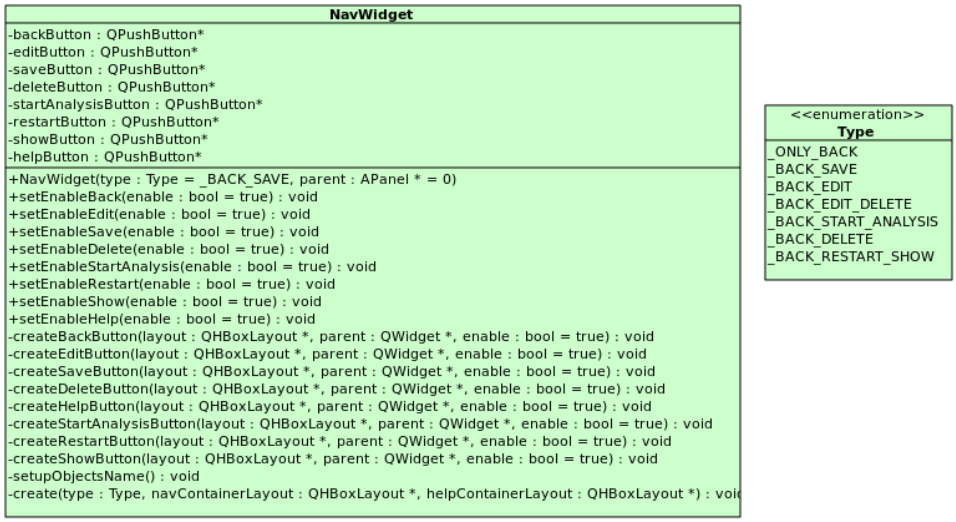

Figura 107: Diagramma Classe NavWidget: attributi e metodi

Descrizione

Classe che rappresenta la componente disposta nella parte bassa del layout della finestra contenente i pulsanti per la navigazione all'indietro, la visualizzazione della guida interattiva e, ove previsti, i pulsanti per la modifica, per la cancellazione e per il salvataggio delle modifiche apportate.

Utilizzo

La classe viene utilizzata da tutte le viste in quanto è sicuramente sempre presente il pulsante per ritornare alla finestra precedente e per l'apertura della guida; gli altri pulsanti saranno presenti ove vi è la possibilità di effettuare operazioni che influiscono sullo stato del sistema come per esempio l'inserimento di un nuovo $\text{Subject}_{\mathbf{G}}$ o la cancellazione di un $\text{Protocol}_{\mathbf{G}}$ o ancora la modifica di un gruppo di $\text{Subject}_{\mathbf{G}}$.

Classi ereditate

- Qt::QWidget.

Attributi

- `-backButton: QPushButton*`

  Descrizione: Pulsante che permette il ritorno alla finestra precedente.

- `-editButton: QPushButton*`

  Descrizione: Pulsante che permette la modifica di alcuni dati visualizzati nella vista.

- `-saveButton: QPushButton*`

Descrizione: Pulsante che permette il salvataggio delle modifiche precedentemente effettuate.

- `-deleteButton: QPushButton*`

  Descrizione: Pulsante che permette la rimozione degli oggetti selezionati.

- `-helpButton: QPushButton*`

  Descrizione: Pulsante che permette la visualizzazione della guida interattiva.

- `-startButton: QPushButton*`

  Descrizione: Pulsante che permette di avviare una nuova analisi.

- `-restartButton: QPushButton*`

  Descrizione: Pulsante che permette di riavviare un'analisi.

- `-showButton: QPushButton*`

  Descrizione: Pulsante che permette visualizzare i risultati delle analisi.

Metodi

- `+ NavWidget(type: Type, parent : QWidget*=0)`

  Descrizione: Costruttore per la classe NavWidget.

  Argomenti:

  - `type: Type`
    suggerisce la tipologia di NavWidget da costruire:
    - solo ritorno alla finestra precedente e visualizzazione guida (es. visualizzazione $\text{Subject}_{\mathbf{G}}$ presenti nel sistema);
    - ritorno alla finestra precedente e salvataggio operazioni (es. per le viste che si occupano di creare nuovi elementi);
    - ritorno alla finestra precedente e possibilità di modifica dei campi (es. visualizzazione gruppi di $\text{Subject}_{\mathbf{G}}$);
    - ritorno alla finestra precedente e possibilità di cancellare elementi selezionati (es. visualizzazione $\text{Protocol}_{\mathbf{G}}$);
    - ritorno alla finestra precedente e possibilità di avviare l'analisi (es. starAnalysis);
    - ritorno alla finestra precedente e possibilità di riavviare l'analisi e visualizzare i risultati.
  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ToolBar.

- `-setupObjectName():void`

  Descrizione: Metodo che ha il compito di impostare i nomi degli oggetti contenuti all'interno del widget *SelectDeselectAll*.

- `-createBackButton(layout:QHBoxLayout*, parent:QWidget*= 0, enable:bool= true): void`

Descrizione: Metodo che ha il compito di creare il campo dati *backButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createEditButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *editButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createSaveButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool= true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *saveButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createDeleteButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *deleteButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createHelpButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *helpButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createStartAnalysisButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *startButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createShowButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *showButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-createRestartButton(layout:QHBoxLayout*, parent:QWidget*=0, enable:bool=true):void`

Descrizione: Metodo che ha il compito di creare il campo dati *restartButton*.

Argomenti:

- `layout: QHBoxLayout*`
  box per dare un'allineamento orizzontale del pulsante;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di NavWidget;
- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableBack(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *back*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableEdit(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *edit*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableSave(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *save*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableDelete(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *delete*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableStartAnalyis(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *startAnalysis*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableRestart(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *restart*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableShow(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *show*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `+setEnableHelp(enable:bool=true):void`

Descrizione: Metodo che ha il compito di impostare la proprietà *enable:* del pulsante *help*.

Argomenti:

- `enable: bool=true`
  caratteristica del pulsante ad essere abilitato alla pressione, di default è true.

- `-create(type: Type, navContainerLayout: QHBoxLayout, helpContainerLayout: QHBoxLayout)`
Costruttore per la classe NavWidget.

Argomenti:

- `type: Type`
  Rappresenta quali pulsanti verranno visualizzati nel widget;
- `navContainerLayout: QHBoxLayout*`
  Rappresenta il layout per i pulsanti di navigazione;
- `helpContainerLayout: QHBoxLayout*`
  Rappresenta il layout per il pulsante di help.

5.3.4 ImageLabel (class)

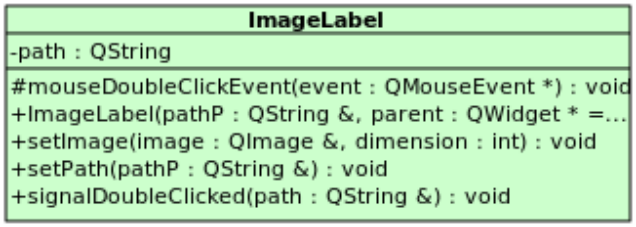

Figura 108: Diagramma Classe ImageLabel: attributi e metodi

Descrizione

Classe che rappresenta l'immagine di un Subject$_G$ o dell'analisi alla con la quale si può interagire facendo doppio click.

Utilizzo

La classe viene utilizzata dalle classi SubjectsView e detailedResultsView per visualizzare le anteprime delle immagini con le quali l'utente può interagire.

Classi ereditate

- Qt::QLabel.

Attributi

- `-path: QString`

  Descrizione: Rappresenta il path dell'immagine.

Metodi

- `+ ImageLabel(pathP: const QString&, parent : QWidget*=0)`

  Descrizione: Costruttore per la classe ImageLabel.

  Argomenti:

  - `pathP: const QString&`
    Rappresenta il path relativo all'immagine.
  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ImageLabel.

- `# mouseDoubleClickEvent(event:QMouseEvent*):void`

  Descrizione: Metodo virtuale della classe QLabel ridefinito per il doppio click del mouse.

Argomenti:

- `event: QMouseEvent*`
  Rappresenta l'evento di pressione del mouse.

- `+ setImage(image:QImage &, dimension:int):void`

Descrizione: Metodo che ha il compito di impostare l'immagine nella label con la dimensione passata come parametro

Argomenti:

- `image: QImage&`
  Rappresenta l'immagine da ridimensionare;
- `dimension: int`
  Rappresenta la dimensione a cui scalare l'immagine.

- `+ signalDoubleClicked(path:const QString &) :void`

Descrizione: Signal_{G} emesso quando sulla lable viene fatto il doppio click con il mouse.

Argomenti:

- `path: const QString&`
  Rappresenta il path dell'immagine contenuta nella label;

5.3.5 ResultMessageWidget (class)

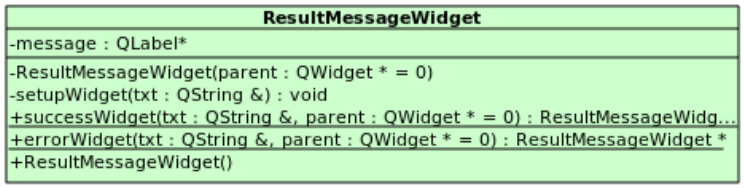

Figura 109: Diagramma Classe ResultMessageWidget: attributi e metodi

Descrizione

Classe che rappresenta il widget che visualizza i messaggi dopo aver fatto un azione di salvataggio, modifica o eliminazione. Il messaggio può essere di successo o di errore.

Utilizzo

La classe viene utilizzata dalle varie viste per segnalare all'utente il successo o errore di un'azione fatta.

Classi ereditate

- Qt::QWidget.

Attributi

- `-message: QLabel*`

  Descrizione: Rappresenta la label per il messaggio di testo da visualizzare.

Metodi

- `-ResultMessageWidget(parent : QWidget*=0)`

  Descrizione: Costruttore per la classe ResultMessageWidget.

  Argomenti:

  - `parent: QWidget*=0`
    Puntatore al QWidget padre di ResultMessageWidget.

- `-setupWidget(txt: const QString&):void`

  Descrizione: Metodo che ha il compito di impostare il componente e il layout del widget

  Argomenti:

  - `txt: const QString&`
    Rappresenta il testo da visualizzare.

- `-successWidget(txt: const QString&,parent: QWidget*=0):ResultMessageWidget*`

Descrizione: Metodo che ha il compito di creare un ResultMessageWidget di successo. Ritorna il puntatore al widget creato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da visualizzare nel widget;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di ResultMessageWidget.

Note:

- il metodo deve essere marcato statico.

- `-errorWidget(txt: const QString&,parent: QWidget*=0):ResultMessageWidget*`

Descrizione: Metodo che ha il compito di creare un ResultMessageWidget di errore. Ritorna il puntatore al widget creato.

Argomenti:

- `txt: const QString&`
  Rappresenta il testo da visualizzare nel widget;
- `parent: QWidget*=0`
  Puntatore al QWidget padre di ResultMessageWidget.

Note:

- il metodo deve essere marcato statico.

6 Specifica componenti Romeo::Controller

6.1 Romeo::Controller

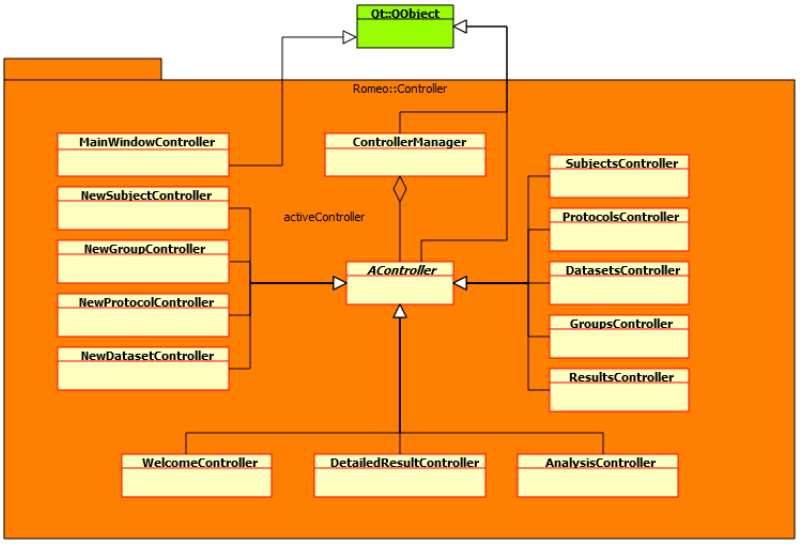

Figura 110: Diagramma package *Romeo::Controller*

6.1.1 AController (abstract)

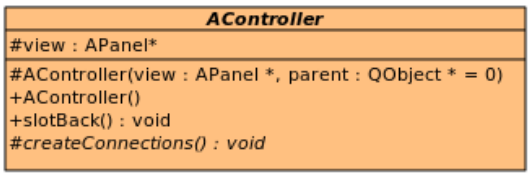

Figura 111: Diagramma classe *AController*

Descrizione: classe astratta che rappresenta un generico controller per un oggetto derivato dalla classe *APanel*

Utilizzo: riceve i signal$_{G}$ dalla vista che sta gestendo e reagisce di conseguenza.

Eredita da:

- Qt::QObject.

Attributi

- `# view : APanel *`

  Descrizione Puntatore all'oggetto *APanel* che l'oggetto *AController* sta controllando. Essendo *APanel* una classe astratta, il tipo dinamico di tale puntatore non sarà mai `APanel *`, ma bensì un puntatore ad una sua sottoclasse.

Metodi

- `# AController(view : APanel *, parent : QObject *)`

  Descrizione: Costruttore protetto che crea un controller sull'oggetto view passato come parametro.

  #### Argomenti

  - `view : APanel *`
    Puntatore all'oggetto *APanel* al quale lavorerà l'oggetto *AController* che verrà costruito.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject* padre dell'oggetto *AController*.

- `# createConnections() : void`

  Descrizione Metodo virtuale puro che fornisce un contratto per la creazione delle varie connect per gestire i signal$_{\mathbf{G}}$ inviati dalla view.

  #### Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale puro.

- `+ slotBack() : void`

  Descrizione Metodo che gestisce la richiesta, da parte dell'utente, di ritornare alla vista precedente.

  #### Note

  - Il metodo deve essere marcato come virtuale.

6.1.2 AnalysisController (class)

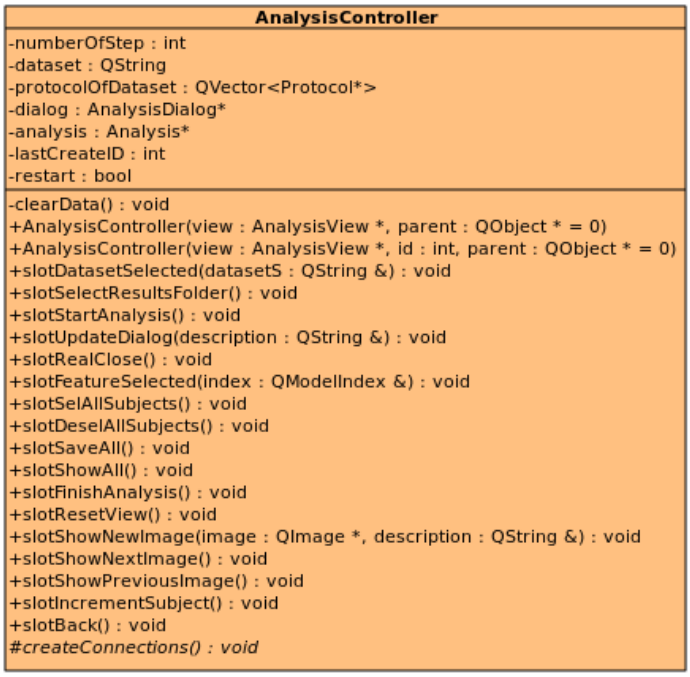

Figura 112: Diagramma classe AnalysisController

Descrizione: classe che rappresenta il controller per un oggetto AnalysisView.

Utilizzo: viene utilizzata per gestire i Signal$_{G}$ emessi da un oggetto Analysisview e reagisce in modo appropriato ogni volta che ne viene catturato uno.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- dataset : QString`

  Descrizione contiene il nome del Dataset$_{G}$ da analizzare

- `- dialog : AnalysisDialog *`

  Descrizione rappresenta il puntatore alla finestra di dialogo per iniziare l'analisi.

- `- analysis : Analysis *`

  Descrizione puntatore all'oggetto analysis

- `- lastCreateID : int`

  Descrizione rappresenta l'id nel database dell'analisi corrente creata.

- `- restart : bool`

Descrizione rappresenta un flag che vale false se l'analisi è una nuova analisi true altrimenti.

- `- protocolsOfDataset : QVector<Protocol*>`

Descrizione rappresenta i $\text{Protocol}_{\mathbf{G}}$ contenuti nel $\text{Dataset}_{\mathbf{G}}$ del quale si vuole fare l'analisi.

Metodi

- `- clearData() : void`

Descrizione metodo che rimuove i dati dell'oggetto controllore

- `- createDialogConnections() : void`

Descrizione metodo che crea le connessioni per la finestra di dialogo di inizio analisi.

