Qilan®

Version 2.8.1

CommonGround Softworks, Inc.

Documentation by Stephen Caine

Table of Contents

License Agreement

SOFTWARE LICENSE AGREEMENT

This software License Agreement ("Agreement") is entered into between CommonGround Softworks, Inc. ("Licensor") and the purchaser ("Customer") of the software.

DEFINITIONS

Software: The term "Software" shall mean the computer program in object code only and any user manuals downloaded from the Licensor's web site or provided by the Licensor. The term "Software" includes any corrections, bug fixes, enhancements, updates or other modifications, including custom modifications, to such computer program and user manuals.

Registration Codes: Registration Codes are values necessary to operate the Software, which are sent by Licensor to the Customer by e-mail or in writing after Customer has received the software.

Qilan Developer: The Qilan Developer is an application package that creates and/or modifies a project file which is a specialized document that defines data logic, input and/or output screens and schema relationships with one or more back-end data bases.

Qilan Engine: The Qilan Engine is a common gateway interface (cgi) and/or fast common gateway interface (fcgi) that runs project files created by the Qilan Developer.

LICENSE

Grant of License: Licensor grants Customer, pursuant to the terms and conditions of this agreement, a perpetual, nonexclusive, nontransferable license to use the Software with the Qilan Developer, if purchased, and Qilan Engine, if purchased.

Restrictions on Use: Customer agrees to use the Software, including the Qilan Developer, if purchased, and Qilan Engine, if purchased, only for Customer's own business. Furthermore, installation of the Qilan engine is limited to one (1) machine per license. Customer shall not permit any parents, subsidiaries, affiliated entities or other third parties to use the Software, including the Qilan Developer, if purchased, and/or Qilan Engine, if purchased. However, if, and only if, Customer is a Value Added Reseller (VAR) and has entered into a VAR agreement with Licensor, then, and only then, may the Customer sell the Software, including the Qilan Developer, if purchased, and the Qilan Engine, if purchased, to a parent(s), subsidiary(s), affiliated entity(s) or other third parties. Any sales by a VAR to a third party or parties can only be made if subject to the restriction to the third party buyer(s) that installation of the Qilan engine is limited to one (1) machine per license. Furthermore, the allowance to copy, backup, and transfer the software subject to the terms of this entire agreement shall not in any way modify the

restriction that installation of the Qilan Engine is limited to one machine per license.

Copies: Customer, solely to enable it to use the license for the items purchased, including the Software, including the Qilan Developer, and the Qilan Engine, may make only those copies necessary for archival and back-up purposes provided that any copy made shall include Licensor's copyright and any other proprietary notices. Customer shall have no other right to copy, in whole or in part, these items except in the sale of its services to another party in which case such copy shall include Licensor's copyright and other proprietary notices. Any copy of the Software, including but not limited to those named above, made by Customer for whatever reason is the exclusive property of Licensor.

Modifications, Reverse Engineering: Customer agrees that only Licensor shall have the right to alter, maintain, enhance or otherwise modify the Software, Qilan Developer, and/or Qilan Engine. Customer may modify the Software only within its services to another party as part of its business. Customer shall not, at any time, disassemble, decompile or reverse engineer the Software's computer program, including the Qilan Developer, and/or Qilan Engine.

Material Terms and Conditions: Customer specifically agrees that each of terms and conditions of this Section are material and that failure of Customer to comply with these terms and conditions shall constitute sufficient cause of Licensor to terminate this Agreement. The presence of this Subsection shall not be relevant in determining the materiality of any other provision or breach of this Agreement by either party.

DELIVERY, INSTALLATION, DATA CONVERSION, AND ACCEPTANCE

Delivery: It is understood that Licensor shall deliver the Software, including the Qilan Developer, if purchased, and/or the Qilan Engine, if purchased, to Customer through the Internet and by Customer downloading said above items. Acceptance of any or all of these items and installation of Registration Codes means that Customer agrees to all the terms and conditions of this Agreement.

Data Conversion: Customer shall be solely responsible for data conversion, data entry and verification of data.

LICENSE FEE

In General: In consideration for the license(s) granted by Licensor under this Agreement Customer shall pay Licensor fee(s) as set forth on the Customer's web site at the time of purchase.

Taxes: Customer shall, in addition to the other amounts payable under this Agreement, pay all sales, uses, value added or other taxes, federal, state or otherwise, however designated, which are levied or imposed by reason of the transaction contemplated by this Agreement.

OWNERSHIP

Title: Customer and Licensor agree that Licensor owns all proprietary rights, including patent, copyright, trade secret, trademark and other proprietary rights, in and to the Software, including the Qilan Developer, if purchased, and/or Qilan Engine, if purchased, and any corrections, bug fixed, enhancements, updates or other modifications, including custom modifications, to the Software, Developer, and/or Engine, whether made by Licensor or any third party.

Transfers: Under no circumstances shall Customer sell, license, publish, display, or distribute to a third party the Software, including the Qilan Developer, if purchased, and/or Qilan Engine, if purchased, or any copy thereof, in whole or in party, without Licensor's prior written consent. Customer may then make said transfer of any or all of the above stated items to a third party if, and only if:

Any and all original documentation and user manuals are transferred at the same time as the Software, including the Qilan Developer, and/or Qilan Engine;

The transferee agrees to and assumes all responsibilities of this agreement and so acknowledges in writing to the Licensor, and

Customer notifies Licensor in writing prior to the transfer. Licensor reserves the right to void any transfer in its sole discretion.

PROPRIETARY INFORMATION

Customer agrees that the Software, including the Qilan Developer, and Qilan Engine, contains proprietary information, including trade secrets, know-how and confidential information, and that is the exclusive property of Licensor. Customer shall not disclose any such proprietary information concerning the Software, including the Qilan Developer, and/or Qilan Engine without the prior written consent of Licensor.

WARRANTY

Scope of Warranty: Licensor warrants to Customer that, for a period of ninety (90) days commencing upon the sending of the Registration Codes, the Software, including the Qilan Developer, and/or Qilan Engine will substantially comply with the specifications set forth in the Software Manual. During this warranty period, Licensor shall provide the Customer with technical support. After expiration of the warranty period, Licensor may provide support and maintenance for the Software pursuant to the terms of any Maintenance Agreement entered into between the parties.

Disclaimer of any Other Warranty: THE LIMITED WARRANTY SET FORTH IN THE ABOVE SUBSECTION IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

LIMITATION PERIOD

No arbitration or other action under this Agreement, unless involving death or personal injury, may be brought by either party against the other more than one (1) year after the cause of actions arises.

