

Beyond QTable

When to Use an Advanced Table Widget

The screenshot shows a window titled "stocks" containing a "Stock Activity" table. The table displays stock prices for "Today" and "One Day Ago". The "Today" section has a "Show Gains In" dropdown set to "Green". The "One Day Ago" section has a "Show Gains In" dropdown set to "Green" and a "Green" color selector. The table columns are "Stock", "Current", "High", and "Low". The data is as follows:

| | Stock | Today | | One Day Ago | |
|----|--------|---------|---------|-------------|---------|
| | | Current | High | Current | High |
| 1 | NYSE | 8676.60 | 8676.60 | 8540.22 | 8544.00 |
| 2 | NASDAQ | 1507.27 | 1507.27 | 1501.34 | 1511.40 |
| 3 | ATT | \$39.61 | \$39.61 | \$40.10 | \$41.00 |
| 4 | VYZ | \$33.78 | \$33.78 | \$34.33 | \$35.32 |
| 5 | DAS | \$32.08 | \$32.38 | \$31.80 | \$32.28 |
| 6 | RWD | \$21.99 | \$21.99 | \$22.33 | \$23.20 |
| 7 | FWY | \$17.06 | \$17.06 | \$17.20 | \$17.50 |
| 8 | AMD | \$12.19 | \$12.19 | \$12.33 | \$12.33 |
| 9 | CWS | \$6.93 | \$6.93 | \$7.00 | \$7.24 |
| 10 | TEE | \$4.62 | \$4.62 | \$4.55 | \$4.80 |

**Integrated Computer
Solutions Incorporated**

The User Interface Company™

201 Broadway
Cambridge, MA 02139
617.621.0060
info@ics.com
www.ics.com



Beyond QTable: When to Use an Advanced Table Widget

Table of Contents

| | |
|---|-----------|
| Preface | 3 |
| Architecture | 5 |
| Basic Structure/Layout | 6 |
| Initial Visible Table Size | 6 |
| Freezing Rows/Columns..... | 7 |
| Major/Minor Columns/Rows | 8 |
| Importing/Exporting Data | 8 |
| Headers | 9 |
| Top, Bottom, Left, Right Headers..... | 9 |
| Multiple Rows/Columns of Headers..... | 10 |
| Content in Headers Can Span Cells | 10 |
| All Four Corner Cells Can Contain Content | 11 |
| More Flexible Setting of Basic Attributes | 11 |
| Headers Can Contain Widgets | 11 |
| Cell Editing..... | 12 |
| Specifying Edit Policy | 12 |
| Force a Cell to Go Out of Edit Mode..... | 13 |
| Display | 13 |
| Cell Attributes Can Be Set at Cell, Row, Column or Table..... | 13 |
| Flexible Options to Display Text that Overflows a Cell..... | 14 |
| Greater Control of Gridlines and Cell Borders | 15 |
| Cursor Can Be Set at Cell, Row, Column, and Table | 16 |
| Sorting | 16 |
| Printing..... | 17 |
| Print Areas and Titles | 17 |
| Non-WYSIWYG Printing is Important | 17 |
| Conclusion | 18 |
| Appendix 1: Estimated Savings | 19 |
| Qt Training, Advanced Qt Components and Testing Tools..... | 21 |
| QicsTable™ | 21 |
| GraphPak™ | 21 |
| KD Executor | 21 |
| About ICS..... | 22 |

Copyright © 2004 Integrated Computer Solutions, Inc. All rights reserved. This document may not be reproduced without written consent from Integrated Computer Solutions, Inc. Qt is a registered trademark of TrollTech AS. QicsTable and GraphPak are trademarks of Integrated Computer Solutions, Inc. All other trademarks are property of their owners.

Preface

The purpose of this paper is to compare and contrast the programmatic and qualitative differences between the basic QTable widget provided with the Qt toolkit and the advanced QicsTable component available from Integrated Computer Solutions, Inc. The information provided below is intended to help you, the Qt developer, decide if the QTable widget provided for free with the Enterprise Edition of Qt is sufficient for your needs or if the QicsTable provides sufficient value to justify buying it. When applicable, we provide suggestions on how you may be able to retrofit QTable to provide similar functionality as that provided by QicsTable.

It should be expected that new versions of QTable may adopt some of the features of QicsTable and that QicsTable will likewise continue to add new features. Make sure you have the latest version of this document by visiting: <http://www.ics.com/qt> before proceeding. We're very interested in correcting any errors. If you spot one, please tell us by sending a note to info@ics.com.

To help you organize your evaluation, we have grouped key functionalities in the following areas:

- **Architecture**
- **Basic Structure/Layout**
- **Importing/Exporting**
- **Headers**
- **Cell Editing**
- **Display**
- **Sorting**
- **Printing**

As we look at each area, we will explain its significance and then characterize the similarities and differences between the two components. We also try to provide an estimate of how long it will take to retrofit any advanced feature of QicsTable into QTable. By adding up all the extra costs, you can then calculate whether QicsTable makes sense for your particular project. Appendix 1 provides a table to help keep track.

For example, a key difference between QicsTable and QTable is one of philosophy. QTable assumes that you will subclass and extend the API to handle many commonly needed attributes (e.g., text alignment in each cell, font choices, colors, etc.). In contrast, QicsTable provides a broad API that significantly reduces the need to subclass. We believe that reducing the need to subclass will save you coding time now and maintenance time later. You may wish to estimate your costs for creating specialized subclasses and note it on the table in **Appendix 1**.

