ExamMgr Plugin Interface Reference Manual

0.7

Generated by Doxygen 1.4.3

Tue Dec 20 11:57:02 2005

Contents

Chapter 1

ExamMgr Plugin Interface Directory Hierarchy

1.1 ExamMgr Plugin Interface Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

ExamMgr Plugin Interface Hierarchical Index

2.1 ExamMgr Plugin Interface Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

ExamMgr Plugin Interface Class Index

3.1 ExamMgr Plugin Interface Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

ExamMgr Plugin Interface Page Index

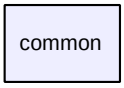
4.1 ExamMgr Plugin Interface Related Pages

Here is a list of all related documentation pages:

Chapter 5

ExamMgr Plugin Interface Directory Documentation

5.1 common/ Directory Reference



Files

- file **apphelper.h**
- file **configaccess.h**
- file **coursecount.h**
- file **courses.h**
- file **degrees.h**
- file **examexportplugin.h**
- file **examimportplugin.h**
- file **exams.h**
- file **importexamselector.h**
- file **klausur.h**
- file **notenplugin.h**
- file **pluginbase.h**
- file **pluginbase_defines.h**
- file **pluginloader.h**
- file **program_version.h**
- file **projectaccess.h**
- file **projectaccessbackendplugin.h**
- file **Statistics.h**
- file **student.h**

Chapter 6

ExamMgr Plugin Interface Class Documentation

6.1 AppHelper Class Reference

`#include <apphelper.h>`

6.1.1 Detailed Description

Some useful helper functions.

Definition at line 18 of file apphelper.h.

Static Public Member Functions

- static QString stripFilenameFromPath (const QString &path)

  Returns the path without filename.

- static QString stripPathFromFileName (const QString &path)

  Returns the filename without path.

- static QString suffixFromFileName (const QString &filename)

  Returns the suffix from the filename without (!) leading ".

6.1.2 Member Function Documentation

6.1.2.1 static QString AppHelper::suffixFromFileName (const QString & *filename*) [inline, static]

Returns the suffix from the filename without (!) leading ".

"

Definition at line 44 of file apphelper.h.

The documentation for this class was generated from the following file:

- apphelper.h

6.2 ConfigAccess Class Reference

`#include <configaccess.h>`

6.2.1 Detailed Description

Generic data store.

This class stores any data which may be interesting to save. Loading and saving of the data is done automatically. The information contained is available for the application and for every plugin or component. Thus it might be used to for simple commmunication between them! This class behaves like the singleton pattern! Use config() to get the instance.

Definition at line 34 of file configaccess.h.

Public Types

Non constum idenfiers for configuration settings.

*Custom values may be used if they are greater than "*_CUSTOM_ENTRIES"*

- enum String_Configs {
  PATH_TO_TEMPLATES = 0, **PATH_TO_DPA_OUTPUTDIR**, **PATH_TO_AUSHANG_-DIR**, **PATH_TO_PROJECT_FILES**,
  PATH_TO_EXCEL_FILES, **PATH_TO_CSV_FILES**, **PATH_TO_LATEX_FILES**, **PATH_-TO_HIS_OUTPUT_DIR**,
  LAST_USED_PLUGIN_NAME, **LAST_EMPHASIS**, **S_CUSTOM_ENTRIES** = 100 }
- enum String_List_Configs { **FILE_HISTORY** = 0, **PLUGIN_NAMES**, **SL_CUSTOM_-ENTRIES** = 100 }
- enum String_Map_Configs { **COURSES** = 0, **DEGREES**, **MARK_INTERVALS**, **SM_-CUSTOM_ENTRIES** = 100 }
- enum StringList_Map_Configs { **EXAMS** = 0, **SLM_CUSTOM_ENTRIES** = 100 }

Public Member Functions

- void **loadNonDefaultConfig** ()
- bool isLoadedSuccessfully ()

  Returns whether the config file could be loaded successfully.

Access Strings

Returns or changes the string which is identified by "identifier". "identifier" is defined by String_Configs or a custom number..

See also:
String_Configs

- QString & **stringData** (int identifier)
- QString & **operator[ ]** (int identifier)
- void **setStringData** (int identifier, const QString &value)

Access string lists.

"identifier" is defined by String_List_Configs

See also:
String_List_Configs

- QStringList & **stringListData** (int identifier)
- void **setStringListData** (int identifier, const QStringList &value)

Accesses data of type QMap<QString, QString>

Typically it is used for table information. The Identifier defines the data, as defined by String_Map_Configs

See also:
String_Map_Configs

- QMap< QString, QString > & **stringMap** (int identifier)
- void **setStringMap** (int identifier, const QMap< QString, QString > &data)

Access StringList-Maps of type QMap<QString, QStringList>

"identifier" is defined by StringList_Map_Configs

See also:
StringList_Map_Configs

- QMap< QString, QStringList > & **stringListMap** (int identifier)
- void **setStringListMap** (int identifier, const QMap< QString, QStringList > &data)

Access plugin configration by using the plugin name as identifier

- QStringList pluginNamesStored ()
- QMap< int, QString > & pluginConfig (const QString &identifier)
- void setPluginConfig (const QString &identifier, const QMap< int, QString > &data)
  Set the configuration for the plugin, defined by "identifier".
- QMap< int, QString > & pluginFieldNames (const QString &identifier)
  Function to request the field names of the configuration.
- void setPluginFieldNames (const QString &identifier, const QMap< int, QString > &data)
  Saves the fieldnames for the given identifier.
- void removePluginData (const QString &identifier)
  Removes the complete plugin data (the configuration and the field names).

Static Public Member Functions

- static QString pathAndNameOfDefaultConfigDir ()
  Returns the name including the path of the configuration file;.
- static ConfigAccess & **config** ()

6.2.2 Member Function Documentation

6.2.2.1 QMap<int, QString>& ConfigAccess::pluginConfig (const QString & *identifier*)

Returns:
The configuration of the plugin, identified by "identifier"

6.2.2.2 QMap<int, QString>& ConfigAccess::pluginFieldNames (const QString & *identifier*)

Function to request the field names of the configuration.