- `# createConnections() : void`

Descrizione metodo che ha il compito di creare le connessioni tra la vista che l'oggetto controllore ha associata e il controller stesso.

Note

- Il metodo deve essere marcato come costante.

- `+ AnalysisController(analysis : Analysis *, parent : QObject *)`

Descrizione: costruttore per la creazione del controller di una nuova analisi

Argomenti

- `analysis : Analysis *`
  puntatore alla view associata al controller.
- `parent : QObject *`
  rappresenta il parent dell'oggetto controller in costruzione.

- `+ AnalysisController(analysis : Analysis *, id : int, parent : QObject *)`

Descrizione: costruttore per la classe che ha il compito di creare il controller per un'analisi già esistente da far ripartire.

Argomenti

- `analysis : Analysis *`
  puntatore alla vista associata al controller in costruzione.
- `id : int`
  rappresenta l'id univoco dell'oggetto analisi all'interno del database.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in costruzione

- `+ slotDatasetSelected(datasetS : const QString &) : void`

Descrizione: Slot$_{G}$ che riceve il segnale dalla view associata quando l'utente seleziona un dataset.

Argomenti

- `datasetS : const QString &`
  rappresenta il nome del Dataset$_{G}$ selezionato.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotSelectedResultsFolder() : void`

Descrizione Slot$_{G}$ che riceve il segnale dalla view associata quando l'utente seleziona il pulsante per la cartella in cui verranno esportati i risultati che l'analisi produrrà.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotStartAnalysis() : void`

Descrizione Slot$_{G}$ che riceve il segnale dalla view associata quando l'utente seleziona il pulsante per iniziare l'analisi.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotUpdateDialog(description : const QString &) : void`

Descrizione: Slot$_{G}$ che riceve il segnale dalla view associata quando la finestra di dialogo dell'analisi necessita di aggiornare la vista.

Argomenti

- `description : const QString &`
  rappresenta la descrizione testuale da impostare alla view.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotRealClose() : void`

Descrizione Slot$_{G}$ che riceve il segnale dalla view associata quando la finestra di dialogo ha terminato l'analisi e imposta i puntatori analysis e dialog a null.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotFeatureSelected(index : const QModelIndex &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona le Feature$_{\mathbf{G}}$ da visualizzare o salvare.

Argomenti

- `index : const QModelIndex &`
  rappresenta l'indice della tabella selezionata dall'utente.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotSelAllSubjects() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per selezionare tutti i Subject$_{\mathbf{G}}$ presenti nel Dataset$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotDeselAllSubject() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per deselezionare tutti i Subject$_{\mathbf{G}}$ presenti nel Dataset$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotSaveAll() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per salvare tutte le Feature$_{\mathbf{G}}$ presenti.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowAll() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per visualizzare tutte le Feature$_{\mathbf{G}}$ presenti.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotFinishAnalysis() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'analisi è stata terminata.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotResetView() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata per ripristinare la view.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowNewImage(image : QImage *, description : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando la nuova immagine dell'analisi è pronta per essere mostrata.

Argomenti

- `image : QImage *`
  rappresenta l'immagine da mostrare.
- `description : const QString &`
  rappresenta la descrizione dell'immagine passata come parametro.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowNextImage() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per visualizzare l'immagine successiva dalla finestra di dialogo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowPreviousImage() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante per visualizzare l'immagine precedente dalla finestra di dialogo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotIncrementSubject() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata per aumentare il numero di Subject$_{\mathbf{G}}$da analizzare.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotBack() : void`

Descrizione Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante back per tornare alla view precedente.

Note

- Il metodo deve essere marcato come virtuale.
- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.3 ControllerManager (class)

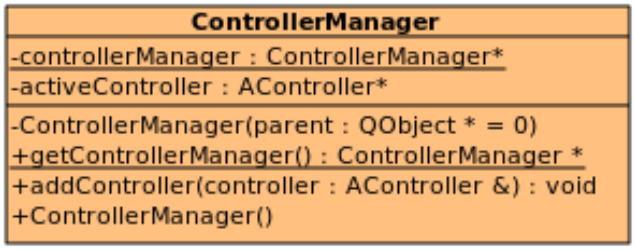

Figura 113: Diagramma classe *ControllerManager*

Descrizione: classe che si occupa di cancellare i controller dalla memoria, quando non sono più necessari. È implementata tramite il design patterng$_G$ Singleton.

Utilizzo: si occupa di gestire ed eliminare i controller.

Eredita da:

- Qt::QObject.

Attributi

- `- activeController : AController *`

  Descrizione Puntatore al controller attivo in un determinato istante.

- `- static controllerManager : ControllerManager *`

  Descrizione Puntatore al campo dati statico che rappresenta l'unica istanza della classe ControllerManager. Tale istanza verrà creata in modo *lazy*, ossia non sarà creata prima di quando viene richiesta per la prima volta.

Metodi

- `- ControllerManager(parent : QObject *)`

  Descrizione: Costruttore privato, come previsto dal design patterng$_G$ Singleton, della classe *ControllerManger*. Essendo privato, non sarà possibile creare direttamente oggetti di tipo *ControllerManager*. Per ottenere l'unica istanza di *ControllerManger* si dovrà utilizzare il metodo `getControllerManager()`.

  Argomenti

  - `parent : QObject *`
    Puntatore al *QObject* padre dell'oggetto *ControllerManger*.

- `+ getControllerManger() : ControllerManager *`

  Descrizione Metodo statico che, come previsto dal design patterng$_G$ Singleton, ritorna l'unica istanza della classe *ControllerManager* esistente. Se tale istanza non è ancora esitente, il metodo si occuperà di crearla.

- `+ addController(controller : const AController &) : void`

Descrizione: Metodo che permette di impostare l'oggetto di tipo *AController* attivo in un dato instante. Se al momento della chiamata è già presente un oggetto *AController*, esso viene eliminato dalla memoria. In questo modo non si ha memoria inutilizzata.

Argomenti

– `controller : const AController &`
Rappresenta l'oggetto *AController* da gestire.

6.1.4 DatasetsController (class)

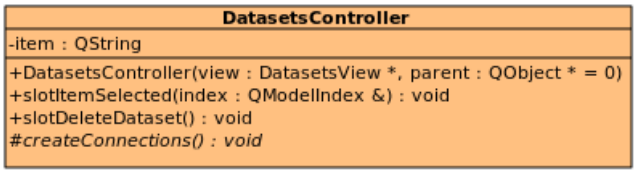

Figura 114: Diagramma classe *DatasetsController*

Descrizione: classe rappresentante un controller per un oggetto di tipo *DatasetsView*.

Utilizzo: gestisce i signal$_G$ emessi da un oggetto *DatasetsView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- item : QString`

  Descrizione Rappresenta il nome del Dataset$_G$ selezionato dall'utente nella vista associata al controller.

Metodi

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_G$ dell'oggetto *DatasetsController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ DatasetsController(view : DatasetsView *, parent : QObject *)`

  Descrizione: Costruttore protetto della classe *DatasetsController*. Si occupa di costruire un oggetto di tipo *DatasetsController*, associandolo alla propria vista.

  Argomenti

  - `view : DatasetsView *`
    Puntatore all'oggetto di tipo *DatasetView*, rappresentante la vista che verrà controllata dall'oggetto *DatasetsController*.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *DatasetsController*.

- `+ slotItemSelected(index : const QModelIndex &) : void`

Descrizione: Metodo pubblico che mostra i dettagli del Dataset$_{G}$ selezionato dall'utente, nella vista associata.

Argomenti

- `index : const QModelIndex &`
  Indice dell'item associato al Dataset$_{G}$ selezionato dall'utente.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotDeleteDataset() : void`

Descrizione Elimina il Dataset$_{G}$ selezionato dall'utente dal database.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `- featuresInfo(feats : QVector<AFeature*>, txtProtocols : QString &) : void`

Descrizione: aggiunge le informazioni riguardanti le Feature$_{G}$ alla stringa passata come parametro.

Argomenti

- `feats : QVector<AFeature*>`
  rappresenta puntatori alle Feature$_{G}$ presenti nei Protocol$_{G}$ del Datase$_{G}$ preso in considerazione.
- `txtProtocols : QString &`
  rappresenta la stringa al quale vengono aggiunte le informazioni relative alle Feature$_{G}$.

Note

- Il metodo deve essere marcato come costante.

- `- algorithmInfo(alg : AAlgorithm *, txtProtocols : QString &) : void`

Descrizione: aggiunge le informazioni riguardanti l'algoritmo di cluster$_{G}$ alla stringa passata come parametro.

Argomenti

- `alg : AAlgorithm *`
  puntatore all'algoritmo contenuto nel Dataset$_{G}$ selezionato.
- `txtProtocols : QString &`
  stringa dove verranno aggiunte le informazioni riguardanti l'algoritmo.

Note

- Il metodo deve essere marcato come costante.

6.1.5 DetailedResultController (class)

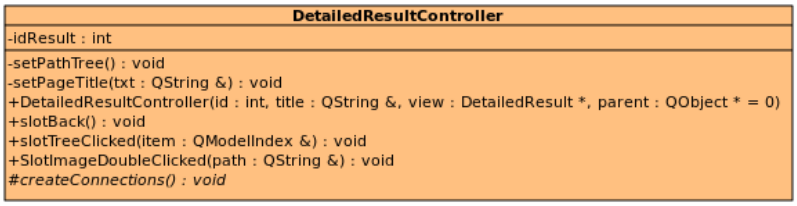

Figura 115: Diagramma classe DetailedResultController

Descrizione: La classe estende la classe astratta *AController* e ha il compito di catturare gli input dell'utente nella vista dei dettaglio risultati

Utilizzo: viene utilizzata per gestire i Signal$_G$ emessi da un oggetto DetailedResultView e reagisce in modo appropriato

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- idResult : int`

  Descrizione rappresenta l'id dell'analisi con cui è identificata all'interno del database.

Metodi

- `- setPathTree() : void`

  Descrizione metodo che ha il compito di impostare il path della vista ad albero.

- `- setPageTitle(txt : const QString &) : void`

  Descrizione: metodo che imposta il titolo della pagina nella parte alta della finestra.

  Argomenti

  - `txt : const QString &`
    rappresenta il testo da mettere come titolo.

  Note

  - Il metodo deve essere marcato come costante.

- `# createConnections() : void`

  Descrizione metodo che ha il compito di impostare le connessioni dell'oggetto.

Note

- Il metodo deve essere marcato come costante.
- Il metodo deve essere marcato come virtuale.

- `+ DetailedResultController(id : int, title : const QString &, view : DetailedResult*, parent : QObject *)`

Descrizione: Costruttore della classe, che ha il compito di creare l'oggetto e le connessioni alla view associata.

Argomenti

- `id : int`
  rappresenta l'id associato all'analisi di cui si stanno visualizzando i risultati.
- `title : const QString &`
  rappresenta il titolo da mettere alla pagina.
- `view : DetailedResult*`
  rappresenta la view a cui il controller in costruzione verrà associata.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in creazione.

- `+ slotBack() : void`

Descrizione $\text{Slot}_{\mathbf{G}}$ che riceve il segnale della view associata quando l'utente preme il pulsante per tornare indietro.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotTreeClicked(item : const QModelIndex &) : void`

Descrizione: $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quanto l'utente preme sull'albero.

Argomenti

- `item : const QModelIndex &`
  rappresenta l'elemento selezionato dall'albero.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotImageDoubleClicked(path : const QString &) : void`

Descrizione: $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente fa il doppio click sull'immagine/video.

Argomenti

- `path : const QString &`
  rappresenta il path del file selezionato.

Note

– Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.6 GroupsController (class)

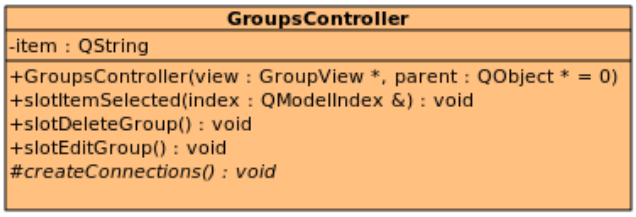

Figura 116: Diagramma classe *GroupsController*

Descrizione: classe che rappresenta un controller per un oggetto di tipo *GroupsView*.

Utilizzo: gestisce i signal$_{\mathbf{G}}$ emessi da un oggetto *GroupsView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- item : QString`

  Descrizione Rappresenta il nome del gruppo di Subject$_{\mathbf{G}}$ selezionato dall'utente nella vista associata al controller.

Metodi

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_{\mathbf{G}}$ dell'oggetto *GroupsController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ GroupsController(view : GroupView *, parent : QObject *)`

  Descrizione: Costruttore pubblico della classe *GroupsController*. Si occupa di costruire un oggetto di tipo *GroupsController*, associandolo alla propria vista.

  Argomenti

  - `view : GroupView *`
    Puntatore all'oggetto di tipo *GroupView*, rappresentante la vista che verrà controllata dall'oggetto *GroupsController*.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *GroupsController*.

- `+ slotItemSelected(index : const QModelIndex &) : void`

Descrizione: Metodo pubblico che mostra i dettagli del gruppo di Subject$_{\mathbf{G}}$ selezionato dall'utente, nella vista associata.

Argomenti

- `index : const QModelIndex &`
  Indice dell'item associato al gruppo di Subject$_{\mathbf{G}}$ selezionato dall'utente.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotDeleteGroup() : void`

Descrizione Elimina il gruppo di Subject$_{\mathbf{G}}$ selezionato dall'utente dal database.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotEditGroup() : void`

Descrizione Metodo che permette all'utente di poter modificare il gruppo di Subject$_{\mathbf{G}}$ selezionato. Il metodo reindirizza l'utente ad un altra vista in cui può effettuare le dovute modifiche.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.7 MainWindowController (class)

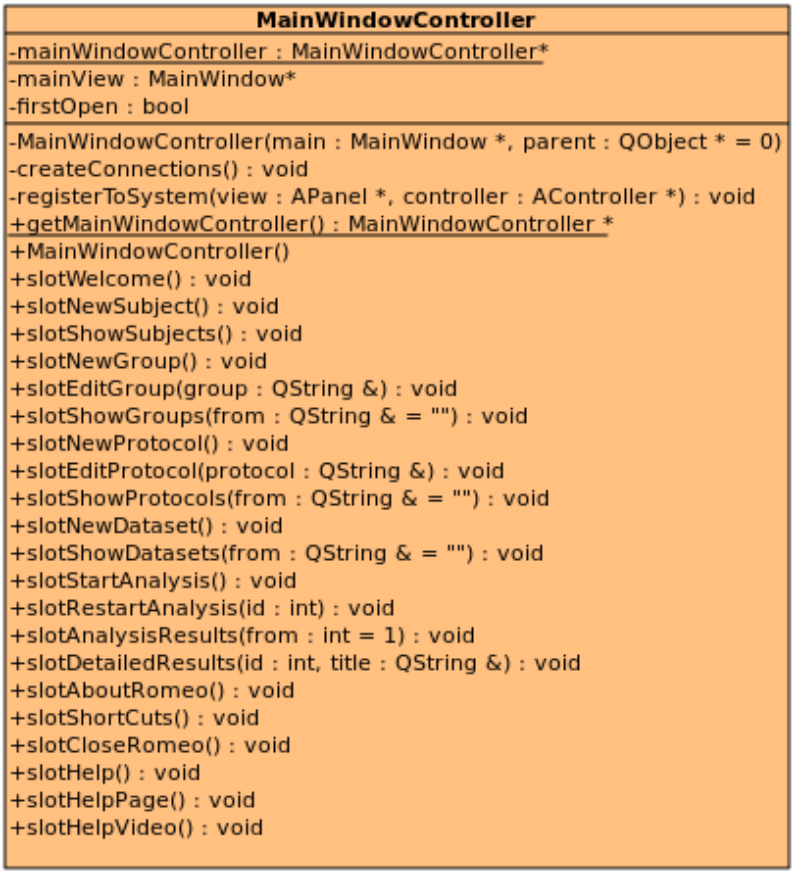

Figura 117: Diagramma classe MainViewController

Descrizione: classe che rappresenta un controller per un oggetto *MainWindowController*. Gestisce i vari input utente nella finestra principale del programma

Utilizzo: viene utilizzata per gestire i Signal_{G} emessi da un oggetto MainWindow. La classe MainWindowController recepisce i Signal_{G} emessi dalla view associata e reagisce in modo appropriato

Eredita da:

- Qt::QObject.

Attributi

- `- mainView : MainWindow*`

  Descrizione rappresenta la main Window gestita dal mainWindowController.

- `- firtOpen : bool`

  Descrizione indica se la mainWindow è visualizzata per la prima volta (true).

- `- static mainWindowController : MainWindowController*`

Descrizione rappresenta l'unica istanza di mainWindowController disponibile in quanto è stata implementando rispettando il design pattern$_G$ Singleton.

Note

- Il metodo deve essere marcato come static

Metodi

- `- MainWindowController(main : MainWindow*, parent : QObject *)`

  Descrizione: costruttore per l'oggetto della classe MainWindowController in cui gli verrà associata la view passata come parametro.