Annotations for the Stock Activity window:

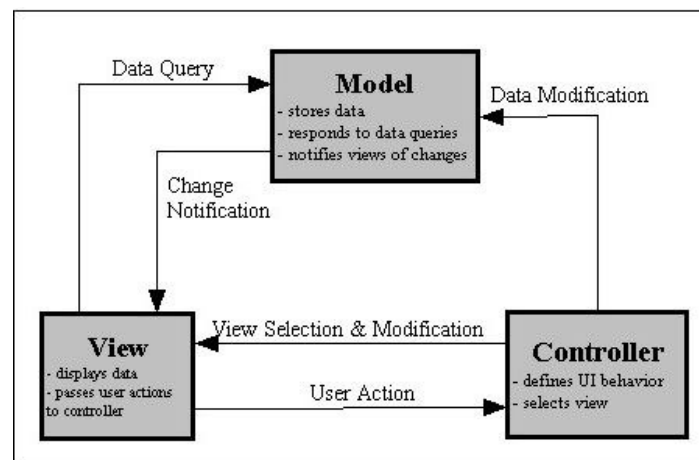
- Corners can contain widgets**: Points to the "Auto Updates" checkbox in the top-left corner.
- Header content can span cells and carry unique formatting**: Points to the "Today" and "One Day Ago" headers.
- Headers can contain widgets**: Points to the "Show Gains in" dropdown menus.
- Major/Minor Columns and rows**: Points to the "Stock" column and the row indices (0-9).
- Columns and rows can be "frozen"**: Points to the row index column.
- Expanded cell formatting and edit control**: Points to the cell containing the value "8676.60".
- Greater control of Gridlines and Borders**: Points to the grid structure.

| | Stock | Today | | | One Day Ago | | |
|---|--------|---------------------|---------|---------|---------------------|---------|---------|
| | | Show Gains in Green | | | Show Gains in Green | | |
| | | Current | High | Low | Current | High | Low |
| 0 | NYSE | 8676.60 | 8676.60 | 8676.60 | 8540.22 | 8544.00 | 8502.89 |
| 1 | NASDAQ | 1507.27 | 1507.27 | 1507.27 | 1501.34 | 1511.40 | 1499.61 |
| 2 | ATT | \$39.61 | \$39.61 | \$39.61 | \$40.10 | \$41.00 | \$38.35 |
| 3 | XYZ | \$33.78 | \$33.78 | \$33.78 | \$34.33 | \$35.32 | \$33.90 |
| 4 | DAS | \$32.38 | \$32.38 | \$32.38 | \$31.80 | \$32.28 | \$31.44 |
| 5 | RMD | \$21.99 | \$21.99 | \$21.99 | \$22.33 | \$23.20 | \$22.00 |
| 6 | HWY | \$17.06 | \$17.06 | \$17.06 | \$17.20 | \$17.50 | \$16.50 |
| 7 | AMD | \$12.19 | \$12.19 | \$12.19 | \$12.33 | \$12.33 | \$12.10 |
| 8 | CWS | \$6.93 | \$6.93 | \$6.93 | \$7.00 | \$7.24 | \$6.99 |
| 9 | TEE | \$4.62 | \$4.62 | \$4.62 | \$4.55 | \$4.80 | \$4.11 |

A table highlighting a few of the advanced features available (without subclassing) in QicsTable™

Architecture

The architecture of a table widget can have a significant impact on its performance, scalability, and flexibility. For example, if a table is only designed to handle data organized in rows, and the particular data being loaded is column-oriented, the performance of the table could be abysmal. The Model-View-Controller (MVC) architecture has long been recognized as a superior design pattern for applications. In this terminology, the Model refers to the actual data, the View represents the visual display of the application, and the Controller is the actual application that processes the data and updates the View.



MVC architecture allows optimized data Models and simplifies multiple Views into the same Model.

In QicsTable, the abstract class QicsDataModel represents the Model. This class defines the common interface that all data models must support. The QicsDataModelDefault class is the default implementation of the QicsDataModel interface. If QicsDataModelDefault does not meet your needs, you may create your own implementation of the QicsDataModel API. For example, your data might be in a database. In such cases, you can override the default QicsDataModel and fetch information in appropriate chunks from the database.

Many applications need to implement multiple Views. For example, it is common for applications to have several windows open, each window looking at a different segment of the database. If the application was not built using the MVC design, and the data is read into each table separately, then a table needs to know how to tell the other table that some data has changed and that the table needs to get a new copy of the data. This communication mechanism between multiple processes is a notorious source of errors.

The MVC controller architecture of QicsTable makes this much simpler. Since two or more Views can share the same data Model, then the Model just needs to tell all the active Views when an update has been made to the data. This form of communication has to exist anyway, and connecting the data

to the different Views using a single signal mechanism eliminates the redundant error prone mechanism.

Basic Structure/Layout

When creating a table, one of your main concerns is putting a set of rows and columns on a screen, and perhaps eventually on a printer. While both QTable and QicsTable provide the expected basic APIs, QicsTable provides several unique capabilities that may be crucial to your application.

Available in both QTable and QicsTable:

- Specify initial visible table size in terms of x, y pixels
- Insert, delete and move rows/columns
- Specify row/column height and width

Available only in QicsTable:

- Setting the initial visible table size in terms of number of rows/columns
- Ability to freeze rows/columns
- Capability to create major/minor rows/columns

Initial Visible Table Size

When your table is initially displayed, how big should it be? With QTable, you will need to think of this initial size in terms of pixels (e.g., 200x400 pixels). In many cases, this would be reasonable. For example, maybe your total window is 600x800 pixels and you want your table to occupy 25% of that space (300x400 pixels).

