# PROYECTO FINAL ALSE

Generated by Doxygen 1.9.8

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

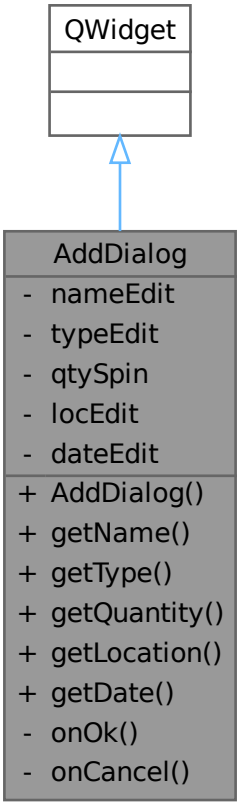
# Class Documentation

## 4.1 AddDialog Class Reference

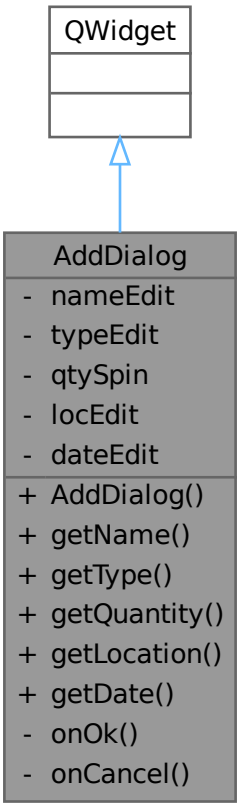
Diálogo modal para la creación y edición de componentes.

`#include <mainwindow.h>`

Inheritance diagram for AddDialog:



Collaboration diagram for AddDialog:




### Signals

- void accepted ()
- void cancelled ()

### Public Member Functions

- AddDialog (QWidget *parent=nullptr)

  *Constructor de la clase AddDialog.*
- QString getName () const

  *Obtiene el nombre ingresado por el usuario.*
- QString getType () const

  *Obtiene el tipo de componente ingresado.*
- int getQuantity () const

  *Obtiene la cantidad seleccionada.*
- QString getLocation () const

  *Obtiene la ubicación física del componente.*
- QDate getDate () const

  *Obtiene la fecha de adquisición seleccionada.*

### Private Slots

- void onOk ()

  *Slot llamado al presionar "Agregar/Aceptar".*
- void onCancel ()

  *Slot llamado al presionar "Cancelar".*

### Private Attributes

- QLineEdit * nameEdit
- QLineEdit * typeEdit
- QSpinBox * qtySpin
- QLineEdit * locEdit
- QDateEdit * dateEdit

## 4.1.1 Detailed Description

Diálogo modal para la creación y edición de componentes.

Esta clase hereda de QWidget (usado como ventana) y proporciona un formulario con validaciones básicas para ingresar Nombre, Tipo, Cantidad, Ubicación y Fecha.

Definition at line 28 of file mainwindow.h.

## 4.1.2 Constructor & Destructor Documentation

### 4.1.2.1 AddDialog()

```
AddDialog::AddDialog (
            QWidget * parent = nullptr )  [explicit]
```

Constructor de la clase AddDialog.

Configura el layout vertical, inicializa los widgets de entrada (QLineEdit, QSpinBox, etc.) y conecta los botones de Aceptar/Cancelar a sus respectivos slots.

**Parameters**

| | |
|---|---|
| *parent* | Widget padre (opcional), por defecto es nullptr. |

Definition at line 76 of file mainwindow.cpp.

```
00076                                   : QWidget(parent)
00077 {
00078     setWindowTitle("Agregar componente");
00079     QVBoxLayout *v = new QVBoxLayout(this);
00080
00081     // Campos de entrada
00082     nameEdit = new QLineEdit(); nameEdit->setPlaceholderText("Nombre");
00083     typeEdit = new QLineEdit(); typeEdit->setPlaceholderText("Tipo");
00084     qtySpin = new QSpinBox(); qtySpin->setRange(0, 1000000); qtySpin->setValue(1);
00085     locEdit = new QLineEdit(); locEdit->setPlaceholderText("Ubicación");
00086     dateEdit = new QDateEdit(QDate::currentDate()); dateEdit->setCalendarPopup(true);
00087
00088     QPushButton *btnOk = new QPushButton("Agregar");
```

```
00089     QPushButton *btnCancel = new QPushButton("Cancelar");
00090
00091     // Construcción del formulario
00092     v->addWidget(new QLabel("Nombre:"));     v->addWidget(nameEdit);
00093     v->addWidget(new QLabel("Tipo:"));       v->addWidget(typeEdit);
00094     v->addWidget(new QLabel("Cantidad:"));   v->addWidget(qtySpin);
00095     v->addWidget(new QLabel("Ubicación:")); v->addWidget(locEdit);
00096     v->addWidget(new QLabel("Fecha adquisición:")); v->addWidget(dateEdit);
00097
00098     // Botonera
00099     QHBoxLayout *h = new QHBoxLayout();
00100     h->addWidget(btnOk);
00101     h->addWidget(btnCancel);
00102     v->addLayout(h);
00103
00104     // Conexiones de señales
00105     connect(btnOk, &QPushButton::clicked, this, &AddDialog::onOk);
00106     connect(btnCancel, &QPushButton::clicked, this, &AddDialog::onCancel);
00107
00108     // Asignación de nombres de objeto para búsqueda dinámica (findChild)
00109     nameEdit->setObjectName("nameEdit");
00110     typeEdit->setObjectName("typeEdit");
00111     qtySpin->setObjectName("qtySpin");
00112     locEdit->setObjectName("locEdit");
00113     dateEdit->setObjectName("dateEdit");
00114 }
```

References dateEdit, locEdit, nameEdit, onCancel(), onOk(), qtySpin, and typeEdit.

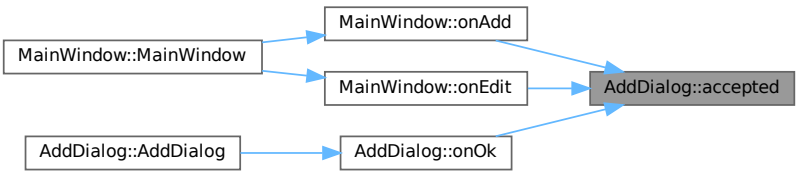
Here is the call graph for this function:


### 4.1.3 Member Function Documentation

#### 4.1.3.1 accepted

```
void AddDialog::accepted ( )  [signal]
```

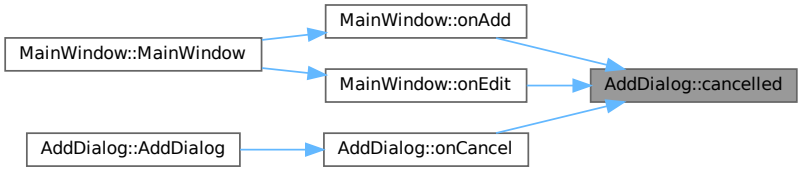
Here is the caller graph for this function:



#### 4.1.3.2 cancelled

```
void AddDialog::cancelled ( )  [signal]
```

Here is the caller graph for this function:




#### 4.1.3.3 getDate()

```
QDate AddDialog::getDate ( ) const
```

Obtiene la fecha de adquisición seleccionada.

**Returns**

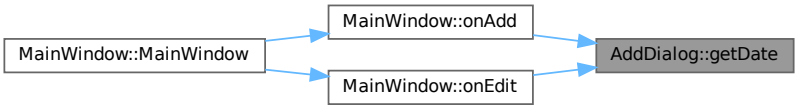
Objeto QDate con la fecha.

Definition at line 144 of file mainwindow.cpp.

```
00144 { return dateEdit->date(); }
```

References dateEdit.

Here is the caller graph for this function:



#### 4.1.3.4 getLocation()

```
QString AddDialog::getLocation ( ) const
```

Obtiene la ubicación física del componente.

**Returns**

QString con la ubicación.

Definition at line 138 of file mainwindow.cpp.

```
00138 { return locEdit->text().trimmed(); }
```

References locEdit.

Here is the caller graph for this function:


#### 4.1.3.5 getName()

```
QString AddDialog::getName ( ) const
```

Obtiene el nombre ingresado por el usuario.

**Returns**

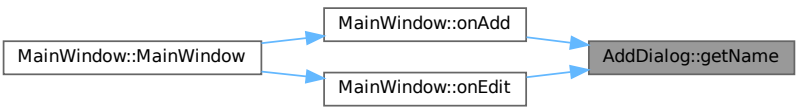
QString con el texto limpio de espacios al inicio y final (trimmed).

Definition at line 120 of file mainwindow.cpp.

```
00120 { return nameEdit->text().trimmed(); }
```

References nameEdit.

Here is the caller graph for this function:



#### 4.1.3.6 getQuantity()

```
int AddDialog::getQuantity ( ) const
```

Obtiene la cantidad seleccionada.

**Returns**

Entero con el valor del QSpinBox.

Definition at line 132 of file mainwindow.cpp.

```
00132 { return qtySpin->value(); }
```

References qtySpin.

Here is the caller graph for this function:


#### 4.1.3.7 getType()

```
QString AddDialog::getType ( ) const
```

Obtiene el tipo de componente ingresado.

**Returns**

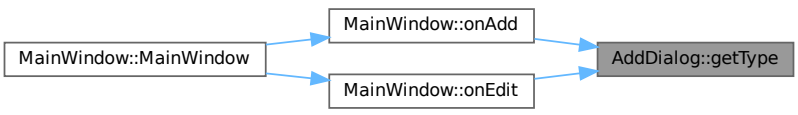
QString con el tipo (trimmed).

Definition at line 126 of file mainwindow.cpp.

```
00126 { return typeEdit->text().trimmed(); }
```

References typeEdit.

Here is the caller graph for this function:



#### 4.1.3.8 onCancel

```
void AddDialog::onCancel ( )  [private], [slot]
```

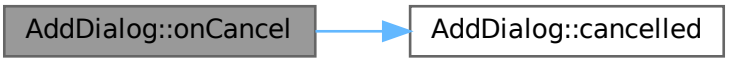
Slot llamado al presionar "Cancelar".

Emite la señal *cancelled()* para cerrar el diálogo sin cambios.

Definition at line 156 of file mainwindow.cpp.
```
00156 { emit cancelled(); }
```

References cancelled().

Here is the call graph for this function:




Here is the caller graph for this function:


#### 4.1.3.9 onOk

```
void AddDialog::onOk ( )  [private], [slot]
```

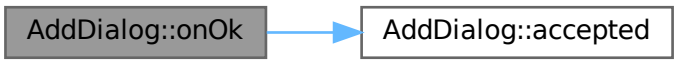
Slot llamado al presionar "Agregar/Aceptar".

Emite la señal *accepted()* para notificar a la ventana principal.

Definition at line 150 of file mainwindow.cpp.
```
00150 { emit accepted(); }
```

References accepted().

Here is the call graph for this function:




Here is the caller graph for this function:


### 4.1.4 Member Data Documentation

#### 4.1.4.1 dateEdit

```
QDateEdit* AddDialog::dateEdit  [private]
```

Definition at line 54 of file mainwindow.h.

#### 4.1.4.2 locEdit

```
QLineEdit* AddDialog::locEdit  [private]
```

Definition at line 53 of file mainwindow.h.

#### 4.1.4.3 nameEdit

```
QLineEdit* AddDialog::nameEdit  [private]
```

Definition at line 50 of file mainwindow.h.

#### 4.1.4.4 qtySpin

```
QSpinBox* AddDialog::qtySpin  [private]
```

Definition at line 52 of file mainwindow.h.

#### 4.1.4.5 typeEdit

```
QLineEdit* AddDialog::typeEdit  [private]
```

Definition at line 51 of file mainwindow.h.

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

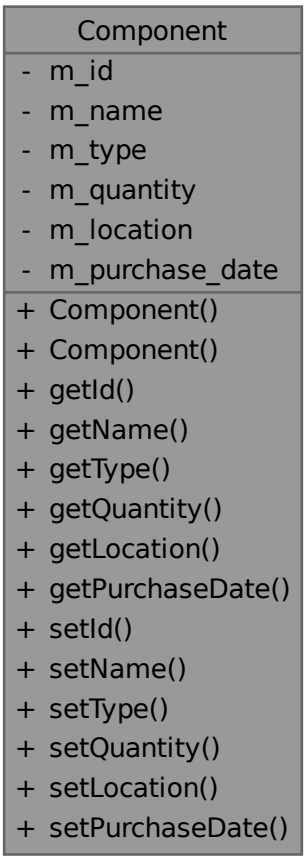
- include/mainwindow.h
- src/mainwindow.cpp

## 4.2 Component Class Reference

Representa un componente almacenado en el inventario.

```
#include <component.h>
```

Collaboration diagram for Component:



### Public Member Functions

- Component ()

  *Constructor por defecto.*
- Component (int id, const QString &name, const QString &type, int quantity, const QString &location, const QString &purchase_date)

  *Constructor con parámetros.*
- int getId () const

  *Obtiene el ID del componente.*
- QString getName () const

  *Obtiene el nombre del componente.*
- QString getType () const

  *Obtiene el tipo del componente.*
- int getQuantity () const

  *Obtiene la cantidad disponible del componente.*
- QString getLocation () const

  *Obtiene la ubicación del componente en el almacén.*
- QString getPurchaseDate () const

  *Obtiene la fecha de compra del componente.*
- void setId (int id)

  *Establece el ID del componente.*
- void setName (const QString &name)

  *Establece el nombre del componente.*
- void setType (const QString &type)

  *Establece el tipo del componente.*
- void setQuantity (int quantity)

  *Establece la cantidad del componente.*
- void setLocation (const QString &location)

  *Establece la ubicación del componente.*
- void setPurchaseDate (const QString &purchase_date)

  *Establece la fecha de compra del componente.*

### Private Attributes

- int m_id

  *Identificador único del componente.*
- QString m_name

  *Nombre del componente.*
- QString m_type

  *Tipo o categoría del componente.*
- int m_quantity

  *Cantidad disponible en inventario.*
- QString m_location

  *Ubicación física del componente.*
- QString m_purchase_date

  *Fecha de compra del componente.*

### 4.2.1 Detailed Description

Representa un componente almacenado en el inventario.

Esta clase modela un componente físico dentro del sistema de inventario. Incluye información básica como su identificador, nombre, tipo, cantidad, ubicación y fecha de compra. Provee métodos para obtener y modificar cada uno de estos atributos.

Definition at line 15 of file component.h.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 Component() [1/2]

```
Component::Component ( )
```

Constructor por defecto.

Inicializa todos los campos a valores base (0 o vacío).

Inicializa el componente con valores neutrales:

- id = 0
- cantidad = 0

Los demás campos quedan vacíos.

Definition at line 12 of file component.cpp.

```
00013     : m_id(0),
00014       m_quantity(0)
00015 {
00016 }
```

#### 4.2.2.2 Component() [2/2]

```
Component::Component (
            int id,
            const QString & name,
            const QString & type,
            int quantity,
            const QString & location,
            const QString & purchase_date )
```

Constructor con parámetros.

Constructor que inicializa todos los campos del componente.

**Parameters**

| | |
|---|---|
| *id* | Identificador único del componente. |
| *name* | Nombre del componente. |
| *type* | Tipo o categoría del componente. |
| *quantity* | Cantidad disponible en inventario. |
| *location* | Ubicación física dentro del almacén. |
| *purchase_date* | Fecha de compra del componente. |
| *id* | Identificador único del componente. |
| *name* | Nombre del componente. |

Definition at line 28 of file component.cpp.

```
00034     : m_id(id),
00035       m_name(name),
00036       m_type(type),
00037       m_quantity(quantity),
00038       m_location(location),
00039       m_purchase_date(purchase_date)
00040 {
00041 }
```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 getId()

```
int Component::getId ( ) const
```

Obtiene el ID del componente.

**Returns**

ID almacenado.

Definition at line 47 of file component.cpp.

```
00047 { return m_id; }
```

References m_id.

#### 4.2.3.2 getLocation()

```
QString Component::getLocation ( ) const
```

Obtiene la ubicación del componente en el almacén.

Obtiene la ubicación física del componente.

**Returns**

Ubicación almacenada.

Definition at line 71 of file component.cpp.

```
00071 { return m_location; }
```

References m_location.

#### 4.2.3.3 getName()

```
QString Component::getName ( ) const
```

Obtiene el nombre del componente.

**Returns**

Nombre actual.

Definition at line 53 of file component.cpp.

```
00053 { return m_name; }
```

References m_name.

#### 4.2.3.4 getPurchaseDate()

```
QString Component::getPurchaseDate ( ) const
```

Obtiene la fecha de compra del componente.

Obtiene la fecha de adquisición.

**Returns**

Fecha en formato QString.

Definition at line 77 of file component.cpp.
```
00077 { return m_purchase_date; }
```

References m_purchase_date.

#### 4.2.3.5 getQuantity()

```
int Component::getQuantity ( ) const
```

Obtiene la cantidad disponible del componente.

Obtiene la cantidad disponible.