NO CONSEQUENTIAL DAMAGES

Licensor shall not be liable to Customer for indirect, special, incidental, exemplary or consequential damage (including, without limitation, lost profits) related to this Agreement or resulting from Customer's use or inability to use the Software, including the Qilan Developer, and/or Qilan Engine arising from any cause of actions whatsoever, including contact, warranty, strict liability, or negligence, even if Licensor has been noticed of the possibility of such damages.

LIMITATION ON RECOVERY

Under no circumstance shall the liability of Licensor to Customer exceed the amounts paid by Customer to Licensor under this Agreement.

INDEMNIFICATION

Licensor shall indemnify and hold harmless Customer from and against any claims, including reasonable legal fees and expenses, based upon infringement of any United States copyright or patent by the Software, including the Qilan Developer, and/or Qilan Engine. Customer agrees to cooperate fully with Licensor during such proceedings. Licensor shall defend and settle, at its sole expense, all proceedings arising out of the forgoing. In the event of such infringement, Licensor shall replace, in whole or in part, the Software, including the Qilan Developer, and/or Qilan Engine with a substantially compatible and functionally equivalent computer program or modify the Software to avoid the infringements.

TERM AND TERMINATION

Effective Date: This Agreement and the license granted hereunder shall take effect immediately upon receipt of the Registration Codes for the Software.

Termination: Each party shall have the right to terminate this Agreement and the license granted herein upon the occurrence of the following events (an 'Event of Default'):

In the event the other party violates any provisions of this Agreement; or

In the event the other party terminates or suspends its business, becomes subject to any bankruptcy or insolvency proceeding under Federal or state statute, becomes insolvent or subject to direct control by a trustee, receiver, or similar authority, or has wound up or

liquidated, voluntarily or otherwise.

Notice and Opportunity to Cure: Upon the occurrence of an Event of Default, a party shall deliver to the defaulting party a Notice of Intent to Terminate that identifies the Event of Default. If the Event of Default remains uncured for thirty (30) days, the party may terminate this Agreement and the license granted herein by delivering to the defaulting party a Notice of Termination, by e-mail or in writing, that identifies the effective date of the termination, which date shall not by less than thirty (30) days after the date of delivery of the Notice of Intent to Terminate.

Procedure: Within ten (10) days after termination of the license, Customer shall delete or destroy all copies of the Software, including the Qilan Developer, and/or Qilan Engine.

ASSIGNMENT

Customer shall not assign or otherwise transfer the Software, including the Qilan Developer, and/or Qilan Engine or this Agreement to anyone, including any parent, subsidiaries, affiliated entities or third party, or as part of the sale of any portion of its business, or pursuant to any merger, consolidation or reorganization, without Licensor's prior written consent.

FORCE MAJEURE

Neither party shall be in default or otherwise liable for any delay in or failure of this performance under this Agreement if such delay or failure arises by any reason beyond its reasonable control, including any act of God, any acts of the common enemy, the elements, earthquakes, floods, fires, epidemics, riots, failures or delay in transportation or communications, or any act or failure to act by the other party or such other party's employees, agents or contractors; provided, however, that lack of funds shall not be deemed to be a reason beyond a party's reasonable control. The parties will promptly inform and consult with each other as to any of the above causes, which in their judgment may or could be the cause of a delay in the performance of this Agreement.

ARBITRATION

The parties shall settle any controversy arising out of this Agreement by arbitration in New Hampshire in accordance with the rules of the American Arbitration Association. A single arbitrator shall be agreed upon by the parties, if the parties cannot agree upon an arbitrator in witting thirty (30) days, then the parties agree that a single arbitrator shall be appointed by the Americans Arbitration Association. The arbitrator may award attorneys' fees and costs as part of the award. The award of the arbitrator shall be binding and may be entered as a judgment in any court of competent jurisdiction.

GENERAL PROVISIONS

Complete Agreement: The parties agree that this Agreement is the Complete and

exclusive statement of the agreement between the parties, which supersedes and merges all prior proposals understandings, and all other agreements, oral or writing, between the parties relating to this Agreement.

Amendment: This Agreement may not be modified, altered or amended except by written instrument duly executed by both parties.

Waiver: The waiver or failure of either party to exercise in any respect any right provided for in this Agreement shall not be deemed a waiver of any further right under this Agreement

Severability: If any provision of this Agreement is invalid, illegal or unenforceable under any applicable statute or rule of law, it is to that extent to be deemed omitted. The remainder of the Agreement shall be valid and enforceable to the maximum extent possible.

Governing Law: This Agreement and performance hereunder shall be governed by the laws of the State of New Hampshire.

Read and Understood: Each party acknowledges that it has read and understands this Agreement and agrees to be bound by its terms.

Read and Understood: Each party acknowledges that it has read and understands this Agreement and agrees to be bound by its terms.

Acknowledgements

Qilan binary code includes work based on Software developed by Omni Development.

Original, unmodified binary code is available at:

http://www.omnigroup.com/community/developer/sourcecode/

Omni Development makes no warranties express or implied, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. Under no circumstances shall Omni Development be liable to you or any other person for any indirect, special, incidental, or consequential damages of any kind related to or arising out of your use of the software, even if Omni Development has been informed of the possibility of such damages.

We wish to thank Scott Keith (OpenBase International) and Geert Clemmenson (FrontBase Software) for their cooperation and assistance in the development of the Qilan Adapters.

We also want to personally thank the many developers and supporters who believed in the Qilan vision.

Documentation Icons

Instruction/Guidance

Techniques are explored and developed using concrete examples.

Helpful Hints

Ideas and suggestions to make using Qilan easier.

Special Feature

Unique Qilan capabilities or other distinctive characteristics.

Caution

What to look out for or other items needing special attention.

Hacker's Advice

When you want to get 'technical'. Suggestions to enhance Qilan's functionality.

Knowledge of UNIX Helpful

Suggested readings or topics that might prove helpful.

Knowledge of HTML Helpful

Suggested readings or topics that might prove helpful.

Introduction (Pronounced: kai´·lan)

Qilan is a database system designed to operate on a web server. It is a complete environment for the creation of user entry/output screens, logical data manipulation and data base access. With Qilan, the developer can easily link SQL database engines with the world wide web.

Qilan can be used by developers who don't like to program. Developers do not have to know or rely upon SQL or remember complicated or intricate symbols to create complex queries, manipulate data, or access databases.