However, do you know into how many rows and columns this translates ? In most applications, tables need to be a certain size based on the data that needs to be presented. Tables that are too small can be difficult to use because you need to utilize scroll bars. On the other hand, making the table too big means wasted screen real estate that could be better used elsewhere.

QicsTable allows you to set the initially visible height and width of your table in terms of the number of rows or columns you want to display. In short, QicsTable asks you to decide on the appropriate functional size of the table and it does the translation into the number of pixels. This mechanism also gracefully handles changes in fonts, border widths, grids, or even changes in the height of particular rows. QicsTable will automatically adjust the visible size of the table to compensate as the specific attributes of cells change.

Using QTable, calculating the initial visible size of the table based on the number of rows and columns you want displayed is cumbersome and involves determining the height and width of each cell, the size of any borders and shadows, the size of row/column headers, and the size of the border of the table. In practice, many developers find themselves guessing at the size and then compiling and tweaking the application a dozen or so times until the size is right.

Alternatively, the QicsTable member functions `QicsTable::setVisibleRows` and `setVisibleColumns()` allow you to specify the number of row and columns you want to initially see. (Of course you can calculate number of pixels with QicsTable if you need/want too.) If you think that setting the table size based on the number of rows and columns is simpler, faster, and less error prone than calculating pixels, then add your estimated effort costs to the table in **Appendix 1**.

Freezing Rows/Columns

If you have ever created a large spreadsheet, then you've likely used the ability to "freeze" a set of rows/columns. This allowed you to scroll through your data while maintaining one or more rows or columns "frozen" in place for reference. Typically, you might use this to remind you of the contents of a particular row or column. Alternately, you might use this if you wanted to compare a specific data point to another one elsewhere in the table.

Implement this functionality using QTable would involve embedding a "table within a table" and setting up a scheme for moving data back and forth between them. Conceptually this isn't hard, but getting signals/slots working well between them would take some time.

Alternatively, through a simple API, QicsTable provides the ability to set a certain set of rows and/or columns as frozen. To see exactly how to do this, read the description of the following functions in the documentation available online at <http://www.ics.com/qt/qicstable/index.html>:

- `QicsTable::freezeTopRows()`
- `QicsTable::freezeBottomRows()`
- `QicsTable::freezeLeftColumns()`
- `QicsTable::freezeRightColumns()`.

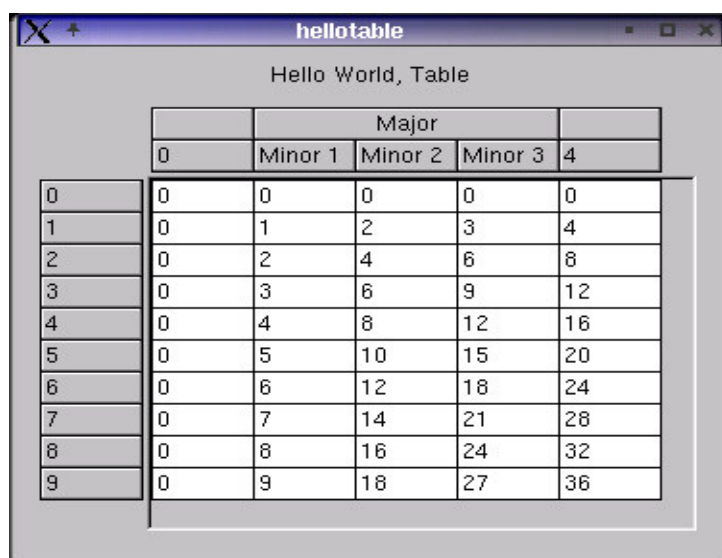
| | Col 0 | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 | Col 7 | Col 8 | Col 9 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Row 6 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Row 7 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Row 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Row 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Row 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Row 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Row 4 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Row 5 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Row 8 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Row 9 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Row 10 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Row 11 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

QicsTable allows you to easily "freeze" rows and columns.

Major/Minor Columns/Rows

In creating tables, adjacent columns (and/or rows) are often related. For example, you might have three columns labeled “R”, “G”, “B” that refer to the mix of the primary colors that produce the actual color being displayed. Alternately, you might have a series of vectors where you want to display each vector’s direction and magnitude. It would be natural to place these related columns (or rows) adjacent to each other in your table. This works fine as long as the end user does not want to change the layout. For example, an end user might want to drag the vector so that it is next to another vector to allow a visual comparison. This is not a problem, as long as he/she remembers to select both columns. You can also provide a facility to hide/show columns and your users need only to remember to select both columns. Many users, however, will select one column, move it somewhere convenient for them at the time, forget where they positioned it, and call you up to ask for help finding the missing column.

You could program your application so that if a column is selected, the other related columns in the set (e.g., the entire “Major Column”) are selected too, however it is likely that you only want to select a single column to manipulate. With QTable, there is no easy solution to provide your users with Major/Minor columns/rows. In QicsTable, this is taken care of cleanly using header cells that span multiple rows or columns.



| | | Major | | | |
|---|---|---------|---------|---------|----|
| | 0 | Minor 1 | Minor 2 | Minor 3 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 |
| 2 | 0 | 2 | 4 | 6 | 8 |
| 3 | 0 | 3 | 6 | 9 | 12 |
| 4 | 0 | 4 | 8 | 12 | 16 |
| 5 | 0 | 5 | 10 | 15 | 20 |
| 6 | 0 | 6 | 12 | 18 | 24 |
| 7 | 0 | 7 | 14 | 21 | 28 |
| 8 | 0 | 8 | 16 | 24 | 32 |
| 9 | 0 | 9 | 18 | 27 | 36 |

Major/Minor columns and rows are standard in QicsTable.