**Returns**

Cantidad en inventario.

Definition at line 65 of file component.cpp.
```
00065 { return m_quantity; }
```

References m_quantity.

#### 4.2.3.6 getType()

```
QString Component::getType ( ) const
```

Obtiene el tipo del componente.

**Returns**

Tipo actual.

Definition at line 59 of file component.cpp.
```
00059 { return m_type; }
```

References m_type.

#### 4.2.3.7 setId()

```
void Component::setId (
            int id )
```

Establece el ID del componente.

Establece un nuevo ID.

**Parameters**

| | |
|---|---|
| *id* | Nuevo identificador. |

Definition at line 83 of file component.cpp.

```
00083 { m_id = id; }
```

References m_id.

#### 4.2.3.8 setLocation()

```
void Component::setLocation (
            const QString & location )
```

Establece la ubicación del componente.

Establece una nueva ubicación.

**Parameters**

| | |
|---|---|
| *location* | Ubicación actualizada. |

Definition at line 107 of file component.cpp.

```
00107 { m_location = location; }
```

References m_location.

#### 4.2.3.9 setName()

```
void Component::setName (
            const QString & name )
```

Establece el nombre del componente.

Establece un nuevo nombre.

**Parameters**

| | |
|---|---|
| *name* | Nombre actualizado. |

Definition at line 89 of file component.cpp.

```
00089 { m_name = name; }
```

References m_name.

#### 4.2.3.10 setPurchaseDate()

```
void Component::setPurchaseDate (
            const QString & purchase_date )
```

Establece la fecha de compra del componente.

Establece una nueva fecha de adquisición.

**Parameters**

| | |
|---|---|
| *purchase_date* | Fecha actualizada. |

Definition at line 113 of file component.cpp.

```
00113 { m_purchase_date = purchase_date; }
```

References m_purchase_date.

#### 4.2.3.11 setQuantity()

```
void Component::setQuantity (
            int quantity )
```

Establece la cantidad del componente.

Cambia la cantidad disponible.

**Parameters**

| | |
|---|---|
| *quantity* | Nueva cantidad. |

Definition at line 101 of file component.cpp.

```
00101 { m_quantity = quantity; }
```

References m_quantity.

#### 4.2.3.12 setType()

```
void Component::setType (
            const QString & type )
```

Establece el tipo del componente.

Establece un nuevo tipo de componente.

**Parameters**

| | |
|---|---|
| *type* | Tipo actualizado. |

Definition at line 95 of file component.cpp.

```
00095 { m_type = type; }
```

References m_type.

### 4.2.4 Member Data Documentation

#### 4.2.4.1 m_id

```
int Component::m_id  [private]
```

Identificador único del componente.

Definition at line 78 of file component.h.

#### 4.2.4.2 m_location

```
QString Component::m_location  [private]
```

Ubicación física del componente.

Definition at line 82 of file component.h.

#### 4.2.4.3 m_name

```
QString Component::m_name  [private]
```

Nombre del componente.

Definition at line 79 of file component.h.

#### 4.2.4.4 m_purchase_date

```
QString Component::m_purchase_date  [private]
```

Fecha de compra del componente.

Definition at line 83 of file component.h.

#### 4.2.4.5 m_quantity

```
int Component::m_quantity  [private]
```

Cantidad disponible en inventario.

Definition at line 81 of file component.h.

#### 4.2.4.6 m_type

```
QString Component::m_type  [private]
```

Tipo o categoría del componente.

Definition at line 80 of file component.h.

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

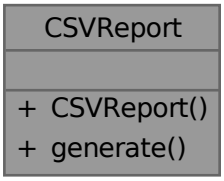
- include/component.h
- src/component.cpp

## 4.3 CSVReport Class Reference

Clase encargada de generar reportes CSV del inventario.

`#include <report.h>`

Collaboration diagram for CSVReport:




### Public Member Functions

- CSVReport ()
  *Constructor por defecto.*
- bool generate (const QList< InventoryItem > &items, const QString &filePath)
  *Genera un archivo CSV con la lista de items proporcionada.*

### 4.3.1 Detailed Description

Clase encargada de generar reportes CSV del inventario.

Esta clase ofrece un método principal para crear un archivo CSV que contiene todos los campos relevantes de cada InventoryItem.

Definition at line 32 of file report.h.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 CSVReport()

```
CSVReport::CSVReport ( )
```

Constructor por defecto.

Constructor por defecto de la clase CSVReport.

No realiza ninguna operación específica, pero se define para mantener consistencia en la estructura del proyecto.

Definition at line 10 of file report.cpp.

```
00011 {
00012 }
```

### 4.3.3 Member Function Documentation

#### 4.3.3.1 generate()

```
bool CSVReport::generate (
            const QList< InventoryItem > & items,
            const QString & filePath )
```

Genera un archivo CSV con la lista de items proporcionada.

Genera un archivo CSV con la información del inventario.

El archivo generado contiene encabezados y cada fila representa un elemento del inventario con su información completa.

**Parameters**

| | |
|---|---|
| *items* | Lista de elementos del inventario a exportar. |
| *filePath* | Ruta completa donde se guardará el archivo CSV. |

**Returns**

true si el archivo se generó correctamente, false en caso contrario.

Abre o crea un archivo en la ruta indicada y escribe una fila con los encabezados seguida por una línea por cada elemento del inventario. Los campos de texto son encapsulados entre comillas dobles.

Ejemplo de formato generado:

```
ID;Nombre;Tipo;Cantidad;Ubicacion;FechaAdquisicion
1;"Resistencia";"Electrónico";50;"Caja A";"2024-03-01"
```

**Parameters**

| | |
|---|---|
| *items* | Lista de elementos del inventario. |
| *filePath* | Ruta completa del archivo CSV a generar. |

**Returns**

`true` si el archivo fue generado correctamente, `false` si no fue posible abrirlo para escritura.

Definition at line 33 of file report.cpp.

```
00035 {
00036     QFile file(filePath);
00037
00038     if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
00039         qDebug() « "No se puede abrir archivo CSV:" « filePath;
00040         return false;
00041     }
00042
00043     QTextStream out(&file);
00044
00045     // Encabezados del CSV
00046     out « "ID;Nombre;Tipo;Cantidad;Ubicacion;FechaAdquisicion\n";
00047
00048     // Datos de cada item
00049     for (const InventoryItem &it : items) {
00050         out « it.id « ";"
00051             « "\"" « it.nombre « "\";"
00052             « "\"" « it.tipo « "\";"
```

```
00053             « it.cantidad « ";"
00054             « "\"" « it.ubicacion « "\";"
00055             « "\"" « it.fechaAdquisicion « "\"\n";
00056     }
00057
00058     file.close();
00059     return true;
00060 }
```

Here is the caller graph for this function:


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

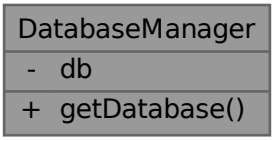
- include/report.h
- src/report.cpp

## 4.4 DatabaseManager Class Reference

Clase encargada de gestionar la conexión con la base de datos.

`#include <DatabaseManager.h>`

Collaboration diagram for DatabaseManager:




### Static Public Member Functions

- static QSqlDatabase getDatabase ()

  *Obtiene la base de datos principal del sistema.*

### Static Private Attributes

- static QSqlDatabase db = QSqlDatabase()

  *Instancia estática de la base de datos administrada.*

### 4.4.1 Detailed Description

Clase encargada de gestionar la conexión con la base de datos.

Esta clase implementa un patrón similar a Singleton para proporcionar una única instancia de conexión a la base de datos SQLite utilizada por la aplicación. El método estático getDatabase retorna una referencia a dicha conexión evitando crear múltiples instancias.

Se utiliza QSqlDatabase para manejar la apertura y configuración de la base de datos según lo requiera Qt.

Definition at line 18 of file DatabaseManager.h.

### 4.4.2 Member Function Documentation

#### 4.4.2.1 getDatabase()

```
QSqlDatabase DatabaseManager::getDatabase ( )  [static]
```

Obtiene la base de datos principal del sistema.

Obtiene y gestiona la conexión a la base de datos SQLite.

Si la conexión aún no se ha inicializado, se configura y se abre. En llamadas posteriores, devuelve la misma instancia ya configurada.

**Returns**

Instancia global de QSqlDatabase utilizada por el sistema.

Este método garantiza que la base de datos:

- Sea creada solo una vez (patrón singleton estático).
- Se configure con el driver "QSQLITE".
- Utilice como archivo local "inventario.db".
- Abra la conexión si aún no lo está.

Si ocurre un error al abrir la base de datos, el mensaje es mostrado mediante qDebug().

**Returns**

Objeto QSqlDatabase listo para ser utilizado.

Definition at line 27 of file DatabaseManager.cpp.

```
00028 {
00029     // Si la instancia aún no es válida, configurar el driver
00030     if (!db.isValid()) {
00031         db = QSqlDatabase::addDatabase("QSQLITE");
00032         db.setDatabaseName("inventario.db");
00033     }
00034
00035     // Abrir la base de datos si aún no está abierta
00036     if (!db.isOpen()) {
00037         if (!db.open()) {
00038             qDebug() « "ERROR al abrir la base de datos:" « db.lastError();
00039         } else {
00040             qDebug() « "Base de datos abierta correctamente.";
00041         }
00042     }
00043
00044     return db;
00045 }
```

References db.

Here is the caller graph for this function:


### 4.4.3 Member Data Documentation

#### 4.4.3.1 db

```
QSqlDatabase DatabaseManager::db = QSqlDatabase()  [static], [private]
```

Instancia estática de la base de datos administrada.

Inicialización del objeto estático de base de datos.

Esta variable almacena la conexión única utilizada por toda la aplicación. Es configurada en getDatabase.

Se inicializa como una instancia vacía. La configuración real (tipo de driver y nombre de archivo) se realiza dentro de getDatabase().

Definition at line 38 of file DatabaseManager.h.

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

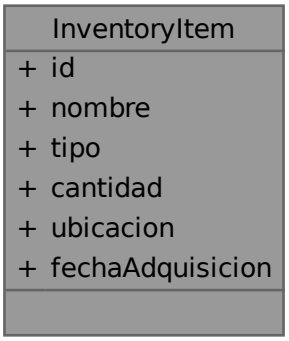
- include/DatabaseManager.h
- src/DatabaseManager.cpp

## 4.5 InventoryItem Struct Reference

Estructura que representa un elemento del inventario.

`#include <InventoryManager.h>`

Collaboration diagram for InventoryItem:




**Public Attributes**

- int id
- QString nombre
- QString tipo
- int cantidad
- QString ubicacion
- QString fechaAdquisicion

### 4.5.1 Detailed Description

Estructura que representa un elemento del inventario.

Esta estructura es declarada en otro archivo (InventoryManager.h), y se utiliza aquí solo como referencia para exportar datos.

Definition at line 13 of file InventoryManager.h.

### 4.5.2 Member Data Documentation

#### 4.5.2.1 cantidad

`int InventoryItem::cantidad`

Definition at line 17 of file InventoryManager.h.

#### 4.5.2.2 fechaAdquisicion

```
QString InventoryItem::fechaAdquisicion
```

Definition at line 19 of file InventoryManager.h.

#### 4.5.2.3 id

```
int InventoryItem::id
```

Definition at line 14 of file InventoryManager.h.

#### 4.5.2.4 nombre

```
QString InventoryItem::nombre
```

Definition at line 15 of file InventoryManager.h.

#### 4.5.2.5 tipo

```
QString InventoryItem::tipo
```

Definition at line 16 of file InventoryManager.h.

#### 4.5.2.6 ubicacion

```
QString InventoryItem::ubicacion
```

Definition at line 18 of file InventoryManager.h.

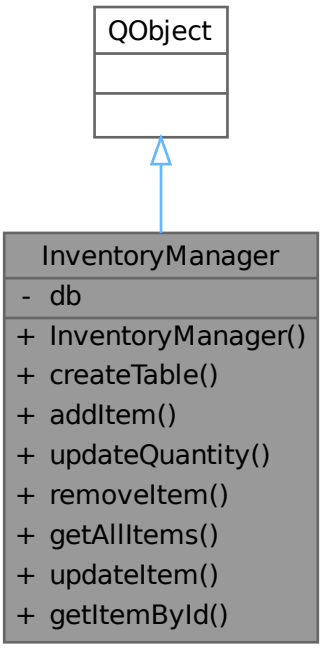
The documentation for this struct was generated from the following file:

- include/InventoryManager.h

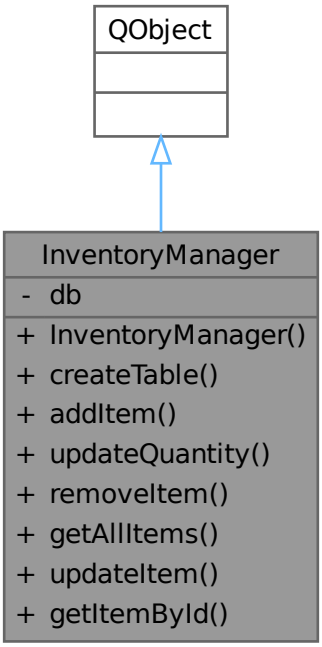
## 4.6 InventoryManager Class Reference

`#include <InventoryManager.h>`

Inheritance diagram for InventoryManager:



Collaboration diagram for InventoryManager:




### Public Member Functions

- InventoryManager (QSqlDatabase database, QObject *parent=nullptr)

  *Constructor de InventoryManager.*
- bool createTable ()

  *Crea la tabla principal del inventario si no existe.*
- bool addItem (const QString &nombre, const QString &tipo, int cantidad, const QString &ubicacion, const QString &fechaAdquisicion)

  *Inserta un nuevo elemento en la tabla inventario.*
- bool updateQuantity (int id, int newQuantity)

  *Actualiza únicamente la cantidad de un elemento identificado por id.*
- bool removeItem (int id)

  *Elimina un elemento del inventario.*
- QList< InventoryItem > getAllItems ()

  *Obtiene todos los registros almacenados en la tabla inventario.*
- bool updateItem (int id, const QString &nombre, const QString &tipo, int cantidad, const QString &ubicacion, const QString &fechaAdquisicion)

  *Actualiza todos los campos de un elemento del inventario.*
- InventoryItem getItemById (int id)

  *Obtiene un único elemento del inventario según su id.*

### Private Attributes

- QSqlDatabase db

### 4.6.1 Detailed Description

Definition at line 29 of file InventoryManager.h.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 InventoryManager()

```
InventoryManager::InventoryManager (
            QSqlDatabase database,
            QObject * parent = nullptr )  [explicit]
```

Constructor de InventoryManager.

Inicializa el gestor de inventario con la base de datos proporcionada. Se verifica que la base de datos sea válida; de no serlo, la ejecución del programa se detiene mediante qFatal, ya que el inventario depende completamente del acceso a la base de datos.

**Parameters**

| | |
|---|---|
| *database* | Conexión a la base de datos SQLite. |
| *parent* | Objeto padre opcional según el sistema de jerarquía de Qt. |

Definition at line 17 of file InventoryManager.cpp.

```
00018     : QObject(parent), db(database)
00019 {
00020     if (!db.isValid()) {
00021         qFatal("ERROR FATAL: Base de datos no válida. "
00022                "Asegúrate de usar DatabaseManager::getDatabase().");
00023     }
00024 }
```

References db.

### 4.6.3 Member Function Documentation

#### 4.6.3.1 addItem()

```
bool InventoryManager::addItem (
            const QString & nombre,
            const QString & tipo,
            int cantidad,
            const QString & ubicacion,
            const QString & fechaAdquisicion )
```

Inserta un nuevo elemento en la tabla inventario.

**Parameters**

| | |
|---|---|
| *nombre* | Nombre del componente. |
| *tipo* | Tipo o categoría. |
| *cantidad* | Cantidad disponible. |
| *ubicacion* | Ubicación física. |
| *fechaAdquisicion* | Fecha de adquisición. |

**Returns**

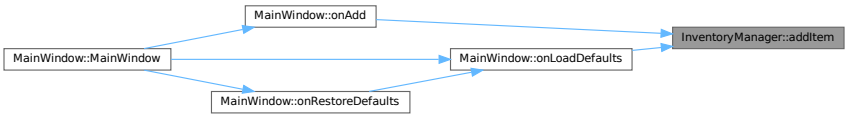
true si la operación fue exitosa, false si el INSERT falló.

Definition at line 67 of file InventoryManager.cpp.

```
00072 {
00073     QSqlQuery query(db);
00074
00075     query.prepare(
00076         "INSERT INTO inventario "
00077         "(nombre, tipo, cantidad, ubicacion, fechaAdquisicion) "
00078         "VALUES (?, ?, ?, ?, ?)"
00079     );
00080
00081     query.addBindValue(nombre);
00082     query.addBindValue(tipo);
00083     query.addBindValue(cantidad);
00084     query.addBindValue(ubicacion);
00085     query.addBindValue(fechaAdquisicion);
00086
00087     return query.exec();
00088 }
```

References db.