Qilan works with your existing databases and web development tools. Developers merely need to create an access to their database. Qilan provides the tools to build complete professional web-based applications based on that data.

Qilan is fully extensible. Internet users can interact with scripts (UNIX, Perl, C, etc.) to run external routines, acquire data or execute automated processes.

The Qilan development environment is a native Mac OS X application. The Qilan engine is designed to run in concert with a web server as a traditional CGI or FastCGI on Mac OS X.

A FastCGI is a high performance, scaleable, web server interface that offers many advantages over the traditional CGI. We encourage you to read about FastCGIs and learn about how to optimize performance, and make the most of the many available features. My favorites include speed improvements, load distribution and the ability to have a web server in one location and the fcgi in another.

Here is a web site that offers lots of useful information:

http://www.FastCGI.com/

Refer to the Appendix, at the end of this manual, for specific instructions as to how to obtain, install and configure FastCGI.

Hardware/Software Requirements

Qilan (v2.8.1) runs native on Mac OS X or Mac OS X Server; version 10.3.x or higher (Darwin Kernel) is required. We suggest you have a minimum of 128 megabytes of RAM. Additional RAM should be considered if you plan to build a complex project or serve a large number of web users.

Did you know you can develop a Qilan project, server web users, run a database and use your computer for other tasks, all at the same time! Actually, Mac OS X makes this all possible. If you plan to do this, get yourself lots of RAM.

Qilan consists of two separate applications: The Qilan Developer and the Qilan Engine.

Qilan Developer (Qilan.app) is used to create Qilan projects and is automatically installed in `/Applications`. A project contains input/output screens (HTML), links to back-end databases and data logic.

For development (Qilan Developer), we suggest a 19-inch or larger monitor

The Qilan Engine runs as either a traditional cgi (common gateway interface) or as a fcgi (fast common gateway interface). An engine is automatically installed into `/Library/WebServer/CGI-Executables` and `Library/WebServer/FCGI-Executables`. Mac OS X automatically installs the Apache web server (sometimes referred by 'httpd') by default, but you may need to activate it in the Sharing Preferences Panel. Refer to your system documentation for details.

If you choose to operate Qilan as a cgi, no additional installations or configuration is necessary. The engine is referenced as follows:

```
cgi-bin/qilan.cgi
```

If you choose to operate Qilan as a FastCGI, please refer to the Appendix for installation and configuration details. Installation of FastCGI will modify your httpd.config file and add additional items to `/Library/Qilan/`. The fcgi engine is referenced as follows:

```
fcgi-bin/qilan.fcgi
```

We recommend the use of a high quality backend SQL engine for optimum performance. Databases may be located on any machine on the same Internet or Intranet as the Apache web server. Communication is via TCP/IP. Databases, such as OpenBase, FrontBase, FileMaker, Helix RADE, MS SQL, MS Access, MySQL, Paradox, Informix, Sybase and Oracle, amongst others, are accessible via Qilan. Please check our website at `www.qilan.com` for an updated list of supported databases.

Qilan can access multiple databases, local or remote, singularly or simultaneously. All that is required is logon information and appropriate database schema.

Qilan communicates to most databases via JDBC (Java DataBase Connectivity). Qilan will automatically install compatible database JDBC adapters. Database companies frequently update their software so be sure to obtain the most recent JDBC adapter from your database supplier.

> *Helix RADE is an exception, as it does not natively support JDBC. We supply a bridge application, "Osmosis Gateway", that links Qilan with Helix RADE. Please refer to the Appendix for additional connectivity information.*

We suggest you visit `http://java.sun.com/products/jdbc/` for information about JDBC technology and updated listings of JDBC drivers.

If you download a newer driver than those supplied with Qilan, be sure you place the driver in `/Library/Java/Extensions` and name it exactly as the one you are replacing.

When replacing JDBC drivers, insure permissions are correctly set. Qilan requires that at a minimum, the owner, group and everyone be granted *read* access.

Installation

Installing Qilan is easy and straightforward.

Insure you are logged in as a “System Administrator”. You can verify your user status by opening the User Preference Panel.

Have you registration keys handy, as the installer will request them.

The Qilan install package is typically compressed, so you might have to decompress it before it can be installed. To do this, locate UnStuffit® or OpenUp® and decompress the file.

UnStuffit is included with Mac OS X; OpenUp is available from www.stepwise.com.

After the file has been expanded, its icon will change to a package. Now, double click on the package icon and the installer program will be launched.

The installation process is automated and will lead you through a number of screens.

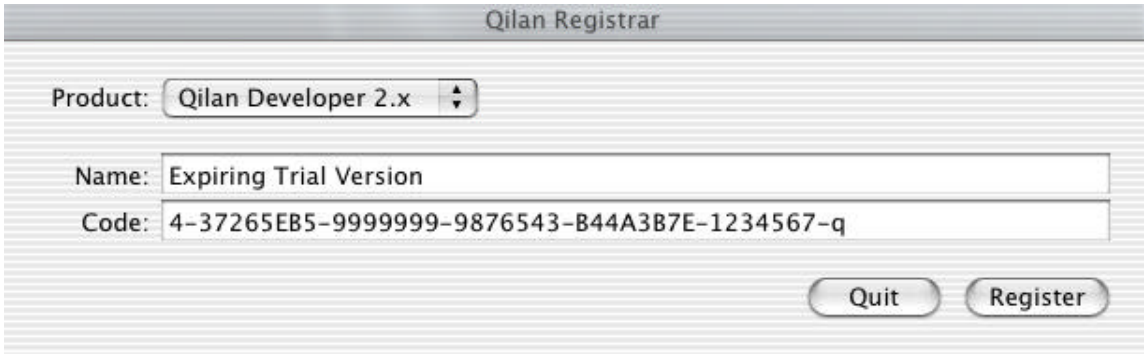
Registration

Qilan is registered by registrant name, product type/version, IP or MAC address, and expiration date (if any). These items are then used to generate a registration code. Codes can be obtained by visiting the CommonGround website, www.qilan.com, and navigating to 'Registration Codes'.

If a DEMO code is requested or provided, the Qilan engine will only respond to local requests and may be subject to expiration.

The Qilan Developer is not linked to a specific machine although a valid name and code is required during registration. The Qilan Engine (cgi/fcgi), in addition to a valid name and code, is linked to a specific IP or MAC address.

During the installation process, you will be asked to register. Have your registration name and codes available. The Registrar application will open automatically.