Fieldnames are the human readable version of the field numbers returned by pluginConfig()

Parameters:
identifier The "name" of the plugin, returned by pluginNamesStored()

Returns:
The map to request the human readable name of the field with given number

6.2.2.3 QStringList ConfigAccess::pluginNamesStored ()

Returns:
List of all plugin names, which stored its information. These strings may be used as idenfiers for requesting stored configuration.

See also:
pluginConfig()

6.2.2.4 void ConfigAccess::removePluginData (const QString & *identifier*)

Removes the complete plugin data (the configuration and the field names).

Parameters:
identifier "Name" of the plugin to remove its data.

6.2.2.5 void ConfigAccess::setPluginConfig (const QString & *identifier*, const QMap< int, QString > & *data*)

Set the configuration for the plugin, defined by "identifier".

Parameters:
identifier The "name" of the plugin, returned by pluginNamesStored()
data The data ..

6.2.2.6 void ConfigAccess::setPluginFieldNames (const QString & *identifier*, const QMap< int, QString > & *data*)

Saves the fieldnames for the given identifier.

Parameters:
identifier The "name" of the plugin, returned by pluginNamesStored()
data The data ..

The documentation for this class was generated from the following file:

- configaccess.h

6.3 CourseCount Class Reference

`#include <coursecount.h>`

6.3.1 Detailed Description

Simple container class to store course statistics.

Definition at line 22 of file coursecount.h.

Public Member Functions

- **CourseCount** (const QString &courseNumber, const QString &exam_nr, const QString &degreeNr, uint count)
- **CourseCount** (const CourseCount &copy)
- CourseCount & **operator=** (const CourseCount &copy)
- QString **courseNumber** () const
- QString **examNr** () const
- QString **degreeNr** () const
- uint **courseCount** () const
- void **inc** ()
- QStringList **members** () const
- void **addMember** (const QString &mbr)
- void **setMembers** (const QStringList &members)

The documentation for this class was generated from the following file:

- coursecount.h

6.4 Courses Class Reference

`#include <courses.h>`

6.4.1 Detailed Description

Container class to store courses.

This class supports serialization and converting of its internal data from/to a map.

Definition at line 28 of file courses.h.

Public Member Functions

- Courses ()
- **Courses** (const Courses &copy)
- void **addCourse** (const QString &number, const QString &name)
- void **remCourseByNumber** (const QString &number)
- QStringList **courseNumbers** () const
- QStringList **courseNames** () const
- const QString & **courseNameByNumber** (const QString &number) const
- QString **courseNumberByName** (const QString &name) const

Serialization

The content of the class is written into or created out of a stream.

- ∗void **serialize** (QDataStream &stream) const
- bool **deserialize** (QDataStream &stream)

Import/Export

Converting the content of the class from/to a map. This methods may be used to store its content into the class ConfigAccess

See also:
ConfigAccess

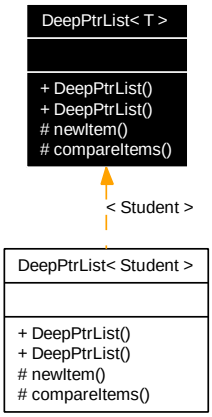
- ∗QMap< QString, QString > **toMap** () const
- void **fromMap** (const QMap< QString, QString > &map)

The documentation for this class was generated from the following file:

- courses.h

6.5 DeepPtrList< T > Class Template Reference

Inheritance diagram for DeepPtrList< T >:



6.5.1 Detailed Description

template<class T> class DeepPtrList< T >

Definition at line 45 of file klausur.h.

Public Member Functions

- DeepPtrList ()
- DeepPtrList (const T &copy)

Protected Member Functions

- QPtrCollectionType newItem (QPtrCollectionType pItem)
- int compareItems (QPtrCollectionType item1, QPtrCollectionType item2)

The documentation for this class was generated from the following file:

- klausur.h

6.6 Degrees Class Reference

`#include <degrees.h>`

6.6.1 Detailed Description

Container class to store degrees.

This class supports serialization and converting of its internal data from/to a map.

Definition at line 28 of file degrees.h.

Public Member Functions

- Degrees ()
- **Degrees** (const Degrees &copy)
- void **addDegree** (const QString &number, const QString &name)
- void **remDegreeByNumber** (const QString &number)
- QStringList **degreeNumbers** () const
- QStringList **degreeNames** () const
- const QString & **degreeNameByNumber** (const QString &number) const
- QString **degreeNumberByName** (const QString &name) const

Serialization

The content of the class is written into or created out of a stream.

- ∗void **serialize** (QDataStream &stream) const
- bool **deserialize** (QDataStream &stream)

Import/Export

Converting the content of the class from/to a map. This methods may be used to store its content into the class ConfigAccess

See also:
ConfigAccess

- ∗QMap< QString, QString > **toMap** () const
- void **fromMap** (const QMap< QString, QString > &map)

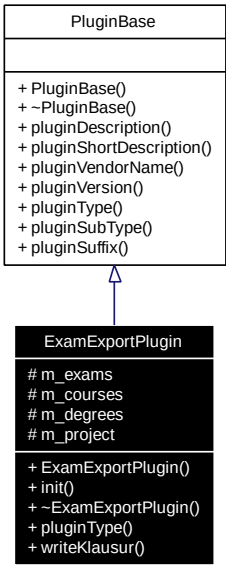
The documentation for this class was generated from the following file:

- degrees.h

6.7 ExamExportPlugin Class Reference

`#include <examexportplugin.h>`

Inheritance diagram for ExamExportPlugin:



6.7.1 Detailed Description

Abstract base class to export data.

Definition at line 32 of file examexportplugin.h.

Public Types

- enum PluginTypes { **NOTENPLUGIN** = 0, **PROJECT_ACCESS**, **EXPORT_DATA**, **IMPORT_-DATA** }

Public Member Functions

- virtual bool init (const ProjectAccess &project)=0

  Init plugin.

- virtual ~ExamExportPlugin ()
- virtual PluginBase::PluginTypes pluginType ()

  Type of plugin.

- virtual bool writeKlausur ()=0

  Write Klausur in special format.

- virtual QString pluginDescription ()=0

  Long description about the plugin.

- virtual QString pluginShortDescription ()=0

  Very short description about the plugin (one line, maximum 20 Chars).