Here is the caller graph for this function:




#### 4.6.3.2 createTable()

```
bool InventoryManager::createTable ( )
```

Crea la tabla principal del inventario si no existe.

La tabla contiene los campos:

- id (PRIMARY KEY AUTOINCREMENT)
- nombre
- tipo
- cantidad
- ubicacion
- fechaAdquisicion

**Returns**

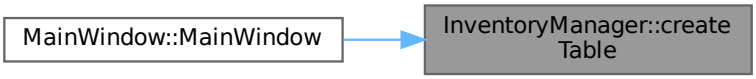
true si la tabla se creó o ya existía; false si hubo error en la ejecución.

Definition at line 39 of file InventoryManager.cpp.

```
00040 {
00041     QSqlQuery query(db);
00042
00043     QString sql =
00044         "CREATE TABLE IF NOT EXISTS inventario ("
00045         "id INTEGER PRIMARY KEY AUTOINCREMENT,"
00046         "nombre TEXT NOT NULL,"
00047         "tipo TEXT NOT NULL,"
00048         "cantidad INTEGER NOT NULL,"
00049         "ubicacion TEXT NOT NULL,"
00050         "fechaAdquisicion TEXT NOT NULL"
00051         ");";
00052
00053     return query.exec(sql);
00054 }
```

References db.

Here is the caller graph for this function:




#### 4.6.3.3 getAllItems()

```
QList< InventoryItem > InventoryManager::getAllItems ( )
```

Obtiene todos los registros almacenados en la tabla inventario.

**Returns**

Lista con todos los InventoryItem encontrados.

Definition at line 127 of file InventoryManager.cpp.

```
00128 {
00129     QList<InventoryItem> items;
00130     QSqlQuery query(db);
00131
00132     query.exec("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion FROM inventario");
00133
00134     while (query.next()) {
00135         InventoryItem it;
00136         it.id = query.value(0).toInt();
00137         it.nombre = query.value(1).toString();
00138         it.tipo = query.value(2).toString();
00139         it.cantidad = query.value(3).toInt();
00140         it.ubicacion = query.value(4).toString();
00141         it.fechaAdquisicion = query.value(5).toString();
00142         items.append(it);
00143     }
00144     return items;
00145 }
```

References InventoryItem::cantidad, db, InventoryItem::fechaAdquisicion, InventoryItem::id, InventoryItem::nombre, InventoryItem::tipo, and InventoryItem::ubicacion.

Here is the caller graph for this function:


#### 4.6.3.4 getItemById()

```
InventoryItem InventoryManager::getItemById (
            int id )
```

Obtiene un único elemento del inventario según su id.

**Parameters**

| | |
|---|---|
| *id* | Identificador buscado. |

**Returns**

Estructura InventoryItem con los datos encontrados. Si no existe, retorna un objeto vacío con valores por defecto.

Definition at line 192 of file InventoryManager.cpp.

```
00193 {
00194     InventoryItem it;
00195
00196     QSqlQuery query(db);
00197     query.prepare("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion "
00198                   "FROM inventario WHERE id = ?");
00199     query.addBindValue(id);
00200
00201     if (!query.exec() || !query.next()) {
00202         return it;  // vacío si no se encuentra
00203     }
00204
00205     it.id = query.value(0).toInt();
00206     it.nombre = query.value(1).toString();
00207     it.tipo = query.value(2).toString();
00208     it.cantidad = query.value(3).toInt();
00209     it.ubicacion = query.value(4).toString();
00210     it.fechaAdquisicion = query.value(5).toString();
00211
00212     return it;
00213 }
```

References InventoryItem::cantidad, db, InventoryItem::fechaAdquisicion, InventoryItem::id, InventoryItem::nombre, InventoryItem::tipo, and InventoryItem::ubicacion.

Here is the caller graph for this function:

#### 4.6.3.5 removeItem()

```
bool InventoryManager::removeItem (
            int id )
```

Elimina un elemento del inventario.

**Parameters**

| | |
|---|---|
| *id* | Identificador del elemento a borrar. |

**Returns**

true si el registro fue eliminado correctamente.

Definition at line 114 of file InventoryManager.cpp.

```
00115 {
00116     QSqlQuery query(db);
00117     query.prepare("DELETE FROM inventario WHERE id = ?");
00118     query.addBindValue(id);
00119     return query.exec();
00120 }
```

References db.

Here is the caller graph for this function:


#### 4.6.3.6 updateItem()

```
bool InventoryManager::updateItem (
            int id,
            const QString & nombre,
            const QString & tipo,
            int cantidad,
            const QString & ubicacion,
            const QString & fechaAdquisicion )
```

Actualiza todos los campos de un elemento del inventario.

**Parameters**

| | |
|---|---|
| *id* | Identificador del registro a actualizar. |
| *nombre* | Nuevo nombre. |
| *tipo* | Nuevo tipo. |
| *cantidad* | Nueva cantidad. |
| *ubicacion* | Nueva ubicación. |
| *fechaAdquisicion* | Nueva fecha de adquisición. |

**Returns**

true si el registro fue modificado correctamente.

Definition at line 159 of file InventoryManager.cpp.

```
00165 {
00166     QSqlQuery query(db);
00167
00168     query.prepare(
00169         "UPDATE inventario SET "
00170         "nombre = ?, tipo = ?, cantidad = ?, ubicacion = ?, fechaAdquisicion = ? "
00171         "WHERE id = ?"
00172     );
00173
00174     query.addBindValue(nombre);
00175     query.addBindValue(tipo);
00176     query.addBindValue(cantidad);
00177     query.addBindValue(ubicacion);
00178     query.addBindValue(fechaAdquisicion);
00179     query.addBindValue(id);
00180
00181     return query.exec();
00182 }
```

References db.

Here is the caller graph for this function:


#### 4.6.3.7 updateQuantity()

```
bool InventoryManager::updateQuantity (
            int id,
            int newQuantity )
```

Actualiza únicamente la cantidad de un elemento identificado por id.

**Parameters**

| | |
|---|---|
| *id* | Identificador del registro a actualizar. |
| *newQuantity* | Nueva cantidad a asignar. |

**Returns**

true si la operación fue exitosa, false si falló.

Definition at line 98 of file InventoryManager.cpp.

```
00099 {
00100     QSqlQuery query(db);
00101     query.prepare("UPDATE inventario SET cantidad = ? WHERE id = ?");
00102     query.addBindValue(newQuantity);
00103     query.addBindValue(id);
00104     return query.exec();
00105 }
```

References db.

### 4.6.4 Member Data Documentation

#### 4.6.4.1 db

```
QSqlDatabase InventoryManager::db  [private]
```

Definition at line 89 of file InventoryManager.h.

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

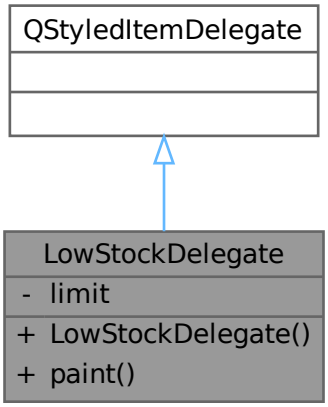
- include/InventoryManager.h
- src/InventoryManager.cpp

## 4.7 LowStockDelegate Class Reference

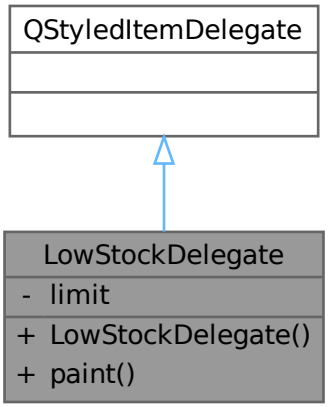
Delegate que resalta en rojo los elementos con bajo stock.

```
#include <delegate.h>
```

Inheritance diagram for LowStockDelegate:



Collaboration diagram for LowStockDelegate:




**Public Member Functions**

- LowStockDelegate (int threshold, QObject ∗parent=nullptr)

  *Constructor del delegado de bajo stock.*
- void paint (QPainter ∗painter, const QStyleOptionViewItem &option, const QModelIndex &index) const override

  *Sobrescribe el método paint para aplicar resaltado.*

**Private Attributes**

- int limit

  *Umbral mínimo de cantidad para marcar el texto en rojo.*

### 4.7.1 Detailed Description

Delegate que resalta en rojo los elementos con bajo stock.

Esta clase hereda de QStyledItemDelegate y se encarga de modificar la apariencia de las celdas en un QTableView o QListView cuando la cantidad de un componente es menor a un umbral definido.

Útil para visualizar rápidamente inventario crítico en interfaces Qt.

Definition at line 17 of file delegate.h.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 LowStockDelegate()

```
LowStockDelegate::LowStockDelegate (
            int threshold,
            QObject * parent = nullptr )  [inline]
```

Constructor del delegado de bajo stock.

**Parameters**

| | |
|---|---|
| *threshold* | Umbral mínimo de cantidad antes de marcar el texto en rojo. |
| *parent* | Objeto padre opcional según la jerarquía Qt. |

Definition at line 26 of file delegate.h.

```
00027         : QStyledItemDelegate(parent), limit(threshold) {}
```

## 4.7.3 Member Function Documentation

### 4.7.3.1 paint()

```
void LowStockDelegate::paint (
            QPainter * painter,
            const QStyleOptionViewItem & option,
            const QModelIndex & index ) const  [inline], [override]
```

Sobrescribe el método paint para aplicar resaltado.

Cambia el color del texto a rojo cuando la columna correspondiente a la cantidad (columna 3 del modelo) tiene un valor menor al umbral definido en limit.

**Parameters**

| | |
|---|---|
| *painter* | Objeto empleado para realizar el dibujo. |
| *option* | Opciones de estilo de la celda. |
| *index* | Índice del elemento dentro del modelo. |

Definition at line 40 of file delegate.h.

```
00043     {
00044         QStyleOptionViewItem opt(option);
00045
00046         // Columna 3 → cantidad del componente
00047         if (index.column() == 3 && index.data().toInt() < limit) {
00048             opt.palette.setColor(QPalette::Text, Qt::red);
00049         }
00050
00051         QStyledItemDelegate::paint(painter, opt, index);
00052     }
```

References limit.

## 4.7.4 Member Data Documentation

### 4.7.4.1 limit

```
int LowStockDelegate::limit  [private]
```

Umbral mínimo de cantidad para marcar el texto en rojo.

Definition at line 55 of file delegate.h.

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

- include/delegate.h

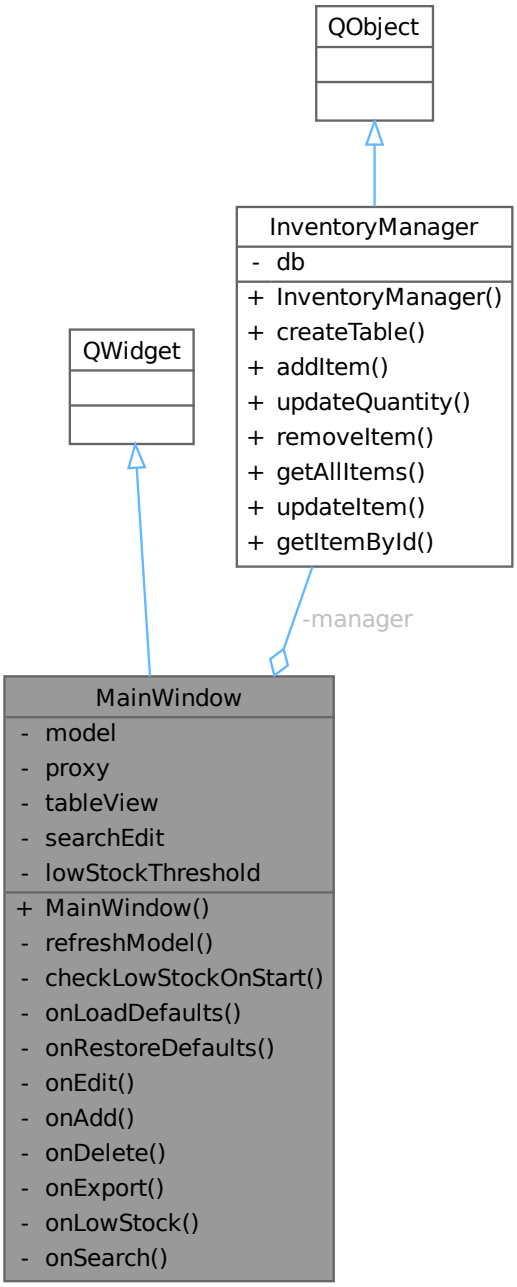
## 4.8 MainWindow Class Reference

Controlador principal de la aplicación de gestión de inventario.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:

Collaboration diagram for MainWindow:




### Public Member Functions

- MainWindow (QSqlDatabase db, QWidget ∗parent=nullptr)

  *Constructor de la ventana principal.*

### Private Slots

- void onEdit ()

*Inicia el proceso de edición del componente seleccionado.*

- void onAdd ()

  *Abre el diálogo para agregar un nuevo componente al inventario.*
- void onDelete ()

  *Elimina el registro seleccionado en la tabla.*
- void onExport ()

  *Genera y exporta un reporte del inventario en formato CSV.*
- void onLowStock ()

  *Analiza el stock actual y resalta visualmente los ítems críticos.*
- void onSearch (const QString &text)

  *Filtra la tabla en tiempo real según el texto ingresado.*

**Private Member Functions**

- void refreshModel ()

  *Actualiza la vista recargando los datos desde la base de datos SQL.*
- void checkLowStockOnStart ()
- void onLoadDefaults ()

  *Carga un conjunto de datos de prueba (Seed Data).*
- void onRestoreDefaults ()

  *Restaura la base de datos a su estado original (vacía y luego recargada).*

**Private Attributes**

- InventoryManager manager
- QSqlQueryModel * model
- QSortFilterProxyModel * proxy
- QTableView * tableView
- QLineEdit * searchEdit
- const int lowStockThreshold = 5

### 4.8.1 Detailed Description

Controlador principal de la aplicación de gestión de inventario.

Esta clase orquesta la interacción entre el usuario y la base de datos. Sus responsabilidades incluyen:

- Inicializar la base de datos y la interfaz de usuario.
- Gestionar el modelo de datos (QSqlQueryModel) y el filtrado (QSortFilterProxyModel).
- Manejar eventos de botones (CRUD, exportación, restauración).
- Aplicar delegados personalizados para la visualización (LowStockDelegate).

Definition at line 65 of file mainwindow.h.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 MainWindow()

```
MainWindow::MainWindow (
            QSqlDatabase db,
            QWidget * parent = nullptr )  [explicit]
```

Constructor de la ventana principal.

Inicializa la conexión a la base de datos, configura el layout principal, crea los modelos de datos y conecta todas las señales de la interfaz.