  Argomenti

  - `main : MainWindow*`
    rappresenta la view che verrà gestita dal controller in creazione.
  - `parent : QObject *`
    rappresenta il parent dell'oggetto in creazione.

- `- createConnections() : void`

  Descrizione metodo che ha il compito di creare tutte le connessioni per l'oggetto della classe in questione.

  Note

  - Il metodo deve essere marcato come costante.

- `- registerToSystem(view : APanel *, controller : AController *) : void`

  Descrizione: metodo che ha il compito di impostare i contenuti principali della MainWindow con la view passatao. Inoltre passa l'istanza del controller al metodo *addController* della classe *ControllerManager*.

  Argomenti

  - `view : APanel *`
    rappresenta la view da visualizzare come contenuto principale nella mainWindow.
  - `controller : AController *`
    rappresenta il controller associato alla view. passata come parametro

  Note

  - Il metodo deve essere marcato come costante.

- `+ getMainWindowController() : MainWindowController*`

  Descrizione metodo statico che ritorna l'unica istanza statica della classe MainWindowController stessa.

Note

- Il metodo deve essere marcato come static

- `+ slotWelcome() : void`

  Descrizione Slot$_{G}$ che permette all'utente di ritornare alla pagina iniziale (WelcomeView).

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotNewSubject() : void`

  Descrizione Slot$_{G}$ che permette all'utente di andare alla view per la creazione di un nuovo Subject$_{G}$ all'interno del sistema.

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotShowSubjects() : void`

  Descrizione Slot$_{G}$ che permette all'utente di raggiungere la pagina per la visualizzazione dei subject$_{G}$ attualmente presenti nel sistema.

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotNewGroup() : void`

  Descrizione Slot$_{G}$ che permette all'utente di raggiungere la pagina per la creazione di un nuovo gruppo di Subject$_{G}$ all'interno del sistema.

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotShowGroups(from : const QString & = "") : void`

  Descrizione: Slot$_{G}$ che permette all'utente di raggiungere la pagina per la visualizzazione dei gruppi d subject$_{G}$ attualmente presenti nel sistema.

  Argomenti

  - `from : const QString & = ""`
    rappresenta la vista dalla quale l'utente decide di voler visualizzare i gruppi di Subject$_{G}$ presenti nel sistema.

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotEditGroups(group : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente modificare un gruppo di Subject$_{\mathbf{G}}$ attualmente presenti nel sistema.

Argomenti

- `group : const QString &`
  rappresenta il nome del gruppo da modificare

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotNewProtocol() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di raggiungere la pagina per la creazione di un nuovo Protocol$_{\mathbf{G}}$ all'interno del sistema.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotEditProtocol(protocol : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente modificare un Protocol$_{\mathbf{G}}$ esistente.

Argomenti

- `protocol : const QString &`
  rappresenta il nome del Protocol$_{\mathbf{G}}$ che l'utente vuole modificare.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowProtocols(from : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente di raggiungere la pagina per la visualizzazione dei Protocol$_{\mathbf{G}}$ attualmente presenti nel sistema.

Argomenti

- `from : const QString &`
  rappresenta la view dalla quale l'utente vuole visualizzare i Protocol$_{\mathbf{G}}$ attualmente presenti nel sistema.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotNewDataset() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di raggiungere la pagina per la creazione di un nuovo Dataset$_{\mathbf{G}}$ nel sistema.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShowDatasets(from : const QString & = "") : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente di raggiungere la pagina per la visualizzazione dei Dataset$_{\mathbf{G}}$ attualmente presenti nel sistema.

Argomenti

- `from : const QString & = ""`
  rappresenta la view dalla quale l'utente vuole visualizzare i Dataset$_{\mathbf{G}}$ presenti nel sistema.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotStartAnalysis() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di iniziare una nuova analisi.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotRestartAnalysis(id : int) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente di far ripartire un'analisi precedentemente avviata.

Argomenti

- `id : int`
  rappresenta l'id dell'analisi che l'utente vuole far ripartire.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotAnalysisResults(from : int = 1) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare i dettagli delle precedenti analisi eseguite.

Argomenti

- `from : int = 1`
  rappresenta l'id dell'analisi dal quale partire.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotDetailedResults(id : int, title : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare i dettagli di una specifica analisi.

Argomenti

- `id : int`
  rappresenta l'id dell'analisi della quale l'utente vuole visualizzarne i dettagli.
- `title : const QString &`
  rappresenta il titolo della pagina del dettaglio dei risultati da mettere in alto alla view.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotAboutRomeo() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare le informazioni relative al prodotto Romeo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotShortCuts() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare gli shortcut utilizzabili all'interno del prodotto Romeo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotCloseRomeo() : void`

Descrizione Slot$_{\mathbf{G}}$ eseguito quando l'utente decide di uscire dall'applicazione Romeo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotHelp() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare il manuale d'uso di Romeo.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotHelpPage() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare una determinata pagina del manuale d'uso per Romeo.

Note

– Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotHelpVideo() : void`

Descrizione Slot$_{\mathbf{G}}$ che permette all'utente di visualizzare la guida video all'utilizzo di Romeo.

Note

– Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.8 NewDatasetController (class)

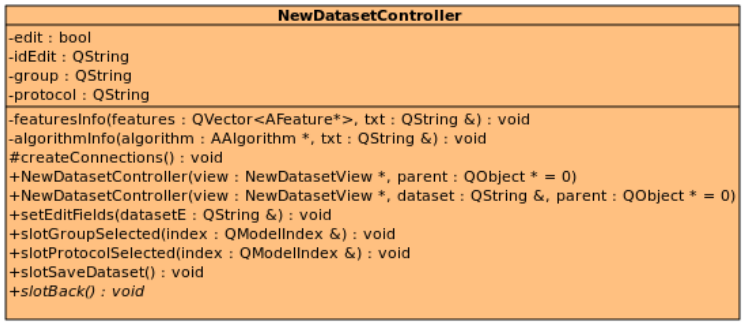

Figura 118: Diagramma classe NewDatasetController

Descrizione: classe che rappresenta un controller per un oggetto NewDatasetView e ha il compito di creare un nuovo Protocol$_{\mathbf{G}}$.

Utilizzo: viene utilizzata per gestire i Signal$_{\mathbf{G}}$ emessi dall'oggetto NewDatasetView associato e reagisce in modo appropriato una volta recepito.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- edit : bool`

  Descrizione rappresenta un flag che vale false se è un nuovo Dataset$_{\mathbf{G}}$ altrimenti vale true.

- `- idEdit : QString`

  Descrizione rappresenta l'id del Dataset$_{\mathbf{G}}$ da modificare.

- `- group : QString`

  Descrizione rappresenta il gruppo contenuto all'interno del Dataset$_{\mathbf{G}}$.

- `- protocol : QString`

  Descrizione rappresenta il protocol selezionato contenuto all'interno del Dataset$_{\mathbf{G}}$.

Metodi

- `- featureInfo(features : QVector<AFeature*>, txt : QString &) : void`

  Descrizione: metodo che imposta le informazioni legate alle Feature$_{\mathbf{G}}$ contenute nel Dataset$_{\mathbf{G}}$.

Argomenti

- `features : QVector<AFeature*>`
  rappresenta l'insieme delle Feature$_{\mathbf{G}}$ contenute nel Dataset$_{\mathbf{G}}$.
- `txt : QString &`
  rappresenta la stringa alla quale vengono aggiunte le informazioni relative alle Feature$_{\mathbf{G}}$ contenute nel Dataset$_{\mathbf{G}}$

Note

- Il metodo deve essere marcato come costante.

- `- algorithmInfo(algorithm : AAlgorihm *, txt : QString &) : void`

Descrizione: metodo che imposta le informazioni legate all'algoritmo di cluster$_{\mathbf{G}}$ del Protocol$_{\mathbf{G}}$ contenuto dentro al Dataset$_{\mathbf{G}}$.

Argomenti

- `algorithm : AAlgorihm *`
  rappresenta l'algoritmo contenuto nel Protocol$_{\mathbf{G}}$.
- `txt : QString &`
  rappresenta la stringa nella quale verranno inserite le informazioni dell'algoritmo di cluster$_{\mathbf{G}}$.

Note

- Il metodo deve essere marcato come costante.

- `# createConnections() : void`

Descrizione metodo che crea le connessioni tra la vista associata all'oggetto controller in questione.

Note

- Il metodo deve essere marcato come costante.

- `+ NewDatasetController(view : MainWindowController*, dataset : const QString &, parent : Q`

Descrizione: costruttore per la classe NewDatasetController che riceve come parametro la view associata da gestire.

Argomenti

- `view : MainWindowController*`
  rappresenta la view che verrà associata all'oggetto in creazione.
- `dataset : const QString &`
  rappresenta il nome del Dataset$_{\mathbf{G}}$ da modificare.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in creazione.

- `+ NewDatasetController(view : NewDatasetView*, parent : QObject *)`

Descrizione: costruttore per la classe NewDatasetController al quale viene passata la view che gestirà, e il Dataset$_{G}$ che l'utente vuole modificare.

Argomenti

- `view : NewDatasetView*`
  rappresenta la view che verrà associata al controller in creazione.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in creazione.

- `+ setEditFields(datasetE : const QString &) : void`

Descrizione: metodo che imposta i campi dati per poter essere modificati dall'utente.

Argomenti

- `datasetE : const QString &`
  rappresenta il nome del Dataset$_{G}$ da modificare.

- `- slotGroupSelected(index : const QModelIndex &, index : const QModelIndex &) : void`

Descrizione: Slot$_{G}$ che riceve il segnale della selezione, da parte dell'utente, del gruppo di Subject$_{G}$.

Argomenti

- `index : const QModelIndex &`
  rappresenta l'indice della tabella cui si trova il gruppo di Subject$_{G}$ selezionato.
- `index : const QModelIndex &`
  rappresenta l'indice della tabella cui si trova il Protocol$_{G}$ selezionato.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotProtocolSelected() : void`

Descrizione Slot$_{G}$ che riceve il segnale della selezione, da parte dell'utente, del Protocol$_{G}$.

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotSaveDataset() : void`

Descrizione Slot$_{G}$ che riceve il segnale della selezione, da parte dell'utente, del pulstante per il salvataggio delle modifiche (save).

Note

- Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotBack() : void`

Descrizione Slot$_{\mathbf{G}}$ virtuale che implementa il contratto di ritorno alla view precedente e reagisce al segnale della pressione del pulsante back dalla view. In particolare ritorna alla finestra principale se le modifiche fatte riguardano la creazione di un nuovo Dataset$_{\mathbf{G}}$ altrimenti ritorna alla view che mostra tutti i Dataset$_{\mathbf{G}}$ presenti all'interno del sistema.

Note

- Il metodo deve essere marcato come virtuale.
- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.9 NewGroupController (class)

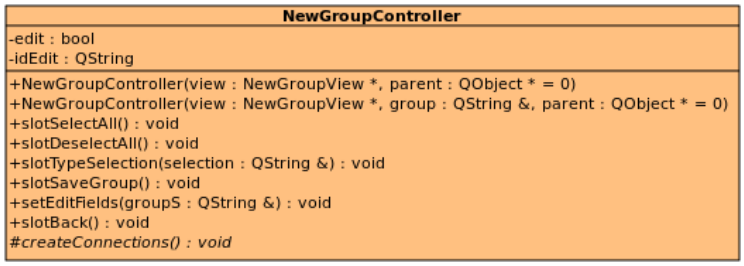

Figura 119: Diagramma classe *NewGroupController*

Descrizione: classe rappresentante un controller per un oggetto di tipo *NewGroupView*.

Utilizzo: gestisce i signal$_{\mathbf{G}}$ emessi da un oggetto *NewGroupView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `# edit : bool`

  Descrizione Rappresenta il tipo di operazione svolta dal controller. Nel caso in cui valga `true`, il controller si occuperà di gestire la modifica di un gruppo di Subject$_{\mathbf{G}}$. In caso contrario, gestirà la creazione di un nuovo gruppo.

- `- idEdit : QString`

  Descrizione Stringa rappresentante il nome del gruppo di Subject$_{\mathbf{G}}$ che si vuole eventualmente modificare.

Metodi

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_{\mathbf{G}}$ dell'oggetto *NewGroupController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ NewGroupController(view : NewGroupView *, parent : QObject *)`

  Descrizione: Costruttore della classe *NewGroupController*. Si occupa di costruire un oggetto di tipo *NewGroupController*, associandolo alla propria vista. Tale costruttore viene utilizzato quando è necessario gestire una vista di creazione di un nuovo gruppo di Subject$_{\mathbf{G}}$.

Argomenti

- `view : NewGroupView *`
  Puntatore all'oggetto di tipo *NewGroupView*, rappresentante la vista che verrà controllata dall'oggetto *NewGroupController*.
- `parent : QObject *`
  Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *NewGroupController*.

- `+ NewGroupController(view : NewGroupView *, group : const QString &, parent : QObject *)`

Descrizione: Costruttore della classe *NewGroupController*. Si occupa di costruire un oggetto di tipo *NewGroupController*, associandolo alla propria vista. Tale costruttore viene utilizzato quando è necessario gestire una modifica ad un gruppo di $\text{Subject}_{\mathbf{G}}$ preesistente.

Argomenti

- `view : NewGroupView *`
  Puntatore all'oggetto di tipo *NewGroupView*, rappresentante la vista che verrà controllata dall'oggetto *NewGroupController*.
- `group : const QString &`
  Stringa rappresentante il nome del gruppo di $\text{Subject}_{\mathbf{G}}$ di cui il controller deve gestire le modifiche.
- `parent : QObject *`
  Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *NewGroupController*.

- `+ slotSelectAll() : void`

Descrizione Metodo che permette di selezionare tutti i $\text{Subject}_{\mathbf{G}}$ presenti nel sistema, per aggiungerli al gruppo di $\text{Subject}_{\mathbf{G}}$ che l'utente sta creando.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotDeselectAll() : void`

Descrizione Metodo che permette di deselezionare tutti i $\text{Subject}_{\mathbf{G}}$ presenti nella view.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotTypeSelection(selection : const QString &) : void`

Descrizione: Metodo che gestisce il cambiamento del tipo di gruppo che l'utente vuole creare (2D, 2D-t, 3D, 3D-t). Al cambiamento del tipo di gruppo da parte dell'utente, il controller si occupa di aggiornare i $\text{Subject}_{\mathbf{G}}$ presenti nel sistema, con quelli della tipologia selezionata.

Argomenti

- `selection : const QString &`
  Stringa rappresentante il nuovo tipo di gruppo di Subject$_{\mathbf{G}}$ che l'utente vuole creare.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotSaveGroup() : void`

Descrizione Metodo che gestise il salvataggio del gruppo di Subject$_{\mathbf{G}}$ che l'utente sta creando. Nel caso in cui i dati inseriti dall'utente non siano corretti, o siano incompleti, il controller crea una vista per segnalare all'utente il problema.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ setEditFields(groupS : const QString &) : void`

Descrizione: Metodo che permette di visualizzare i dati di un determinato gruppo di Subject$_{\mathbf{G}}$, nella vista associata.

Argomenti

- `groupS : const QString &`
  Stringa rappresentante il nome del gruppo di Subject$_{\mathbf{G}}$ di cui si vogliono visualizzare i dati nella rispettiva vista.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotBack() : void`

Descrizione Metodo che permette di ritornare alla vista precedentemente visualizzata. Tale vista cambierà in base alla tipologia di operazione che l'utente sta compiendo (modifica o creazione di un gruppo).

Note

- Il metodo deve essere marcato come virtuale.
- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `- create_Update(group : GroupOfSubject*, name : QString &) : void`

Descrizione: metodo che ha il compito di salvare o aggiornare un gruppo di Subject$_{\mathbf{G}}$ all'interno del database,

Argomenti

- `group : GroupOfSubject*`
  puntatore al gruppo da salvare o aggiornare.
- `name : QString &`
  riferimento al nome del gruppo passato come parametro.

6.1.10 NewProtocolController (class)

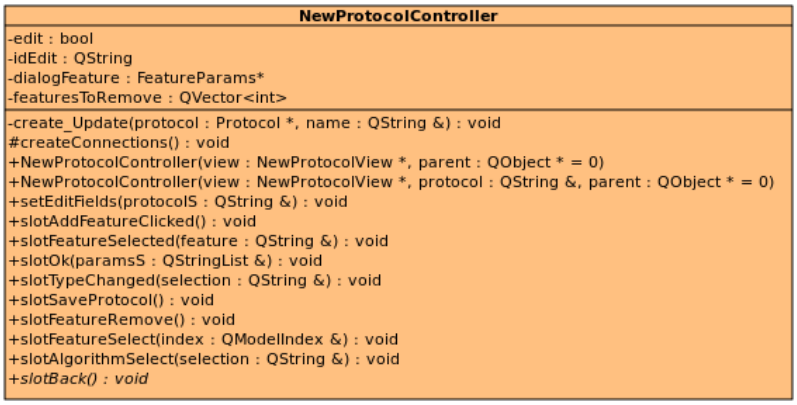

Figura 120: Diagramma classe NewProtocolController

Descrizione: classe che rappresenta un controller per un oggetto NewProtocolView e ha il compito di gestire un nuovo Protocol$_G$

Utilizzo: viene utilizzata per gestire i Signal$_G$ emessi da un oggetto NewProtocolview e reagisce in modo appropriato ad essi una volta catturati

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- edit : bool`

  Descrizione rappresenta un flag che vale false se si sta creando un nuovo Protocol$_G$, true altrimenti.