- virtual QString pluginVendorName ()=0

  Vendor name.

- virtual QString pluginVersion ()=0

  Version.

- virtual int pluginSubType ()

  Optional subtype.

Static Public Member Functions

- static QString pluginSuffix ()

  Returns the string that suffixes plugins (like "dll" for Windows and "dylib" for MacOSX).

Protected Attributes

- Exams m_exams
- Courses m_courses
- Degrees m_degrees
- ProjectAccess m_project

6.7.2 Member Function Documentation

6.7.2.1 virtual bool ExamExportPlugin::init (const ProjectAccess & *project*) [pure virtual]

Init plugin.

Returns:
false if initialization failed or is cancelled (e.g. the user pressed any cancel button)

6.7.2.2 virtual QString PluginBase::pluginShortDescription () [pure virtual, inherited]

Very short description about the plugin (one line, maximum 20 Chars).

This description may have a special meaning, depending to the plugin (see ProjectAccessBackendPlugin for an example).

6.7.2.3 virtual int PluginBase::pluginSubType () [virtual, inherited]

Optional subtype.

Meaning is plugin specific..

Returns:
Always 0 if nothing is defined.

6.7.2.4 virtual PluginBase::PluginTypes ExamExportPlugin::pluginType () [inline, virtual]

Type of plugin.

See also:
PluginTypes

Implements PluginBase.

Definition at line 45 of file examexportplugin.h.

6.7.2.5 virtual QString PluginBase::pluginVersion () [pure virtual, inherited]

Version.

For instance "V1.0"

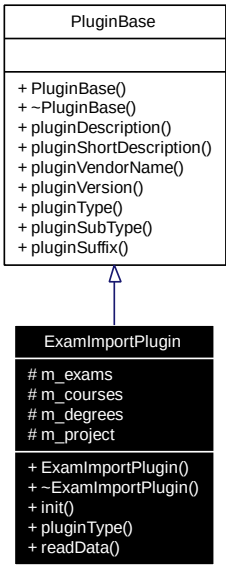
The documentation for this class was generated from the following file:

- examexportplugin.h

6.8 ExamImportPlugin Class Reference

`#include <examimportplugin.h>`

Inheritance diagram for ExamImportPlugin:



6.8.1 Detailed Description

Abstract base class to import data.

Definition at line 35 of file examimportplugin.h.

Public Types

- enum PluginTypes { **NOTENPLUGIN** = 0, **PROJECT_ACCESS**, **EXPORT_DATA**, **IMPORT_-DATA** }

Public Member Functions

- virtual ~ExamImportPlugin ()
- virtual bool init (const ProjectAccess &project)=0

  Init plugin.

- virtual PluginBase::PluginTypes pluginType ()

  Type of plugin.

- virtual bool readData ()=0

  Read data which is stored in the current project.

- virtual QString pluginDescription ()=0

  Long description about the plugin.

- virtual QString pluginShortDescription ()=0

  Very short description about the plugin (one line, maximum 20 Chars).

- virtual QString pluginVendorName ()=0

  Vendor name.

- virtual QString pluginVersion ()=0

  Version.

- virtual int pluginSubType ()

  Optional subtype.

Static Public Member Functions

- static QString pluginSuffix ()

  Returns the string that suffixes plugins (like "dll" for Windows and "dylib" for MacOSX).

Protected Attributes

- Exams m_exams
- Courses m_courses
- Degrees m_degrees
- ProjectAccess m_project

6.8.2 Member Function Documentation

6.8.2.1 virtual bool ExamImportPlugin::init (const ProjectAccess & *project*) [pure virtual]

Init plugin.

Returns:
false if initialization failed or is cancelled (e.g. the user pressed any cancel button)

6.8.2.2 virtual QString PluginBase::pluginShortDescription () [pure virtual, inherited]

Very short description about the plugin (one line, maximum 20 Chars).

This description may have a special meaning, depending to the plugin (see ProjectAccessBackendPlugin for an example).

6.8.2.3 virtual int PluginBase::pluginSubType () [virtual, inherited]

Optional subtype.

Meaning is plugin specific..

Returns:
Always 0 if nothing is defined.

6.8.2.4 virtual PluginBase::PluginTypes ExamImportPlugin::pluginType () [inline, virtual]

Type of plugin.

See also:
PluginTypes

Implements PluginBase.

Definition at line 47 of file examimportplugin.h.

6.8.2.5 virtual QString PluginBase::pluginVersion () [pure virtual, inherited]

Version.

For instance "V1.0"

6.8.2.6 virtual bool ExamImportPlugin::readData () [pure virtual]

Read data which is stored in the current project.

Returns:
false if something went wrong.

The documentation for this class was generated from the following file:

- examimportplugin.h

6.9 Exams Class Reference

`#include <exams.h>`

6.9.1 Detailed Description

Container class to store exams.

This class supports serialization and converting of its internal data from/to a map.

Definition at line 27 of file exams.h.

Public Member Functions

- Exams ()
- **Exams** (const Exams &copy)
- void **addExam** (const QString &examName)
- void **addExamNumber** (const QString &examName, const QString &number)
- bool **removeExam** (const QString &examName)
- bool **removeExamNumber** (const QString &eExamName, const QString &number)
- bool **renameExam** (const QString &oldName, const QString &newName)
- bool **renameExamNumber** (const QString examName, const QString &oldNumber, const QString &newNumber)
- QStringList **examNames** () const
- QStringList **examNumbers** (const QString &examName) const
- QString **examNameByNumber** (const QString &examNumber) const

Serialization

The content of the class is written into or created out of a stream.

- ∗void **serialize** (QDataStream &stream) const
- bool **deserialize** (QDataStream &stream)

Import/Export

Converting the content of the class from/to a map. This methods may be used to store its content into the class ConfigAccess

See also:
ConfigAccess

- ∗QMap< QString, QStringList > **toMap** () const
- void **fromMap** (const QMap< QString, QStringList > &map)

The documentation for this class was generated from the following file:

- exams.h

6.10 ImportExamSelector Class Reference

6.10.1 Detailed Description

Definition at line 24 of file importexamselector.h.