Select the product type: Qilan Developer or Qilan Engine; then type/paste your registration name and code as provided by CommonGround Softworks, Inc. ***The name under which the product was registered and code must be entered exactly as provided***. Click 'Register' when you are finished with your entries.

The Registrar will not alert you to invalid registration codes. You will be informed of invalid registrations upon opening the Qilan application or accessing a Qilan webpage.

When the installer displays the license agreement, take a moment and read it thoroughly. It contains important information as to your rights and obligations. CommonGround Softworks, Inc. wants you to enjoy and prosper with Qilan, but will aggressively enforce the provisions of the license agreement.

When you are ready to continue, click, "Continue". The installation process will finish automatically.

QilanRegistrar is used to register the Qilan Developer and/or the Qilan Engine independent of the installation process. It may be used separately if additional registrations are desired or necessary. It is installed in:

`/Applications`

The Registrar is a very simple application. When launched, you will be asked to identify the product you wish to license and enter your registration name and code. If a license exists (current or expired), it will be shown. After you make your entries, click "Register", and then quit.

Updating a Project

Qilan projects may need to be updated due to the inclusion of new features or changes the file format. When this becomes necessary, a dialog will be presented upon opening the project with the Qilan Developer.

Updating a project file will temporarily convert the file, then open it. If you wish to make the update permanent, choose File > Save. This will discard the temporary copy. Optionally, choosing File > Save As… will save a copy of the updated project without affecting the original (unupdated) project.

Closing the project without saving will maintain the project without it being updated.

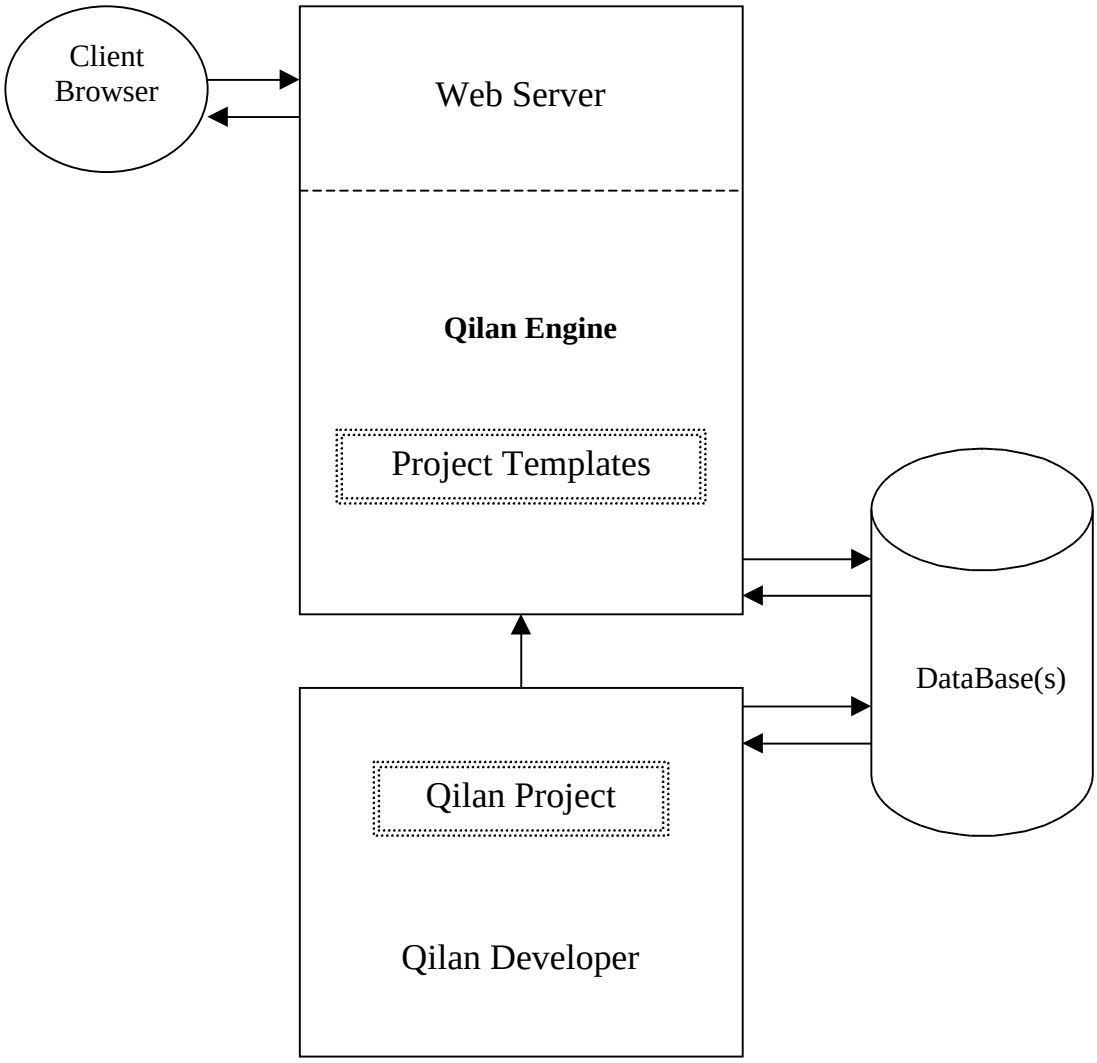
Updating a project file will make is unreadable by earlier versions of the Qilan Developer and Qilan Engine.

Plug-Ins

From time to time, additional features and/or adapters may be made available from CommonGround Softworks via plug-ins. Plug-ins are specialized files useable only by Qilan. Both the Qilan Developer and Qilan Engine require separate plug-in files.

Plug-Ins are placed in `/Library/Qilan`.