**Parameters**

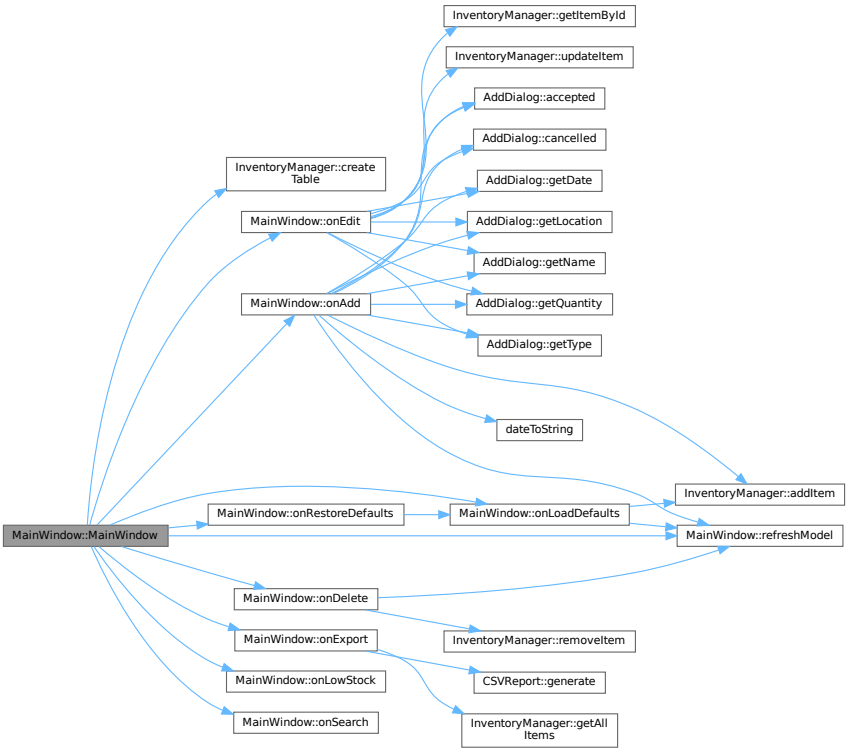
| | |
|---|---|
| *db* | Referencia a la conexión de base de datos QSqlDatabase ya inicializada. |
| *parent* | Widget padre (opcional). |

Definition at line 184 of file mainwindow.cpp.

```
00185     : QWidget(parent), manager(db)
00186 {
00187     setWindowTitle("Gestión de Inventario");
00188     resize(1000, 600);
00189
00190     QVBoxLayout *mainLayout = new QVBoxLayout(this);
00191     QHBoxLayout *topLayout = new QHBoxLayout();
00192
00193     // -- Inicialización de Widgets --
00194     searchEdit = new QLineEdit(); searchEdit->setPlaceholderText("Buscar...");
00195     QPushButton *btnAdd = new QPushButton("Agregar");
00196     QPushButton *btnDelete = new QPushButton("Eliminar seleccionado");
00197     QPushButton *btnExport = new QPushButton("Exportar CSV");
00198     QPushButton *btnLowStock = new QPushButton("Revisar stock bajo");
00199     QPushButton *btnLoadDefaults = new QPushButton("Cargar base por defecto");
00200     QPushButton *btnRestore = new QPushButton("Restaurar base original");
00201     QPushButton *btnEdit = new QPushButton("Editar");
00202
00203     // -- Construcción del Layout Superior --
00204     topLayout->addWidget(btnLoadDefaults);
00205     topLayout->addWidget(btnRestore);
00206     topLayout->addWidget(btnEdit);
00207     topLayout->addWidget(new QLabel("Buscar:"));
00208     topLayout->addWidget(searchEdit);
00209     topLayout->addWidget(btnAdd);
00210     topLayout->addWidget(btnDelete);
00211     topLayout->addWidget(btnLowStock);
00212     topLayout->addWidget(btnExport);
00213
00214     mainLayout->addLayout(topLayout);
00215
00216     // -- Configuración del Modelo MVC --
00217     model = new QSqlQueryModel(this);
00218
00219     // Configuración del Proxy para filtrado y ordenamiento
00220     proxy = new QSortFilterProxyModel(this);
00221     proxy->setSourceModel(model);
00222     proxy->setFilterCaseSensitivity(Qt::CaseInsensitive);
00223     proxy->setFilterKeyColumn(-1); // Filtrar en todas las columnas
00224
00225     // -- Configuración de la Tabla (Vista) --
00226     tableView = new QTableView();
00227     tableView->setModel(proxy);
00228     tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
00229     tableView->setSelectionMode(QAbstractItemView::SingleSelection);
00230     tableView->setEditTriggers(QAbstractItemView::NoEditTriggers); // Edición solo vía diálogo
00231
00232     // Inyección del delegado para resaltar stock bajo
00233     tableView->setItemDelegate(new LowStockDelegate(lowStockThreshold, this));
00234     mainLayout->addWidget(tableView);
00235
00236     // -- Conexiones de Señales y Slots --
00237     connect(btnAdd, &QPushButton::clicked, this, &MainWindow::onAdd);
00238     connect(btnDelete, &QPushButton::clicked, this, &MainWindow::onDelete);
00239     connect(btnExport, &QPushButton::clicked, this, &MainWindow::onExport);
00240     connect(btnLowStock, &QPushButton::clicked, this, &MainWindow::onLowStock);
00241     connect(searchEdit, &QLineEdit::textChanged, this, &MainWindow::onSearch);
00242     connect(btnLoadDefaults, &QPushButton::clicked, this, &MainWindow::onLoadDefaults);
00243     connect(btnRestore, &QPushButton::clicked, this, &MainWindow::onRestoreDefaults);
00244     connect(btnEdit, &QPushButton::clicked, this, &MainWindow::onEdit);
00245
00246     // Inicialización de la tabla en BD si no existe
00247     if (!manager.createTable()) {
00248         QMessageBox::critical(this, "Error Crítico", "No se pudo crear o verificar la tabla de
      inventario en la base de datos.");
00249     }
00250
00251     refreshModel();
00252 }
```

References InventoryManager::createTable(), lowStockThreshold, manager, model, onAdd(), onDelete(), onEdit(), onExport(), onLoadDefaults(), onLowStock(), onRestoreDefaults(), onSearch(), proxy, refreshModel(), searchEdit, and tableView.

Here is the call graph for this function:




## 4.8.3 Member Function Documentation

### 4.8.3.1 checkLowStockOnStart()

```
void MainWindow::checkLowStockOnStart ( )  [private]
```

### 4.8.3.2 onAdd

```
void MainWindow::onAdd ( )  [private], [slot]
```

Abre el diálogo para agregar un nuevo componente al inventario.

Crea una instancia de AddDialog y conecta su señal de aceptación para insertar el nuevo registro mediante InventoryManager.

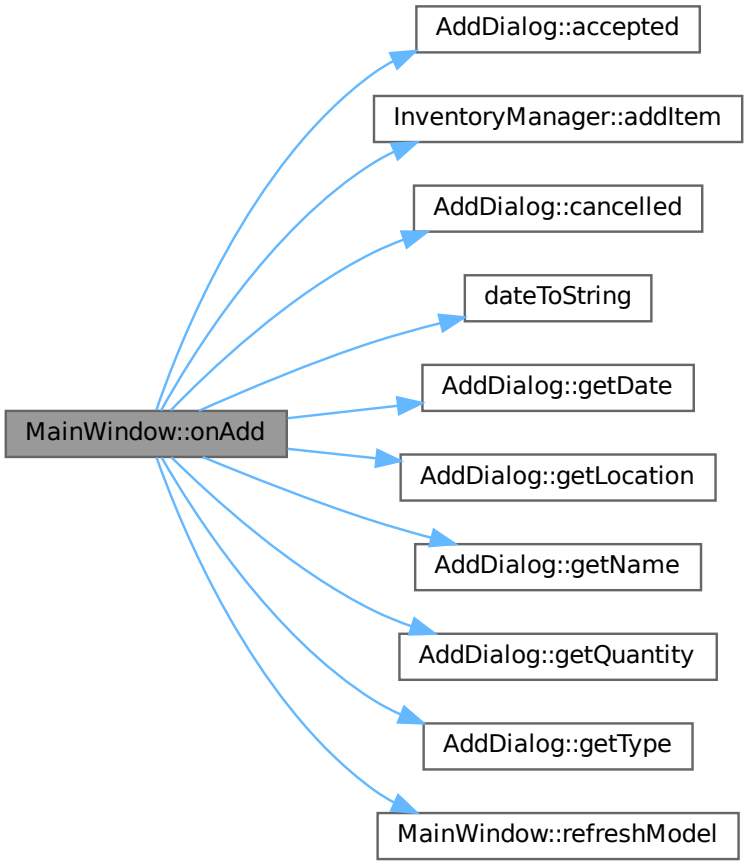
Definition at line 323 of file mainwindow.cpp.

```
00324 {
00325     AddDialog *dlg = new AddDialog();
00326     dlg->setWindowModality(Qt::ApplicationModal);
00327     dlg->show();
00328
00329     connect(dlg, &AddDialog::accepted, this, [this, dlg]() {
00330         if (dlg->getName().isEmpty() || dlg->getType().isEmpty()) {
00331             QMessageBox::warning(this, "Validación", "Nombre y tipo son obligatorios.");
00332             return;
00333         }
00334
00335         Component c(0,
```

```
00336                     dlg->getName(),
00337                     dlg->getType(),
00338                     dlg->getQuantity(),
00339                     dlg->getLocation(),
00340                     dateToString(dlg->getDate()));
00341
00342         if (!manager.addItem(c.getName(), c.getType(), c.getQuantity(), c.getLocation(),
      c.getPurchaseDate())) {
00343             QMessageBox::critical(this, "Error", "No se pudo insertar el componente en la base de
      datos.");
00344         } else {
00345             refreshModel();
00346         }
00347         dlg->close();
00348         dlg->deleteLater();
00349     });
00350
00351     connect(dlg, &AddDialog::cancelled, dlg, &AddDialog::close);
00352 }
```

References AddDialog::accepted(), InventoryManager::addItem(), AddDialog::cancelled(), dateToString(), AddDialog::getDate(), AddDialog::getLocation(), AddDialog::getName(), AddDialog::getQuantity(), AddDialog::getType(), manager, and refreshModel().

Here is the call graph for this function:



Here is the caller graph for this function:


#### 4.8.3.3 onDelete

```
void MainWindow::onDelete ( )  [private], [slot]
```

Elimina el registro seleccionado en la tabla.

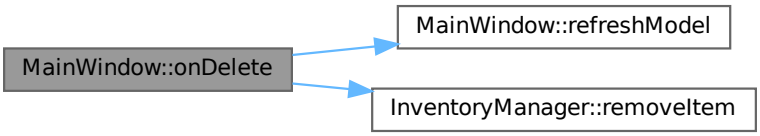
Muestra un cuadro de confirmación antes de proceder. Si se confirma, solicita al InventoryManager la eliminación por ID.

Definition at line 359 of file mainwindow.cpp.

```
00360 {
00361     QModelIndexList sel = tableView->selectionModel()->selectedRows();
00362     if (sel.isEmpty()) {
00363         QMessageBox::information(this, "Eliminar", "Selecciona una fila para eliminar.");
00364         return;
00365     }
00366
00367     QModelIndex idx = sel.first();
00368     QModelIndex src = proxy->mapToSource(idx);
00369     int id = model->data(model->index(src.row(), 0)).toInt();
00370
00371     if (QMessageBox::question(this, "Confirmar Eliminación",
00372                               QString("¿Estás seguro de eliminar el registro con ID %1?").arg(id)) ==
      QMessageBox::Yes)
00373     {
00374         if (!manager.removeItem(id)) {
00375             QMessageBox::critical(this, "Error", "No se pudo eliminar el registro.");
00376         } else {
00377             refreshModel();
00378         }
00379     }
00380 }
```

References manager, model, proxy, refreshModel(), InventoryManager::removeItem(), and tableView.

Here is the call graph for this function:



Here is the caller graph for this function:


#### 4.8.3.4 onEdit

```
void MainWindow::onEdit ( )  [private], [slot]
```

Inicia el proceso de edición del componente seleccionado.

Recupera el ID del item seleccionado en la tabla, consulta sus datos actuales desde la base de datos, y abre un AddDialog precargado con dicha información.

**Note**

Utiliza una función lambda para manejar la señal `accepted` del diálogo, capturando el ID para realizar el `updateItem`.

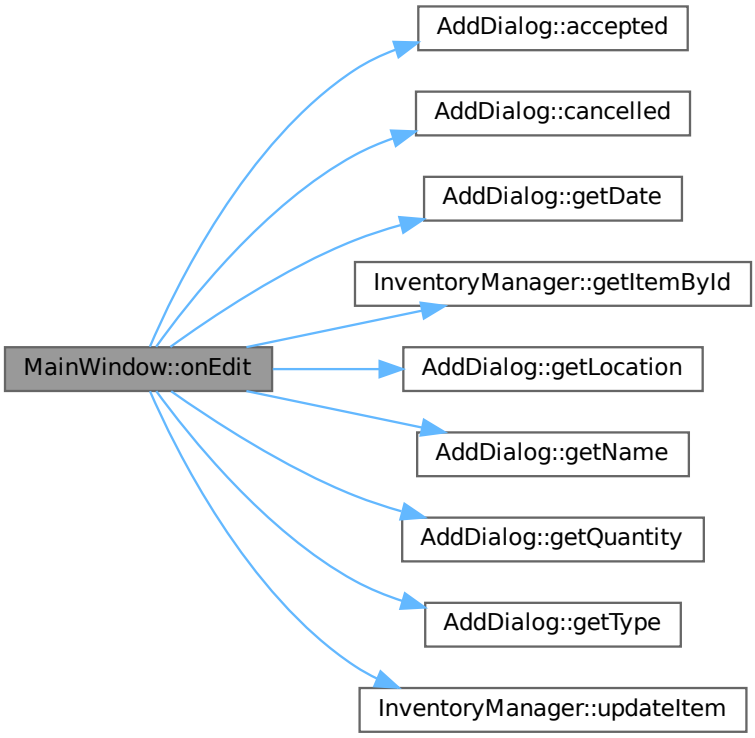
Definition at line 263 of file mainwindow.cpp.

```
00264 {
00265     QModelIndexList sel = tableView->selectionModel()->selectedRows();
00266     if (sel.isEmpty()) {
00267         QMessageBox::information(this, "Editar", "Por favor, selecciona una fila primero.");
00268         return;
00269     }
00270 
00271     // Mapeo del índice del proxy al índice del modelo fuente para obtener el ID real
00272     QModelIndex idx = sel.first();
00273     QModelIndex src = proxy->mapToSource(idx);
00274     int id = model->data(model->index(src.row(), 0)).toInt();
00275 
00276     InventoryItem it = manager.getItemById(id);
00277 
00278     AddDialog *dlg = new AddDialog();
00279     dlg->setWindowModality(Qt::ApplicationModal);
00280 
00281     // Precargar datos en los widgets del diálogo usando findChild
00282     dlg->findChild<QLineEdit*>("nameEdit")->setText(it.nombre);
00283     dlg->findChild<QLineEdit*>("typeEdit")->setText(it.tipo);
00284     dlg->findChild<QSpinBox*>("qtySpin")->setValue(it.cantidad);
00285     dlg->findChild<QLineEdit*>("locEdit")->setText(it.ubicacion);
00286     dlg->findChild<QDateEdit*>("dateEdit")->setDate(QDate::fromString(it.fechaAdquisicion,
      "yyyy-MM-dd"));
00287 
00288     dlg->show();
00289 
00290     // Lógica al aceptar el diálogo
00291     connect(dlg, &AddDialog::accepted, this, [this, dlg, id]() {
00292 
00293         if (dlg->getName().isEmpty() || dlg->getType().isEmpty()) {
00294             QMessageBox::warning(this, "Validación", "Nombre y tipo son campos obligatorios.");
00295             return;
00296         }
00297 
00298         if (!manager.updateItem(
00299                 id,
00300                 dlg->getName(),
00301                 dlg->getType(),
00302                 dlg->getQuantity(),
00303                 dlg->getLocation(),
00304                 dlg->getDate().toString("yyyy-MM-dd")))
00305         {
```

```
00306             QMessageBox::critical(this, "Error", "Fallo al actualizar el componente en la base de
      datos.");
00307         } else {
00308             refreshModel();
00309         }
00310
00311         dlg->close();
00312         dlg->deleteLater();
00313     });
00314
00315     connect(dlg, &AddDialog::cancelled, dlg, &AddDialog::close);
00316 }
```

References AddDialog::accepted(), AddDialog::cancelled(), InventoryItem::cantidad, InventoryItem::fechaAdquisicion, AddDialog::getDate(), InventoryManager::getItemById(), AddDialog::getLocation(), AddDialog::getName(), AddDialog::getQuantity(), AddDialog::getType(), manager, model, InventoryItem::nombre, proxy, tableView, InventoryItem::tipo, InventoryItem::ubicacion, and InventoryManager::updateItem().

Here is the call graph for this function:




Here is the caller graph for this function:

#### 4.8.3.5 onExport

```
void MainWindow::onExport ( )  [private], [slot]
```

Genera y exporta un reporte del inventario en formato CSV.

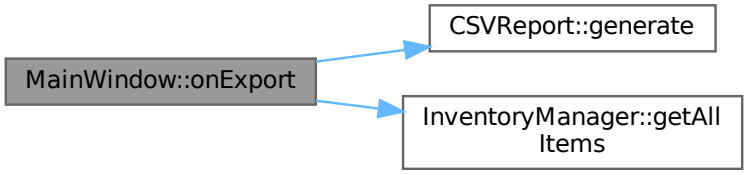
Abre un diálogo de sistema para seleccionar la ruta de guardado y utiliza la clase helper CSVReport para generar el archivo.

Definition at line 470 of file mainwindow.cpp.

```
00471 {
00472     QString filename = QFileDialog::getSaveFileName(
00473         this, "Guardar Reporte CSV", "reporte.csv", "Archivos CSV (*.csv)");
00474
00475     if (filename.isEmpty()) return;
00476
00477     CSVReport report;
00478     // Obtiene todos los items mediante el manager y genera el archivo
00479     if (report.generate(manager.getAllItems(), filename)) {
00480         QMessageBox::information(this, "Éxito", "El reporte ha sido exportado correctamente.");
00481     } else {
00482         QMessageBox::critical(this, "Error de Exportación", "No se pudo escribir el archivo en la ruta
      seleccionada.");
00483     }
00484 }
```

References CSVReport::generate(), InventoryManager::getAllItems(), and manager.

Here is the call graph for this function:




Here is the caller graph for this function:

#### 4.8.3.6 onLoadDefaults()

```
void MainWindow::onLoadDefaults ( )  [private]
```

Carga un conjunto de datos de prueba (Seed Data).

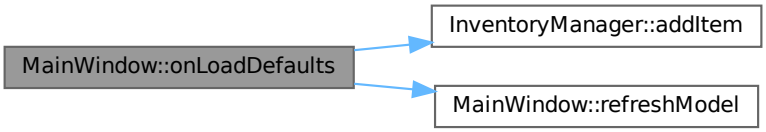
Inserta aproximadamente 50 ítems predefinidos en la base de datos. Útil para pruebas y demostraciones iniciales.