Public Member Functions

- **ImportExamSelector** (QWidget ∗parent=0, const char ∗name=0, bool modal=FALSE, WFlags fl=0)

Public Attributes

- QLabel ∗ textLabel1
- QComboBox ∗ m_pExamSelector
- QPushButton ∗ m_pOkButton

Protected Slots

- virtual void **languageChange** ()

Protected Attributes

- QGridLayout ∗ ImportExamSelectorLayout
- QHBoxLayout ∗ layout2

The documentation for this class was generated from the following file:

- importexamselector.h

6.11 Klausur Class Reference

`#include <klausur.h>`

6.11.1 Detailed Description

Container class to store exams.

This class supports serialization and converting of its internal data from/to a map.

Definition at line 74 of file klausur.h.

Public Types

- enum NonCustomTypes {
  IDENTIFIER, **DATE**, **ROOM**, **BUILDING**,
  PO_SEMESTER, **MAX_POINTS**, **NUM_TASKS**, **MIN_PASS**,
  MIN_BEST, **EMPHASIS** }
  Identifier numbers used in the non custom map, returned by nonCustomMap() and setNonCustomMap().

Public Member Functions

- **Klausur** (const Klausur &copy)
- Klausur & **operator=** (const Klausur &copy)
- void init ()
  Initialize this object.

- void **setBezeichnung** (const QString &v)
- void **setDatum** (const QString &v)
- void **setRaum** (const QString &v)
- void **setGebaeude** (const QString &v)
- void **setMembers** (const DeepPtrList< Student > &v)
- void **setPruefungsSemester** (const QString &v)
- void **setAnzAufgaben** (const QString &v)
- void **setMaxPoints** (const QString &v)
- void **setMinBestehen** (const QString &v)
- void **setMinBestnote** (const QString &v)
- void **setEmphasis** (const QString &v)
- void **changed** (bool changed)
- bool **isStudentAlreadyStored** (const QString &Matr_Nr)
- void **addMember** (const Student &v)
- void **removeMember** (const QString &Matr_Nr)
- void **replaceMember** (const Student &v, const QString &Matr_Nr)
- QString **bezeichnung** () const
- QString **datum** () const
- QString **raum** () const
- QString **gebaeude** () const
- DeepPtrList< Student > **members** () const

- QString **pruefungsSemester** () const
- QString **anzAufgaben** () const
- QString **maxPoints** () const
- QString **minBestehen** () const
- QString **minBestnote** () const
- QString **emphasis** () const
- bool **isChanged** () const
- Student & **memberByMatrNr** (const QString &matr_nr) const
- uint studCount () const

  Get Number of registered students.

- uint anwCount () const

  Get Number of students which wrote the exam.

- QString **customValueForKey** (const QString &key) const
- QValueList< Student > **courseMembers** (const QString &courseNumber)
- bool **isEmpty** () const
- bool **isNull** () const

Import/Export

Converting the content of the class from/to a map. The nonCustomMap() and setNonCustomMap() functions allow a direct access to the data which is used by the application itself and accessable by the values in NonCustomTypes. Customvalues are undefined attributes which may be stored but is not used by the application. Use customMap(), setCustomMap() and setCustomValueForKey() to access or load/store them.

- *QMap< int, QString > **nonCustomMap** () const
- QMap< QString, QString > **customMap** () const
- void **setNonCustomMap** (const QMap< int, QString > &map)
- void **setCustomMap** (const QMap< QString, QString > &v)
- void **setCustomValueForKey** (const QString &key, const QString &value)

Serialization

The content of the class is written into or created out of a stream.

- *void **serialize** (QDataStream &stream)
- void **deserialize** (QDataStream &stream)

Static Public Attributes

- static Klausur null

6.11.2 Member Function Documentation

6.11.2.1 void Klausur::init ()

Initialize this object.

After initialization, "isNull()" returns false;

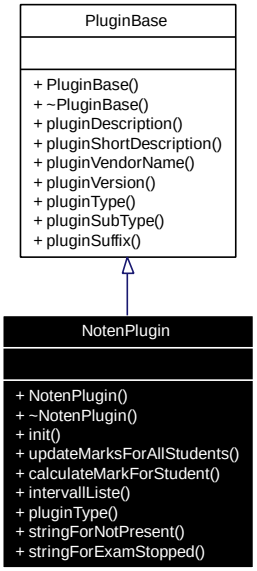
The documentation for this class was generated from the following file:

- klausur.h

6.12 NotenPlugin Class Reference

`#include <notenplugin.h>`

Inheritance diagram for NotenPlugin:



6.12.1 Detailed Description

Base class to calculate marks.

Use this for plugins

Definition at line 26 of file notenplugin.h.

Public Types

- enum PluginTypes { **NOTENPLUGIN** = 0, **PROJECT_ACCESS**, **EXPORT_DATA**, **IMPORT_-DATA** }

Public Member Functions

- virtual void init (Klausur *klausur, int emphasis)=0

  Init Plugin.

- virtual void updateMarksForAllStudents ()=0

  The marks of all students are adjusted in the "klausur" set by the constructor.

- virtual QString calculateMarkForStudent (const Student &student)=0

  Calculation of marks for a given student.

- virtual QArray< double > intervallListe (double prozentBestMark, double prozentWorstMark, uint numElements)=0

Creates an interval list.

- PluginBase::PluginTypes pluginType ()

  Plugin spezific method.

- virtual QString pluginDescription ()=0

  Long description about the plugin.

- virtual QString pluginShortDescription ()=0

  Very short description about the plugin (one line, maximum 20 Chars).

- virtual QString pluginVendorName ()=0

  Vendor name.

- virtual QString pluginVersion ()=0

  Version.

- virtual int pluginSubType ()

  Optional subtype.

Static Public Member Functions

- static const QString stringForNotPresent ()

  Official string for students not present in the exam.

- static const QString stringForExamStopped ()

  Official string for students started the exam but leave before end.

- static QString pluginSuffix ()

  Returns the string that suffixes plugins (like "dll" for Windows and "dylib" for MacOSX).

6.12.2 Member Function Documentation

6.12.2.1 virtual QString NotenPlugin::calculateMarkForStudent (const Student & *student*) [pure virtual]

Calculation of marks for a given student.

This function calculated the mark for the given student, regarding the settings in the class "Klausur".