The Qilan Schema Overview



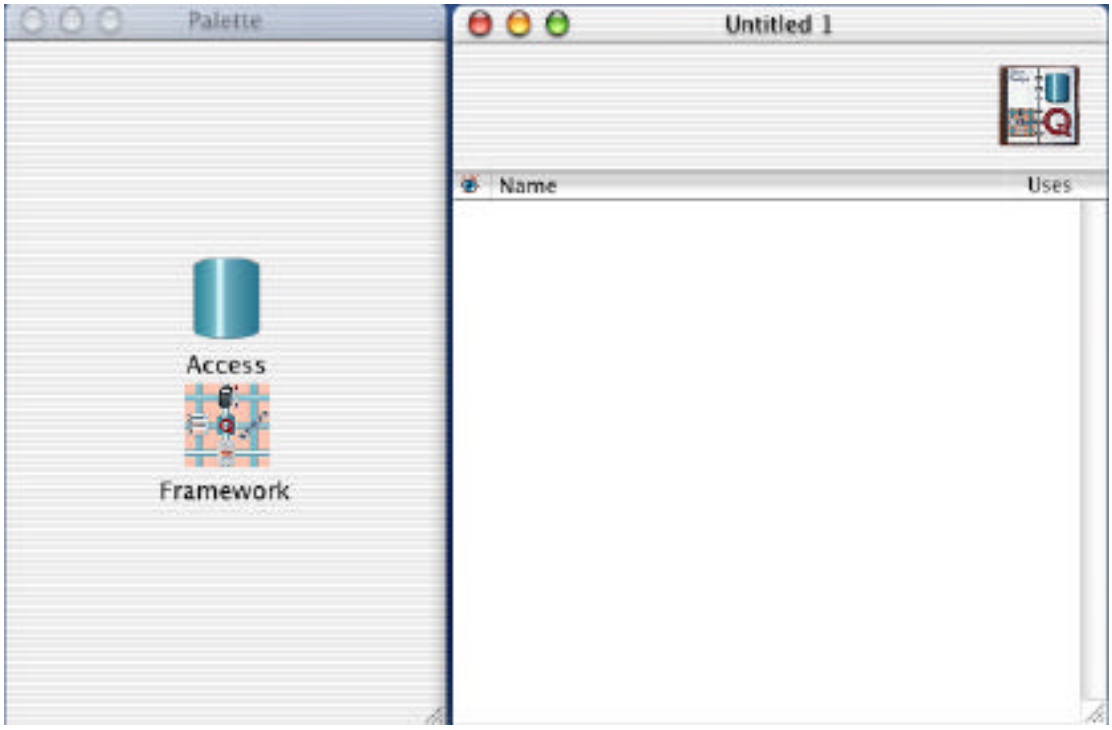
A client browser accesses a project template on the web server. The page is processed by the Qilan Engine (qilan.cgi). Processing may include database access; followed by a client return.

Project templates are specially formatted WebTemplates ‘exported’ by the Qilan Project. The Qilan Project imports/exports database schema, controls and manipulates database logic and formats the client response.

Qilan Icons and Windows

Object Hierarchy

Qilan uses an object hierarchy. A Qilan object can be seen either closed (as an icon), or open (as a window).



Windows function as ‘containers’. For example, the project window (shown above with the palette to the left) contains the access, the access window contains the table, and the table window contains fields, relationships and abacus expressions. There is an implied dependency relationship, namely, the table window is dependent upon the access window and the access window is dependent upon the project window. Visually however, any window may remain open (or docked) regardless of the state of parent window.

If you have multiple windows open, refer to the window title bar for the full path (parent to child). If you quickly want to return to or open the parent window just select, Icon > Parent Window, while a child window is active or an icon is highlighted.

Icons, which are used by or contained within other objects, cannot be discarded. Icons used by another object, will display a value greater than zero in its 'home' window under the "Uses" column. You can ascertain where and how an icon is used by double clicking the 'Uses' column. See, "Icon Uses" in the following section.

Icons, which contain other objects, for example, the access, and framework icons, cannot be thrown away if they hold other objects.

Project

The Project window contains all the elements necessary for access, control, design and manipulation of database data. It is the main (parent) window and contains Access and Framework icons. If the project window is closed, you will be prompted to save your work then all of it's related windows will close.

Access

The access icon refers to an existing database. It contains logon information, a representation of a database schema (tables/fields) and Qilan objects used to manipulate database data.

Access Table

An Access Table is a graphical presentation of an external database structure, namely the “table” or “relation”. A database consists of one or more tables, which may be logically related to one another.

A database table does not really exist in Qilan, but rather is represented by Qilan. The actual table structure exists in the database. This approach allows you the freedom to create logic without having to worry about connection issues.

When Qilan references a table in a database, it does so via its ‘External Name’.

> External Name: This is the name of the table, as it will be known by the database. Qilan will link to this table, by name, when data is retrieved, updated, deleted or created. Therefore, you should not change this name unless you also change the corresponding table name in the database. If you are creating a new table, insure the table created in the database has the same name. External Names are case sensitive.

External table names are required and cannot be left blank; they also must be unique for each database. You will receive errors if they are left empty or are not unique. It is also suggested that names not contain spaces and be kept as short as possible.

Qilan references an access table by its name, as entered in the Access window.

Access Field

Fields represent the storage mechanism for data elements and are represented within tables. Fields are designed to store data as dates, flags, strings or numbers (floating point and integers).

Field types specified by the database are accessible by Qilan.

A table field does not really exist in Qilan, but rather are represented by Qilan. The actual field exists in the database.

When Qilan references a field in a database, it does so via its 'External Name'.

> External Name: This is the name of the field, as it will be known by the database. Qilan will link to this field, by name, when data is retrieved, updated, deleted or created. Therefore, you should not change this name unless you also change the corresponding field name in the database. If you are creating a new field, insure the field created in the database has the same name. External Names are case sensitive.

External field names are required and cannot be left blank; they also must be unique for each table. You will receive errors if they are empty or are not unique. It is also suggested that names not contain spaces and be kept as short as possible. SQL compliant databases will return errors if reserved words are used as field names. Please refer to the Appendix for a complete list of reserved words.

Unlike tables, most databases offer additional field specifications, such as type, width, precision, etc. Please refer to Field Specifications for more information.

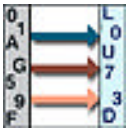
Access Relationship

A relationship defines a link or connection between record fields; there is an implied state of equality. The link may be within the same table or between two different tables, but always within the same database.

Relationships are not actual database structures, as with tables or fields, but derived values created by Qilan.

Qilan goes beyond the traditional relationship concept used by many databases. For example, depending up the sophistication of the database, Qilan will support concatenated, derived and 'on-the-fly' links. In addition, Qilan supports relationship semantics.

Access DataFlow



DataFlows define data movement. Based on a relationship link, a dataflow uses that link to replace data in the target table. Dataflows are triggered or activated by specific 'Q' tags on the WebTemplate. Access dataflows are used specifically to move data from one table to another table within the same database.

DataFlows are not actual database structures, as with tables or fields, but derived values created by Qilan.

Access Abacus

Abacus expressions manipulate database data. They are used to query, format, logically handle data fields and/or process other abacus expressions.