- `- idEdit : QString`

  Descrizione rappresenta l'id del Protocol$_G$ che l'utente sta modificando.

- `- dialogFeature : FeatureParams*`

  Descrizione rappresenta la finestra di dialogo adibita all'aggiunta di una Feature$_G$.

- `- featuresToRemove : QVector<int>`

  Descrizione rappresenta l'insieme di id delle Feature$_G$ che l'utente vuole rimuovere dal Protocol$_G$.

Metodi

- `- create_Update(protocol : Protocol*, parent : QObject *) : void`

  Descrizione: metodo che ha il compito di salvare o aggiornare il Protocol$_G$ in questione.

Argomenti

- `protocol : Protocol*`
  rappresenta il Protocol$_G$ da salvare o modificare a seconda che il Protocol$_G$ sia in creazione o precedentemente creato e ora in modifica.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in creazione.

- `# createConnections() : void`

Descrizione metodo che ha il compito di creare tutte le connessioni tra l'oggetto controller e la view associata ad esso da gestire.

Note

- Il metodo deve essere marcato come costante.

- `+ NewProtocolController(view : NewProtocolView*, parent : QObject *)`

Descrizione: costruttore per un oggetto NewProtocolController a cui viene passata la view gestita dall'oggetto che si sta creando.

Argomenti

- `view : NewProtocolView*`
  rappresenta la view che verrà associata all'oggetto controller in creazione.
- `parent : QObject *`
  rappresenta il parent dell'oggetto in creazione.

- `+ NewProtocolController(view : NewProtocolView*, protocol : const QString &, parent : QObj`

Descrizione: costruttore per l'oggetto controller che gestisce la view NewProtocolView quando l'utente vuole modificare il Protocol$_G$ precedentemente selezionato.

Argomenti

- `view : NewProtocolView*`
  rappresenta la view che verrà associata all'oggetto controller in creazione.
- `protocol : const QString &`
  stringa che rappresenta il nome del Protocol$_G$ che l'utente ha scelto di modificare
- `parent : QObject *`
  rappresenta il parent del controller in creazione.

- `+ setEditFields(protocolS : const QString &) : void`

Descrizione: metodo che ha il compito di impostare i campi in modo tale che l'utente possa modificarne i valori.

Argomenti

- `protocolS : const QString &`
  la stringa rappresenta il nome del Protocol$_G$ i cui campi devono essere preparati per la possibile modifica da parte dell'utente.

- `+ slotAddFeatureClicked() : void`

Descrizione Slot$_G$ che ha il compito di aggiungere la Feature$_G$ selezionata dalla finestra di dialogo contenente le Feature$_G$ disponibili.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `+ slotFeatureSelected(feature : const QString &) : void`

Descrizione: Slot$_G$ che riceve il segnale quando l'utente seleziona una Feature$_G$ dalla finestra di dialogo per l'inserimento di una nuova Feature$_G$

Argomenti

- `feature : const QString &`
  rappresenta il nome della Feature$_G$ selezionata.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `+ slotOk(paramsS : const QStringList&) : void`

Descrizione: Slot$_G$ che ha il compito di aggiungere al Protocol$_G$ la Feature$_G$ aggiunta dall'utente.

Argomenti

- `paramsS : const QStringList&`
  rappresenta la lista dei valori dei parametri della Feature$_G$ da aggiungere al Protocol$_G$ in creazione.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `+ slotTypeChanged(selection : const QString &) : void`

Descrizione: Slot$_G$ che ha il compito di modificare il tipo associato al Protocol$_G$ che l'utente vuole andare a creare. Lo Slot$_G$ aggiorna la view associata che ha lanciato il Signal$_G$ con le informazioni appropriate.

Argomenti

- `selection : const QString &`
  rappresenta il tipo, che l'utente ha selezionato, associato al Protocol$_G$ in creazione.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotSaveProtocol() : void`

Descrizione Slot$_{\mathbf{G}}$ che ha il compito di salvare il Protocol$_{\mathbf{G}}$ nell'applicativo Romeo una volta che l'utente preme sul pulsante save della view associata al controller che riceve il Signal$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotFeatureRemove() : void`

Descrizione Slot$_{\mathbf{G}}$ che ha il compito di rimuovere dal Protocol$_{\mathbf{G}}$ le Feature$_{\mathbf{G}}$ selezionate dall'utente nella view associata al controller che riceve il Signal$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotFeatureSelect(index : const QModelIndex &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona una Feature$_{\mathbf{G}}$ dalla tabella contenente tutte le Feature$_{\mathbf{G}}$ presenti nell'applicativo Romeo.

Argomenti

- `index : const QModelIndex &`
  rappresenta l'indice della tabella che è stato selezionato.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotAlgorithm(selection : const QString &) : void`

Descrizione: Slot$_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona l'algoritmo di cluster$_{\mathbf{G}}$.

Argomenti

- `selection : const QString &`
  rappresenta il nome dell'algortimo di cluster$_{\mathbf{G}}$ che l'utente ha selezionato.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotBack() : void`

Descrizione Slot$_{\mathbf{G}}$ virtuale che ha il compito di ritornare alla precedente view visualizzata dall'utente.

Note

- Il metodo deve essere marcato come virtuale.
- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

6.1.11 NewSubjectController (class)

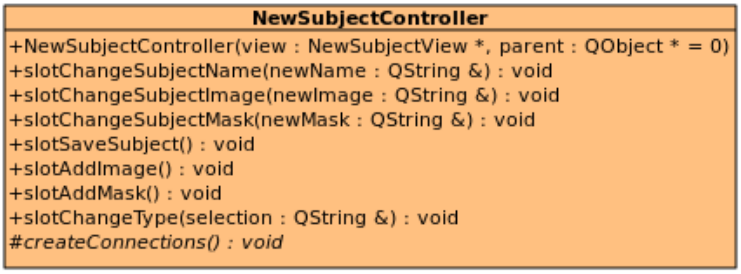

Figura 121: Diagramma classe *NewSubjectController*

Descrizione: classe rappresentante un controller per un oggetto di tipo *NewSubjectView*.

Utilizzo: gestisce i signal$_G$ emessi da un oggetto *NewSubjectView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Metodi

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_G$ dell'oggetto *NewSubjectController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ NewSubjectController(view : NewSubjectView *, parent : QObject *)`

  Descrizione: Costruttore pubblico della classe *NewSubjectController*. Si occupa di costruire un oggetto di tipo *NewSubjectController*, associandolo alla propria vista.

  Argomenti

  - `view : NewSubjectView *`
    Puntatore all'oggetto di tipo *NewSubjectView*, rappresentante la vista che verrà controllata dall'oggetto *NewSubjectController*.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *NewSubjectController*.

- `+ slotChangeSubjectName(newName : const QString &) : void`

Descrizione: Metodo che gestisce il cambiamento di nome del Subject$_{\mathbf{G}}$ che l'utente sta creando.

Argomenti

- `newName : const QString &`
  Stringa rappresentante il nuovo nome che l'utente vuole assegnare al Subject$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotChangeSubjectImage(newImage : const QString &) : void`

Descrizione: Metodo che gestisce il cambiamento dell'immagine del Subject$_{\mathbf{G}}$ che l'utente sta creando.

Argomenti

- `newImage : const QString &`
  Stringa rappresentante la nuova immagine che l'utente vuole assegnare al Subject$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotChangeSubjectMask(newMask : const QString &) : void`

Descrizione: Metodo che gestisce il cambiamento della maschera del Subject$_{\mathbf{G}}$ che l'utente sta creando.

Argomenti

- `newMask : const QString &`
  Stringa rappresentante la nuova maschera che l'utente vuole assegnare al Subject$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotSaveSubject() : void`

Descrizione Metodo che si occupa di salvare nel database il Subject$_{\mathbf{G}}$ che l'utente sta creando.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotAddImage() : void`

Descrizione Metodo che crea la finestra di dialogo che permette all'utente di selezionare dal filesystem, il file contenente l'immagine da associare al Subject$_{\mathbf{G}}$ che l'utente sta creando.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotAddMask() : void`

Descrizione Metodo che crea la finestra di dialogo che permette all'utente di selezionare dal filesystem, il file contenente la maschera da associare al Subject$_{\mathbf{G}}$ che l'utente sta creando.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `+ slotChangeType(selection : const QString &) : void`

Descrizione: Metodo che gestisce il cambiamento del tipo di Subject$_{\mathbf{G}}$ che l'utente sta creando.

Argomenti

- `selection : const QString &`
  Stringa rappresentante il nuovo tipo che l'utente vuole assegnare al Subject$_{\mathbf{G}}$.

Note

- Il metodo è uno slot$_{\mathbf{G}}$ Qt$_{\mathbf{G}}$.

- `- creationDone(name : QString &, imagePath : QFileInfo&, image : QString &, mask : QString`

Descrizione: metodo che ha il compito di copiare il file caricato alla creazione del Subject$_{\mathbf{G}}$ da parte dell'utente e visualizza il messaggio di salvataggio.

Argomenti

- `name : QString &`
  rappresenta il nome del Subject$_{\mathbf{G}}$ creato.
- `imagePath : QFileInfo&`
  rappresenta il path a cui reperire l'immagine selezionata dall'utente$_{\mathbf{G}}$.
- `image : QString &`
  rappresenta il nome dell'immagine selezionata dall'utente.
- `mask : QString &`
  rappresenta il nome della maschera selezionata dall'utente.
- `maskPath : QFileInfo&`
  rappresenta il path a cui reperire la maschera selezionata dall'utente.

6.1.12 ProtocolsController (class)

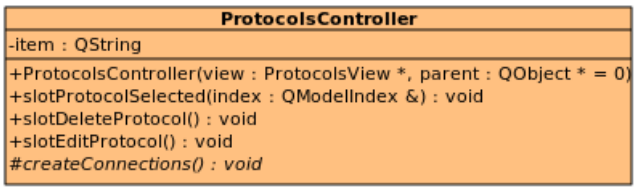

Figura 122: Diagramma classe *ProtocolsController*

Descrizione: classe rappresentante un controller per un oggetto di tipo *ProtocolsView*.

Utilizzo: gestisce i signal$_G$ emessi da un oggetto *ProtocolsView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- item : QString`

  Descrizione Rappresenta il nome del Protocol$_G$ selezionato dall'utente nella vista associata al controller.

Metodi

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_G$ dell'oggetto *ProtocolsController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ ProtocolsController(view : ProtocolsView *, parent : QObject *)`

  Descrizione: Costruttore pubblico della classe *ProtocolsController*. Si occupa di costruire un oggetto di tipo *ProtocolsController*, associandolo alla propria vista.

  Argomenti

  - `view : ProtocolsView *`
    Puntatore all'oggetto di tipo *ProtocolsView*, rappresentante la vista che verrà controllata dall'oggetto *ProtocolsController*.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *ProtocolsController*.

- `+ slotProtocolSelected(index : const QModelIndex &) : void`

Descrizione: Metodo che, successivamente alla richiesta dell'utente, imposta la vista con i dati del Protocol$_G$ selezionato.

Argomenti

- `index : const QModelIndex &`
  Indice dell'item associato al Protocol$_G$ selezionato dall'utente.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `+ slotDeleteProtocol() : void`

Descrizione Metodo che si occupa di gestire l'eliminazione del Protocol$_G$ selezionato dall'utente.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `+ slotEditProtocol() : void`

Descrizione Metodo che permette all'utente di poter modificare il Protocol$_G$ selezionato.

Note

- Il metodo è uno slot$_G$ Qt$_G$.

- `- featuresInfo(feats : QVector<AFeature*>) : void`

Descrizione: metodo che ha il compito di caricare le informazioni relative alle Feature$_G$ contenute nel Protocol$_G$ selezionato.

Argomenti

- `feats : QVector<AFeature*>`
  rappresenta l'insieme delle Feature$_G$ contenute nel Protocol$_G$ associato al controller in esame.

Note

- Il metodo deve essere marcato come costante.

- `- algorithmInfo(algorithm : AAlgorihm *) : void`

Descrizione: metodo che carica le informazioni relative all'algoritmo di cluster$_G$ contenuto dentro al Protocol$_G$ selezionato.

Argomenti

- `algorithm : AAlgorihm *`
  rappresenta l'algoritmo contenuto dentro al Protocol$_G$ selezionato.

Note

– Il metodo deve essere marcato come costante.

6.1.13 ResultsController (class)

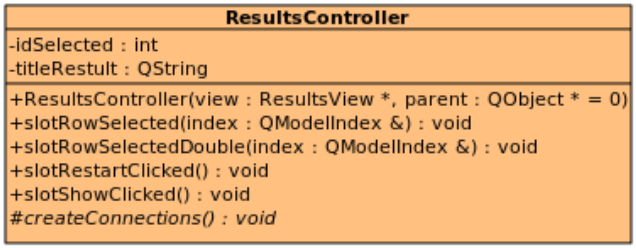

Figura 123: Diagramma classe ResultsController

Descrizione: classe che rappresenta un controller per un oggetto ResultsView.

Utilizzo: viene utilizzata per gestire i Signal_{G} emessi da un oggetto ResultsView e reagisce in modo appropriato a quelli recepiti.

Eredita da:

- Romeo::Controller::AController.

Attributi

- `- idSelected : int`

  Descrizione rappresenta l'id dell'analisi selezionata all'interno del database.

- `- titleResult : QString`

  Descrizione rappresenta il titolo da visualizzare nella parte alta della view.

Metodi

- `# createConnections() : void`

  Descrizione metodo virtuale ridefinito che ha il compito di creare tutte le connessioni.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

- `+ ResultsController(view : ResultsView*, parent : QObject *)`

  Descrizione: Costruttore per la classe che associa l'oggetto in creazione con la view passata come parametro.

Argomenti

- `view : ResultsView*`
  rappresenta la view a cui il controller in creazione verrà associata.
- `parent : QObject *`
  rappresenta il parent del oggetto in creazione.

- `- slotRowSelected(index : const QModelIndex &) : void`

Descrizione: $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona la riga relativa a un'analisi.

Argomenti

- `index : const QModelIndex &`
  rappresenta l'indice di riga della tabella selezionata.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotRowSelectedDouble(index : const QModelIndex &) : void`

Descrizione: $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona con il doppio click la riga relativa a un'analisi.

Argomenti

- `index : const QModelIndex &`
  rappresenta l'indice di riga della tabella selezionata.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotRestarClicked() : void`

Descrizione $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente preme il pulsante di restart.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

- `+ slotShowClicked() : void`

Descrizione $\text{Slot}_{\mathbf{G}}$ che riceve il segnale dalla view associata quando l'utente seleziona il pulsante per visualizzare i risultati dell'analisi.

Note

- Il metodo è uno $\text{slot}_{\mathbf{G}}$ $\text{Qt}_{\mathbf{G}}$.

6.1.14 SubjectsController (class)

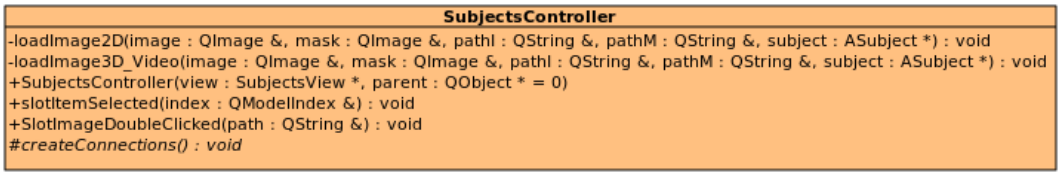

Figura 124: Diagramma classe SubjectsController

Descrizione: classe che rappresenta il controller per un oggetto SubjectView.

Utilizzo: Viene utilizzata per gesitre i Signal$_{G}$ emessi da un oggetto SubjectsView e reagisce in modo appropriato ogni volta che ne cattura uno.

Eredita da:

- Romeo::Controller::AController.

Metodi

- `- loadImage2D(image : QImage&, mask : QImage&, pathI : QString &, pathM : QString &, subje`

  Descrizione: metodo che carica un'immagine 2D associata al Subject$_{G}$

  Argomenti

  - `image : QImage&`
    rappresenta l'immagine da caricare
  - `mask : QImage&`
    rappresenta la maschera da caricare
  - `pathI : QString &`
    rappresenta il path relativo all'immagine all'interno del filesystem.
  - `pathM : QString &`
    rappresenta il path relativo alla maschera all'interno del filesystem.
  - `subject : ASubject*`
    rappresenta il puntatore al Subject$_{G}$ a cui sono associati immagine e maschera.

- `- loadImage3D_Video(image : QImage&, mask : QImage&, pathI : QString &, pathM : QString &, subject : ASubject*) : void`

  Descrizione: metodo che carica un'immagine 3D o un video 2D/3D associata al Subject.