6.12.2.2 virtual QString PluginBase::pluginShortDescription () [pure virtual, inherited]

Very short description about the plugin (one line, maximum 20 Chars).

This description may have a special meaning, depending to the plugin (see ProjectAccessBackendPlugin for an example).

6.12.2.3 virtual int PluginBase::pluginSubType () [virtual, inherited]

Optional subtype.

Meaning is plugin specific..

Returns:
Always 0 if nothing is defined.

6.12.2.4 PluginBase::PluginTypes NotenPlugin::pluginType () [virtual]

Plugin spezific method.

Return type of this plugin

Returns:
PluginBase::NOTENPLUGIN

Implements PluginBase.

6.12.2.5 virtual QString PluginBase::pluginVersion () [pure virtual, inherited]

Version.

For instance "V1.0"

6.12.2.6 static const QString NotenPlugin::stringForExamStopped () [inline, static]

Official string for students started the exam but leave before end.

Todo
Change this to a virtual method.

Plugins should implement it by themselve.

Definition at line 66 of file notenplugin.h.

6.12.2.7 static const QString NotenPlugin::stringForNotPresent () [inline, static]

Official string for students not present in the exam.

Todo
Change this to a virtual method.

Plugins should implement it by themselve.

Definition at line 58 of file notenplugin.h.

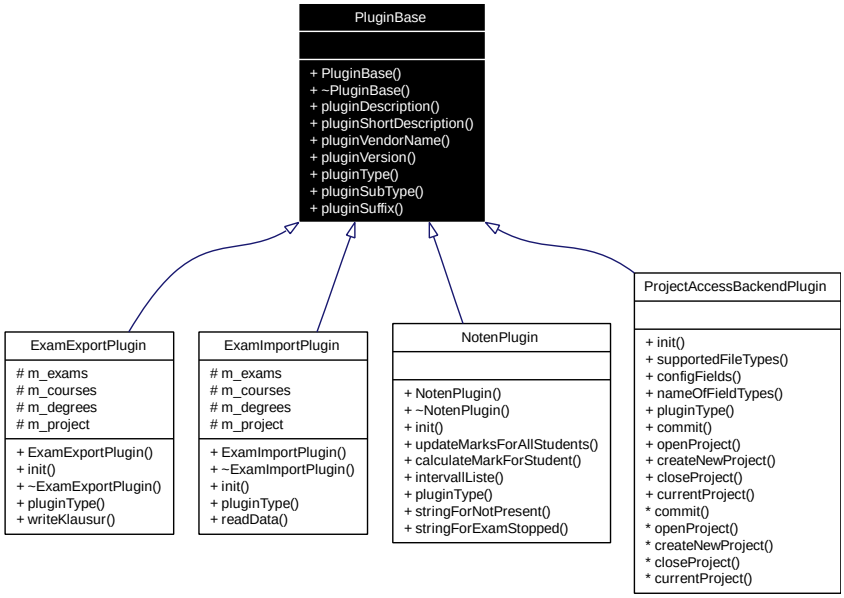
The documentation for this class was generated from the following file:

- notenplugin.h

6.13 PluginBase Class Reference

`#include <pluginbase.h>`

Inheritance diagram for PluginBase:



6.13.1 Detailed Description

The base class for all plugins.

It defines the standart interface for Plugins, loadable by the PluginLoader Class! Any change in the default interface has to cause a change in "G_PLUGIN_INTERFACE_VERSION", otherwise we could not detect old plugins or changes in the QT-Version.

How to use this plugin interface: Define a new abstract base class, defining the plugin interface which is inherited from class "PluginBase". Create a new subclass, which must implement the interface and has to use "Q_EXPORT_PLUGIN(<PLUGINCLASS>)" to export its symbols. The pluginloader should be able to load the plugin and to return an object pointer.

Definition at line 36 of file pluginbase.h.

Public Types

- enum PluginTypes { **NOTENPLUGIN** = 0, **PROJECT_ACCESS**, **EXPORT_DATA**, **IMPORT_-DATA** }

Public Member Functions

- virtual QString pluginDescription ()=0

  Long description about the plugin.

- virtual QString pluginShortDescription ()=0

Very short description about the plugin (one line, maximum 20 Chars).

- virtual QString pluginVendorName ()=0

  Vendor name.

- virtual QString pluginVersion ()=0

  Version.

- virtual PluginBase::PluginTypes pluginType ()=0

  Type of plugin.

- virtual int pluginSubType ()

  Optional subtype.

Static Public Member Functions

- static QString pluginSuffix ()

  Returns the string that suffixes plugins (like "dll" for Windows and "dylib" for MacOSX).

6.13.2 Member Function Documentation

6.13.2.1 virtual QString PluginBase::pluginShortDescription () [pure virtual]

Very short description about the plugin (one line, maximum 20 Chars).

This description may have a special meaning, depending to the plugin (see ProjectAccessBackendPlugin for an example).

6.13.2.2 virtual int PluginBase::pluginSubType () [virtual]

Optional subtype.

Meaning is plugin specific..

Returns:
Always 0 if nothing is defined.

6.13.2.3 virtual PluginBase::PluginTypes PluginBase::pluginType () [pure virtual]

Type of plugin.

See also:
PluginTypes

Implemented in ExamExportPlugin, ExamImportPlugin, NotenPlugin, and ProjectAccessBackendPlugin.

6.13.2.4 virtual QString PluginBase::pluginVersion () [pure virtual]

Version.

For instance "V1.0"

The documentation for this class was generated from the following file:

- pluginbase.h

6.14 PluginLoader Class Reference

`#include <pluginloader.h>`

6.14.1 Detailed Description

Platform independent loading of plugins.

The base of the plugin infrastructure. Everything which wants to load plugins uses it.

Definition at line 23 of file pluginloader.h.

Public Member Functions

- **PluginLoader** (const QString &plugin_dir=QString::null)
- QString defaultPluginName ()

  Returns the name of the plugin set to default.

- QString currentLoadedPluginName ()

  Returns the filename with path (!) of the plugin, which is currently loaded!

- void setPluginDir (const QString &plugin_dir)

  Change the default plugin directory (directory of the application).

- void setDefaultPluginName (const QString &plugin_nmae)

  Sets the plugin name, which is used in the future as default.