Abacus expressions are not actual database structures, as with tables or fields, but derived values created by Qilan.

Framework

Frameworks are Qilan structures that are analogous to database tables. They may contain fields, abacus expressions, relationships, dataflows and most importantly, WebTemplates. In contrast with Access tables, there are two very important differences:

- Although frameworks contain persistent structural elements, such as fields, they are only accessed during qilan.cgi execution; and

- Framework 'data' is temporary. Frameworks do not store data, but rather act upon it. The WebTemplate, when properly configured, triggers all actions, which occur during qilan.cgi execution.

You can create as many frameworks as you need. Now that's said, when do need to create more than one? This is not a simple answer. A framework can serve many uses from design organization to functional groupings. As a general rule, think of a framework as the background 'support' for a WebTemplate. If one or more WebTemplates tend to share the same set of objects (fields, abacus expressions, etc.), then they should all be in the same framework. However, do not over do it! In later chapters, we will discuss how to share data between frameworks.

Framework Field

Framework fields represent transient storage objects for HTML INPUTs, assigned values and various input/output mechanisms. They are not 'typed' and can be of any size (up to the maximum file size for your operating system). Framework fields can be designated as session variables. Please refer to the section on Sessions for more information.

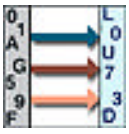
Framework Abacus

Abacus expressions manipulate data. They are used to evaluate, format, logically handle data fields and/or process other abacus expressions.

Framework Relationship

A relationship defines a link or connection between framework data and table record fields; there is an implied state of equality. The link is always between framework data and a specific database table.

Framework DataFlow



DataFlows define data movement. Based on a relationship link, a dataflow uses that link to replace data in the target table. Dataflows are triggered or activated by specific 'Q' tags on the WebTemplate.

Framework WebTemplate

WebTemplates have four primary functions:

- WebTemplates format user and database data (input/output). Qilan defaults to using HTML, but with proper modifications, any DTD can be used.

- WebTemplates retrieve database data. Data can be queried using a flag formatted abacus icon or with standard SQL. Fields or derived values are accessible as well as standard summary functions.

- WebTemplates store/update database data. Using relationships and dataflows, data can be created, updated, deleted or transferred from one database to another.

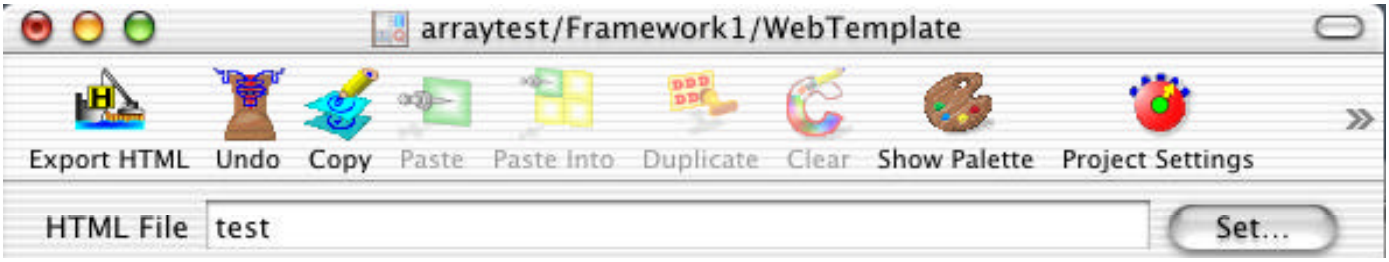
- WebTemplates logically manipulate data and program execution. Using the wide variety of 'Q' tags, logical steps can be created to test, evaluate, loop or control procedural constructs.

A Framework may have as many WebTemplates as the designer requires. Icons created in the Framework may be used by more than one WebTemplate.

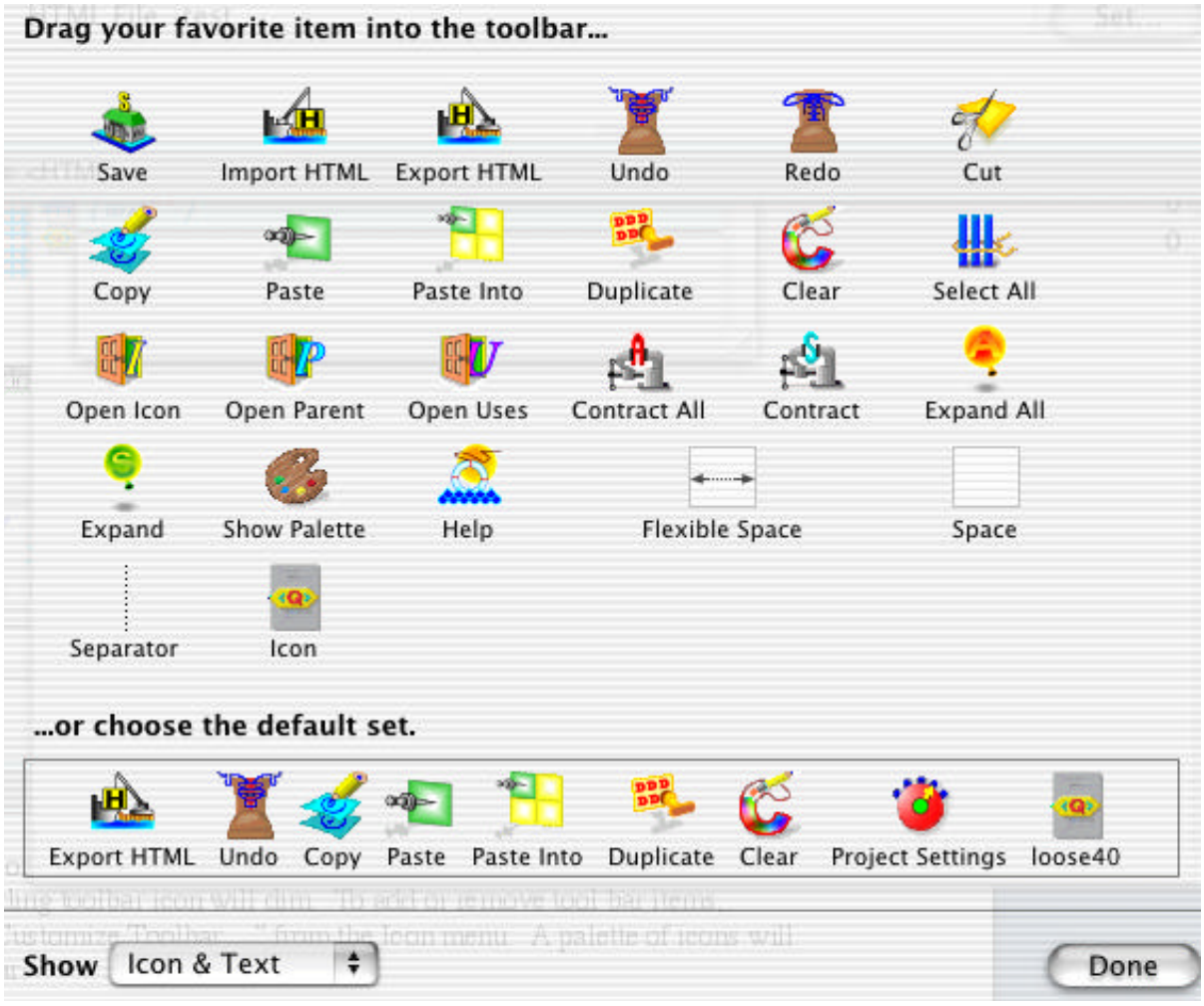
In a very real way, the WebTemplate is where is 'all comes together'. The Qilan engine executes the WebTemplate. All your formatting, actions, procedures and logic must be present or linked to the WebTemplate. Later on you will learn more about WebTemplate 'Q' tags. It is by strategically using these tags that Qilan objects are accessed and triggered.