  Argomenti

  - `image : QImage&`
    rappresenta l'immagine 3D o il video 2D/3D da caricare.
  - `mask : QImage&`
    rappresenta la maschera da caricare.

  - `pathI : QString &`
    rappresenta il path relativo al file da caricare all'interno del filesystem
  - `pathM : QString &`
    rappresenta il path relativo alla maschera all'interno del filesystem.
  - `subject : ASubject*`
    rappresenta il Subject$_{G}$ a cui associare i file passati come parametri ai metodi.

- `+ SubjectsController(view : SubjectView*, parent : QObject *)`

  Descrizione: costruttore che ha il compito di costruire il controller associato alla view passata come parametro.

  Argomenti

  - `view : SubjectView*`
    rappresenta il puntatore alla view cui il controller in costruzione verrà associato.
  - `parent : QObject *`
    rappresenta il parent di SubjectsController.

- `# createConnections() : void`

  Descrizione metodo che ha il compito di creare tutte le connessioni per il controller.

  Note

  - Il metodo deve essere marcato come virtuale.
  - Il metodo deve essere marcato const.

- `+ slotItemSelected(item : const QModelIndex &) : void`

  Descrizione: slot$_{G}$ che mostra all'utente i dettagli del Subject$_{G}$ selezionato nella view a cui il controller è associato.

  Argomenti

  - `item : const QModelIndex &`
    rappresenta l'indice dell'elemento associato al Subject$_{G}$ selezionato

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

- `+ slotImageClicked(path : QString) : void`

  Descrizione: Slot$_{G}$ che riceve il Signal$_{G}$ di doppio click dell'utente sull'anteprima dell'immagine o del video.

  Argomenti

  - `path : QString`
    rappresenta il path del file selezionato

  Note

  - Il metodo è uno slot$_{G}$ Qt$_{G}$.

6.1.15 WelcomeController (class)

Figura 125: Diagramma classe *WelcomeController*

Descrizione: classe che rappresenta un controller per un oggetto di tipo *WelcomeView*.

Utilizzo: gestisce i signal$_G$ emessi da un oggetto *WelcomeView* ed agisce conseguentemente in modo appropriato.

Eredita da:

- Romeo::Controller::AController.

Metodi

- `+ WelcomeController(view : WelcomeView *, parent : QObject *)`

  Descrizione: Costruttore protetto della classe *WelcomeController*. Si occupa di costruire un oggetto di tipo *WelcomeController*, associandolo alla propria vista.

  Argomenti

  - `view : WelcomeView *`
    Puntatore all'oggetto di tipo *WelcomeView*, rappresentante la vista che verrà controllata dall'oggetto *WelcomeController*.
  - `parent : QObject *`
    Puntatore all'oggetto *QObject*, rappresentante il padre dell'oggetto *WelcomeController*.

- `# createConnections() : void`

  Descrizione Metodo che si occupa di creare le connessioni tra i segnali emessi dalla vista associata e gli slot$_G$ dell'oggetto *WelcomeController*. Per l'implementazione del metodo, la classe utilizza gli slot forniti dalla classe *MainWindowController*.

  Note

  - Il metodo deve essere marcato come costante.
  - Il metodo deve essere marcato come virtuale.

7 Diagrammi di sequenza

Di seguito sono riportati i diagrammi di sequenza delle operazioni principali di Romeo.

7.1 Creazione di un nuovo Subject

Il diagramma seguente, descrive la sequenza di operazioni che vengono effettuate quando si procede all'inserimento di un nuovo Subject$_{\mathbf{G}}$. Esso pone enfasi sulle operazioni di creazione delle varie viste necessarie, tralasciando, per agevolare la lettura, le interazioni con la base di dati. La sequenza delle operazioni viene scatenata dall'utente quando seleziona, dalla finestra principale di Romeo (*WelcomePage*), il comando *New Subject*, tramite l'apposito pulsante. Vengono successivamente effettuate le seguenti operazioni:

1. Viene emesso il signal$_{\mathbf{G}}$ *CreateNewSubject* del *WelcomeController*;
2. Il *WelcomeController* richiama lo slot$_{\mathbf{G}}$ *SlotNewSubject* del *MainWindowController*;
3. Il *MainWindowController* crea una nuova istanza della vista *NewSubjectView* che permette all'utente di interagire col sistema per inserire il Subject$_{\mathbf{G}}$. In seguito *MainWindowController* crea un'istanza di *NewSubjectController*;
4. *MainWindowController* invoca il proprio metodo *RegisterToSystem*, il quale sostituisce la vista corrente (*WelcomePage*) con la nuova vista *NewSubjectView*, cancellandola. In seguito registra anche il nuovo controller *NewSubjectController* invocando il metodo *addController* di *ControlManager*, cancellando anche il controller relativo alla finestra principale;

L'utente a questo punto visualizza la vista che permette l'inserimento delle informazioni del Subject$_{\mathbf{G}}$, quali: *Nome*, *Tipo*, *Immagine/Video* e *Maschera*. L'aggiunta dei file relativi all'immagine e alla maschera prevede le seguenti informazioni:

1. Emissione del signal$_{\mathbf{G}}$ *addImage/addMask*, raccolto dal *NewSubjectController* che procede lanciando lo slot$_{\mathbf{G}}$ *slotAddImage/slotAddMask*; quest'ultimo crea una finestra di dialogo che permette all'utente di selezionare il file relativo all'immagine;
2. Dopo aver caricato il file, il controller invocherà il metodo *setImagePath/setMaskPath*, indicando il percorso del file selezionato.

Una volta inserite le informazioni sul Subject$_{\mathbf{G}}$, l'utente potrà salvare il subject$_{\mathbf{G}}$ premendo il pulsante *Save*. La pressione del pulsate provoca le seguenti operazioni:

1. Emmissione del signal$_{\mathbf{G}}$ *saveObject*, raccolto dal *NewSubjectController* che procede lanciando lo slot$_{\mathbf{G}}$ *slotSaveSubject*, il quale salva il Subject$_{\mathbf{G}}$ appena creato.

Al fine di agevolare la lettura del diagramma sono state omesse le operazioni di aggiornamento della *StatusBar* di *MainWindow*.

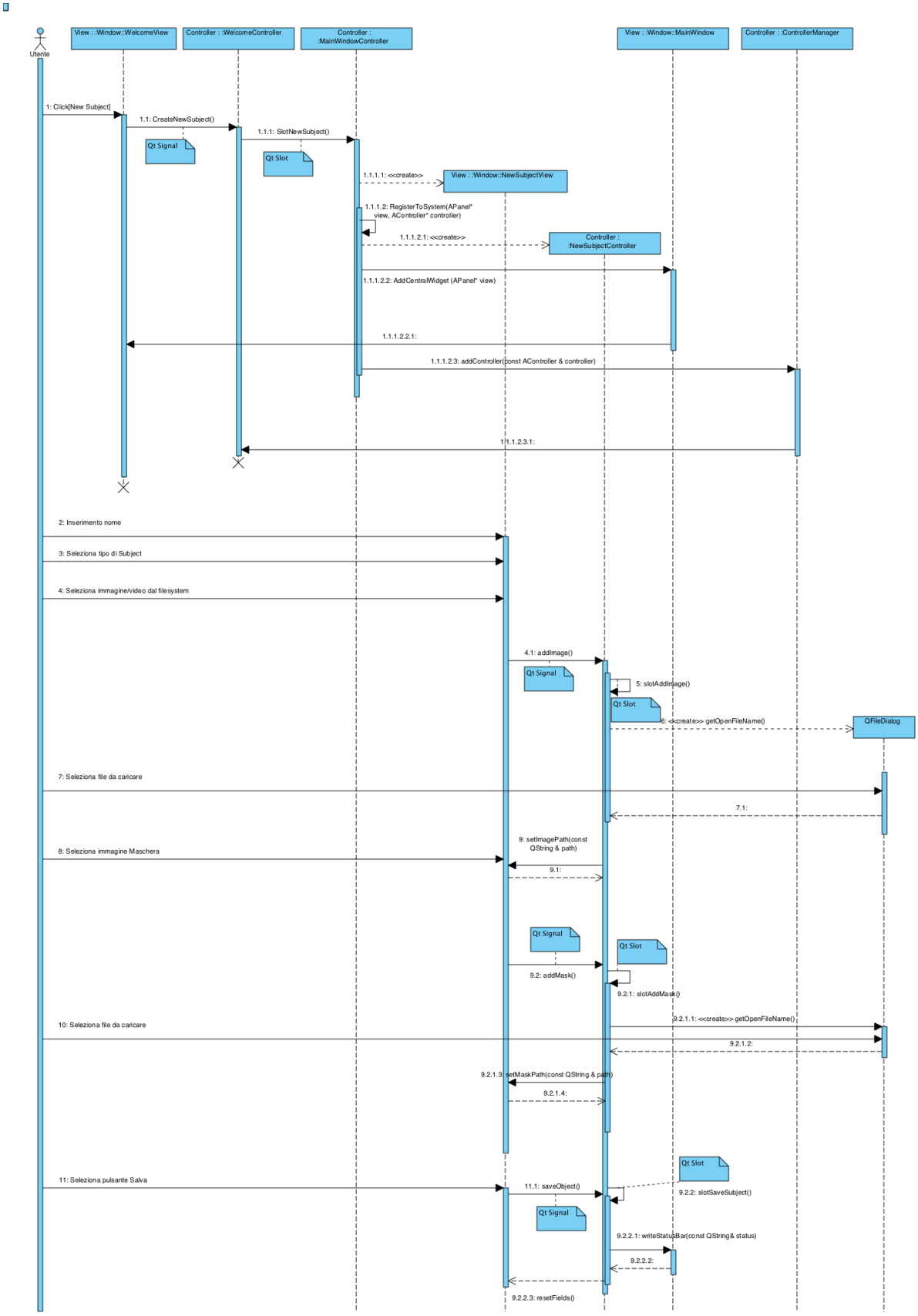

Figura 126: Diagramma di sequenza: creazione Subject

7.2 Creazione di un nuovo Protocol

Il seguente diagramma pone enfasi sulle interazioni tra vista, controller e base di dati. La situazione descritta presuppone che l'utente si trovi nella vista di creazione di un Protocol$_{\mathbf{G}}$, visto che queste sono descritte nel diagramma precedente. L'utente interagisce con la vista inserendo il *Nome* e il *Tipo* del Protocol$_{\mathbf{G}}$. Esso può inoltre aggiungere una o più Feature$_{\mathbf{G}}$, premendo il pulsante *AddFeature*. In seguito alla pressione del pulsante, vengono effettuate le seguenti operazioni:

1. *NewProtocolView* emette il signal$_{\mathbf{G}}$ *addFeatureClicked*, raccolto da *NewProtocolController*;
2. Il controller reagisce invocando lo slot$_{\mathbf{G}}$ *slotAddFeatureClicked*, che interagisce con *FeatCreator* per recuperare la lista delle Feature$_{\mathbf{G}}$ presenti (in questo caso del primo ordine), con i relativi parametri.

A questo punto l'utente interagisce con una finestra di dialogo, che gli permette di selezionare la Feature$_{\mathbf{G}}$ da aggiungere e di settare i vari parametri. In seguito alla pressione del pulsante *Ok*, vengono scatenate le seguenti operazioni:

1. Emissione del signal *Ok* da parte del dialogo *FeatureParams*, che ritorna la lista dei parametri al *NewProtocolController*;
2. Il controller reagisce invocando lo slot$_{\mathbf{G}}$ *slotOk*, recuperando la Feature$_{\mathbf{G}}$ da aggiungere. In seguito aggiunge la Feature$_{\mathbf{G}}$ al *NewProtocolFeatureTableModel* e distrugge la vista del dialogo.

L'utente può aggiungere una o più Feature; verranno quindi ripetute le operazioni precedenti, finché l'utente non seleziona il pulsante *Save*. In seguito vengono eseguite le seguenti operazioni:

1. Viene emesso il signal$_{\mathbf{G}}$ *saveObject*, raccolto dal *NewProtocolController*;
2. Il controller reagisce invocando lo slot$_{\mathbf{G}}$ *slotSaveProtocol*, che recupera dalla vista *NewProtocolView* le informazioni sul *Nome*, il *Tipo* e le *Feature* che compongono il Protocol$_{\mathbf{G}}$ da salvare;
3. Il controller in seguito crea un oggetto *ProtocolDAO*, che consente al sistema di salvare il Protocol$_{\mathbf{G}}$ sulla base di dati, tramite il metodo *createProtocol*;
4. Il controller poi si preoccupa di reimpostare le varie i campi dati delle *form* di inserimento, consentendo la successiva creazione di un nuovo Protocol$_{\mathbf{G}}$;
5. Il controller infine invoca il metodo *writeStatusBar* della *MainWindow*, che informerà l'utente dell'avvenuto inserimento.

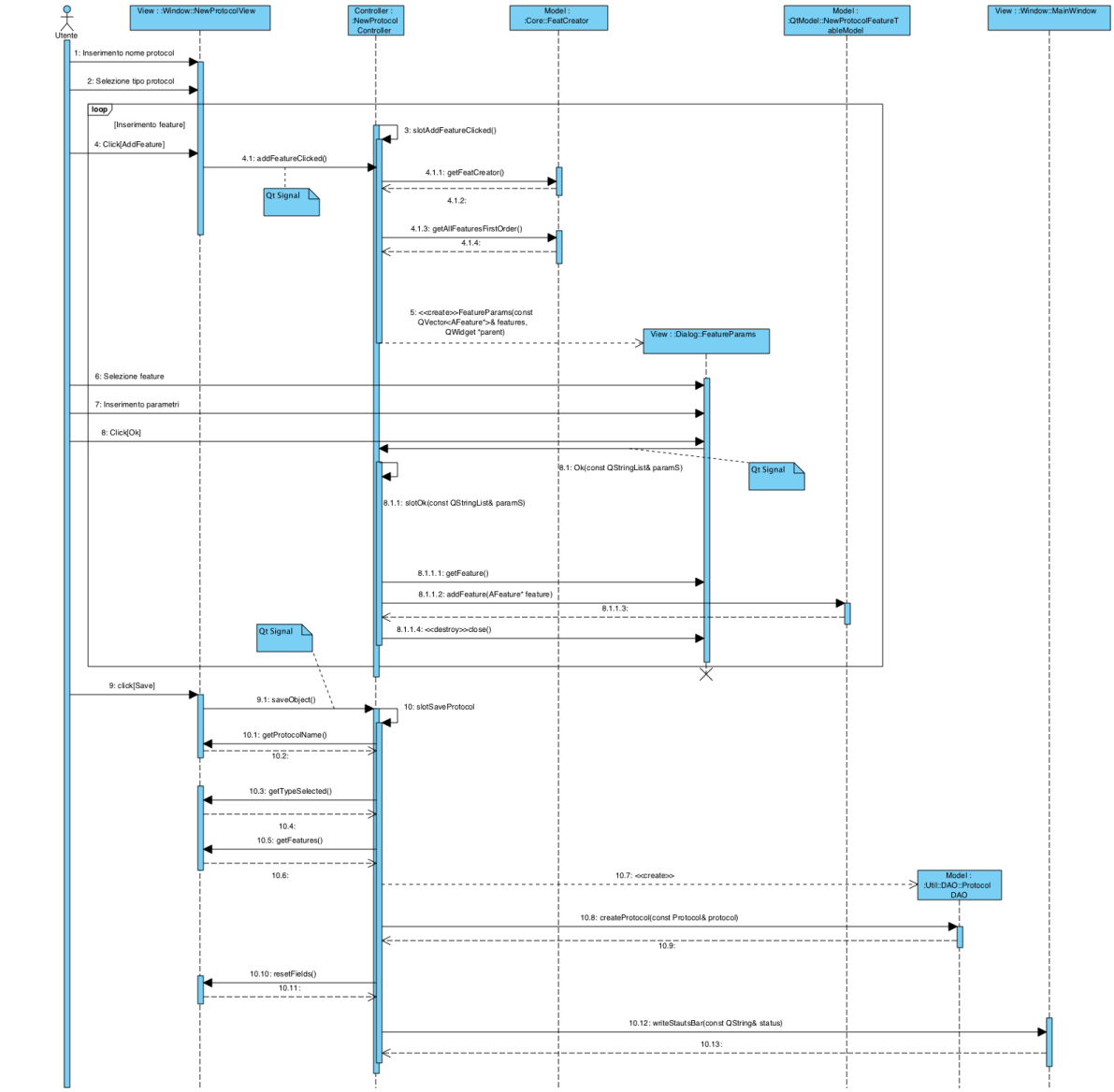

Figura 127: Diagramma di sequenza: creazione Protocol

7.3 Avvio di un'analisi

Il diagramma seguente descrive la sequenza di operazioni che vengono effettuate quando si procede all'avvio di un'analisi. La situazione descritta presuppone che l'utente si trovi nella vista di avvio analisi. L'utente potrà selezionare il *Dataset* sul quale effettuare l'analisi, un insieme di Subject$_{\mathbf{G}}$, le Feature$_{\mathbf{G}}$ di cui vogliamo esportare i risultati e quelle di cui vogliamo visualizzare l'anteprima durante l'analisi. Dovrà inoltre selezionare la cartella di destinazione dei risultati; questo comporterà le seguenti operazioni:

1. Emissione di un signal$_{\mathbf{G}}$ da parte della *AnalysisView*, raccolto dall'*AnalysisController*;
2. Il controller reagisce invocando lo slot$_{\mathbf{G}}$ *slotSelectResultsFolder*, il quale crea una nuova finestra di dialogo dal quale l'utente può specificare la cartella di destinazione;
3. In seguito il controller invoca il metodo *setDestinationFolder* della vista il quale imposta il campo in cui viene visualizzato il path di esportazione selezionato.