Definition at line 387 of file mainwindow.cpp.

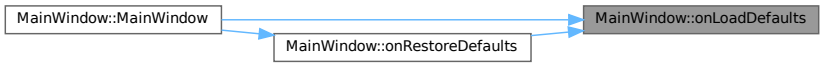
```
00388 {
00389     QList<QList<QString» defaults = {
00390       {"Resistor 10k", "Electrónico", "100", "Cajón A1", "2024-01-10"},
00391       {"Capacitor 100nF", "Electrónico", "80", "Cajón A1", "2024-01-12"},
00392       {"Diodo 1N4148", "Electrónico", "150", "Cajón A2", "2024-02-01"},
00393       {"LED Rojo 5mm", "Electrónico", "200", "Cajón A2", "2024-03-05"},
00394       {"Transistor 2N3904", "Electrónico", "120", "Cajón A3", "2024-03-10"},
00395       {"Potenciómetro 10k", "Electrónico", "15", "Cajón B1", "2024-02-20"},
00396       {"Motor DC 6V", "Electrónico", "10", "Estante B2", "2024-03-03"},
00397       {"Sensor HC-SR04", "Sensor", "12", "Estante C1", "2024-02-28"},
00398       {"Arduino Uno", "Microcontrolador", "4", "Estante C2", "2023-11-22"},
00399       {"Protoboard", "Herramienta", "10", "Estante C3", "2023-12-10"},
00400       {"Raspberry Pi Pico", "Microcontrolador", "6", "Estante C2", "2024-01-30"},
00401       {"Relé 5V", "Electrónico", "30", "Cajón A3", "2024-01-11"},
00402       {"Cable Dupont (m/m)", "Accesorio", "300", "Cajón A1", "2024-01-05"},
00403       {"Cable Dupont (h/h)", "Accesorio", "300", "Cajón A1", "2024-01-05"},
00404       {"Batería 9V", "Electrónico", "25", "Estante D1", "2024-02-10"},
00405       {"Multímetro Digital", "Instrumento", "3", "Mesa Taller", "2023-12-10"},
00406       {"Osciloscopio", "Instrumento", "1", "Mesa Taller", "2023-09-10"},
00407       {"Fuente DC 30V", "Instrumento", "2", "Mesa Taller", "2023-09-10"},
00408       {"Sensor PIR SR505", "Sensor", "20", "Estante C1", "2024-01-05"},
00409       {"Sensor DHT11", "Sensor", "15", "Estante C1", "2024-01-08"},
00410       {"Sensor DHT22", "Sensor", "10", "Estante C1", "2024-01-08"},
00411       {"ESP32-CAM", "Microcontrolador", "7", "Estante C2", "2024-02-01"},
00412       {"ESP8266", "Microcontrolador", "10", "Estante C2", "2024-02-02"},
00413       {"Cámara Web USB", "Computación", "5", "Estante D2", "2023-11-11"},
00414       {"Router TP-Link", "Red", "3", "Estante D2", "2023-12-01"},
00415       {"Switch Lógico", "Electrónico", "100", "Cajón A4", "2024-01-22"},
00416       {"Cables USB", "Accesorio", "30", "Cajón B3", "2023-12-28"},
00417       {"Rollo Cinta Aislante", "Herramienta", "20", "Cajón B3", "2023-12-30"},
00418       {"Soldador 60W", "Herramienta", "2", "Mesa Taller", "2023-12-15"},
00419       {"Estaño", "Consumible", "10", "Cajón B1", "2023-12-15"},
00420       {"Alcohol Isopropílico", "Limpieza", "4", "Estante E1", "2024-01-02"},
00421       {"Guantes Nitrilo", "Laboratorio", "200", "Armario F1", "2024-01-01"},
00422       {"Termómetro Digital", "Laboratorio", "3", "Estante E2", "2023-12-10"},
00423       {"Placa de Calentamiento", "Laboratorio", "1", "Estante E3", "2023-11-20"},
00424       {"Cronómetro", "Laboratorio", "2", "Cajón B4", "2023-12-09"},
00425       {"Lupa de Banco", "Herramienta", "2", "Mesa Taller", "2023-10-15"},
00426       {"Extensión Eléctrica", "Hogar", "8", "Estante D1", "2023-10-10"},
00427       {"Bombillo LED 12W", "Hogar", "20", "Estante D1", "2023-09-10"},
00428       {"Tomacorriente", "Hogar", "40", "Estante D1", "2023-09-10"},
00429       {"Interruptor de Pared", "Hogar", "40", "Estante D1", "2023-09-10"},
00430       {"Control Remoto IR", "Electrónico", "15", "Cajón A2", "2024-02-01"},
00431       {"Buzzer 5V", "Electrónico", "40", "Cajón A3", "2024-02-03"},
00432       {"Servo SG90", "Electrónico", "15", "Cajón A3", "2024-02-10"},
00433       {"Switch de Palanca", "Electrónico", "50", "Cajón A4", "2024-02-11"},
00434       {"Jack DC 5.5mm", "Electrónico", "60", "Cajón A4", "2024-02-11"},
00435       {"Pinzas de Cocodrilo", "Accesorio", "50", "Cajón B3", "2024-01-01"},
00436       {"Termistor NTC 10k", "Sensor", "80", "Cajón A2", "2024-01-20"}
00437     };
00438
00439     for (const auto &item : defaults) {
00440         manager.addItem(item[0], item[1], item[2].toInt(), item[3], item[4]);
00441     }
00442
00443     refreshModel();
00444     QMessageBox::information(this, "Carga Completa", "Se han cargado exitosamente los componentes por
      defecto.");
00445 }
```

References InventoryManager::addItem(), manager, and refreshModel().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 4.8.3.7 onLowStock

`void MainWindow::onLowStock ( ) [private], [slot]`

Analiza el stock actual y resalta visualmente los ítems críticos.

1. Itera sobre todas las filas del modelo proxy.
2. Compara la cantidad con `lowStockThreshold`.
3. Si es menor, modifica el `Qt::BackgroundRole` de la celda para pintarla de rojo claro.
4. Genera una lista de nombres de productos bajos en stock y la muestra en un MessageBox.

Definition at line 494 of file mainwindow.cpp.

```
00495 {
00496     QStringList lowStockItems;
00497
00498     for (int r = 0; r < proxy->rowCount(); r++) {
00499         // Índice de la columna 'Cantidad' (columna 3)
00500         QModelIndex idxQty = proxy->index(r, 3);
00501         int qty = idxQty.data().toInt();
00502
00503         if (qty < lowStockThreshold) {
00504             // Pintar toda la fila (iterando columnas)
00505             for (int c = 0; c < proxy->columnCount(); c++) {
00506                 tableView->model()->setData(
00507                     proxy->index(r, c),
00508                     QColor(255, 200, 200), // Rojo claro
00509                     Qt::BackgroundRole
00510                 );
00511             }
00512
00513             // Agregar nombre a la lista de alerta
00514             QModelIndex idxName = proxy->index(r, 1);
00515             lowStockItems << idxName.data().toString();
00516         }
00517     }
00518
00519     if (!lowStockItems.isEmpty()) {
```

```
00520         QMessageBox::warning(this, "Alerta de Stock Bajo",
00521             "Los siguientes ítems están por debajo del mínimo:\n\n" + lowStockItems.join("\n"));
00522     } else {
00523         QMessageBox::information(this, "Stock Saludable", "No hay elementos con stock bajo.");
00524     }
00525 }
```

References lowStockThreshold, proxy, and tableView.

Here is the caller graph for this function:


#### 4.8.3.8 onRestoreDefaults()

```
void MainWindow::onRestoreDefaults ( )  [private]
```

Restaura la base de datos a su estado original (vacía y luego recargada).

**Warning**

Esta acción ejecuta un `DELETE FROM inventario`, borrando todos los datos existentes.

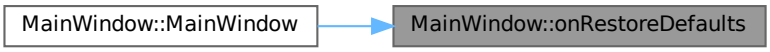
Definition at line 451 of file mainwindow.cpp.

```
00452 {
00453     if (QMessageBox::question(this, "Restaurar Fábrica",
00454                               "¿Seguro que deseas borrar TODO el inventario y restaurar los datos por
      defecto?\nEsta acción no se puede deshacer.")
00455         != QMessageBox::Yes)
00456         return;
00457     QSqlQuery q;
00458     q.exec("DELETE FROM inventario");
00459
00460
00461     onLoadDefaults();
00462     QMessageBox::information(this, "Restauración", "La base de datos ha sido restaurada.");
00463 }
```

References onLoadDefaults().

Here is the call graph for this function:

Here is the caller graph for this function:




#### 4.8.3.9 onSearch

```
void MainWindow::onSearch (
            const QString & text )  [private], [slot]
```

Filtra la tabla en tiempo real según el texto ingresado.

**Parameters**

| | |
|---|---|
| *text* | Cadena de búsqueda. Se busca coincidencia en cualquier columna. |

Definition at line 531 of file mainwindow.cpp.

```
00532 {
00533     proxy->setFilterFixedString(text);
00534 }
```

References proxy.

Here is the caller graph for this function:


#### 4.8.3.10 refreshModel()

```
void MainWindow::refreshModel ( )  [private]
```

Actualiza la vista recargando los datos desde la base de datos SQL.

Ejecuta de nuevo la consulta `SELECT` sobre el modelo y reasigna los nombres de las cabeceras para una visualización amigable.

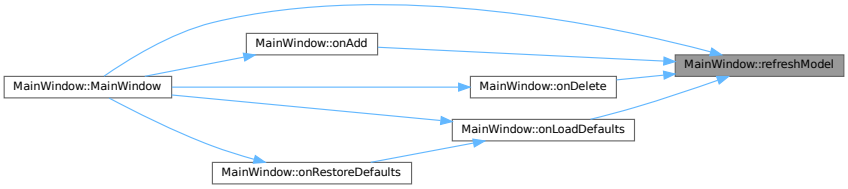
Definition at line 541 of file mainwindow.cpp.

```
00542 {
00543     model->setQuery("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion "
00544                     "FROM inventario ORDER BY id DESC");
00545
```

```
00546     model->setHeaderData(0, Qt::Horizontal, "ID");
00547     model->setHeaderData(1, Qt::Horizontal, "Nombre");
00548     model->setHeaderData(2, Qt::Horizontal, "Tipo");
00549     model->setHeaderData(3, Qt::Horizontal, "Cantidad");
00550     model->setHeaderData(4, Qt::Horizontal, "Ubicación");
00551     model->setHeaderData(5, Qt::Horizontal, "Fecha Adquisición");
00552
00553     tableView->resizeColumnsToContents();
00554 }
```

References model, and tableView.

Here is the caller graph for this function:




## 4.8.4 Member Data Documentation

### 4.8.4.1 lowStockThreshold

```
const int MainWindow::lowStockThreshold = 5  [private]
```

Definition at line 111 of file mainwindow.h.

### 4.8.4.2 manager

```
InventoryManager MainWindow::manager  [private]
```

Definition at line 105 of file mainwindow.h.

### 4.8.4.3 model

```
QSqlQueryModel* MainWindow::model  [private]
```

Definition at line 106 of file mainwindow.h.

### 4.8.4.4 proxy

```
QSortFilterProxyModel* MainWindow::proxy  [private]
```

Definition at line 107 of file mainwindow.h.

#### 4.8.4.5 searchEdit

```
QLineEdit* MainWindow::searchEdit  [private]
```

Definition at line 109 of file mainwindow.h.

#### 4.8.4.6 tableView

```
QTableView* MainWindow::tableView  [private]
```

Definition at line 108 of file mainwindow.h.

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

- include/mainwindow.h
- src/mainwindow.cpp

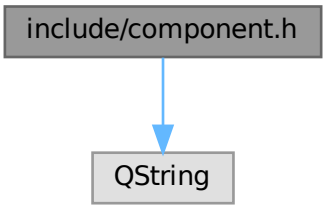
# Chapter 5

# File Documentation

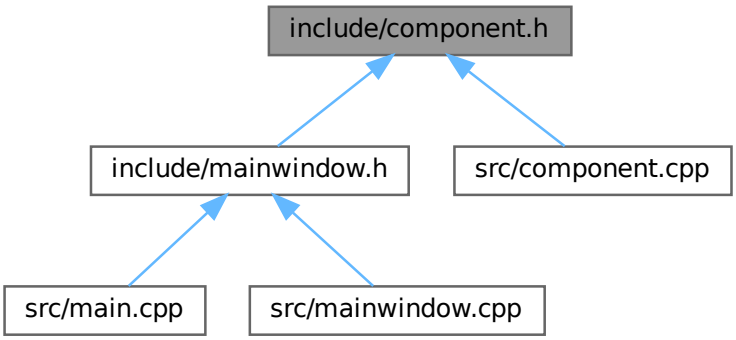
## 5.1 include/component.h File Reference

```
#include <QString>
```

Include dependency graph for component.h:




This graph shows which files directly or indirectly include this file:



### Classes

- class Component

  *Representa un componente almacenado en el inventario.*

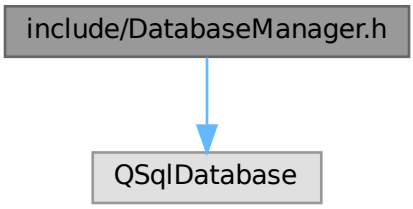
## 5.2 component.h

Go to the documentation of this file.

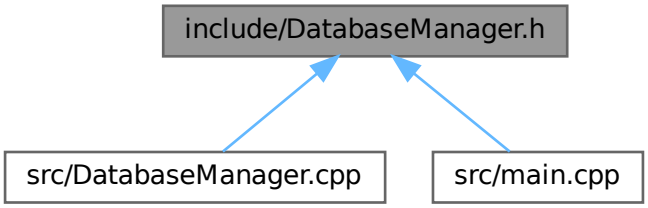
```
00001 #ifndef COMPONENT_H
00002 #define COMPONENT_H
00003
00004 #include <QString>
00005
00015 class Component
00016 {
00017 public:
00023     Component();
00024
00034     Component(int id,
00035               const QString &name,
00036               const QString &type,
00037               int quantity,
00038               const QString &location,
00039               const QString &purchase_date);
00040
00042     int getId() const;
00043
00045     QString getName() const;
00046
00048     QString getType() const;
00049
00051     int getQuantity() const;
00052
00054     QString getLocation() const;
00055
00057     QString getPurchaseDate() const;
00058
00060     void setId(int id);
00061
00063     void setName(const QString &name);
00064
00066     void setType(const QString &type);
00067
00069     void setQuantity(int quantity);
00070
00072     void setLocation(const QString &location);
00073
00075     void setPurchaseDate(const QString &purchase_date);
00076
00077 private:
00078     int m_id;
00079     QString m_name;
00080     QString m_type;
00081     int m_quantity;
00082     QString m_location;
00083     QString m_purchase_date;
00084 };
00085
00086 #endif // COMPONENT_H
```

## 5.3 include/DatabaseManager.h File Reference

`#include <QSqlDatabase>`
Include dependency graph for DatabaseManager.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class DatabaseManager

  *Clase encargada de gestionar la conexión con la base de datos.*

## 5.4 DatabaseManager.h

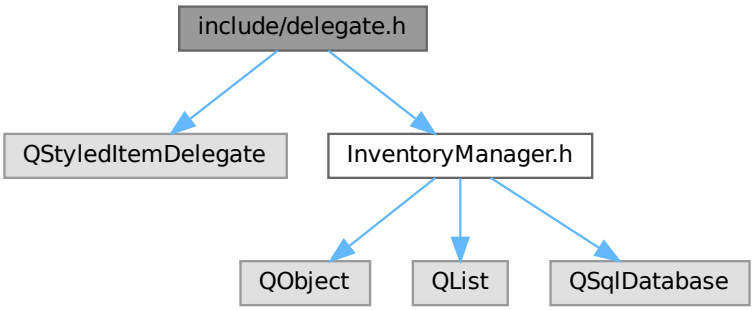
Go to the documentation of this file.

```
00001 #ifndef DATABASEMANAGER_H
00002 #define DATABASEMANAGER_H
00003
00004 #include <QSqlDatabase>
00005
00018 class DatabaseManager
00019 {
00020 public:
00029     static QSqlDatabase getDatabase();
00030
00031 private:
00038     static QSqlDatabase db;
00039 };
00040
00041 #endif // DATABASEMANAGER_H
```

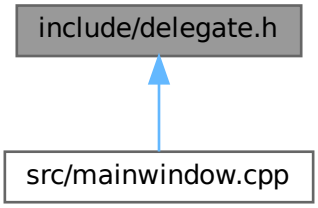
## 5.5 include/delegate.h File Reference

```
#include <QStyledItemDelegate>
#include "InventoryManager.h"
```

Include dependency graph for delegate.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class LowStockDelegate

  *Delegate que resalta en rojo los elementos con bajo stock.*

## 5.6 delegate.h

Go to the documentation of this file.

