Model Subclassing

Model subclassing involves creating new models that inherit from the QAbstractItemModel, QAbstractListModel and QAbstractTableModel classes that provide default implementations to many of the functions required for their corresponding data structures.

The functions of new model subclasses that are required for implementation can be divided into three categories:

- Item data handling: Functions that enables views and delegations of model dimensions, examinations of items and retrieval of data to queries.
- Navigation and index creation: Hierarchal structures require functions that allow for the proper navigation and creation of model indexes for views to interact efficiently with.
- Drag and drop support and MIME type handling: Models inherit functions that control the way that internal and external drag and drop operations are performed. Theses functions allow items of data to be described in terms of MIME types that other components and applications can understand.

Item Handling:

Defining levels of access models have to their own data.

Read only: requires the following functions to be properly implemented in subclass.

- **flags():** Enables other components to obtain information about each item provided by model.
- **data():** Enables data to be supplied to views and delegates.
- **headerData():** Allows for views with information to show in their headers. Information is only be retrieved by views that have the ability to display header information.
- **rowCount():** Provides the number of rows of data exposed by model.
- **columnCount():** Provides the number of columns of data exposed by model. Direct subclasses of QAbstractTableModel and QAbstractItemModel must implement this function, where as QAbstractListModel already implements it.

Editable Items: requires the following functions to be properly implemented in subclasses.

- **Flags():** Must return an appropriate combination of flags for each item. In particular, the value returned by this function must include Qt::ItemIsEditable in addition to the values applied to items in a read-only model
- **setData():** Used to modify the item of data associated with a specified model index. To be able to accept user input, provided by user interface elements, this function must handle data associated with Qt::EditRole. The implementation may also accept data associated with many different kinds of roles specified by Qt::ItemDataRole. After changing the item of data, models must emit the dataChanged() signal to inform other components of the change.
- **setHeaderData():** Used to modify horizontal and vertical header information. After changing the item of data, models must emit the headerDataChanged() signal to inform other components of the change.

Resizable Models: requires the following functions to be properly implemented in subclass.

- **insertRows():** Used to add new rows and items of data to all types of model. Implementations must call beginInsertRows() before inserting new rows into any underlying data structures, and call endInsertRows() immediately afterwards.
- **removeRows():** Used to remove rows and the items of data they contain from all types of model. Implementations must call beginRemoveRows() before inserting new columns into any underlying data structures, and call endRemoveRows() immediately afterwards.
- **insertColumns():** Used to add new columns and items of data to table models and hierarchical models. Implementations must call beginInsertColumns() before rows are removed from any underlying data structures, and call endInsertColumns() immediately afterwards.
- **removeColumns():** Used to remove columns and the items of data they contain from table models and hierarchical models. Implementations must call beginRemoveColumns() before columns are removed from any underlying data structures, and call endRemoveColumns() immediately afterwards.

Navigation and Model index Creation:

Defining functions for Hierarchal navigation and indexing.

Parents and Children:

- **index():** Given a model index for a parent item, this function allows views and delegates to access children of that item. If no valid child item - corresponding to the specified row, column, and parent model index, can be found, the function must return QModelIndex(), which is an invalid model index.
- **parent():** Provides a model index corresponding to the parent of any given child item. If the model index specified corresponds to a top-level item in the model, or if there is no valid parent item in the model, the function must return an invalid model index, created with the empty QModelIndex() constructor.

Drag and Drop Support and MIME Type Handling:

Defining further behaviours for drag and drop operations when default operations are insufficient.

MIME data:

Built in models and views utilize MIME type data to pass on information about model indexes. When defining drag and drop for custom models it is possible to redefine the following function to allow for the exportation of items of data with specialized formats.

- **mimeData():** Can be reimplemented to return data in specialized formats.

Accepting dropped data:

To properly utilize the default implementations of QAbstractItemModel::dropMimeData() which attempts to insert data into a model as either siblings of items or as children, the subclass must provide reimplementation's of the following functions:

- **insertRows():** Enables model to automatically insert new data using existing implementation.
- **insertColumns():** Enables model automatically insert new data using existing implementation.
- **setData():** Enables new rows and columns to be populated with data.
- **setItemData():** Provides more efficient support for the population of new items.

The following functions must be reimplemented to allow subclasses to accept other forms of data.

- **supportedDropActions():** Indicates types of drag and drop actions accepted by model.
- **mimeTypes():** returns list of MIME types that can be decoded and handled by the model.
- **dropMimeData():** Performs decoding and defines how the model will deal with the transferred data.

In order to ensure drag operations work properly, it is important to reimplement the following functions that remove data from the model:

- **removeRows()**
- **removeRow()**
- **removeColumns()**
- **removeColumn()**

For more information on functions that require proper implementation for each category visit https://doc.qt.io/archives/4.6/model-view-model-subclassing.html