- QValueList< PluginDescriptor > listOfAvailablePlugins (PluginBase::PluginTypes type, int subtype=0) const

  Return list of available plugins with given type.

- template<class T> T * plugin (PluginBase::PluginTypes type, const QString &plugin_name=QString::null, QWidget *pParent=NULL)

  Returns a pointer to an object of the class contained in the plugin.

Classes

- struct PluginDescriptor

  Something like the fingerprint of a plugin.

6.14.2 Member Function Documentation

6.14.2.1 QValueList<PluginDescriptor> PluginLoader::listOfAvailablePlugins (PluginBase::PluginTypes *type*, int *subtype* = 0) const

Return list of available plugins with given type.

This function will not change the internal state of this class!

6.14.2.2 template<class T> T * PluginLoader::plugin (PluginBase::PluginTypes *type*, const QString & *plugin_name* = QString::null, QWidget * *pParent* = NULL)

Returns a pointer to an object of the class contained in the plugin.

Parameters:

type Defines, what kind of plugin should be loaded as defined in PluginBase::PluginTypes

plugin_name The filename of the plugin. If no name is given, the default plugin of the given type is loaded. The default plugin is found by the following name convention: default_<type>.

pParent If your plugin is a graphical component, it is possible to set the pointer to the parent widget.

Definition at line 84 of file pluginloader.h.

The documentation for this class was generated from the following file:

- pluginloader.h

6.15 PluginLoader::PluginDescriptor Struct Reference

`#include <pluginloader.h>`

6.15.1 Detailed Description

Something like the fingerprint of a plugin.

Is created by listOfAvailablePlugins()

Definition at line 27 of file pluginloader.h.

Public Attributes

- QString pluginName
- QString pluginDescriptionShort
- QString pluginDescription
- QString pluginVendor
- QString pluginVersion
- int pluginSubType

The documentation for this struct was generated from the following file:

- pluginloader.h

6.16 ProjectAccess Class Reference

`#include <projectaccess.h>`

6.16.1 Detailed Description

Class to access project data.

This class uses explicit data sharing! If you copy this object, you will only do a shallow copy. Thus, there still exists one project after a copy operation and changing the project content in one object will change it in the other, too. A real (deep) copy is not supported, as a project should not be opened more than once!

The project access is realized via a backend plugin which is loaded by the loadBackend∗ methods (bridge pattern). Using the signal projectChanged(), all connected observer objects will be notified if something changed (Observer-Pattern).

Definition at line 33 of file projectaccess.h.

Public Types

- enum BackendTypes { **NOT_SPECIFIED** = 0, FILE_ACCESS, ADVANCED_ACCESS }

  List of available backend types.

- enum ConfigTypes { FILENAME = 0, PASSWORD, **CUSTOM_CONFIG_TYPES** = 100 }

  List of predefined config types.

Signals

- void projectChanged ()

  Signal occures if the state of the project or its data was changed!

Public Member Functions

- ProjectAccess (const ProjectAccess &copy)

  Shallow copy constructor of the instance.

- ProjectAccess & operator= (const ProjectAccess &copy)

  Shallow copy of the instance.

- bool loadBackendBySuffix (const QString &suffix)

  Load a backend plugin which is compatible to the given suffix.

- bool loadBackendByFilename (const QString &filename)

  Load the backend plugin with the given filename.

- BackendTypes backendType ()

  Returns the type of the loaded backend plugin.

- bool init (const QMap< int, QString > accessinfo)

  Initializer.

- QMap< int, QString > configFields ()

  Returns a map with all of the config information this backend needs to access the ressource.

- QMap< int, QString > nameOfFieldTypes ()

  To create a human readable configuration widget, use this translation map.

- QString shortDescriptionOfCurrentPlugin ()

  Returns the short description of the current loaded plugin.

- bool createNewProject ()

  Create a new project.

- bool openProject ()

  Open a project ressource.

- bool commit ()

  Committing changes.

- bool closeProject ()

  Close the current project.

Access functions

Access the data of the current project via this functions. If no valid project is opened, a reference to a Klausur is returned which is "null".

Returns:
Reference to the current project data or Klausur::null

- ∗Klausur & **current** () const
- const Klausur & **constCurrent** () const

6.16.2 Member Enumeration Documentation

6.16.2.1 enum ProjectAccess::BackendTypes

List of available backend types.

Enumerator:

FILE_ACCESS A backend plugin which just accesses file based project files and therefore just need a filename to configure.

ADVANCED_ACCESS A backend plugin which needs various information (url, username, password) to access the project data successfully.

Definition at line 41 of file projectaccess.h.

6.16.2.2 enum ProjectAccess::ConfigTypes

List of predefined config types.

These special types are handled especially. It is highly recommended to use them, otherwise unexpected things may happen. Please use *CUSTOM_CONFIG_TYPES* to start with your own configuration types!

See also:
configFields()

Enumerator:

FILENAME Used for backend plugins of the type *FILE_ACCESS* to store the filename.

PASSWORD Used to store the filename.

This will not saved by the class ConfigAccess as long as we don't have a cypher-mechanism implemented!

Definition at line 55 of file projectaccess.h.

6.16.3 Member Function Documentation

6.16.3.1 BackendTypes ProjectAccess::backendType ()

Returns the type of the loaded backend plugin.

See also:
BackendTypes

6.16.3.2 bool ProjectAccess::closeProject ()

Close the current project.

It saves the data and frees allocated ressources.

Returns:
true if saving was successful

6.16.3.3 bool ProjectAccess::commit ()

Committing changes.

The exact behavior may change, depending to the backend plugin. On databases, a real commit is executed and the data is stored into the database. On file based backends, the data is stored into the file. In every case, the signal projectChanged() is emmitted!

Returns:
true if everything works well.

See also:
projectChanged()

6.16.3.4 QMap<int, QString> ProjectAccess::configFields ()

Returns a map with all of the config information this backend needs to access the ressource.

The human readable meaning (or name) of the type id's are returned by the function nameOfFieldTypes(). The types are defined by the enum ConfigTypes and by the plugin itself.

See also:
nameOfFieldTypes(), ConfigTypes

6.16.3.5 bool ProjectAccess::createNewProject ()

Create a new project.