Importing/Exporting Data

The moment you present your application data as table, it is almost a certainty that your end users will want to either import their spreadsheet data into your table, or to export the table so they can do some custom processing on the data.

If you’ve chosen to use QTable, you will need to write your own comma separated values (CSV) import/export functions. There are a number of CSV converters listed on <http://sourceforge.net>, perhaps one of those will meet your requirements, both technical and legal (you cannot use a GPL converter with an application that uses a commercially licensed version of Qt.). Unless you have

familiarity with handling the CSV format (remember you have to add quotes and deal with existing quotes in the data), you should allow several days to get importing/exporting working.

QicsTable has built-in support for importing and exporting CSV files. Checkout `QicsDataModel::readASCII()` and `QicsDataModel::writeASCII()` for details. Basically, this enables import/export of any delimited file (comma, tab, semicolon, etc.) into or out of QicsTable.

Headers

All table widgets provide header rows and columns that label the data that is stored in each column and row. Spreadsheets such as Excel are relatively inflexible and are locked to letters (e.g., “A”, “B”, “C”) across the top and numbers (e.g., “1”, “2”, “3”, etc.) going down the rows. However, you have much more flexibility with these labels when you use table widgets such as QTable or QicsTable. For example, both tables provide you with the ability to substitute icons for the numbers and/or letters. One of the benefits of icons, if well designed, is that they are understandable by people who might not understand the original language of the application. Both widgets, of course, also allow you to use text to describe the column and/or row.

In addition to these common features, QicsTable provides the following that you might find useful:

- Headers can be placed on the top, bottom, left, and right
- Multiple rows/columns of headers
- Content in headers can span cells
- Four corner cells that can contain content
- Very flexible setting of basic attributes (e.g., fonts, colors, alignment etc.)
- Headers can contain widgets (text entry, comboboxes, etc.)

Modifying QTable to add features to the headers is a difficult task. QTable uses its own undocumented QHeader subclass (QTableHeader). Because there is no documentation for this class, a thorough reading of the QTableHeader source code will most likely be required in order to understand the class's behavior. In addition, since the use of the QTableHeader class is "hard-wired" into QTable, you will need to make a subclass of QTable that uses your new header class.

Top, Bottom, Left, Right Headers

When you think of a spreadsheet, you naturally think of headers being only on the top and left of the table. Why? Because spreadsheets need to be open ended to the right and below as your model expands. QTable follows this model yet also allows the headers to be placed at the top and left sides of the table.

However, applications can have more defined parameters and the view port into the table is typically of a fixed size. In this environment, there may be reasons to have headers below and to the right of the table in addition to, or even instead of, the top and bottom. With QicsTable, you can turn on/turn off headers on all four sides of the table.

Adding the additional bottom and right headers into QTable wouldn't be too difficult, given Qt's flexible layout management and the fact that QTable already has code in place for the top and left headers. Regardless, adapting the existing code, extending the table API, and testing the final code would take many developers several days.

Multiple Rows/Columns of Headers

QTable, like QicsTable, allows you to wrap a line of text to create multiple lines within a header row or column. In addition to this option, when you use QicsTable, you can have multiple rows (or columns) of headers. This feature provides increased flexibility in labeling your data with multiple lines of information.

For many applications needing to wrap text across multiple lines, you can probably use QTable's existing support for multiple lines within a header. Depending on the complexity of your header, it may take a couple tries to get the right spacing and to get things to align according to your design. For most developers, this means that approximately 2 hours per application need to be allocated to fine-tuning. Clearly, if all you needed were multiple lines of text for a column (or row), then it would be hard to justify the extra cost of QicsTable, and you could probably get by with QTable.

However, this feature becomes critical to if you plan to use the Major/Minor rows/columns feature mentioned above because the selection of the minor or major column (or row) depends on what line of the Header the user selects. Multiple header lines functionality is also important to support the Spanned cells and Widgets in the Header features discussed below.

Content in Headers Can Span Cells

If you use Excel or other spreadsheet applications, you are probably familiar with the content of a cell overwriting or "spanning" across multiple cells. Spanning can be a very useful function that helps makes the overall relationship between multiple columns or row clearer. For example, using QicsTable's multiple header row feature, you might label one set of columns with "R", "G", and "B". You can then span the three columns in the column header above with the words "Primary Color Mixture". This would make it clear to a reader the meaning of "R", "G", and "B".

QTable does not provide any ability to specify cell spanning in the Header. Only in the main body of the table is the developer able to specify that the contents of one cell overlaps/overwrites the contents of the adjacent cell.

This functionality should sound similar to the Major/Minor rows and columns capability described in the previous section. Major and Minor rows and columns are a semantic labeling that users give to a table organization. Syntactically, this is accomplished by using spanning cells in the headers.

Extending QTable to support spanned cells in the Header would require a major modification to the QTableHeader class. Our estimate is that the modification would require at least two weeks of work to develop and debug.

All Four Corner Cells Can Contain Content

The position of a table's scrollbars naturally creates four little corner cells at the top left, top right, bottom right, and bottom left. For example, the top left corner of an Excel spreadsheet contains a control that is used to select the whole table. QTable allows developers to put a string or icon in the bottom right corner of the table. However, QTable does not provide any direct mechanism to use the top left, top right, and bottom left corners.