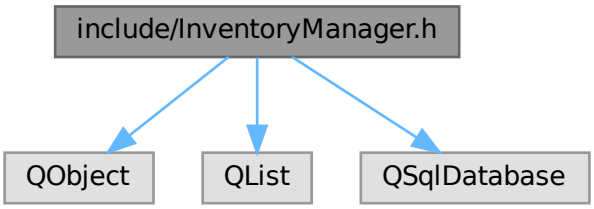
```
00001 #ifndef LOWSTOCKDELEGATE_H
00002 #define LOWSTOCKDELEGATE_H
00003
00004 #include <QStyledItemDelegate>
00005 #include "InventoryManager.h"
00006
00017 class LowStockDelegate : public QStyledItemDelegate
```

```
00018 {
00019 public:
00026     LowStockDelegate(int threshold, QObject *parent = nullptr)
00027         : QStyledItemDelegate(parent), limit(threshold) {}
00028
00040     void paint(QPainter *painter,
00041                const QStyleOptionViewItem &option,
00042                const QModelIndex &index) const override
00043     {
00044         QStyleOptionViewItem opt(option);
00045
00046         // Columna 3 → cantidad del componente
00047         if (index.column() == 3 && index.data().toInt() < limit) {
00048             opt.palette.setColor(QPalette::Text, Qt::red);
00049         }
00050
00051         QStyledItemDelegate::paint(painter, opt, index);
00052     }
00053
00054 private:
00055     int limit;
00056 };
00057
00058 #endif // LOWSTOCKDELEGATE_H
```

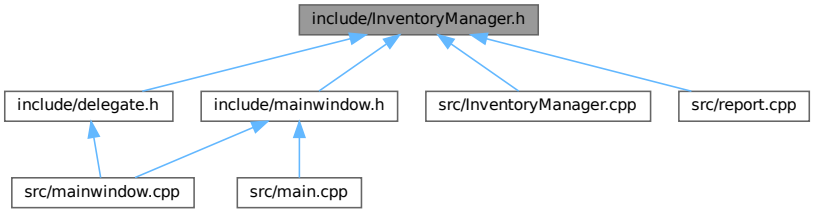
## 5.7 include/InventoryManager.h File Reference

```
#include <QObject>
#include <QList>
#include <QSqlDatabase>
```

Include dependency graph for InventoryManager.h:




This graph shows which files directly or indirectly include this file:



### Classes

- struct InventoryItem

  *Estructura que representa un elemento del inventario.*
- class InventoryManager

## 5.8 InventoryManager.h

Go to the documentation of this file.

```
00001 #ifndef INVENTORYMANAGER_H
00002 #define INVENTORYMANAGER_H
00003
00004 #include <QObject>
00005 #include <QList>
00006 #include <QSqlDatabase>
00007
00008 /*
00009  * Estructura que representa un ítem dentro del inventario.
00010  * Contiene toda la información necesaria para leer o escribir
00011  * un registro desde la base de datos.
00012  */
00013 struct InventoryItem {
00014     int id;                     // Identificador único del ítem (PRIMARY KEY)
00015     QString nombre;             // Nombre del componente o elemento
00016     QString tipo;               // Tipo o categoría del ítem
00017     int cantidad;               // Cantidad disponible en inventario
00018     QString ubicacion;          // Ubicación física dentro del almacén
00019     QString fechaAdquisicion;   // Fecha en que se adquirió el ítem
00020 };
00021
00022 /*
00023  * Clase InventoryManager
00024  * ----------------------
00025  * Administra la comunicación entre la aplicación y la base de datos SQLite.
00026  * Permite crear la tabla, agregar, editar, eliminar y consultar ítems.
00027  * Se utiliza como capa intermedia entre la lógica del programa y el motor SQL.
00028  */
00029 class InventoryManager : public QObject
00030 {
00031     Q_OBJECT
00032
00033 public:
00034     /*
00035      * Constructor del administrador de inventario.
00036      * Recibe una referencia a la base de datos ya inicializada.
00037      */
00038     explicit InventoryManager(QSqlDatabase database,
00039                               QObject *parent = nullptr);
00040
00041     /*
00042      * Crea la tabla del inventario si no existe.
00043      * Retorna true si la operación es exitosa.
00044      */
00045     bool createTable();
00046
00047     /*
00048      * Inserta un nuevo item en la base de datos.
00049      * Los parámetros corresponden a cada columna de la tabla.
00050      */
00051     bool addItem(const QString &nombre,
00052                  const QString &tipo,
00053                  int cantidad,
00054                  const QString &ubicacion,
00055                  const QString &fechaAdquisicion);
00056
00057     /*
00058      * Actualiza únicamente la cantidad de un ítem según su ID.
00059      */
00060     bool updateQuantity(int id, int newQuantity);
00061
00062     /*
00063      * Elimina un ítem del inventario por ID.
00064      */
00065     bool removeItem(int id);
00066
00067     /*
00068      * Obtiene todos los ítems de la tabla en forma de lista.
00069      */
00070     QList<InventoryItem> getAllItems();
```

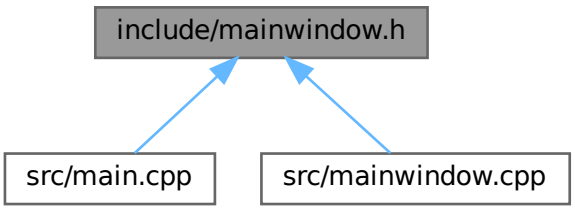
```
00071
00072     /*
00073      * Actualiza un ítem completo según su ID.
00074      */
00075     bool updateItem(int id,
00076                     const QString &nombre,
00077                     const QString &tipo,
00078                     int cantidad,
00079                     const QString &ubicacion,
00080                     const QString &fechaAdquisicion);
00081
00082     /*
00083      * Devuelve un solo ítem según su ID.
00084      * Si no existe, retorna un struct vacío.
00085      */
00086     InventoryItem getItemById(int id);
00087
00088 private:
00089     QSqlDatabase db;    // Conexión activa a la base de datos SQLite
00090 };
00091
00092 #endif // INVENTORYMANAGER_H
```

## 5.9 include/mainwindow.h File Reference

```
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QDateEdit>
#include <QTableView>
#include <QSqlQueryModel>
#include <QSortFilterProxyModel>
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QMessageBox>
#include "InventoryManager.h"
#include "component.h"
#include "report.h"
```

Include dependency graph for mainwindow.h:


This graph shows which files directly or indirectly include this file:



### Classes

- class AddDialog

  *Diálogo modal para la creación y edición de componentes.*
- class MainWindow

  *Controlador principal de la aplicación de gestión de inventario.*

## 5.10 mainwindow.h

Go to the documentation of this file.

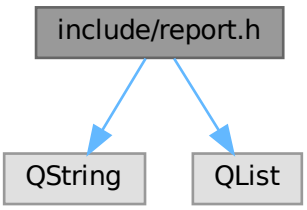
```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QWidget>
00005 #include <QLineEdit>
00006 #include <QSpinBox>
00007 #include <QDateEdit>
00008 #include <QTableView>
00009 #include <QSqlQueryModel>
00010 #include <QSortFilterProxyModel>
00011 #include <QPushButton>
00012 #include <QVBoxLayout>
00013 #include <QHBoxLayout>
00014 #include <QLabel>
00015 #include <QMessageBox>
00016
00017 #include "InventoryManager.h"
00018 #include "component.h"
00019 #include "report.h"
00020
00021 /*
00022  * Clase AddDialog
00023  * ---------------
00024  * Ventana pequeña para agregar un nuevo ítem al inventario.
00025  * Contiene campos de entrada para nombre, tipo, cantidad, ubicación y fecha.
00026  * Emite señales accepted() y cancelled() según la acción del usuario.
00027  */
00028 class AddDialog : public QWidget {
00029     Q_OBJECT
00030 public:
00031     explicit AddDialog(QWidget *parent = nullptr);
00032
00033     // Métodos para obtener los valores ingresados por el usuario
00034     QString getName() const;
00035     QString getType() const;
00036     int getQuantity() const;
00037     QString getLocation() const;
00038     QDate getDate() const;
00039
00040 signals:
00041     void accepted();   // Se emite cuando el usuario confirma
00042     void cancelled();  // Se emite cuando el usuario cancela
00043
00044 private slots:
00045     void onOk();       // Maneja el botón de OK
00046     void onCancel();   // Maneja el botón de cancelar
00047
00048 private:
00049     // Widgets de entrada
00050     QLineEdit *nameEdit;
00051     QLineEdit *typeEdit;
00052     QSpinBox  *qtySpin;
00053     QLineEdit *locEdit;
00054     QDateEdit *dateEdit;
00055 };
00056
00057
00058 /*
00059  * Clase MainWindow
00060  * ----------------
00061  * Es la ventana principal del sistema de inventario.
00062  * Se encarga de mostrar los datos, filtrarlos, agregar, editar,
00063  * eliminar, exportar reportes y manejar el estado del inventario.
00064  */
00065 class MainWindow : public QWidget {
00066     Q_OBJECT
00067 public:
00068     // Recibe la base de datos ya abierta y lista para usarse
```

```
00069     explicit MainWindow(QSqlDatabase db, QWidget *parent = nullptr);
00070
00071 private slots:
00072     // Acciones asociadas a los botones de la UI
00073     void onEdit();
00074     void onAdd();
00075     void onDelete();
00076     void onExport();
00077     void onLowStock();
00078     void onSearch(const QString &text);  // Filtro de búsqueda en tiempo real
00079
00080 private:
00081     /*
00082      * Refresca los datos del modelo leyendo desde InventoryManager.
00083      * Se usa después de agregar, editar o eliminar ítems.
00084      */
00085     void refreshModel();
00086
00087     /*
00088      * Revisa al iniciar si hay items con pocas existencias.
00089      * Si los hay, muestra una alerta al usuario.
00090      */
00091     void checkLowStockOnStart();
00092
00093     /*
00094      * Carga datos iniciales por defecto para pruebas.
00095      * (Dependiendo de tu implementación en el .cpp)
00096      */
00097     void onLoadDefaults();
00098
00099     /*
00100      * Restaura valores iniciales del sistema.
00101      */
00102     void onRestoreDefaults();
00103
00104 private:
00105     InventoryManager manager;       // Administrador de inventario (capa de BD)
00106     QSqlQueryModel *model;          // Modelo base conectado a SQL
00107     QSortFilterProxyModel *proxy;   // Modelo para búsqueda y filtrado
00108     QTableView *tableView;          // Tabla que muestra los ítems
00109     QLineEdit *searchEdit;          // Barra de búsqueda
00110
00111     const int lowStockThreshold = 5;  // Cantidad mínima antes de considerarse "bajo stock"
00112 };
00113
00114 #endif // MAINWINDOW_H
```

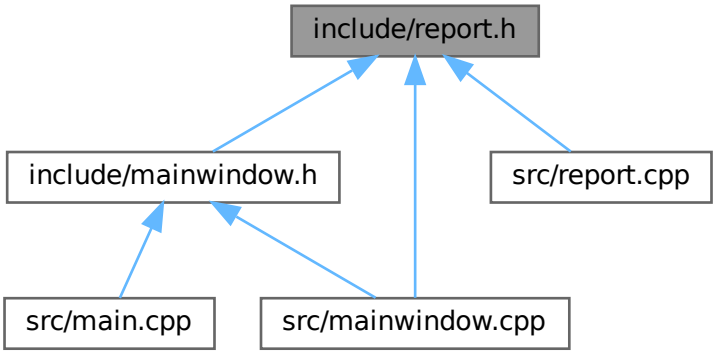
## 5.11 include/report.h File Reference

```
#include <QString>
#include <QList>
```

Include dependency graph for report.h:



This graph shows which files directly or indirectly include this file:




### Classes

- class CSVReport

  *Clase encargada de generar reportes CSV del inventario.*

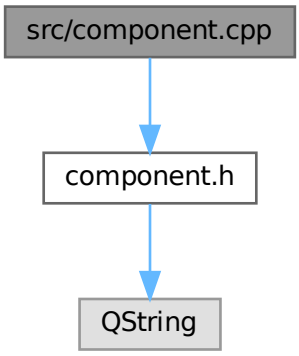
## 5.12 report.h

Go to the documentation of this file.

```
00001
00010 #ifndef CSVREPORT_H
00011 #define CSVREPORT_H
00012
00013 #include <QString>
00014 #include <QList>
00015
00023 struct InventoryItem;
00024
00032 class CSVReport
00033 {
00034 public:
00041     CSVReport();
00042
00054     bool generate(const QList<InventoryItem> &items,
00055                   const QString &filePath);
00056 };
00057
00058 #endif // CSVREPORT_H
00059
```

## 5.13 src/component.cpp File Reference

`#include "component.h"`
Include dependency graph for component.cpp:




## 5.14 component.cpp

Go to the documentation of this file.

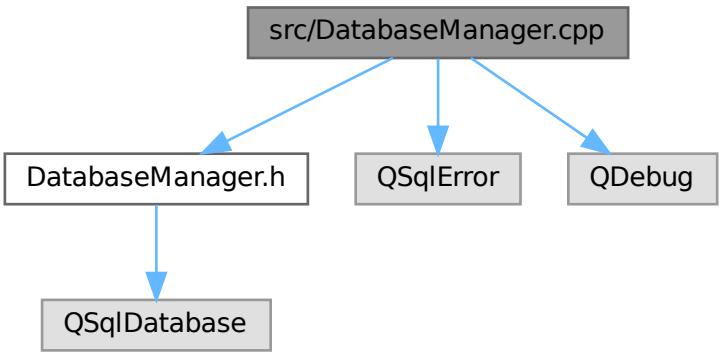
```
00001 #include "component.h"
00002
00012 Component::Component()
00013     : m_id(0),
00014       m_quantity(0)
00015 {
00016 }
00017
00028 Component::Component(int id,
00029                      const QString &name,
00030                      const QString &type,
00031                      int quantity,
00032                      const QString &location,
00033                      const QString &purchase_date)
00034     : m_id(id),
00035       m_name(name),
00036       m_type(type),
00037       m_quantity(quantity),
00038       m_location(location),
00039       m_purchase_date(purchase_date)
00040 {
00041 }
00042
00047 int Component::getId() const { return m_id; }
00048
00053 QString Component::getName() const { return m_name; }
00054
00059 QString Component::getType() const { return m_type; }
00060
00065 int Component::getQuantity() const { return m_quantity; }
00066
00071 QString Component::getLocation() const { return m_location; }
00072
00077 QString Component::getPurchaseDate() const { return m_purchase_date; }
00078
00083 void Component::setId(int id) { m_id = id; }
00084
00089 void Component::setName(const QString &name) { m_name = name; }
00090
00095 void Component::setType(const QString &type) { m_type = type; }
00096
```

```
00101 void Component::setQuantity(int quantity) { m_quantity = quantity; }
00102
00107 void Component::setLocation(const QString &location) { m_location = location; }
00108
00113 void Component::setPurchaseDate(const QString &purchase_date) { m_purchase_date = purchase_date; }
```

## 5.15 src/DatabaseManager.cpp File Reference

```
#include "DatabaseManager.h"
#include <QSqlError>
#include <QDebug>
```

Include dependency graph for DatabaseManager.cpp:




## 5.16 DatabaseManager.cpp

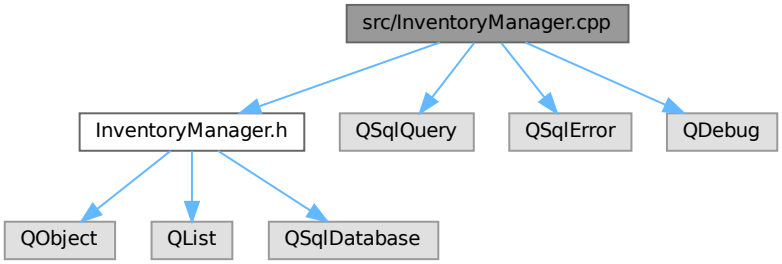
Go to the documentation of this file.

```
00001 #include "DatabaseManager.h"
00002 #include <QSqlError>
00003 #include <QDebug>
00004
00011 QSqlDatabase DatabaseManager::db = QSqlDatabase();
00012
00027 QSqlDatabase DatabaseManager::getDatabase()
00028 {
00029     // Si la instancia aún no es válida, configurar el driver
00030     if (!db.isValid()) {
00031         db = QSqlDatabase::addDatabase("QSQLITE");
00032         db.setDatabaseName("inventario.db");
00033     }
00034
00035     // Abrir la base de datos si aún no está abierta
00036     if (!db.isOpen()) {
00037         if (!db.open()) {
00038             qDebug() « "ERROR al abrir la base de datos:" « db.lastError();
00039         } else {
00040             qDebug() « "Base de datos abierta correctamente.";
00041         }
00042     }
00043
00044     return db;
00045 }
```

## 5.17 src/InventoryManager.cpp File Reference

```
#include "InventoryManager.h"
#include <QSqlQuery>
#include <QSqlError>
#include <QDebug>
```