The real behaviour of this function is controlled by the backend plugin! Thus, you have to load a plugin with *loadBackendBy∗()* and use the function *init()* to configure it, first!

Returns:
true if everything was ok.

See also:
loadBackendByFilename(), loadBackendBySuffix(), init()

6.16.3.6 bool ProjectAccess::init (const QMap< int, QString > *accessinfo*)

Initializer.

All information is stored in the given accessinfo to open the project. This method must be used before any project can be opened.

See also:
backendType(), configFields(), openProject()

6.16.3.7 bool ProjectAccess::loadBackendByFilename (const QString & *filename*)

Load the backend plugin with the given filename.

This commmand searches for the file in the default plugin directory (defined in the class "PluginLoader")

Parameters:
filename The name of the library file

Returns:
true if the backend was loaded successfully, false if not

6.16.3.8 bool ProjectAccess::loadBackendBySuffix (const QString & *suffix*)

Load a backend plugin which is compatible to the given suffix.

Parameters:
suffix The suffix of the file which should be loaded and defines the filetype (without leading ".")

Returns:
false if no backend was found

6.16.3.9 bool ProjectAccess::openProject ()

Open a project ressource.

This just works after loading a backend plugin with *loadBackendBy∗()* and after configuration with the function *init()*.

Returns:
true if everything was ok.

See also:
loadBackendByFilename(), loadBackendBySuffix(), init()

6.16.3.10 QString ProjectAccess::shortDescriptionOfCurrentPlugin ()

Returns the short description of the current loaded plugin.

If the backend is of type *FILE_ACCESS*, this string must contains a valid string for the file selection mechanism (e.g.: "∗.kl") Otherwise the string may contain anything custom..

See also:
backendType()

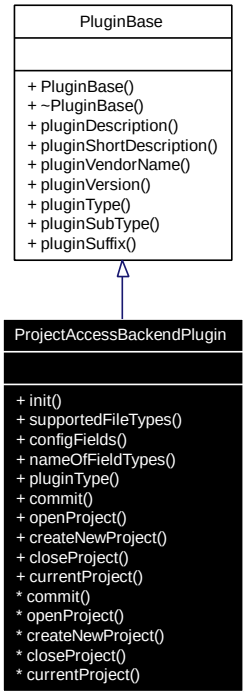
The documentation for this class was generated from the following file:

- projectaccess.h

6.17 ProjectAccessBackendPlugin Class Reference

`#include <projectaccessbackendplugin.h>`

Inheritance diagram for ProjectAccessBackendPlugin:



6.17.1 Detailed Description

Base class for a backend plugin.

Plugins inherented by this class implement how to access the project data. This may be a simple fileaccess implementation or any sophisticated database connectivity!

Definition at line 26 of file projectaccessbackendplugin.h.

Public Types

- enum PluginTypes { **NOTENPLUGIN** = 0, **PROJECT_ACCESS**, **EXPORT_DATA**, **IMPORT_-DATA** }

Public Member Functions

- virtual bool init (const QMap< int, QString > accessinfo)=0

  Initializer.

- virtual QStringList supportedFileTypes ()=0

  Returns a list of all FileTypes this backend supports.

- virtual QMap< int, QString > configFields ()=0

  Returns a map with the config information this backend needs to access the ressource.

- virtual QMap< int, QString > nameOfFieldTypes ()=0

  To create a human readable configuration widget, the fieldtypes used by this backend is translated to text.

- PluginBase::PluginTypes pluginType ()

  Type of plugin.

- virtual QString pluginDescription ()=0

  Long description about the plugin.

- virtual QString pluginShortDescription ()=0

  Very short description about the plugin (one line, maximum 20 Chars).

- virtual QString pluginVendorName ()=0

  Vendor name.

- virtual QString pluginVersion ()=0

  Version.

- virtual int pluginSubType ()

  Optional subtype.

These functions are accessed directy by ProjectAccess.

See also:
ProjectAccess

- virtual bool **commit** ()=0
- virtual bool **openProject** ()=0
- virtual bool **createNewProject** ()=0
- virtual bool **closeProject** ()=0
- virtual Klausur & **currentProject** ()=0

Static Public Member Functions

- static QString pluginSuffix ()

  Returns the string that suffixes plugins (like "dll" for Windows and "dylib" for MacOSX).

6.17.2 Member Function Documentation

6.17.2.1 virtual QMap<int, QString> ProjectAccessBackendPlugin::configFields () [pure virtual]

Returns a map with the config information this backend needs to access the ressource.

The human readable meaning (or name) of the type id's are returned by the function nameOfFieldTypes(). The types are defined by the enum ConfigTypes and by the plugin itself.

See also:
nameOfFieldTypes(), ConfigTypes

6.17.2.2 virtual bool ProjectAccessBackendPlugin::init (const QMap< int, QString > *accessinfo*) [pure virtual]

Initializer.

If more information is needed to access a ressource, all information is stored in the given accessinfo

See also:
backendType(), configFields()

6.17.2.3 virtual QString PluginBase::pluginShortDescription () [pure virtual, inherited]

Very short description about the plugin (one line, maximum 20 Chars).

This description may have a special meaning, depending to the plugin (see ProjectAccessBackendPlugin for an example).

6.17.2.4 virtual int PluginBase::pluginSubType () [virtual, inherited]

Optional subtype.

Meaning is plugin specific..

Returns:
Always 0 if nothing is defined.

6.17.2.5 PluginBase::PluginTypes ProjectAccessBackendPlugin::pluginType () [virtual]

Type of plugin.

See also:
PluginTypes

Implements PluginBase.

6.17.2.6 virtual QString PluginBase::pluginVersion () [pure virtual, inherited]

Version.

For instance "V1.0"

6.17.2.7 virtual QStringList ProjectAccessBackendPlugin::supportedFileTypes () [pure virtual]

Returns a list of all FileTypes this backend supports.

(This function returns an empty list if this backend is not of type FILE_ACCESS!)

The documentation for this class was generated from the following file:

- projectaccessbackendplugin.h

6.18 Statistics Class Reference

6.18.1 Detailed Description

Definition at line 24 of file Statistics.h.