QicsTable allow you to place a widget, text label, or an icon in any of the four corners. Many users like to put the application or company logo in the top left corner. You might also parent a button to these cells and place the application or company logo on top of it. That way, punching the button can cause some special action (e.g., selection of the whole spreadsheet).

Extending QTable to support the remaining three corners of the table, requires you to modify QTable's layout policy to allow for some extra widgets to be placed in the corners. Our rough estimate is that this will take most developers 2-3 days to fully design, test, and debug.

More Flexible Setting of Basic Attributes

Possibly, some header cells of your table would benefit from a bolder font or by using a font color of white against a dark background. As the default, QTable uses the same attributes (e.g., fonts, colors, etc) for the whole header.

QicsTable provides a direct API to allow you to set these basic attributes for the table, row header, or column header and finally on a cell-by-cell basis. This creates a hierarchy where the more specific attribute will override a more general setting of an attribute (e.g., cell is more specific than an attribute set for a row header). For more about this capability, see `QicsRowHeader` and `QicsColumnHeader` in the documentation at <http://www.ics.com/qt/qicstable/index.html>.

Achieving similar functionality using QTable, with cell-by-cell attribute control or general setting by a row with cell-by-cell override, requires you to subclass QTableHeader to set the appropriate attributes. You will also need to subclass QTable to use the new QTableHeader subclass you've created. If you are doing this for a specific application, and have the option of hardwiring the specific attributes, then you can probably implement a reasonable Header with multiple fonts, colors, etc. within a day or so. If you want a more general case that can be re-used by multiple applications, you should schedule at least a two weeks to design, implement, and test.

Headers Can Contain Widgets

It is very easy to think of headers containing static text or icons. In this role, they are passive elements of an application, helping with end user understanding but not providing any interaction with the application. Indeed, this is the only mode of operation that QTable supports.

QicsTable treats cells in a header the same as cells elsewhere in the table. This means that QicsTable header cells can contain additional objects such as comboboxes and text entry widgets. With QicsTable, the header becomes an alternate, and perhaps key mechanism, to interact with the application.

For example, one could imagine two adjacent columns where a combobox in the header of the second column defines the function that is to be performed on the first column to generate the data in the second column. One or more text field(s) in the header could also be used to define a set of parameters that would be used by this function. This is a powerful paradigm that can simplify the user interface of many applications.

Extending QTable to support spanned cells in the Header requires a major modification to the QTableHeader class. ICS's rough estimate is that the modification would require at least two weeks of work to develop and debug.

Cell Editing

After your end user navigates to a cell, it is likely they will want to edit the data. There are a number of subtle user interactions that can make your application easy or hard to use. Both QTable and QicsTable provide facilities to:

- Specify which cells are editable on a table, row, column, and cell basis
- Control when changes are applied, and if cancelled, the restoring of the original data.
- Force a cell to go into edit mode to allow immediate data entry

QicsTable additionally provides you with APIs to:

- Directly specify Edit policy of single click or double click
- Force a cell to go out of edit mode immediately. This is useful if you need to prevent entry of data that may cause inconsistencies.

Specifying Edit Policy

QicsTable provides you with direct control over setting the Edit policy. You can allow your users to enter Edit mode by the single click process of selecting a cell, or by requiring a double-click (one to select and one to enter Edit mode). The member function `QicsGridCommon::clickToEdit()`, is used to control this behavior. This is a member function that sets the Edit policy for a whole grid. You could provide different Edit policies for your table by implementing multiple adjacent grids that use the same headers.

QTable only provides you with a double-click mode for data entry. If you would like to implement a single click mode, you will need to subclass the `QTableWidgetItem` class and add this behavior.

Force a Cell to Go Out of Edit Mode

Most developers need to worry about when a cell goes into Edit mode. However, there are times when you want to yank back control and prematurely terminate Edit mode. For example, perhaps the underlying data model has received a major update and allowing the end user to continue editing a value might generate an inconsistency in the data.

QicsTable provides the member function `QicsTable::uneditCurrentCell()`. This function allows you to force a particular cell to go out of Edit mode.

QTable lacks a convenient member function to force a cell to go out of Edit mode. To enforce this behavior, you will need to subclass QTable and make the changes at that level.

Display

A table is used to display data in a format that your end user can quickly analyze and turn into the knowledge required for making optimal technical and business decisions. Not all data is equally important and sometimes your end users will appreciate tweaks to the display of data that helps them better understand the relationships between one or more columns/rows. For example, you might use a similar font color for two rows that have related information. Alternately, you might paint negative numbers in red and use black for positive numbers.

QTable allows setting of many display attributes only on a whole-table level.

QicsTable is especially strong in its ability to provide multiple levels of direct control over the presentation of data. For example, you can set a font for the whole table, or for all the data in a row (or column), or for the individual cell. As you would expect, the more specific overrides the more general specification, so a font specified on a specific cell would take precedence over the font specified on the table as a whole.

QicsTable's comprehensive API allows you, without subclassing, to:

- Set common cell attributes on a cell, row, column, or table basis
- Determine the display of text that overflows a cell
- Control display of gridlines and cell borders
- Set the cursor display on a cell, row, column, and table basis

Cell Attributes Can Be Set at Cell, Row, Column or Table

In creating a table, there are many cell attributes that can be set to make the data more readable or to highlight significant events. For example, you might want to set the background and foreground colors of a particular column of cells to help end users monitor their values. You might also want to set the alignment of certain columns to be left justified while other columns need to be right justified. Moreover, one can imagine the desire to have a default, table-wide set of fonts and colors while you reverse these colors on individual cells to indicate critical information.