Include dependency graph for InventoryManager.cpp:




## 5.18 InventoryManager.cpp

Go to the documentation of this file.

```
00001 #include "InventoryManager.h"
00002 #include <QSqlQuery>
00003 #include <QSqlError>
00004 #include <QDebug>
00005
00017 InventoryManager::InventoryManager(QSqlDatabase database, QObject *parent)
00018     : QObject(parent), db(database)
00019 {
00020     if (!db.isValid()) {
00021         qFatal("ERROR FATAL: Base de datos no válida. "
00022                "Asegúrate de usar DatabaseManager::getDatabase().");
00023     }
00024 }
00025
00039 bool InventoryManager::createTable()
00040 {
00041     QSqlQuery query(db);
00042
00043     QString sql =
00044         "CREATE TABLE IF NOT EXISTS inventario ("
00045         "id INTEGER PRIMARY KEY AUTOINCREMENT,"
00046         "nombre TEXT NOT NULL,"
00047         "tipo TEXT NOT NULL,"
00048         "cantidad INTEGER NOT NULL,"
00049         "ubicacion TEXT NOT NULL,"
00050         "fechaAdquisicion TEXT NOT NULL"
00051         ");";
00052
00053     return query.exec(sql);
00054 }
00055
00067 bool InventoryManager::addItem(const QString &nombre,
00068                                const QString &tipo,
00069                                int cantidad,
00070                                const QString &ubicacion,
00071                                const QString &fechaAdquisicion)
00072 {
00073     QSqlQuery query(db);
00074
00075     query.prepare(
00076         "INSERT INTO inventario "
00077         "(nombre, tipo, cantidad, ubicacion, fechaAdquisicion) "
00078         "VALUES (?, ?, ?, ?, ?)"
```

```
00079     );
00080
00081     query.addBindValue(nombre);
00082     query.addBindValue(tipo);
00083     query.addBindValue(cantidad);
00084     query.addBindValue(ubicacion);
00085     query.addBindValue(fechaAdquisicion);
00086
00087     return query.exec();
00088 }
00089
00098 bool InventoryManager::updateQuantity(int id, int newQuantity)
00099 {
00100     QSqlQuery query(db);
00101     query.prepare("UPDATE inventario SET cantidad = ? WHERE id = ?");
00102     query.addBindValue(newQuantity);
00103     query.addBindValue(id);
00104     return query.exec();
00105 }
00106
00114 bool InventoryManager::removeItem(int id)
00115 {
00116     QSqlQuery query(db);
00117     query.prepare("DELETE FROM inventario WHERE id = ?");
00118     query.addBindValue(id);
00119     return query.exec();
00120 }
00121
00127 QList<InventoryItem> InventoryManager::getAllItems()
00128 {
00129     QList<InventoryItem> items;
00130     QSqlQuery query(db);
00131
00132     query.exec("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion FROM inventario");
00133
00134     while (query.next()) {
00135         InventoryItem it;
00136         it.id = query.value(0).toInt();
00137         it.nombre = query.value(1).toString();
00138         it.tipo = query.value(2).toString();
00139         it.cantidad = query.value(3).toInt();
00140         it.ubicacion = query.value(4).toString();
00141         it.fechaAdquisicion = query.value(5).toString();
00142         items.append(it);
00143     }
00144     return items;
00145 }
00146
00159 bool InventoryManager::updateItem(int id,
00160                                   const QString &nombre,
00161                                   const QString &tipo,
00162                                   int cantidad,
00163                                   const QString &ubicacion,
00164                                   const QString &fechaAdquisicion)
00165 {
00166     QSqlQuery query(db);
00167
00168     query.prepare(
00169         "UPDATE inventario SET "
00170         "nombre = ?, tipo = ?, cantidad = ?, ubicacion = ?, fechaAdquisicion = ? "
00171         "WHERE id = ?"
00172     );
00173
00174     query.addBindValue(nombre);
00175     query.addBindValue(tipo);
00176     query.addBindValue(cantidad);
00177     query.addBindValue(ubicacion);
00178     query.addBindValue(fechaAdquisicion);
00179     query.addBindValue(id);
00180
00181     return query.exec();
00182 }
00183
00192 InventoryItem InventoryManager::getItemById(int id)
00193 {
00194     InventoryItem it;
00195
00196     QSqlQuery query(db);
00197     query.prepare("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion "
00198                   "FROM inventario WHERE id = ?");
00199     query.addBindValue(id);
00200
00201     if (!query.exec() || !query.next()) {
00202         return it;  // vacío si no se encuentra
00203     }
00204
00205     it.id = query.value(0).toInt();
```

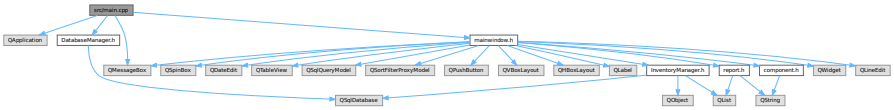
```
00206     it.nombre = query.value(1).toString();
00207     it.tipo = query.value(2).toString();
00208     it.cantidad = query.value(3).toInt();
00209     it.ubicacion = query.value(4).toString();
00210     it.fechaAdquisicion = query.value(5).toString();
00211
00212     return it;
00213 }
```

## 5.19 src/main.cpp File Reference

Punto de entrada principal de la aplicación de inventario.

```
#include <QApplication>
#include <QMessageBox>
#include "DatabaseManager.h"
#include "mainwindow.h"
```

Include dependency graph for main.cpp:




### Functions

- int main (int argc, char ∗argv[ ])

  *Función principal de la aplicación.*

### 5.19.1 Detailed Description

Punto de entrada principal de la aplicación de inventario.

Este archivo inicializa el entorno Qt, establece la conexión con la base de datos mediante DatabaseManager y crea la ventana principal de la aplicación.

Definition in file main.cpp.

### 5.19.2 Function Documentation

#### 5.19.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Función principal de la aplicación.

Inicializa QApplication, intenta abrir la base de datos y crea la ventana principal si la conexión es exitosa.

**Parameters**

| | |
|---|---|
| *argc* | Número de argumentos de línea de comandos. |
| *argv* | Arreglo con los argumentos de línea de comandos. |

**Returns**

int Código de salida de la aplicación.

Definition at line 24 of file main.cpp.

```
00025 {
00026     QApplication app(argc, argv);
00027
00028     // Obtener la conexión a la base de datos desde DatabaseManager
00029     QSqlDatabase db = DatabaseManager::getDatabase();
00030
00031     // Verificar si la base de datos se abrió correctamente
00032     if (!db.isValid() || !db.isOpen()) {
00033         QMessageBox::critical(nullptr, "Error", "No se pudo abrir la base de datos.");
00034         return -1;
00035     }
00036
00037     // Crear y mostrar la ventana principal
00038     MainWindow w(db);
00039     w.show();
00040
00041     return app.exec();
00042 }
```

References DatabaseManager::getDatabase().

Here is the call graph for this function:


## 5.20 main.cpp

Go to the documentation of this file.

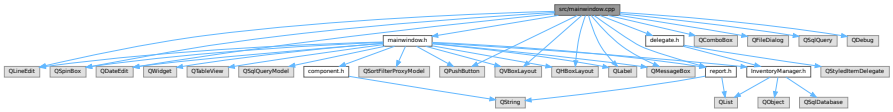
```
00001
00009 #include <QApplication>
00010 #include <QMessageBox>
00011 #include "DatabaseManager.h"
00012 #include "mainwindow.h"
00013
00024 int main(int argc, char *argv[])
00025 {
00026     QApplication app(argc, argv);
00027
00028     // Obtener la conexión a la base de datos desde DatabaseManager
00029     QSqlDatabase db = DatabaseManager::getDatabase();
00030
00031     // Verificar si la base de datos se abrió correctamente
00032     if (!db.isValid() || !db.isOpen()) {
00033         QMessageBox::critical(nullptr, "Error", "No se pudo abrir la base de datos.");
00034         return -1;
00035     }
00036
00037     // Crear y mostrar la ventana principal
00038     MainWindow w(db);
00039     w.show();
00040
00041     return app.exec();
00042 }
00043
```

## 5.21 src/mainwindow.cpp File Reference

Implementación de la lógica de interfaz gráfica para la gestión de inventario.

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QSpinBox>
#include <QComboBox>
#include <QDateEdit>
#include <QPushButton>
#include <QMessageBox>
#include <QFileDialog>
#include <QSqlQuery>
#include <QDebug>
#include "report.h"
#include "delegate.h"
```

Include dependency graph for mainwindow.cpp:




### Functions

- static QString dateToString (const QDate &d)

  *Convierte un objeto QDate a una cadena con formato estándar de base de datos.*
- static Component inventoryItemToComponent (const InventoryItem &it)

  *Adaptador que convierte una estructura de base de datos (InventoryItem) a un objeto de lógica (Component).*

### 5.21.1 Detailed Description

Implementación de la lógica de interfaz gráfica para la gestión de inventario.

**Author**

[Tu Nombre o Equipo de Desarrollo]

**Date**

2024

Este archivo contiene la implementación de las clases:

- AddDialog: Ventana modal para formularios de entrada.
- MainWindow: Ventana principal que orquesta la vista, el modelo SQL y la lógica de negocio.

Se utilizan componentes de Qt como QSqlTableModel, QSortFilterProxyModel y delegates personalizados para manejar la presentación de datos.

Definition in file mainwindow.cpp.

### 5.21.2 Function Documentation

#### 5.21.2.1 dateToString()

```
static QString dateToString (
            const QDate & d )  [static]
```

Convierte un objeto QDate a una cadena con formato estándar de base de datos.

**Parameters**

| | |
|---|---|
| *d* | Fecha a convertir. |

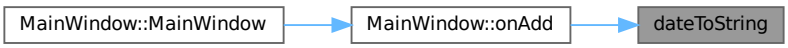
**Returns**

QString con formato "YYYY-MM-DD".

Definition at line 42 of file mainwindow.cpp.

```
00042                                           {
00043     return d.toString("yyyy-MM-dd");
00044 }
```

Here is the caller graph for this function:




#### 5.21.2.2 inventoryItemToComponent()

```
static Component inventoryItemToComponent (
            const InventoryItem & it )  [static]
```

Adaptador que convierte una estructura de base de datos (InventoryItem) a un objeto de lógica (Component).

**Parameters**

| | |
|---|---|
| *it* | Objeto InventoryItem recuperado del InventoryManager. |

**Returns**

Objeto Component listo para ser procesado por la lógica de negocio.

Definition at line 51 of file mainwindow.cpp.

```
00051                                                                   {
00052     return Component(it.id, it.nombre, it.tipo, it.cantidad, it.ubicacion, it.fechaAdquisicion);
00053 }
```

References InventoryItem::cantidad, InventoryItem::fechaAdquisicion, InventoryItem::id, InventoryItem::nombre, InventoryItem::tipo, and InventoryItem::ubicacion.

# 5.22 mainwindow.cpp

Go to the documentation of this file.

```
00001
00015 #include "mainwindow.h"
00016
00017 #include <QVBoxLayout>
00018 #include <QHBoxLayout>
00019 #include <QLabel>
00020 #include <QLineEdit>
00021 #include <QSpinBox>
00022 #include <QComboBox>
00023 #include <QDateEdit>
00024 #include <QPushButton>
00025 #include <QMessageBox>
00026 #include <QFileDialog>
00027 #include <QSqlQuery>
00028 #include <QDebug>
00029
00030 #include "report.h"
00031 #include "delegate.h"
00032
00033 // ============================================================================
00034 // FUNCIONES AUXILIARES ESTÁTICAS
00035 // ============================================================================
00036
00042 static QString dateToString(const QDate &d) {
00043     return d.toString("yyyy-MM-dd");
00044 }
00045
00051 static Component inventoryItemToComponent(const InventoryItem &it) {
00052     return Component(it.id, it.nombre, it.tipo, it.cantidad, it.ubicacion, it.fechaAdquisicion);
00053 }
00054
00055
00056 // ============================================================================
00057 // CLASE ADDDIALOG
00058 // ============================================================================
00059
00076 AddDialog::AddDialog(QWidget *parent) : QWidget(parent)
00077 {
00078     setWindowTitle("Agregar componente");
00079     QVBoxLayout *v = new QVBoxLayout(this);
00080
00081     // Campos de entrada
00082     nameEdit = new QLineEdit(); nameEdit->setPlaceholderText("Nombre");
00083     typeEdit = new QLineEdit(); typeEdit->setPlaceholderText("Tipo");
00084     qtySpin = new QSpinBox(); qtySpin->setRange(0, 1000000); qtySpin->setValue(1);
00085     locEdit = new QLineEdit(); locEdit->setPlaceholderText("Ubicación");
00086     dateEdit = new QDateEdit(QDate::currentDate()); dateEdit->setCalendarPopup(true);
00087
00088     QPushButton *btnOk = new QPushButton("Agregar");
00089     QPushButton *btnCancel = new QPushButton("Cancelar");
00090
00091     // Construcción del formulario
00092     v->addWidget(new QLabel("Nombre:"));    v->addWidget(nameEdit);
00093     v->addWidget(new QLabel("Tipo:"));      v->addWidget(typeEdit);
00094     v->addWidget(new QLabel("Cantidad:"));  v->addWidget(qtySpin);
00095     v->addWidget(new QLabel("Ubicación:")); v->addWidget(locEdit);
00096     v->addWidget(new QLabel("Fecha adquisición:")); v->addWidget(dateEdit);
00097
00098     // Botonera
00099     QHBoxLayout *h = new QHBoxLayout();
00100     h->addWidget(btnOk);
00101     h->addWidget(btnCancel);
00102     v->addLayout(h);
00103
00104     // Conexiones de señales
00105     connect(btnOk, &QPushButton::clicked, this, &AddDialog::onOk);
00106     connect(btnCancel, &QPushButton::clicked, this, &AddDialog::onCancel);
00107
00108     // Asignación de nombres de objeto para búsqueda dinámica (findChild)
00109     nameEdit->setObjectName("nameEdit");
00110     typeEdit->setObjectName("typeEdit");
00111     qtySpin->setObjectName("qtySpin");
00112     locEdit->setObjectName("locEdit");
00113     dateEdit->setObjectName("dateEdit");
00114 }
00115
00120 QString AddDialog::getName() const { return nameEdit->text().trimmed(); }
00121
00126 QString AddDialog::getType() const { return typeEdit->text().trimmed(); }
00127
00132 int AddDialog::getQuantity() const { return qtySpin->value(); }
00133
```

```
00138 QString AddDialog::getLocation() const { return locEdit->text().trimmed(); }
00139
00144 QDate AddDialog::getDate() const { return dateEdit->date(); }
00145
00150 void AddDialog::onOk() { emit accepted(); }
00151
00156 void AddDialog::onCancel() { emit cancelled(); }
00157
00158
00159 // ============================================================================
00160 // CLASE MAINWINDOW
00161 // ============================================================================
00162
00184 MainWindow::MainWindow(QSqlDatabase db, QWidget *parent)
00185     : QWidget(parent), manager(db)
00186 {
00187     setWindowTitle("Gestión de Inventario");
00188     resize(1000, 600);
00189
00190     QVBoxLayout *mainLayout = new QVBoxLayout(this);
00191     QHBoxLayout *topLayout = new QHBoxLayout();
00192
00193     // -- Inicialización de Widgets --
00194     searchEdit = new QLineEdit(); searchEdit->setPlaceholderText("Buscar...");
00195     QPushButton *btnAdd = new QPushButton("Agregar");
00196     QPushButton *btnDelete = new QPushButton("Eliminar seleccionado");
00197     QPushButton *btnExport = new QPushButton("Exportar CSV");
00198     QPushButton *btnLowStock = new QPushButton("Revisar stock bajo");
00199     QPushButton *btnLoadDefaults = new QPushButton("Cargar base por defecto");
00200     QPushButton *btnRestore = new QPushButton("Restaurar base original");
00201     QPushButton *btnEdit = new QPushButton("Editar");
00202
00203     // -- Construcción del Layout Superior --
00204     topLayout->addWidget(btnLoadDefaults);
00205     topLayout->addWidget(btnRestore);
00206     topLayout->addWidget(btnEdit);
00207     topLayout->addWidget(new QLabel("Buscar:"));
00208     topLayout->addWidget(searchEdit);
00209     topLayout->addWidget(btnAdd);
00210     topLayout->addWidget(btnDelete);
00211     topLayout->addWidget(btnLowStock);
00212     topLayout->addWidget(btnExport);
00213
00214     mainLayout->addLayout(topLayout);
00215
00216     // -- Configuración del Modelo MVC --
00217     model = new QSqlQueryModel(this);
00218
00219     // Configuración del Proxy para filtrado y ordenamiento
00220     proxy = new QSortFilterProxyModel(this);
00221     proxy->setSourceModel(model);
00222     proxy->setFilterCaseSensitivity(Qt::CaseInsensitive);
00223     proxy->setFilterKeyColumn(-1); // Filtrar en todas las columnas
00224
00225     // -- Configuración de la Tabla (Vista) --
00226     tableView = new QTableView();
00227     tableView->setModel(proxy);
00228     tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
00229     tableView->setSelectionMode(QAbstractItemView::SingleSelection);
00230     tableView->setEditTriggers(QAbstractItemView::NoEditTriggers); // Edición solo vía diálogo
00231
00232     // Inyección del delegado para resaltar stock bajo
00233     tableView->setItemDelegate(new LowStockDelegate(lowStockThreshold, this));
00234     mainLayout->addWidget(tableView);
00235
00236     // -- Conexiones de Señales y Slots --
00237     connect(btnAdd, &QPushButton::clicked, this, &MainWindow::onAdd);
00238     connect(btnDelete, &QPushButton::clicked, this, &MainWindow::onDelete);
00239     connect(btnExport, &QPushButton::clicked, this, &MainWindow::onExport);
00240     connect(btnLowStock, &QPushButton::clicked, this, &MainWindow::onLowStock);
00241     connect(searchEdit, &QLineEdit::textChanged, this, &MainWindow::onSearch);
00242     connect(btnLoadDefaults, &QPushButton::clicked, this, &MainWindow::onLoadDefaults);
00243     connect(btnRestore, &QPushButton::clicked, this, &MainWindow::onRestoreDefaults);
00244     connect(btnEdit, &QPushButton::clicked, this, &MainWindow::onEdit);
00245
00246     // Inicialización de la tabla en BD si no existe
00247     if (!manager.createTable()) {
00248         QMessageBox::critical(this, "Error Crítico", "No se pudo crear o verificar la tabla de
      inventario en la base de datos.");
00249     }
00250
00251     refreshModel();
00252 }
00253
00263 void MainWindow::onEdit()
00264 {
00265     QModelIndexList sel = tableView->selectionModel()->selectedRows();
```