Public Types

- enum markstat_elements {

  N_10 = 0, **N_13** = 1, **N_17** = 2, **N_20** = 3,

  N_23 = 4, **N_27** = 5, **N_30** = 6, **N_33** = 7,

  N_37 = 8, **N_40** = 9, **N_50_ERSTVERSUCH** = 10, **N_50_ZWEITVERSUCH** = 11,

  BEST_NACHWEIS = 12, **N_50_NACHWEIS** = 13, **NICHT_ERSCHIENEN** = 14, **ANWESEND** = 15,

  __END_MARKS_TABLE }

Public Member Functions

- **Statistics** (const ProjectAccess &project)
- QValueList< CourseCount > courseStatistics (bool groupByExamNr, bool groupByCourseNr, bool groupByDegreeNr) const

  Returns list of all courses (studiengÃ164nge) in this exam.

- void **updateStatistics** (const Courses &courses, const QMap< QString, bool > &visibleCourses, QArray< int > &marks_counter)

6.18.2 Member Function Documentation

6.18.2.1 QValueList<CourseCount> Statistics::courseStatistics (bool *groupByExamNr*, bool *groupByCourseNr*, bool *groupByDegreeNr*) const

Returns list of all courses (studiengÃ164nge) in this exam.

Every entry contains the number and members for every group.

Parameters:

groupByExamNr If true, the result is grouped by the exam number

groupByCourseNr If true, the result is grouped by the course number

groupByDegreeNr If true, the result is grouped by the degree number

The documentation for this class was generated from the following file:

- Statistics.h

6.19 StringListMapContainerFactory< T > Class Template Reference

`#include <configaccess.h>`

6.19.1 Detailed Description

template<class T> class StringListMapContainerFactory< T >

Create an instance of a data container out of the configuration in a simple way.

```
Exams exams = StringListMapContainerFactory<Exams>::create( ConfigAccess::EXAMS );
```

See also:
ConfigAccess

Definition at line 205 of file configaccess.h.

Static Public Member Functions

- static T create (int identifier)

The documentation for this class was generated from the following file:

- configaccess.h

6.20 StringMapContainerFactory< T > Class Template Reference

`#include <configaccess.h>`

6.20.1 Detailed Description

template<class T> class StringMapContainerFactory< T >

Create an instance of a data container out of the configuration in a simple way.

```
Courses courses = StringMapContainerFactory<Courses>::create( ConfigAccess::COURSES )
```

See also:
ConfigAccess

Definition at line 223 of file configaccess.h.

Static Public Member Functions

- static T create (int identifier)

The documentation for this class was generated from the following file:

- configaccess.h

6.21 Student Class Reference

`#include <student.h>`

6.21.1 Detailed Description

Container class to store students.

This class supports serialization and converting of its internal data from/to a map.

Definition at line 33 of file student.h.

Public Types

- enum NonCustomTypes {
  LAST_NAME, **FIRST_NAME**, **MATR_NR**, **DEGREE**,
  COURSE, **PO_NR**, **EXAM_NR**, **EXAM_STATE**,
  EXAM_DATE, **EXAM_TRIAL**, **EXAM_SEMESTER**, **RESULT**,
  PRESENT, **NACHWEIS**, **LAB_NR**, **ORAL_RESULT** }

Public Member Functions

- **Student** (const Student &copy)
- ~Student ()
- Student & **operator=** (const Student &copy)
- void **setNachName** (const QString &v)
- void **setVorName** (const QString &v)
- void **setMatrNr** (const QString &v)
- void **setAbschluss** (const QString &v)
- void **setStudienGang** (const QString &v)
- void **setDPO** (const QString &v)
- void **setPruefungsNr** (const QString &v)
- void **setPruefungsStatus** (const QString &v)
- void **setPruefungsDatum** (const QString &v)
- void **setPruefungsVersuch** (const QString &v)
- void **setSemester** (const QString &v)
- void **setNote** (const QString &v)
- void **setAnwesend** (bool v)
- void **toggleAnwesend** ()
- void **setNachweis** (bool v)
- void **setTeilPts** (uint index, QString &v)
- void **resizeTeilPts** (uint size)
- void **setMuendlNote** (const QString &v)
- void **changed** (bool changed)
- void **setTeilPunkteListe** (const QValueList< QString > &teilpunkte)
- QString **nachName** () const
- QString **vorName** () const
- QString **matrNr** () const
- QString **abschluss** () const

- QString **studiengang** () const
- QString **dpo** () const
- QString **pruefungsNr** () const
- QString **pruefungsStatus** () const
- QString **pruefungsDatum** () const
- QString **pruefungsVersuch** () const
- QString **semester** () const
- QString **note** () const
- bool **anwesend** () const
- bool **nachweis** () const
- QString **teilPts** (uint index) const
- QString **muendlNote** () const
- bool **isChanged** () const
- bool **isEmpty** () const
- QValueList< QString > **teilPunkteListe** () const
- float **sumExercisePoints** () const
- bool **matches** (const QString &s) const
- bool **operator<** (Student &stud)
- bool **operator==** (Student &stud)
- void **clear** ()

Import/Export

Converting the content of the class from/to a map. The nonCustomMap() and setNonCustomMap() functions allow a direct access to the data which is used by the application itself and accessable by the values in NonCustomTypes. Customvalues are undefined attributes which may be stored but is not used by the application. Use customMap(), setCustomMap() and setCustomValueForKey() to access or load/store them.

- ∗QMap< int, QString > **nonCustomMap** () const
- QMap< QString, QString > **customMap** () const
- QString **customValueForKey** (const QString &key) const
- void **setNonCustomMap** (const QMap< int, QString > &map)
- void **setCustomMap** (const QMap< QString, QString > &v)
- void **setCustomValueForKey** (const QString &key, const QString &value)

Serialization

The content of the class is written into or created out of a stream.

- ∗void **serialize** (QDataStream &stream)
- void **deserialize** (QDataStream &stream)

The documentation for this class was generated from the following file:

- student.h

Chapter 7

ExamMgr Plugin Interface Page Documentation

7.1 Todo List

Member NotenPlugin::stringForExamStopped() Change this to a virtual method.

Member NotenPlugin::stringForNotPresent() Change this to a virtual method.

Index