With QicsTable, this is easy. There is a built-in style manager that allows you to set cell attributes table wide, on specific rows/columns, and on individual cells. As you would expect, a row/column setting overrides any table wide defaults and the attributes of individual cells overrides all the global settings.

Because QTable sets these attributes on the whole table, setting the attributes on specific cells requires you to subclass QTableWidgetItem and set the values directly. The lack of a style manager in QTable means that you have to do more work if you want to set a whole row or column to have the same attributes.

Flexible Options to Display Text that Overflows a Cell

Setting the width and height of individual cells is always a tradeoff. If you make the cells wide (or tall) enough for any possible data, they become too big and hard to manipulate. If you make them too small, data is often hidden (or clipped) when it overflows the cell boundary. QTable only provides a “clipped” policy. This means that data that is too large for the cell is clipped (possibly clipped left, right, or left and right depending on the cell’s alignment choice).

QicsTable provides with the ability to “clip” the data, should that be the right choice for your application. Additionally, QicsTable, enables you to specify that when a cell overflows, it extends to the adjacent cell to the right. This way, by organizing lower priority columns (or rows) next to columns with data that could overflow, you can ensure that critical data is always fully displayed.

If you want to clip the data on the cell boundary, QicsTable provides you with the option to add a small marker to the cell indicating that it has been clipped. Furthermore, you can turn on “tool tips” for cells so that when the mouse pointer hovers over a cell that has been clipped, the full, unclipped data pops up in a “balloon”. This provides more rapid access to data compared to clicking down on a particular cell and going into edit mode so that you can scroll back and forth to read the full contents of the cell.

Adding similar capability to QTable would take significant effort. QTableWidgetItem are not subclasses of QWidget, so the “obvious” solution of just using QToolTip::add() won’t work here. Instead, you would want to subclass QToolTip. The new ToolTip would need to be attached to the QTable itself and query the QTable for the correct data to display in the tip.

hellotable

Hello World, Table

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|----|----|----|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | Span region from cell (1,1) 2 rows x 3 columns | | 4 | |
| 2 | 0 | | | 8 | |
| 3 | 0 | | 3 | 6 | 9 |
| 4 | 0 | | 4 | 8 | 12 |
| 5 | 0 | 5 | 10 | 15 | 20 |
| 6 | 0 | 6 | 12 | 18 | 24 |
| 7 | 0 | 7 | 14 | 21 | 28 |
| 8 | 0 | 8 | 16 | 24 | 32 |
| 9 | 0 | 9 | 18 | 27 | 36 |

QicsTable provides the ability for contents of cells to overwrite adjacent cells and also supplies “Tool Tips”. You can also turn on markers in the table to indicate an overflow.

Greater Control of Gridlines and Cell Borders

Most tables have lines (often called gridlines) that partition the cells into rows and columns. Depending on the table, you might want to make these gridlines thicker or thinner. You might also like to change their style so that they create a clear separation between the headers and the cells. However, sometimes these lines are turned off and the boundaries between cells are implied by whitespace.

QTable provides developers with the API to set the style of the gridline only on a table-wide basis. For many applications, this is fine as it avoids drawing attention away from the data. However, you do not have any finer grain control of lines between the cells.

In addition to the single gridline choice that QTable provides, QicsTable adds the concept of a cell border. Each cell can have a border that is a line style of your choice. There is also the ability to offset the border to create whitespace (or eliminate whitespace) between cells. Like most other cell attributes provided by QicsTable, you can set the border and offset on a cell, a row, a column, and a whole table. As you would by now expect, the assigned cell attributes will override any more general setting (row or column or table).

If you want complete control over the lines between cells, first turn off the gridlines. Then use the API provided (see QicsMainGrid and QicsHeader) to set the borders and offsets for your table. Start by setting table-wide defaults, and then gain more control by setting the borders and margins for a specific column. Finish off by setting the borders for a specific cell to provide additional focus.

Providing more flexible border solutions within QTable requires substantial modification. The current table API does not support making modifications on visual properties of entire rows and columns at once, so that functionality needs to be added first. Additionally, to actually draw the borders of a

single row or column differently would require reimplementing either the `QTable::paintCell()` method or making subclasses of `QTableWidgetItem` and reimplementing their individual `paint()` methods.

Cursor Can Be Set at Cell, Row, Column, and Table

The cursor is used to cue the end user on the location for text entry or selection. At first, it seems that having one standard cursor for all cells in a table would be the proper decision. Often the cursor is hard to see, and if it varied depending on which cell is being edited, it might become impossible to discover the location of the cursor in a large and complex table. `QTable` only allows you to set the cursor for the whole table. You do not have any finer grain control over how the cursor might look in a specific row or column.

However, some applications require that the cursor be displayed differently. For example, if a cell contains a picture, then the traditional “I” beam cursor used for text entry would not make sense. Or, if a cell contained a check box, you will probably want an arrow pointer (or something else).

`QicsTable` provides you this additional flexibility to define different cursors on a cell, row, and column basis in addition to specifying an overall table-wide default. And of course, the hierarchy is enforced with the more specific selection overriding the more general (e.g., table) ones.

`QTable` will need substantial modifications to provide a more flexible solution relative to cursors. As mentioned above, `QTableWidgetItem`s are not subclasses of `QWidget`, so the simple solution of calling `QWidget::setCursor()` won’t work here. A possible solution involves having the `QTable` keep track of what item the mouse is currently over and calling the table’s `setCursor()` routine as needed.