A questo punto l'utente può avviare l'analisi premendo il pulsante *Start Analysis*. Questo comporta le seguenti operazioni:

1. Emissione del signal *startAnalysis*, raccolto dall'*AnalysisController*;
2. Il controller reagisce invocando lo slot$_{\mathbf{G}}$ *slotStartAnalysis*.

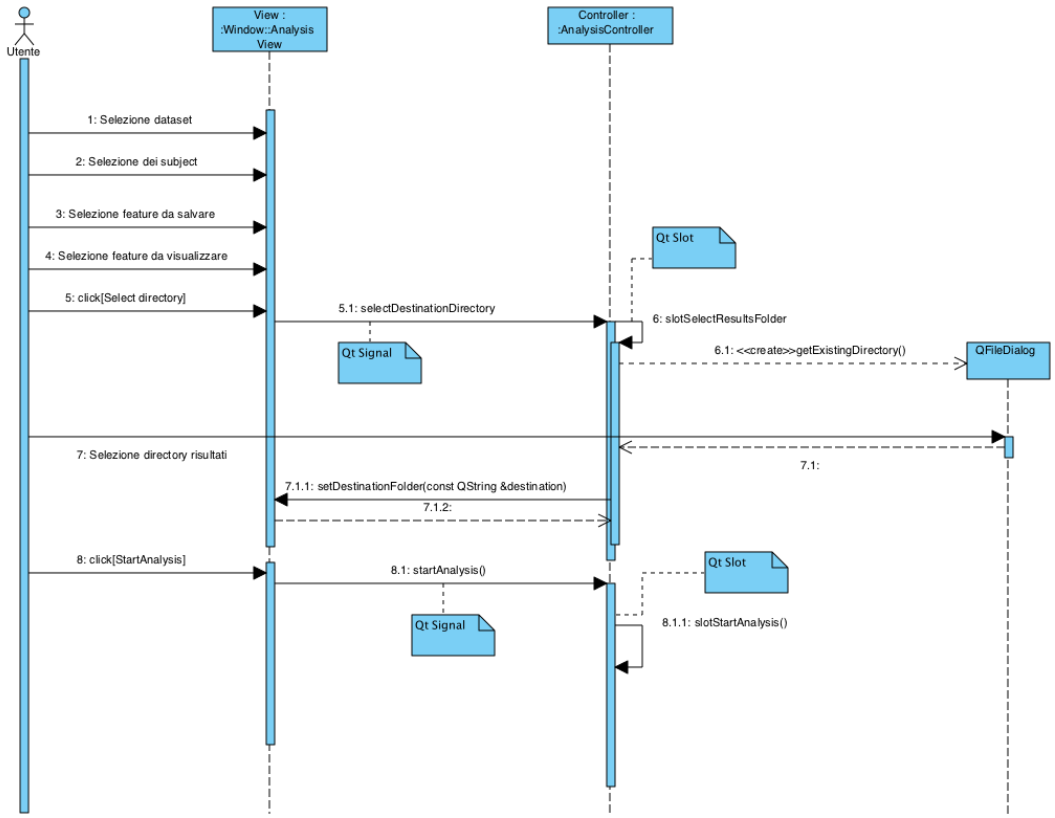

Figura 128: Diagramma di sequenza: esecuzione analisi

7.4 slotStartAnalysis

Il diagramma seguente descrive la sequenza di operazioni che vengono effettuate quando l'utente ha cliccato il pulsante di avvio analisi. Il controller *AnalysisController* esegue queste operazioni:

1. Crea una finestra di dialogo *AnalysisDialog*;
2. Crea un oggetto di tipo *Analysis*;
3. Invoca il metodo *show* di *AnalysisDialog* che mostra il dialogo dell'analisi;
4. Invoca il metodo *start* della classe *Analysis*.

A questo punto la classe *Analysis* chiama il suo metodo *run* che genera un ciclo, sui subject$_{\mathbf{G}}$ su cui viene fatta l'analisi, che esegue le seguenti operazioni:

1. Emette il segnale *beginSubject* che viene ricevuto dal controller *AnalysisController* dal suo slot *slotIncrementSubject*;
2. Lo slot chiama il metodo *incrementCurrentSubject* del dialogo *AnalysisDialog* che si occupa di incrementare gli indici che segnano su quale subject$_{\mathbf{G}}$ stiamo eseguendo l'analisi;
3. In seguito la classe *Analysis* esegue il metodo *executeAnalysisOnSubject* il quale esegue l'analisi sul subject$_{\mathbf{G}}$ corrente;

Dopo aver eseguito l'analisi su tutti i subject$_{\mathbf{G}}$ la classe *Analysis* emette il segnale *resultReady* che viene ricevuto dallo slot *slotFinishAnalysis* del controller *AnalysisController*.
Lo slot chiama il metodo *analysisFinish* del dialogo *AnalysisDialog* che imposta le informazioni del dialogo in modo da segnalare all'utente la corretta terminazione dell'analisi.
L'utente a questo punto può cliccare nel pulsante "Ok" oppure chiudere il dialogo; l'azione emette un segnale che viene ricevuto dallo slot ridefinito *closeEvent* del dialogo *AnalysisDialog*.
Questo slot emette il segnale *closeOnFinish* che viene ricevuto dallo slot *slotResetView* del controller *AnalysisController* che reimposta la vista di avvio analisi e chiude il dialogo *AnalysisDialog*.

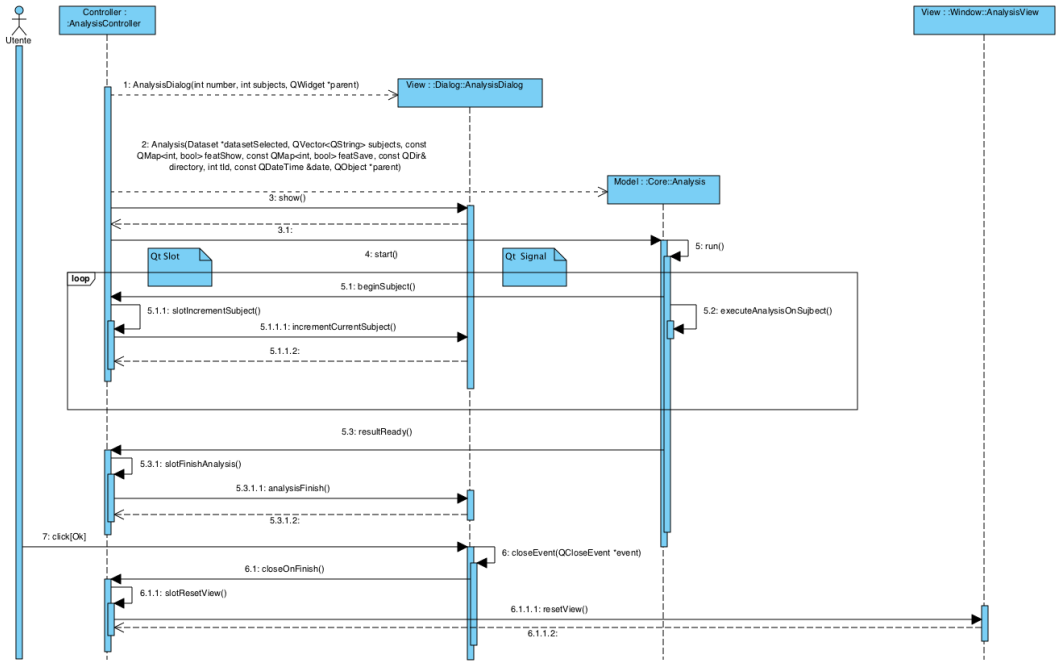

Figura 129: Diagramma di sequenza: slotStartAnalysis

A Tracciamento

A.1 Tracciamento classi-requisiti

Nella tabella sottostante sono presenti delle celle vuote in corrispondenza di alcune classi concrete, a causa del tracciamento dei requisiti all'interfaccia che espone i loro metodi.

Classi	Requisiti
AAlgorithm	
ADatabase	
AFeature	
AlgCreator	
Analysis	R0F10 R0F10.1 R0F10.1.1 R0F10.2 R0F10.2.1 R0F10.3 R0F10.5 R0F12.3 R1F28 R1F30 R1F31
AnalysisDAO	R0F13 R1F29
AnalyzeExporter	
AnalyzeReader	R0F1.2.1.6 R0F1.3.5
APanel	 R0F9
ASubject	R0F1 R0F1.1
ContrastFeature	R0F5.2.5 R0F5.2.5.1 R0F5.2.5.1.1 R0F5.2.5.1.2 R0F5.2.5.2 R0F5.2.5.2.1
CorrelationFeature	R0F5.2.9 R0F5.2.9.1 R0F5.2.9.1.1 R0F5.2.9.1.2 R0F5.2.9.2 R0F5.2.9.2.1
Dataset	R0F8 R0F8.1 R0F8.2 R0F8.3
DatasetDAO	R0F27 R0F27.1 R0F8.1
DatasetsView	R0F27 R0F27.1 R0F27.2
Dialog	R0F1.4

DynamicFeature	
EnergyFeature	R0F5.2.8 R0F5.2.8.1 R0F5.2.8.1.1 R0F5.2.8.1.2 R0F5.2.8.2 R0F5.2.8.2.1
EntropyFeature	R0F5.2.7 R0F5.2.7.1 R0F5.2.7.1.1 R0F5.2.7.1.2 R0F5.2.7.2 R0F5.2.7.2.1
ExecuteAnalysisView	R0F10.3 R0F10.5 R1F31
Exporter	R0F12.1 R0F12.2 R0F12.3 R0F12.4
FeatCreator	
FirstOrderFeature	
FuzzyCMeans	R0F5.4.2 R0F5.4.2.1 R0F5.4.2.1.1 R0F5.4.2.2 R0F5.4.2.2.1 R0F5.4.2.3 R0F5.4.2.3.1 R0F5.4.2.4 R0F5.4.2.4.1
GroupDAO	R0F26 R0F26.1 R0F26.2 R0F3 R0F4 R0F4.2
GroupOfSubject	R0F3 R0F3.1 R2F3.2
GroupsView	R0F26 R0F26.1 R0F26.2 R0F4 R0F4.1 R0F4.2 R0F4.3
Help	R2F14.1.1 R2F14.1.2 R2F14.2 R2F14.2.1
HelpView	R0F14 R0F14.1 R2F14.1.1 R2F14.1.2

	R2F14.2 R2F14.2.1
Hierarchical	R0F5.4.3 R0F5.4.3.1 R0F5.4.3.1.1 R0F5.4.3.2 R0F5.4.3.2.1
HomogeneityFeature	R0F5.2.6 R0F5.2.6.1 R0F5.2.6.1.1 R0F5.2.6.1.2 R0F5.2.6.2 R0F5.2.6.2.1
ImageExporter	
ImageReader	R0F1.2.1.1 R0F1.2.1.2 R0F1.2.1.3 R0F1.3.1 R0F1.3.2 R0F1.3.3
InternalData	
InternalData2D	
InternalData3D	
ISubject	
KMeans	R0F5.4.1 R0F5.4.1.1 R0F5.4.1.1.1 R0F5.4.1.2 R0F5.4.1.2.1 R0F5.4.1.3 R0F5.4.1.3.1 R0F5.4.1.4 R0F5.4.1.4.1
KurtosisFeature	R0F5.2.4 R0F5.2.4.1 R0F5.2.4.1.1 R0F5.2.4.1.2
Log	
MainWindow	R0F9
MaximumFeature	R0F5.2.11 R0F5.2.11.1 R0F5.2.11.1.1 R0F5.2.11.2 R0F5.2.11.2.1
MeanDFeature	R0F5.2.14 R0F5.2.14.1 R0F5.2.14.1.1 R0F5.2.14.2 R0F5.2.14.2.1
MeanFeature	R0F5.2.1 R0F5.2.1.1 R0F5.2.1.1.1 R0F5.2.1.1.2
MenuBar	
MinimumFeature	R0F5.2.12 R0F5.2.12.1

	R0F5.2.12.1.1 R0F5.2.12.2 R0F5.2.12.2.1
NavWidget	
NewDatasetView	R0F8 R0F8.1 R0F8.2 R0F8.3
NewGroupView	R0F3 R0F3.1
NewProtocolView	R0F5 R0F5.1 R0F5.2 R0F5.2.1 R0F5.2.10 R0F5.2.11 R0F5.2.12 R0F5.2.13 R0F5.2.14 R0F5.2.15 R0F5.2.2 R0F5.2.3 R0F5.2.4 R0F5.2.5 R0F5.2.6 R0F5.2.7 R0F5.2.8 R0F5.2.9 R0F5.3 R0F5.4 R0F5.4.1 R0F5.4.2 R0F5.4.3 R2F5.5
NewSubjectView	R0F1 R0F1.1 R0F1.2 R0F1.3
NiftiExporter	
NiftiReader	R0F1.2.1.5 R0F1.3.4
Protocol	R0F5 R0F5.1 R0F5.2 R2F5.5
ProtocolDAO	R0F11 R0F5.1 R0F6
ProtocolsView	R0F6 R0F6.1
ProxySubject	
Reader	R0F1.2 R0F1.2.1 R0F1.3 R0F1.4
RealSubject	R0F1.2

	R0F1.3
RealSubject2D	
RealSubject3D	
ResaultsView	R0F10.4 R0F13
SecondOrderFeature	
SelectDeselectWidget	
SkewnessFeature	R0F5.2.3 R0F5.2.3.1 R0F5.2.3.1.1 R0F5.2.3.1.2
SlopeFeature	R0F5.2.13 R0F5.2.13.1 R0F5.2.13.1.1 R0F5.2.13.2 R0F5.2.13.2.1
StartAnalysisView	R0F12.4 R1F32
StdDeviationFeature	R0F5.2.2 R0F5.2.2.1 R0F5.2.2.1.1 R0F5.2.2.1.2
SubjectDAO	R0F1.1 R0F2 R0F3.1
SubjectsView	R0F2
TimeToPeakFeature	R0F5.2.10 R0F5.2.10.1 R0F5.2.10.1.1 R0F5.2.10.2 R0F5.2.10.2.1
ToolBar	
ValueFeature	R0F5.2.15 R0F5.2.15.1 R0F5.2.15.1.1 R0F5.2.15.2 R0F5.2.15.2.1
VideoReader	R0F1.2.1.4
WelcomeView	

Tabella 3: Tracciamento Classe-Requisito

A.2 Tracciamento requisiti-classi

Requisiti	**Classi**
R0F1	NewSubjectView ASubject
R0F1.1	SubjectDAO NewSubjectView ASubject
R0F1.2	Reader NewSubjectView RealSubject

R0F1.2.1	Reader
R0F1.2.1.1	ImageReader
R0F1.2.1.2	ImageReader
R0F1.2.1.3	ImageReader
R0F1.2.1.4	VideoReader
R0F1.2.1.5	NiftiReader
R0F1.2.1.6	AnalyzeReader
R0F1.3	Reader NewSubjectView RealSubject
R0F1.3.1	ImageReader
R0F1.3.2	ImageReader
R0F1.3.3	ImageReader
R0F1.3.4	NiftiReader
R0F1.3.5	AnalyzeReader
R0F1.4	Reader Dialog
R0F10	Analysis
R0F10.1	Analysis
R0F10.1.1	Analysis
R0F10.2	Analysis
R0F10.2.1	Analysis
R0F10.3	ExecuteAnalysisView Analysis
R0F10.4	ResaultsView
R0F10.5	ExecuteAnalysisView Analysis
R0F11	ProtocolDAO
R0F12	
R0F12.1	Exporter
R0F12.2	Exporter
R0F12.3	Exporter Analysis
R0F12.4	Exporter StartAnalysisView
R0F13	AnalysisDAO ResaultsView
R0F13.1	
R0F13.2	
R0F14	HelpView
R0F14.1	HelpView
R0F2	SubjectDAO SubjectsView
R0F26	GroupDAO GroupsView
R0F26.1	GroupDAO GroupsView
R0F26.2	GroupDAO GroupsView
R0F27	DatasetDAO DatasetsView
R0F27.1	DatasetDAO DatasetsView
R0F27.2	DatasetsView