Customizing the Toolbar

Qilan windows can be customized by adding text and/or icons to the window toolbar.

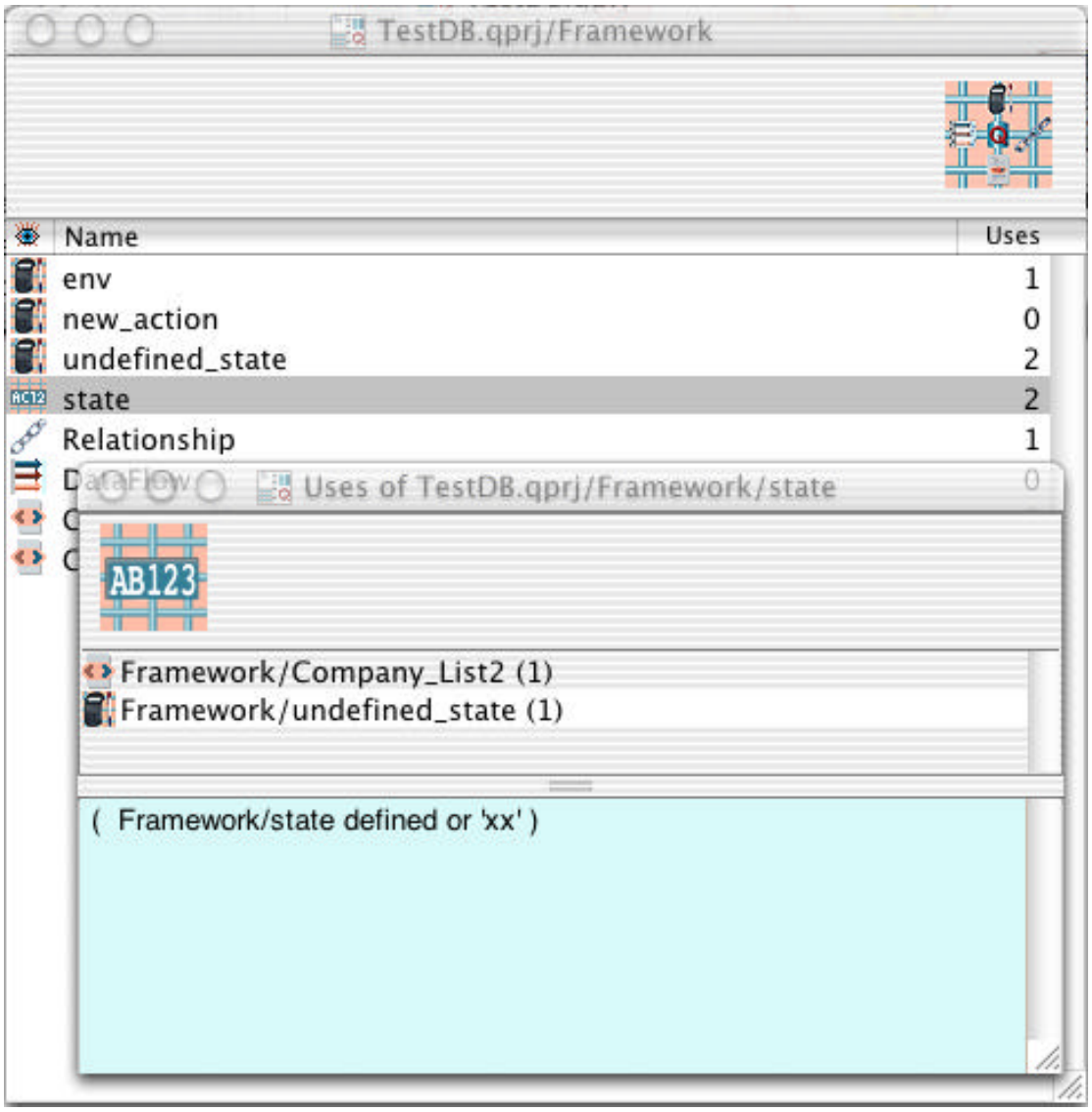


Toolbar icons refer to menu commands. If a menu item is dimmed, the corresponding toolbar icon will dim. To add or remove tool bar items, choose, “Customize Toolbar…” from the Icon menu with the window open and active. A palette of icons will then appear.



Icon Uses

Qilan provides an easy way to determine where and how an icon is used by another icon. This is referred to as, “Uses”. Windows that display a Uses column, Project, Framework, Access and Table, show a numerical value. This number corresponds to the number of times that icon is used by another icon. Double clicking on the number will open the Uses window as shown below:



The Uses window is divided into two panes. The upper pane lists ‘parent’ icons – those icons using the selected (or child) icon. Highlighting a parent icon shows how the ‘child’ icon is used in the lower pane.

While viewing the lower pane, double click on the selected icon to open the enclosing object. The selected icon will be highlight and centered.

This is a great way to locate an icon being used in a WebTemplate. As your projects grow in complexity, an icon can become lost among similarly names icons. This feature comes to the rescue.

Double clicking on a parent icon opens the icon for review or editing.