Sorting

Although every experienced programmer should have programmed a sort at sometime, the good news is that you don’t have to write another one for either `QTable` or `QicsTable`. Both tables have built in sorts that perform quite well on most types of data. In addition, both also provide the ability to implement your own comparison function to handle any special ordering needs.

As long as your data is organized by columns, the sorting functions provided by `QTable` may be adequate for you. `QTable` assumes that your data is column-oriented and that rows only represent different sets of data. So with `QTable` (and `QicsTable`), you can:

- Sort the data in a single column
- Sort the rows in a table by the data in a single column. You can do this directly if the cells are all `TableWidgetItem`s. Otherwise, you will need to subclass.

`QicsTable` extends sorting support to row-oriented data. This means you can easily:

- Sort all the data in a row
- Sort all the columns by the values of a particular row

Implementing row-based sorts in QTable will require modification or subclassing. The low-level pieces (mainly the `swapColumns()` method) exist, but the sort routine itself will need to be implemented. This will probably take about a day to fully test and debug.

Printing

Inevitably, at some point you will have to provide a way for your end user to print your table. In some cases the printout is needed because of regulatory requirements and in others your end users may want the report so they can write notes on it as they analyze their data.

It is tempting to think that printing is only an extension of the table that might already be displayed on the screen. However, there are several additional requirements that need to be satisfied for hardcopy output:

- Ability to specify print areas and titles
- Non-WYSIWYG support (handling of clipping, column width, fonts, etc.)
- Postscript output

QTable's printing support is essentially screen capture. What you see on your screen, you can print. QicsTable provides additional facilities including direct Postscript output. Postscript provides you with the widest possible set of options for hardcopy display.

Print Areas and Titles

QicsTable provides you with the ability to select print areas and titles. This allows you to select a subset of the rows and columns that you want to print. You can also select a set of rows and columns to be the headers for the table. This way, if your table is printed on multiple pages, the headers are reprinted so that the information can be more easily coordinated. In contrast, QTable only allows you to print the screen; you cannot print the whole table.

Adding these features to QTable would require a re-implementation of the portion of the table drawing code to draw to both the screen and the printer. You would also have to write code to do the page layout, which could be a sizable amount of work. Someone familiar with the internals of Qt could do the implementation in a few weeks by subclassing QTable itself. However, the implementation would probably require enough changes to QTable that merging updates and bug fixes provided by Trolltech would require additional work.

Non-WYSIWYG Printing is Important

At first, it probably seems strange to be talking about non-WYSIWYG printing as being an advantage. However, hardcopy is such a different media that just printing it as it appears on the screen can lead to wrong conclusions. For example, on a screen, you might want to clip data that is too big for a cell and put a marker to indicate that the cell overflows. Then a user can just select the cell, put it in Edit mode,

and scroll left or right to see the full content. This, of course, cannot be done on a piece of paper. Although you might suspect a particular piece of data has been clipped, there is no way of knowing except to go back online.

QicsTable is designed to be easily customized for printing. When you decide to make a particular table printable, it creates a parallel object hierarchy that allows you to set the various attributes of the “print table”. This allows you to specify clipping (or non-clipping) policy, reduce font sizes, reduce margins, and change other attributes so that the printout provides the information displayed in a format appropriate for hard copy.

QTable will need substantial modifications to provide non-WYSIWYG printing. One approach would be to create a parallel table connected to the same data. Then you could tune the attributes differently to handle hardcopy output. The programming effort required to keep things in sync between your display table and your printing table could be substantial.

Conclusion

We believe that many developers will find that using QTable will cost additional days, weeks, and in some cases, even months over using QicsTable. If your company does not assign a value to the time you spend developing, or assign a value to having the finished application available sooner (e.g., days, weeks, months), then you should continue using QTable because it is FREE according to your company’s metrics. After all, everything we have done within QicsTable to make it easier for you can be duplicated. *It is just a simple matter of programming...*

However, we developed QicsTable with the belief that most companies do value time to market and the developer’s time and effort. For those companies, we hope that a few of the points we’ve highlighted will demonstrate conclusively the value presented by QicsTable.

For companies that need to do a cost/benefit analysis before licensing new software, we have included a summary sheet in **Appendix 1**. This lists all the highlighted features that QicsTable provides over QTable. For those features that are important to you, we invite you to estimate the effort required to engineer them into QTable and decide whether the cost of purchasing QicsTable justifies the savings it provides.

Appendix 1: Estimated Savings

The table below is provided for your convenience. If you are unsure of the cost-benefit of QicsTable, you can review the list of features, and for those that are important to you, add in your estimate of the effort to retrofit QTable to provide the equivalent facilities.

| Estimated Retrofit Effort | QicsTable Feature | Description |
|---------------------------|---|--|
| | Richer API | QicsTable provides a richer set of member functions that reduces the need to create many specialized classes to set basic things like fonts, colors, etc. |
| | MVC architecture | Model-View-Controller architecture provides clean separation between Model (data) and View (GUI). Allows development of optimized data models (e.g. row vs. column oriented) and multiple Views of same data. |
| | Sizing the Visible Table | The initial visible table can be set in terms of rows and columns and not just by pixel count. Reduces number of “guesses-compiles-adjustments” to get initial table size correct. Also gracefully handles changes in fonts, border widths, etc. |
| | Frozen Rows/Columns | Allows you to freeze a set of rows or columns while scrolling through the rest of table. Useful for comparison of values. |
| | Major/Minor Columns | Provides a hierarchy of columns (or rows) where a major column “contains” a minor column (or other major columns). Selecting a major column also selects all of its minor columns. Provides greater end user control and organization of data. |
| | Import/Export to Spreadsheets | CSV import/export utilities are included within the Data Model class. |
| | Headers on top, bottom, left, and right | In addition to headers (or labels) appearing on top and left like most spreadsheets, QicsTable allows headers to appear on right and bottom. Important when a table has many columns and/or rows of similarly formatted data. |
| | Multiple rows/columns of headers | QicsTable provides multiple rows and columns of headers. Allows headers to contain additional information other than just labels (see below). |
| | Span cells in headers | Cells in headers can span multiple columns or rows to group similar information together visually. |
| | Corner cells can contain content or widgets | Corner cells exist between the row and column headers. In Excel, the top right corner is used to select the whole table. QicsTable allows icons, labels, and widgets to be placed in all four corners. |