R0F3	GroupDAO NewGroupView GroupOfSubject
R0F3.1	SubjectDAO NewGroupView GroupOfSubject
R0F4	GroupDAO GroupsView
R0F4.1	GroupsView
R0F4.2	GroupDAO GroupsView
R0F4.3	GroupsView
R0F5	NewProtocolView Protocol
R0F5.1	ProtocolDAO NewProtocolView Protocol
R0F5.2	NewProtocolView Protocol
R0F5.2.1	NewProtocolView MeanFeature
R0F5.2.1.1	MeanFeature
R0F5.2.1.1.1	MeanFeature
R0F5.2.1.1.2	MeanFeature
R0F5.2.10	NewProtocolView TimeToPeakFeature
R0F5.2.10.1	TimeToPeakFeature
R0F5.2.10.1.1	TimeToPeakFeature
R0F5.2.10.2	TimeToPeakFeature
R0F5.2.10.2.1	TimeToPeakFeature
R0F5.2.11	NewProtocolView MaximumFeature
R0F5.2.11.1	MaximumFeature
R0F5.2.11.1.1	MaximumFeature
R0F5.2.11.2	MaximumFeature
R0F5.2.11.2.1	MaximumFeature
R0F5.2.12	NewProtocolView MinimumFeature
R0F5.2.12.1	MinimumFeature
R0F5.2.12.1.1	MinimumFeature
R0F5.2.12.2	MinimumFeature
R0F5.2.12.2.1	MinimumFeature
R0F5.2.13	NewProtocolView SlopeFeature
R0F5.2.13.1	SlopeFeature
R0F5.2.13.1.1	SlopeFeature
R0F5.2.13.2	SlopeFeature
R0F5.2.13.2.1	SlopeFeature
R0F5.2.14	NewProtocolView MeanDFeature
R0F5.2.14.1	MeanDFeature
R0F5.2.14.1.1	MeanDFeature
R0F5.2.14.2	MeanDFeature
R0F5.2.14.2.1	MeanDFeature
R0F5.2.15	NewProtocolView

	ValueFeature
R0F5.2.15.1	ValueFeature
R0F5.2.15.1.1	ValueFeature
R0F5.2.15.2	ValueFeature
R0F5.2.15.2.1	ValueFeature
R0F5.2.2	NewProtocolView StdDeviationFeature
R0F5.2.2.1	StdDeviationFeature
R0F5.2.2.1.1	StdDeviationFeature
R0F5.2.2.1.2	StdDeviationFeature
R0F5.2.3	NewProtocolView SkewnessFeature
R0F5.2.3.1	SkewnessFeature
R0F5.2.3.1.1	SkewnessFeature
R0F5.2.3.1.2	SkewnessFeature
R0F5.2.4	NewProtocolView KurtosisFeature
R0F5.2.4.1	KurtosisFeature
R0F5.2.4.1.1	KurtosisFeature
R0F5.2.4.1.2	KurtosisFeature
R0F5.2.5	NewProtocolView ContrastFeature
R0F5.2.5.1	ContrastFeature
R0F5.2.5.1.1	ContrastFeature
R0F5.2.5.1.2	ContrastFeature
R0F5.2.5.2	ContrastFeature
R0F5.2.5.2.1	ContrastFeature
R0F5.2.6	NewProtocolView HomogeneityFeature
R0F5.2.6.1	HomogeneityFeature
R0F5.2.6.1.1	HomogeneityFeature
R0F5.2.6.1.2	HomogeneityFeature
R0F5.2.6.2	HomogeneityFeature
R0F5.2.6.2.1	HomogeneityFeature
R0F5.2.7	NewProtocolView EntropyFeature
R0F5.2.7.1	EntropyFeature
R0F5.2.7.1.1	EntropyFeature
R0F5.2.7.1.2	EntropyFeature
R0F5.2.7.2	EntropyFeature
R0F5.2.7.2.1	EntropyFeature
R0F5.2.8	NewProtocolView EnergyFeature
R0F5.2.8.1	EnergyFeature
R0F5.2.8.1.1	EnergyFeature
R0F5.2.8.1.2	EnergyFeature
R0F5.2.8.2	EnergyFeature
R0F5.2.8.2.1	EnergyFeature
R0F5.2.9	NewProtocolView CorrelationFeature
R0F5.2.9.1	CorrelationFeature
R0F5.2.9.1.1	CorrelationFeature
R0F5.2.9.1.2	CorrelationFeature
R0F5.2.9.2	CorrelationFeature
R0F5.2.9.2.1	CorrelationFeature

R0F5.3	NewProtocolView
R0F5.4	NewProtocolView
R0F5.4.1	NewProtocolView KMeans
R0F5.4.1.1	KMeans
R0F5.4.1.1.1	KMeans
R0F5.4.1.2	KMeans
R0F5.4.1.2.1	KMeans
R0F5.4.1.3	KMeans
R0F5.4.1.3.1	KMeans
R0F5.4.1.4	KMeans
R0F5.4.1.4.1	KMeans
R0F5.4.2	NewProtocolView FuzzyCMeans
R0F5.4.2.1	FuzzyCMeans
R0F5.4.2.1.1	FuzzyCMeans
R0F5.4.2.2	FuzzyCMeans
R0F5.4.2.2.1	FuzzyCMeans
R0F5.4.2.3	FuzzyCMeans
R0F5.4.2.3.1	FuzzyCMeans
R0F5.4.2.4	FuzzyCMeans
R0F5.4.2.4.1	FuzzyCMeans
R0F5.4.3	NewProtocolView Hierarchical
R0F5.4.3.1	Hierarchical
R0F5.4.3.1.1	Hierarchical
R0F5.4.3.2	Hierarchical
R0F5.4.3.2.1	Hierarchical
R0F6	ProtocolDAO ProtocolsView
R0F6.1	ProtocolsView
R0F7	
R0F8	NewDatasetView Dataset
R0F8.1	DatasetDAO NewDatasetView Dataset
R0F8.2	NewDatasetView Dataset
R0F8.3	NewDatasetView Dataset
R0F9	MainWindow APanel
R0Q22	
R0Q23	
R0Q24	
R0Q25	
R0Q25.1	
R0V15	
R0V16	
R0V17	
R0V18	
R0V19	
R0V20	

R0V21	
R1F28	Analysis
R1F29	AnalysisDAO
R1F30	Analysis
R1F31	ExecuteAnalysisView Analysis
R1F32	StartAnalysisView
R2F14.1.1	Help HelpView
R2F14.1.2	Help HelpView
R2F14.2	Help HelpView
R2F14.2.1	Help HelpView
R2F3.2	GroupOfSubject
R2F5.5	NewProtocolView Protocol

Tabella 4: Tracciamento Requisiti-Classi

A.3 Tracciamento modulo-test

Metodi	Test
AController::AController()	
AlgorithmDAO::addAlgorithm()	TU10
MainWindow::addCentralWidget()	TU39
AnalysisDialog::addConnect()	TU39
APanel::addConnect()	TU39
DatasetsView::addConnect()	TU39
DetailedResult::addConnect()	TU39
FeatureParams::addConnect()	TU39
GroupView::addConnect()	TU39
NewDatasetView::addConnect()	TU39
NewGroupView::addConnect()	TU39
NewProtocolView::addConnect()	TU39
NewSubjectView::addConnect()	TU39
ProtocolsView::addConnect()	TU39
ResultsView::addConnect()	TU39
SubjectsView::addConnect()	TU39
WelcomeView::addConnect()	TU39
MainWindow::addConnections()	
SubjectsController::addConnections()	
ControllerManager::addController()	
NewProtocolView::addFeatureClicked()	
NewSubjectView::addImage()	
NewSubjectView::addMask()	
DatasetDAO::addProtocol()	TU8
GroupDAO::addSubjectToGroup()	TU7
AlgorithmDAO::AlgorithmDAO()	TU10
AnalysisController::AnalysisController()	
AnalysisDAO::AnalysisDAO()	TU11

AnalysisDialog::AnalysisDialog()	
APanel::APanel()	
APanel::backView()	
ContrastFeature::ContrastFeature()	TU24 TU25
ControllerManager::ControllerManager()	
CorrelationFeature::CorrelationFeature()	TU20 TU21
Log::countLines()	TU36
AnalysisDAO::createAnalysis()	TU11
AnalysisDialog::createButtom()	TU39
DatasetsView::createButtom()	TU39
DetailedResult::createButtom()	TU39
FeatureParams::createButtom()	TU39
GroupView::createButtom()	TU39
NewDatasetView::createButtom()	TU39
NewGroupView::createButtom()	TU39
NewProtocolView::createButtom()	TU39
NewSubjectView::createButtom()	TU39
ResultsView::createButtom()	TU39
SubjectsView::createButtom()	TU39
ProtocolsView::createButton()	TU39
NewDatasetView::createCenter()	TU39
NewProtocolView::createCenter()	TU39
AController::createConnections()	
AnalysisController::createConnections()	
DetailedResultView::createConnections()	
GroupsController::createConnections()	
MainWindowController::createConnections()	
NewGroupController::createConnections()	
NewProtocolController::createConnections()	
NewSubjectController::createConnections()	
ProtocolsControlle::createConnections()	
WelcomeController::createConnections()	
DatasetDAO::createDataset()	TU8
FeatureDAO::createFeature()	TU37
FeatureParams::createFeatureBox()	
FeatureParams::createFirstBox()	
GroupDAO::createGroup()	TU7
AnalysisDialog::createLayout()	
FeatureParams::createLayout()	
NewProtocolView::createLeft()	
AnalysisDialog::createProgress()	
ProtocolDAO::createProtocol()	TU9
NewProtocolView::createRight()	
SubjectDAO::createSubject()	TU6
DatasetsView::createTop()	
AnalysisDialog::createTop()	TU39
DetailedResult::createTop()	TU39
FeatureParams::createTop()	TU39
GroupView::createTop()	TU39
NewDatasetView::createTop()	TU39
NewGroupView::createTop()	TU39
NewProtocolView::createTop()	TU39

NewSubjectView::createTop()	TU39
ProtocolsView::createTop()	TU39
ResultsView::createTop()	TU39
SubjectsView::createTop()	TU39
DatasetDAO::DatasetDAO()	TU8
DatasetsView::DatasetsView()	
AlgorithmDAO::deleteAlgorithm()	TU10
AnalysisDAO::deleteAnalysis()	TU11
DatasetDAO::deleteDataset()	TU8
GroupDAO::deleteGroup()	TU7
APanel::deleteObject()	
ProtocolDAO::deleteProtocol()	TU9
MainWindow::deleteToolBars()	
APanel::deselectAll()	
DetailedResult::DetailedResult	
DetailedResultView::DetailedResultView()	
Dialog::Dialog()	
Dialog::dialogCritical()	
Dialog::dialogInfo()	
Dialog::dialogQuestion()	
APanel::editObject()	
EnergyFeature::EnergyFeature()	TU22 TU23
EntropyFeature::EntropyFeature()	TU28 TU29
GroupDAO::existGroupWithName()	TU7
ProtocolDAO::existProtocolWithName()	TU9
SubjectDAO::existSubjectlWithName()	TU6
DatasetDAO::existSubjectWithName()	TU8
ImageExporter::exportFile()	TU4
AnalyzeExporter::exportFile()	TU5
FeatureDAO::FeatureDAO()	TU37
FeatureParams::FeatureParams()	
FeatureParams::featureSelected()	
FirstOrderFeature::FirstOrderFeature	TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19
FuzzyCMeansAlgorithm::FuzzyCMeansAlgorithm()	TU30 TU31
AlgorithmDAO::getAlgorithmById()	TU10
ProtocolsView::getAlgorithmLabel()	TU38
AlgorithmDAO::getAlgorithmOfProtocol()	TU10
NewProtocolView::getAlgorithmSelected()	TU38
AlgorithmDAO::getAllAlgorithm()	TU10
AnalysisDAO::getAllAnalysis()	TU11
DatasetDAO::getAllDataset()	TU8
DatasetDAO::getAllDatasetName()	TU8
FeatureDAO::getAllFeatureOfProtocol()	TU37
GroupDAO::getAllGroup()	TU7

ProtocolDAO::getAllProtocol()	TU9
ProtocolDAO::getAllProtocolName()	TU9
ProtocolDAO::getAllProtocolOfDataset()	TU9
SubjectDAO::getAllSubject()	TU6
SubjectDAO::getAllSubjectName()	TU6
AnalysisDAO::getAnalysisByDate()	TU11
AnalysisDAO::getAnalysisOfDataset()	TU11
ControllerManager::getControllerManger()	TU38
KMeansAlgorithm::getDistance()	TU34 TU35
FeatureParams::getFeature()	TU38
ProtocolsView::getFeatureLabel()	TU38
FeatureDAO::getFeatureOfProtocol()	TU37
NewProtocolView::getFeaturesTable()	TU38
FuzzyCMeansAlgorithm ::getFuzzyIndex()	TU30 TU31
GroupView::getGroup()	TU38
GroupDAO::getGroupByName()	TU7
NewDatasetView::getGroupInfoLabel()	TU38
DatasetDAO::getGroupOfDataset()	TU8
SubjectDAO::getGroupOfSubject()	TU6
NewDatasetView::getGroupTable()	TU38
NewSubjectView::getImagePath()	
MainWindowController::getMainWindowController()	TU38
NewSubjectView::getMaskPath()	TU38
FuzzyCMeansAlgorithm ::getMaxIteration()	TU30 TU31
KMeansAlgorithm::getMaxIteration()	TU34 TU35
NewGroupView::getName()	
NewGroupView::getNewGroupTableModel()	
FuzzyCMeansAlgorithm ::getNumberOfCluster()	TU30 TU31
KMeansAlgorithm::getNumberOfCluster()	TU34 TU35
KMeansAlgorithm::getNumberOfReplicates()	TU34 TU35
FirstOrderFeature::getParameters()	TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19
SecondOrderFeature::getParameters()	TU20 TU21 TU22 TU23 TU24 TU25 TU26 TU27 TU28

	TU29
ProtocolDAO::getProtocolByName()	TU9
NewDatasetView::getProtocolInfoLabel()	TU38
NewProtocolView::getProtocolName()	TU38
DatasetDAO::getProtocolOfDataset()	TU8
NewDatasetView::getProtocolsTable()	TU38
NewGroupView::getSelectedSubjects()	TU38
SubjectDAO::getSubjectByName()	TU6
NewSubjectView::getSubjectName()	TU38
GroupDAO::getSubjectOfGroup()	TU7
SubjectDAO::getSubjectsByType()	TU6
GroupView::getSubjectsTable()	TU38
NewGroupView::getTableView	TU38
SubjectsView::getTableView()	TU38
FuzzyCMeansAlgorithm ::getThreshold()	TU30 TU31
FirstOrderFeature::getType()	TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19
SecondOrderFeature::getType()	TU20 TU21 TU22 TU23 TU24 TU25 TU26 TU27 TU28 TU29
NewGroupView::getType()	TU38
APanel::getTypeRomeo()	TU38
NewProtocolView::getTypeSelected()	TU38
NewSubjectView::getTypeSelected()	TU38
FirstOrderFeature::getWindowSize()	TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19
SecondOrderFeature::getWindowSize()	TU20 TU21 TU22 TU23 TU24 TU25 TU26 TU27 TU28

	TU29
GroupDAO::GroupDAO()	TU7
GroupsController::GroupsController()	
NewDatasetView::groupSelected()	
GroupView::GroupView()	
DetailedResultView::HelpController()	
APanel::helpView()	
HomogeneityFeature ::HomogeneityFeature ()	TU27
HomogeneityFeature ::HomogeneityFeature()	TU26
ImageExporter::ImageExporter()	TU4
ImageReader::ImageReader()	TU1 TU2
NewSubjectView::imageTextChanded()	
APanel::initializeView()	
GroupView::itemSelected()	TU39
SubjectsView::itemSelected()	TU39
KMeansAlgorithm::KMeansAlgorithm()	TU34 TU35
KurtosisFeature::KurtosisFeature()	TU16 TU17
AnalysisDialog::loadCss()	TU39
APanel::loadCss()	TU39
DatasetsView::loadCss()	TU39
DetailedResult::loadCss()	TU39
FeatureParams::loadCss()	TU39
GroupView::loadCss()	TU39
MainWindow::loadCss()	TU39
NewDatasetView::loadCss()	TU39
NewGroupView::loadCss()	TU39
NewProtocolView::loadCss()	TU39
NewSubjectView::loadCss()	TU39
ProtocolsView::loadCss()	TU39
ResultsView::loadCss()	TU39
SubjectsView::loadCss()	TU39
WelcomeView::loadCss()	TU39
MainWindow::loadCssStatic()	TU39
Log::Log()	TU36
Log::logWriter()	TU36
MainWindow::MainWindow()	
MainWindowController::MainWindowController()	
NewSubjectView::maskTextChanded()	
MeanFeature::MeanFeature()	TU18 TU19
NewSubjectView::nameTextChanded()	
NewProtocolController::NewDatasetController()	
NewDatasetView::NewDatasetView()	
NewGroupController::NewGroupController()	
NewGroupView::NewGroupView()	
NewProtocolController::NewProtocolController()	
NewProtocolView::NewProtocolView()	
NewSubjectController::NewSubjectController()	
NewSubjectView::NewSubjectView()	
FeatureParams::ok()	
ProtocolDAO::ProtocolDAO()	TU9