```
00266     if (sel.isEmpty()) {
00267         QMessageBox::information(this, "Editar", "Por favor, selecciona una fila primero.");
00268         return;
00269     }
00270 
00271     // Mapeo del índice del proxy al índice del modelo fuente para obtener el ID real
00272     QModelIndex idx = sel.first();
00273     QModelIndex src = proxy->mapToSource(idx);
00274     int id = model->data(model->index(src.row(), 0)).toInt();
00275 
00276     InventoryItem it = manager.getItemById(id);
00277 
00278     AddDialog *dlg = new AddDialog();
00279     dlg->setWindowModality(Qt::ApplicationModal);
00280 
00281     // Precargar datos en los widgets del diálogo usando findChild
00282     dlg->findChild<QLineEdit*>("nameEdit")->setText(it.nombre);
00283     dlg->findChild<QLineEdit*>("typeEdit")->setText(it.tipo);
00284     dlg->findChild<QSpinBox*>("qtySpin")->setValue(it.cantidad);
00285     dlg->findChild<QLineEdit*>("locEdit")->setText(it.ubicacion);
00286     dlg->findChild<QDateEdit*>("dateEdit")->setDate(QDate::fromString(it.fechaAdquisicion,
    "yyyy-MM-dd"));
00287 
00288     dlg->show();
00289 
00290     // Lógica al aceptar el diálogo
00291     connect(dlg, &AddDialog::accepted, this, [this, dlg, id]() {
00292 
00293         if (dlg->getName().isEmpty() || dlg->getType().isEmpty()) {
00294             QMessageBox::warning(this, "Validación", "Nombre y tipo son campos obligatorios.");
00295             return;
00296         }
00297 
00298         if (!manager.updateItem(
00299                 id,
00300                 dlg->getName(),
00301                 dlg->getType(),
00302                 dlg->getQuantity(),
00303                 dlg->getLocation(),
00304                 dlg->getDate().toString("yyyy-MM-dd")))
00305         {
00306             QMessageBox::critical(this, "Error", "Fallo al actualizar el componente en la base de
    datos.");
00307         } else {
00308             refreshModel();
00309         }
00310 
00311         dlg->close();
00312         dlg->deleteLater();
00313     });
00314 
00315     connect(dlg, &AddDialog::cancelled, dlg, &AddDialog::close);
00316 }
00317 
00323 void MainWindow::onAdd()
00324 {
00325     AddDialog *dlg = new AddDialog();
00326     dlg->setWindowModality(Qt::ApplicationModal);
00327     dlg->show();
00328 
00329     connect(dlg, &AddDialog::accepted, this, [this, dlg]() {
00330         if (dlg->getName().isEmpty() || dlg->getType().isEmpty()) {
00331             QMessageBox::warning(this, "Validación", "Nombre y tipo son obligatorios.");
00332             return;
00333         }
00334 
00335         Component c(0,
00336                     dlg->getName(),
00337                     dlg->getType(),
00338                     dlg->getQuantity(),
00339                     dlg->getLocation(),
00340                     dateToString(dlg->getDate()));
00341 
00342         if (!manager.addItem(c.getName(), c.getType(), c.getQuantity(), c.getLocation(),
    c.getPurchaseDate())) {
00343             QMessageBox::critical(this, "Error", "No se pudo insertar el componente en la base de
    datos.");
00344         } else {
00345             refreshModel();
00346         }
00347         dlg->close();
00348         dlg->deleteLater();
00349     });
00350 
00351     connect(dlg, &AddDialog::cancelled, dlg, &AddDialog::close);
00352 }
00353 
```

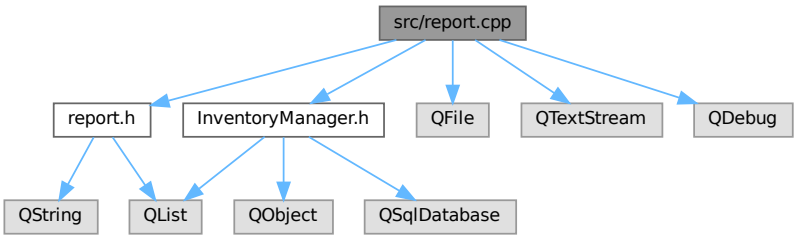
```
00359 void MainWindow::onDelete()
00360 {
00361     QModelIndexList sel = tableView->selectionModel()->selectedRows();
00362     if (sel.isEmpty()) {
00363         QMessageBox::information(this, "Eliminar", "Selecciona una fila para eliminar.");
00364         return;
00365     }
00366
00367     QModelIndex idx = sel.first();
00368     QModelIndex src = proxy->mapToSource(idx);
00369     int id = model->data(model->index(src.row(), 0)).toInt();
00370
00371     if (QMessageBox::question(this, "Confirmar Eliminación",
00372                               QString("¿Estás seguro de eliminar el registro con ID %1?").arg(id)) ==
      QMessageBox::Yes)
00373     {
00374         if (!manager.removeItem(id)) {
00375             QMessageBox::critical(this, "Error", "No se pudo eliminar el registro.");
00376         } else {
00377             refreshModel();
00378         }
00379     }
00380 }
00381
00387 void MainWindow::onLoadDefaults()
00388 {
00389     QList<QList<QString» defaults = {
00390       {"Resistor 10k", "Electrónico", "100", "Cajón A1", "2024-01-10"},
00391       {"Capacitor 100nF", "Electrónico", "80", "Cajón A1", "2024-01-12"},
00392       {"Diodo 1N4148", "Electrónico", "150", "Cajón A2", "2024-02-01"},
00393       {"LED Rojo 5mm", "Electrónico", "200", "Cajón A2", "2024-03-05"},
00394       {"Transistor 2N3904", "Electrónico", "120", "Cajón A3", "2024-03-10"},
00395       {"Potenciómetro 10k", "Electrónico", "15", "Cajón B1", "2024-02-20"},
00396       {"Motor DC 6V", "Electrónico", "10", "Estante B2", "2024-03-03"},
00397       {"Sensor HC-SR04", "Sensor", "12", "Estante C1", "2024-02-28"},
00398       {"Arduino Uno", "Microcontrolador", "4", "Estante C2", "2023-11-22"},
00399       {"Protoboard", "Herramienta", "10", "Estante C3", "2023-12-10"},
00400       {"Raspberry Pi Pico", "Microcontrolador", "6", "Estante C2", "2024-01-30"},
00401       {"Relé 5V", "Electrónico", "30", "Cajón A3", "2024-01-11"},
00402       {"Cable Dupont (m/m)", "Accesorio", "300", "Cajón A1", "2024-01-05"},
00403       {"Cable Dupont (h/h)", "Accesorio", "300", "Cajón A1", "2024-01-05"},
00404       {"Batería 9V", "Electrónico", "25", "Estante D1", "2024-02-10"},
00405       {"Multímetro Digital", "Instrumento", "3", "Mesa Taller", "2023-12-10"},
00406       {"Osciloscopio", "Instrumento", "1", "Mesa Taller", "2023-09-10"},
00407       {"Fuente DC 30V", "Instrumento", "2", "Mesa Taller", "2023-09-10"},
00408       {"Sensor PIR SR505", "Sensor", "20", "Estante C1", "2024-01-05"},
00409       {"Sensor DHT11", "Sensor", "15", "Estante C1", "2024-01-08"},
00410       {"Sensor DHT22", "Sensor", "10", "Estante C1", "2024-01-08"},
00411       {"ESP32-CAM", "Microcontrolador", "7", "Estante C2", "2024-02-01"},
00412       {"ESP8266", "Microcontrolador", "10", "Estante C2", "2024-02-02"},
00413       {"Cámara Web USB", "Computación", "5", "Estante D2", "2023-11-11"},
00414       {"Router TP-Link", "Red", "3", "Estante D2", "2023-12-01"},
00415       {"Switch Lógico", "Electrónico", "100", "Cajón A4", "2024-01-22"},
00416       {"Cables USB", "Accesorio", "30", "Cajón B3", "2023-12-28"},
00417       {"Rollo Cinta Aislante", "Herramienta", "20", "Cajón B3", "2023-12-30"},
00418       {"Soldador 60W", "Herramienta", "2", "Mesa Taller", "2023-12-15"},
00419       {"Estaño", "Consumible", "10", "Cajón B1", "2023-12-15"},
00420       {"Alcohol Isopropílico", "Limpieza", "4", "Estante E1", "2024-01-02"},
00421       {"Guantes Nitrilo", "Laboratorio", "200", "Armario F1", "2024-01-01"},
00422       {"Termómetro Digital", "Laboratorio", "3", "Estante E2", "2023-12-10"},
00423       {"Placa de Calentamiento", "Laboratorio", "1", "Estante E3", "2023-11-20"},
00424       {"Cronómetro", "Laboratorio", "2", "Cajón B4", "2023-12-09"},
00425       {"Lupa de Banco", "Herramienta", "2", "Mesa Taller", "2023-10-15"},
00426       {"Extensión Eléctrica", "Hogar", "8", "Estante D1", "2023-10-10"},
00427       {"Bombillo LED 12W", "Hogar", "20", "Estante D1", "2023-09-10"},
00428       {"Tomacorriente", "Hogar", "40", "Estante D1", "2023-09-10"},
00429       {"Interruptor de Pared", "Hogar", "40", "Estante D1", "2023-09-10"},
00430       {"Control Remoto IR", "Electrónico", "15", "Cajón A2", "2024-02-01"},
00431       {"Buzzer 5V", "Electrónico", "40", "Cajón A3", "2024-02-03"},
00432       {"Servo SG90", "Electrónico", "15", "Cajón A3", "2024-02-10"},
00433       {"Switch de Palanca", "Electrónico", "50", "Cajón A4", "2024-02-11"},
00434       {"Jack DC 5.5mm", "Electrónico", "60", "Cajón A4", "2024-02-11"},
00435       {"Pinzas de Cocodrilo", "Accesorio", "50", "Cajón B3", "2024-01-01"},
00436       {"Termistor NTC 10k", "Sensor", "80", "Cajón A2", "2024-01-20"}
00437     };
00438
00439     for (const auto &item : defaults) {
00440         manager.addItem(item[0], item[1], item[2].toInt(), item[3], item[4]);
00441     }
00442
00443     refreshModel();
00444     QMessageBox::information(this, "Carga Completa", "Se han cargado exitosamente los componentes por
      defecto.");
00445 }
00446
00451 void MainWindow::onRestoreDefaults()
00452 {
```

```
00453     if (QMessageBox::question(this, "Restaurar Fábrica",
00454                               "¿Seguro que deseas borrar TODO el inventario y restaurar los datos por
      defecto?\nEsta acción no se puede deshacer.")
00455         != QMessageBox::Yes)
00456         return;
00457     QSqlQuery q;
00458     q.exec("DELETE FROM inventario");
00459
00460
00461     onLoadDefaults();
00462     QMessageBox::information(this, "Restauración", "La base de datos ha sido restaurada.");
00463 }
00464
00470 void MainWindow::onExport()
00471 {
00472     QString filename = QFileDialog::getSaveFileName(
00473         this, "Guardar Reporte CSV", "reporte.csv", "Archivos CSV (*.csv)");
00474
00475     if (filename.isEmpty()) return;
00476
00477     CSVReport report;
00478     // Obtiene todos los items mediante el manager y genera el archivo
00479     if (report.generate(manager.getAllItems(), filename)) {
00480         QMessageBox::information(this, "Éxito", "El reporte ha sido exportado correctamente.");
00481     } else {
00482         QMessageBox::critical(this, "Error de Exportación", "No se pudo escribir el archivo en la ruta
      seleccionada.");
00483     }
00484 }
00485
00494 void MainWindow::onLowStock()
00495 {
00496     QStringList lowStockItems;
00497
00498     for (int r = 0; r < proxy->rowCount(); r++) {
00499         // Índice de la columna 'Cantidad' (columna 3)
00500         QModelIndex idxQty = proxy->index(r, 3);
00501         int qty = idxQty.data().toInt();
00502
00503         if (qty < lowStockThreshold) {
00504             // Pintar toda la fila (iterando columnas)
00505             for (int c = 0; c < proxy->columnCount(); c++) {
00506                 tableView->model()->setData(
00507                     proxy->index(r, c),
00508                     QColor(255, 200, 200), // Rojo claro
00509                     Qt::BackgroundRole
00510                 );
00511             }
00512
00513             // Agregar nombre a la lista de alerta
00514             QModelIndex idxName = proxy->index(r, 1);
00515             lowStockItems << idxName.data().toString();
00516         }
00517     }
00518
00519     if (!lowStockItems.isEmpty()) {
00520         QMessageBox::warning(this, "Alerta de Stock Bajo",
00521             "Los siguientes ítems están por debajo del mínimo:\n\n" + lowStockItems.join("\n"));
00522     } else {
00523         QMessageBox::information(this, "Stock Saludable", "No hay elementos con stock bajo.");
00524     }
00525 }
00526
00531 void MainWindow::onSearch(const QString &text)
00532 {
00533     proxy->setFilterFixedString(text);
00534 }
00535
00541 void MainWindow::refreshModel()
00542 {
00543     model->setQuery("SELECT id, nombre, tipo, cantidad, ubicacion, fechaAdquisicion "
00544                     "FROM inventario ORDER BY id DESC");
00545
00546     model->setHeaderData(0, Qt::Horizontal, "ID");
00547     model->setHeaderData(1, Qt::Horizontal, "Nombre");
00548     model->setHeaderData(2, Qt::Horizontal, "Tipo");
00549     model->setHeaderData(3, Qt::Horizontal, "Cantidad");
00550     model->setHeaderData(4, Qt::Horizontal, "Ubicación");
00551     model->setHeaderData(5, Qt::Horizontal, "Fecha Adquisición");
00552
00553     tableView->resizeColumnsToContents();
00554 }
```

## 5.23 src/report.cpp File Reference

```
#include "report.h"
#include "InventoryManager.h"
#include <QFile>
#include <QTextStream>
#include <QDebug>
```

Include dependency graph for report.cpp:




## 5.24 report.cpp

Go to the documentation of this file.

```
00001 #include "report.h"
00002 #include "InventoryManager.h"
00003 #include <QFile>
00004 #include <QTextStream>
00005 #include <QDebug>
00006
00010 CSVReport::CSVReport()
00011 {
00012 }
00013
00033 bool CSVReport::generate(const QList<InventoryItem> &items,
00034                          const QString &filePath)
00035 {
00036     QFile file(filePath);
00037
00038     if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
00039         qDebug() « "No se puede abrir archivo CSV:" « filePath;
00040         return false;
00041     }
00042
00043     QTextStream out(&file);
00044
00045     // Encabezados del CSV
00046     out « "ID;Nombre;Tipo;Cantidad;Ubicacion;FechaAdquisicion\n";
00047
00048     // Datos de cada item
00049     for (const InventoryItem &it : items) {
00050         out « it.id « ";"
00051             « "\"" « it.nombre « "\";"
00052             « "\"" « it.tipo « "\";"
00053             « it.cantidad « ";"
00054             « "\"" « it.ubicacion « "\";"
00055             « "\"" « it.fechaAdquisicion « "\"\n";
00056     }
00057
00058     file.close();
00059     return true;
00060 }
```