| Estimated Retrofit Effort | QicsTable Feature | Description |
|---------------------------|---------------------------------------|--|
| | Basic header cell attributes settable | Individual header cells can have unique values for its basic attributes (fonts, colors, alignments, etc.). Useful to call attention to columns or rows. |
| | Widgets in Headers | QicsTable allows widgets to occur in header cells. For example, you could use a combobox to select a function that calculates the value in a row. |
| | Settable Edit Policy | Double-click and single-click user actions are supported for entering cell Edit mode. Double-click is familiar to many spreadsheet users, but single-click is preferred by power users that want to minimize mouse clicks. |
| | Force End of Edit Mode | Edit mode can be terminated by the program. Allows applications to prevent erroneous entry of data. |
| | Flexible setting of cell attributes | Cell attributes (fonts, colors, alignments, etc.) can be set on a cell, row/column, or table wide basis. Simplifies creation and maintenance of consistent styles. |
| | Cell Overflow | QicsTable provides markers indicating that data overflows cell boundaries. Popup tooltips can be enabled to display the full value when mouse pointer hovers over cell. This reduces human errors because of failure to perceive clipping of data. |
| | Gridline and Cell Border controls | In addition to being able to set gridlines on a table wide basis, QicsTable provides borders for individual cells. |
| | Multiple Cursors | Cursors can be set for each cell, row, and column in addition to a table-wide basis. |
| | Row-Oriented Sorting | Can sort all data in a row or sort columns by values in a particular row. Useful when data is row-oriented. Column-oriented sorting is also provided. |
| | Print Areas and Titles | Whole table or contiguous subsections of the table can be printed. Title can print on each page. Provides increased control over printing of table information. |
| | Non-WYSIWYG Printing | Non-WYSIWYG printing help reduces chance of errors resulting from users being unaware that a cell value is not complete. |

_____ Development Effort (in Days) Required to Retrofit QTable
 x _____ Full Cost/Average Developer
 = _____ Total Cost of Using QTable

Qt Training, Advanced Qt Components and Testing Tools

As Trolltech's preferred training partner for North America, ICS provides public and customized on-site training on Qt. See details at

<http://www.ics.com/services/training/?cont=QTtraining>

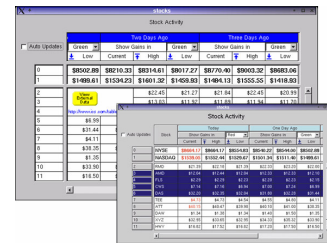
ICS also provides a growing selection of advanced components and testing tools for Qt. Some of our products including:

QicsTable™

Power to easily manipulate the largest data sets and with all of the display and print functions you need to satisfy even the most demanding end-users, this advanced Table component comes with a comprehensive API that makes programming a snap. MVC architecture assures that your application is easy to build, modify, and maintain.

Free trial at

<http://www.ics.com/qt/qicstable/?cont=getqicstable>

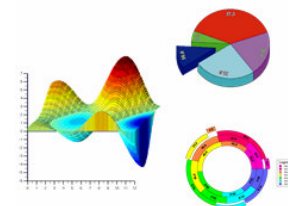


GraphPak™

A collection of powerful charts and graphs that make it easy to visually present complex data.

Free trial at

<http://www.ics.com/qt/graphpak/?detail=downloadA.html>

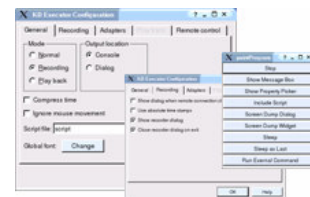


KD Executor

A true cross-platform testing harness that makes it easy to fully test your Qt applications.

Free trial at

<http://www.ics.com/qt/kdexecutor/?cont=download>



About ICS

Driven by the belief that the success of any software application ultimately depends on the quality of the user interface, Integrated Computer Solutions, Inc., The User Interface Company™, of Cambridge, MA, is the world's leading provider of advanced user-interface development tools and services for professional software engineers in the aerospace, petrochemical, transportation, military, communications, entertainment, scientific, and financial industries. Long recognized as the platform of choice for visually developing mission-critical, high-performance Motif applications, ICS' BX series of GUI builders has recently been expanded to provide a complete line of tools that accelerate development of Java™. ICS is the largest independent supplier of add-on products to the Qt multi-platform framework developed by Trolltech. Supporting the software development community since 1987, ICS also provides custom UI development services, custom UI component development, training, consulting, and project porting and implementation services. More information about ICS can be found at <http://www.ics.com>



**Integrated Computer
Solutions Incorporated**

The User Interface Company™

**201 Broadway
Cambridge, MA 02139
617.621.0060
info@ics.com**

www.ics.com