ProtocolsController::ProtocolsController()	
ProtocolsView::protocolSelected()	
ProtocolsView::ProtocolsView()	
ImageReader::readFile2D()	TU1
VideoReader::readFile2D()	TU3
ImageReader::readFile3D()	TU2
VideoReader::readFile3D()	TU3
MainWindowController::registerToSystem()	
FeatureDAO::removeFeature()	TU37
GroupDAO::removeSubjectFromGroup()	TU7
FeatureParams::resetBox()	
NewGroupView::resetFields	
NewProtocolView::resetFields()	
NewSubjectView::resetFields()	
ProtocolsControlle::ResultsController	
ResultsView::ResultsView()	
APanel::saveObject()	
NewSubjectView::saveSubject()	
SecondOrderFeature::SecondOrderFeature()	TU20 TU21 TU22 TU23 TU24 TU25 TU26 TU27 TU28 TU29
APanel::selectAll()	TU38
NewGroupController::setEditFields()	TU38
Log::setEnabled()	TU36
FeatureParams::setFeature()	TU38
NewDatasetView::setGroupInfo()	TU38
NewSubjectView::setImagePath()	TU38
NewSubjectView::setMaskPath()	TU38
NewGroupView::setNSubjects	TU38
FirstOrderFeature::setParameters()	TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19
SecondOrderFeature::setParameters()	TU20 TU21 TU22 TU23 TU24
SecondOrderFeature::setParameters()	TU25 TU26 TU27 TU28 TU29

NewDatasetView::setProtocolInfo()	TU38
SubjectsView::setSubjectImg()	TU38
SubjectsView::setSubjectInfo()	TU38
SubjectsView::setSubjectMask()	TU38
NewSubjectView::setSubjectName()	TU38
MainWindow::setToolBar()	TU38
NewSubjectView::setTypeSelection()	TU38
NewDatasetView::setupCenterLayout()	TU39
SubjectsView::setupInfoBlock()	TU39
APanel::setupLayout()	TU39
DatasetsView::setupLayout()	TU39
DetailedResult::setupLayout()	TU39
FeatureParams::setupLayout()	TU39
GroupView::setupLayout()	TU39
NewDatasetView::setupLayout()	TU39
NewGroupView::setupLayout()	TU39
NewProtocolView::setupLayout()	TU39
NewSubjectView::setupLayout()	TU39
ProtocolsView::setupLayout()	TU39
ResultsView::setupLayout()	TU39
SubjectsView::setupLayout()	TU39
WelcomeView::setupLeftFrame()	TU39
MainWindow::setupMainWindow()	TU39
SubjectsView::setupObjectName()	
WelcomeView::setupObjectName()	
AnalysisDialog::setupObjectName()	TU39
APanel::setupObjectName()	TU39
DatasetsView::setupObjectName()	TU39
DetailedResult::setupObjectName()	TU39
FeatureParams::setupObjectName()	TU39
GroupView::setupObjectName()	TU39
NewDatasetView::setupObjectName()	TU39
NewGroupView::setupObjectName()	TU39
NewProtocolView::setupObjectName()	TU39
NewSubjectView::setupObjectName()	TU39
ProtocolsView::setupObjectName()	TU39
ResultsView::setupObjectName()	TU39
WelcomeView::setupRightFrame()	TU39
SubjectsView::setupTable()	TU39
AnalysisDialog::setupToolTip()	TU39
APanel::setupToolTip()	TU39
DatasetsView::setupToolTip()	TU39
DetailedResult::setupToolTip()	TU39
FeatureParams::setupToolTip()	TU39
GroupView::setupToolTip()	TU39
NewDatasetView::setupToolTip()	TU39
NewGroupView::setupToolTip()	TU39
NewProtocolView::setupToolTip()	TU39
NewSubjectView::setupToolTip()	TU39
ProtocolsView::setupToolTip()	TU39
ResultsView::setupToolTip()	TU39
SubjectsView::setupToolTip()	TU39
WelcomeView::setupToolTip()	TU39
DatasetsView::setupTopLayout()	TU39

DetailedResult::setupTopLayout()	TU39
GroupView::setupTopLayout()	TU39
NewDatasetView::setupTopLayout()	TU39
NewGroupView::setupTopLayout()	TU39
NewProtocolView::setupTopLayout()	TU39
ProtocolsView::setupTopLayout()	TU39
ResultsView::setupTopLayout()	TU39
APanel::setupView()	TU39
DatasetsView::setupView()	TU39
DetailedResult::setupView()	TU39
FeatureParams::setupView()	TU39
GroupView::setupView()	TU39
NewDatasetView::setupView()	TU39
NewGroupView::setupView()	TU39
NewProtocolView::setupView()	TU39
NewSubjectView::setupView()	TU39
ProtocolsView::setupView()	TU39
ResultsView::setupView()	TU39
SubjectsView::setupView()	TU39
WelcomeView::setupView()	TU39
StandardDeviationFeature::singleChannelExecution2D()	TU12
SkewnessFeature::singleChannelExecution2D()	TU14
KurtosisFeature::singleChannelExecution2D()	TU16
MeanFeature::singleChannelExecution2D()	TU18
CorrelationFeature::singleChannelExecution2D()	TU20
EnergyFeature::singleChannelExecution2D()	TU22
ContrastFeature::singleChannelExecution2D()	TU24
HomogeneityFeature ::singleChannelExecution2D()	TU26
EntropyFeature::singleChannelExecution2D()	TU28
FuzzyCMeansAlgorithm ::singleChannelExecution2D()	TU30
KMeansAlgorithm::singleChannelExecution2D()	TU34
StandardDeviationFeature::singleChannelExecution3D()	TU13
SkewnessFeature::singleChannelExecution3D()	TU15
KurtosisFeature::singleChannelExecution3D()	TU17
MeanFeature::singleChannelExecution3D()	TU19
CorrelationFeature::singleChannelExecution3D()	TU21
EnergyFeature::singleChannelExecution3D()	TU23
ContrastFeature::singleChannelExecution3D()	TU25
HomogeneityFeature ::singleChannelExecution3D()	TU27
EntropyFeature::singleChannelExecution3D()	TU29
FuzzyCMeansAlgorithm ::singleChannelExecution3D()	TU31
KMeansAlgorithm::singleChannelExecution3D()	TU35
SkewnessFeature::SkewnessFeature()	TU14 TU15
NewProtocolController::sloSaveDataset()	TU40
MainWindowController::slotAboutRomeo()	TU40
NewProtocolController::slotAddFeatureClicked()	TU40
NewSubjectController::slotAddImage()	TU40
NewSubjectController::slotAddMask()	TU40
MainWindowController::slotAnalysisResults()	TU40
AController::slotBack()	TU40
NewGroupController::slotBack()	TU40
NewSubjectController::slotChangeSubjectImage()	TU40
NewSubjectController::slotChangeSubjectMask()	TU40

NewSubjectController::slotChangeSubjectName()	TU40
NewSubjectController::slotChangeType()	TU40
AnalysisController::slotContinueAnalysis()	TU40
AnalysisController::slotDatasetSelected()	TU40
GroupsController::slotDeleteGroup()	TU40
ProtocolsControlle::slotDeleteProtocol()	TU40
NewGroupController::slotDeselectAll()	TU40
AnalysisController::slotDeselectAllFeatures()	TU40
GroupsController::slotEditGroup()	TU40
AnalysisController::slotExitFromAnalysis()	TU40
ProtocolsControlle::slotExportResult()	TU40
NewProtocolController::slotFeatureSelected()	TU40
NewProtocolController::slotGroupSelected()	TU40
AController::slotHelp()	TU40
GroupsController::slotItemSelected()	TU40
SubjectsController::slotItemSelected()	TU40
MainWindowController::slotNewDataset()	TU40
MainWindowController::slotNewGroup()	TU40
MainWindowController::slotNewProtocol()	TU40
MainWindowController::slotNewSubject()	TU40
DetailedResultView::slotNextPage()	TU40
FeatureParams::slotOk()	TU40
NewProtocolController::slotOk()	TU40
DetailedResultView::slotPreviousPage()	TU40
DetailedResultView::slotProtocolOrder()	TU40
NewProtocolController::slotProtocolSelected()	TU40
ProtocolsControlle::slotProtocolSelected()	TU40
NewGroupController::slotSaveGroup()	TU40
NewProtocolController::slotSaveProtocol()	
NewSubjectController::slotSaveSubject()	TU40
NewGroupController::slotSelectAll()	TU40
AnalysisController::slotSelectAllFeatures()	TU40
DetailedResultView::slotSelectedTreeItem()	TU40
AnalysisController::slotSelectOutputDirectory()	TU40
DetailedResultView::slotSelectResult()	TU40
DetailedResultView::slotShowAll()	TU40
MainWindowController::slotShowDatasets()	TU40
MainWindowController::slotShowGroups()	TU40
MainWindowController::slotShowHelp()	TU40
MainWindowController::slotShowProtocols()	TU40
DetailedResultView::slotShowResults()	TU40
MainWindowController::slotShowSubjects()	TU40
AnalysisController::slotStartAnalysis()	TU40
MainWindowController::slotStartAnalysis()	TU40
DetailedResultView::slotSubjetOrder()	TU40
NewProtocolController::slotTypeChanged()	TU40
NewGroupController::slotTypeSelection()	TU40
ProtocolsControlle::slotViewResult()	TU40
MainWindowController::slotWelcome()	TU40
StandardDeviationFeature::StandardDeviationFeature()	TU12 TU13
APanel::startAnalysis()	
MainWindow::static getMainWindow()	
SubjectDAO::SubjectDAO()	TU6

SubjectsController::SubjectsController()	
SubjectDAO::subjectsOfGroup()	TU6
SubjectsView::SubjectsView()	
NewProtocolView::typeChange()	
NewGroupView::typeSelection	
NewGroupView::updateProxyModel	
VideoReader::VideoReader()	TU3
FeatureParams::visibleBox()	
WelcomeController::WelcomeController()	
WelcomeView::WelcomeView()	
Log::writeLog()	TU36
MainWindow::writeStatusBar()	

Tabella 5: Tracciamento Modulo-Test

A.4 Codice Matlab

Il proponente fornisce alcuni file di codice matlab, che mostrano gli algoritmi di ogni singola feature extractor$_G$ applicata su matrici matlab che rappresentano immagini bidimensionali.

A.4.1 Standard Deviation Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 3D per la feature di Standard Deviation:

```
% std returns the standard deviation value
% within the window (in this example 3D window)
function std=Feat_p1_std(data,size)
% INPUT:
% data: window data
% size: window size
nvol = size^3;
val=Feat_p1_mean(data,size);
ct = 0;
for r=1:size
        for c=1:size
                for s=1:size
                        ct=ct+(data(r,c,s)-val)^2;
                end
        end
end
std = sqrt(ct/nvol);
end
```

A.4.2 Skewness Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 3D per la feature Skewness:

```
% skewness returns the skewness
%within the window (in this example 3D window)
function skew=Feat_p1_skew(data,size)
% INPUT:
% data: window data
% size: window size
nvol = size^3;
val=Feat_p1_mean(data,size);
std=Feat_p1_std(data,size);
ct = 0;
for r=1:size
        for c=1:size
                for s=1:size
                        ct=ct+(data(r,c,s)-val)^3;
                end
        end
end
ct2 = ct/nvol;
skew = ct2/(std^3);
end
```

A.4.3 Mean Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 3D per la feature Mean:

```
% mean returns the mean value
%within the window (in this example 3D window)
function val=Feat_p1_mean(data,size)
% INPUT:
% data: window data
% size: window size
nvol = size^3;
ct = 0;
for r=1:size
        for c=1:size
                for s=1:size
                        ct=ct+data(r,c,s);
                end
        end
end
val = ct/nvol;
end
```

A.4.4 Kurtosis Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 3D per la feature Kurtosis:

```
% kurtosis returns the kurtosis
%within the window (in this example 3D window)
function kurt=Feat_p1_kurt(data,size)
% INPUT:
% data: window data
% size: window size
nvol = size^3;
val=Feat_p1_mean(data,size);
std=Feat_p1_std(data,size);
ct = 0;
for r=1:size
        for c=1:size
                for s=1:size
                        ct=ct+(data(r,c,s)-val)^4;
                end
        end
end
ct2 = ct/nvol;
kurt = ct2/(std^4)-3;
end
```

A.4.5 Contrast Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Contrast:

```
% contrast returns the contrast within the window
%evaluated from GLCM (sempre 2D)
function contr=Feat_p1_contr(data,size)
% INPUT:
```

```
% data: GLCM
% size: GLCM size
ct = 0;
for i=1:size
        for j=1:size
                ct = ct+data(i,j)*(i-j)^2;
        end
end
contr = ct;
end
```

A.4.6 Correlation Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Correlation:

```
% corr returns the correlation within the window evaluated from GLCM (sempre 2D)
function corr=Feat_p1_corr(data,size)
% INPUT:
% data: GLCM
% size: GLCM size
% FORMULA:
% Mi and Mj are vectors
Mi = meanIndexI(data,size);
Si = stdIndexI(data,size,Mi);

Mj = meanIndexJ(data,size);
Sj = stdIndexJ(data,size,Mj);

ct2 = 0;
for i=1:size
        for j = 1:size
                ct1 = data(i,j) * (i-Mi(i))*(j-Mj(j));
                ct2 = ct2 + ct1 / (Si(i) * Sj(j));
        end
end

corr = ct2;

%-------------------------------------------------------------
function S = stdIndexI(data,size,m)

ct = 0;
for i=1:size
        for j=1:size
                ct = ct + data(i,j)*(i-m(i))^2;
        end
        S(i) = sqrt(ct);
        ct = 0;
end

%-------------------------------------------------------------
function M = meanIndexI(data,size)

ct = 0;
for i=1:size
        for j=1:size
```

```
                ct = ct+ i * data(i,j);
        end
        M(i) = ct;
        ct = 0;
end

end%-------------------------------------------------------

function S = stdIndexJ(data,size,m)

ct = 0;
for j=1:size
        for i=1:size
                ct = ct + data(i,j)*(j-m(j))^2;
        end
        S(j) = sqrt(ct);
        ct = 0;
end

%-------------------------------------------------------
function M = meanIndexJ(data, size)

ct = 0;
for j=1:size
        for i=1:size
                ct = ct + j*data(i,j);
        end
        M(j) = ct;
        ct = 0;
end

end
```

A.4.7 Energy Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Energy:

```
% energy returns the energy within
%the window evaluated from GLCM (sempre 2D)
function energy=Feat_p1_energy(data,size)
% INPUT:
% data: GLCM
% size: GLCM size
ct = 0;
for i=1:size
        for j=1:size
                ct = ct+data(i,j)^2;
        end
end
energy = ct;
end
```

A.4.8 Entropy Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Entropy:

```
% entropy returns the entropy within
%the window evaluated from GLCM (sempre 2D)
function entropy=Feat_p1_entropy(data,size)
% INPUT:
% data: GLCM
% size: GLCM size
ct = 0;
for i=1:size
        for j=1:size
                ct = ct+data(i,j)*ln(data(i,j));
        end
end
entropy = -ct;
end
```

A.4.9 Homogeneity Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Homogeneity:

```
% homogeneity returns the homogeneity within
% the window evaluated from GLCM (sempre 2D)
function homo=Feat_p1_homo(data,size)
% INPUT:
% data: GLCM
% size: GLCM size
ct = 0;
for i=1:size
        for j=1:size
                ct = ct+data(i,j)*(1/(1+abs(i-j)));
        end
end
homo = ct;
end
```

A.4.10 Area Under the Curve Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Area Under the Curve in un video 2D:

```
% AUC computes the area under the curve in an interval of frames
function AUC=Clust_par_auc(data,nrow,posin,posend)
% INPUT:
% data: data matrix
% nrow: nr of curves (nr of voxels)
% posin: index of first frame
% posend: index of last frame
AUC=zeros(nrow,1);
for i=1:nrow
    AUC(i)=trapz([posin:posend],data(i,posin:posend),2);
end
end
```

A.4.11 Maximum Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Maximum in un video 2D:

```
% max searches the maximum value in an interval of frames
function maxval=Clust_par_max(data,nrow,posin,posend)
% INPUT:
% data: data matrix
% nrow: nr of curves (nr of voxels)
% posin: index of first frame
% posend: index of last frame
maxval=zeros(nrow,1);
for i=1:nrow
maxval(i)=max(data(i,posin:posend));
end
end
```

A.4.12 Minimum Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Minimum in un video 2D:

```
% min searches the min value in an interval of frames
function val=Clust_par_val(data,nrow,posin,posend)
% INPUT:
% data: data matrix
% nrow: nr of curves (nr of voxels)
% posin: index of first frame
% posend: index of last frame
minval=zeros(nrow,1);
for i=1:nrow
minval(i)=min(data(i,posin:posend));
end
end
```

A.4.13 Mean Dynamic Feature

Questa funzione matlab rappresenta l'algoritmo per calcolare il valore di un pixel 2D per la feature Mean Dynamic in un video 2D:

```
% mean returns the mean value in an interval of frames
function val=Clust_par_mean(data,nrow,posin,posend)
% INPUT:
% data: data matrix
% nrow: nr of curves (nr of voxels)
% posin: index of first frame
% posend: index of last frame
meanval=zeros(nrow,1);
for i=1:nrow
meanval(i)=average(data(i,posin:posend));
end
end
```