If you want to see where and how a parent icon is used, in other words, see its parent, highlight the parent icon and select, “Uses…” from the Icon menu. Alternately, highlight the parent icon and use the command key equivalent: command–option–U.

Inside other windows, such as the abacus or WebTemplate, the Uses window can be accessed by first highlighting the icon, then selecting, “Uses…” from the Icon menu. Alternately, highlight the icon and use the command key equivalent: command–option–U.

Parent Icon

All icons, except the Project itself, belong to a parent. For example, WebTemplates belong to the Framework and tables to the Access. When you have many windows open on the screen at the same time, it is often useful to identify the parent icon. Qilan will open the parent icon for any icon in any window by first selecting the icon, then choosing, “Parent Icon…” from the Icon menu.

Alternately, you may use the key combination, control – command – P.

Copy/Paste Icon

Qilan icons can be duplicated, copied, pasted or dragged within or between windows, or between projects. When an icon is created via a paste, drag or duplication, Qilan will also attempt to create referenced icons. For example, if an abacus expression contains a field, that field will also be created when the abacus expression is copied from one Framework to another Framework.

Copy/Paste operates differently depending upon whether a parent or child icon is copied. Parent icons, Frameworks, Accesses and Tables 'own' other icons. When a Framework is copied, for example, all the child icons and their dependencies are also copied.

However, if a child icon is copied, such as a WebTemplate, referenced icons are created in name only. For example, if a WebTemplate, containing an abacus expression, is copied from one Framework to another, a new WebTemplate and abacus will be created, but the abacus itself will be empty.

To insure the contents of the parent icon is copied, copy or drag the parent and child icons together by holding down the command key, then selecting the desired icons.

If an icon is duplicated, pasted or dragged into a window containing an icon of the same name, the existing icon will be maintained and not replaced. The newly created icon will be appended by a numerical increment (i.e., field, field1, field2, etc.). The default naming convention for duplicated icons is to add a numerical increment to the name of original icon.

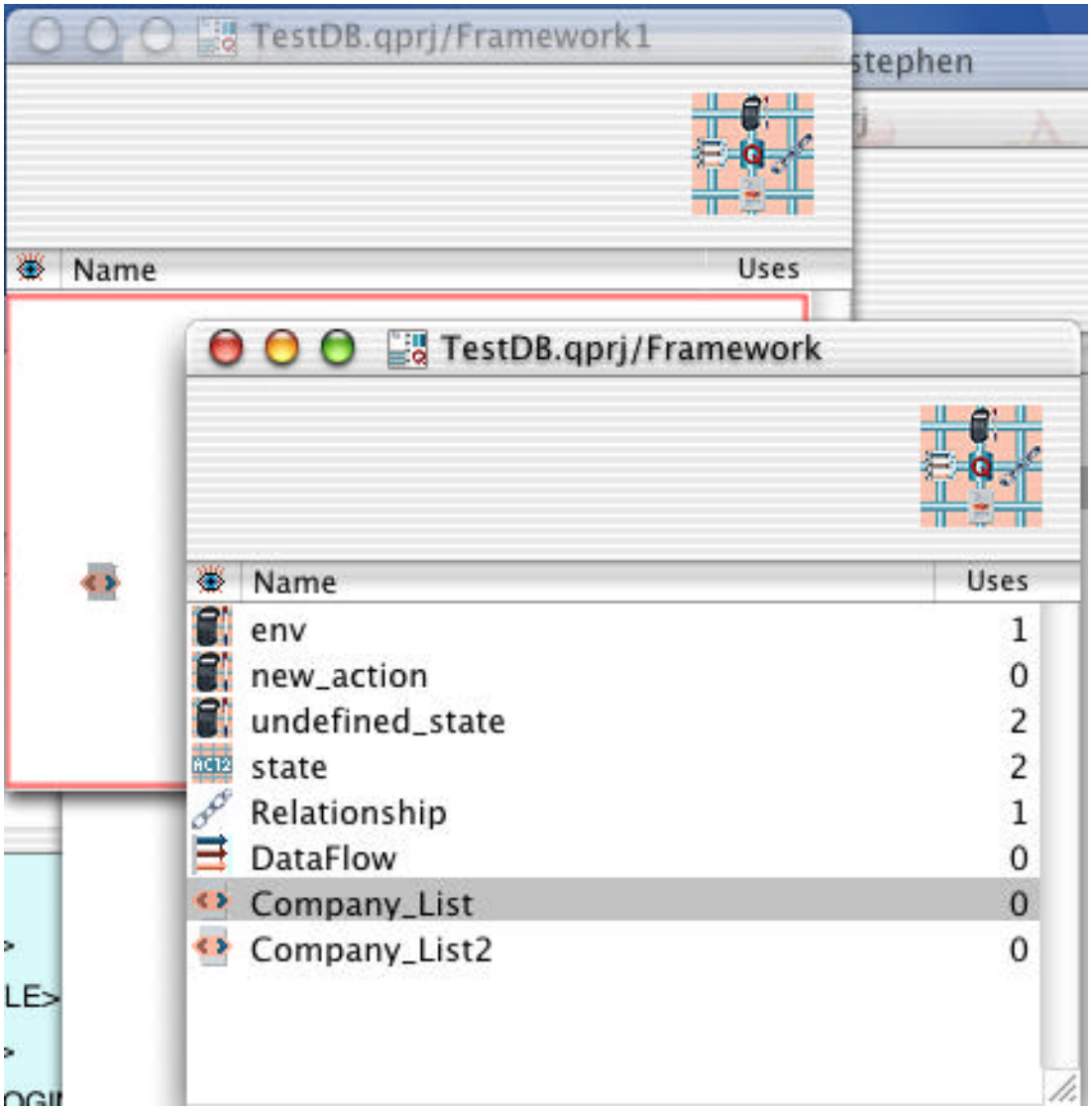
Icons created, pasted or duplicated in an Access or Table, will be created with the same external names and parameters as the original icons. Before you export the database schema, insure you review icon parameters so as to avoid duplicate external names. Qilan names and links will be maintained.

Expressions created in abacus windows can be copied/pasted or dragged into other abacus windows. To copy an expression, highlight the surrounding parenthesis then select Copy, from the Edit menu. You may select any portion of an expression, including just an operand or icon. To paste the expression, open an empty abacus window and select Paste, from the Edit menu. If the abacus you are pasting into has an existing expression, the newly pasted expression will be placed into the lower portion of the window.

Dragging an Icon

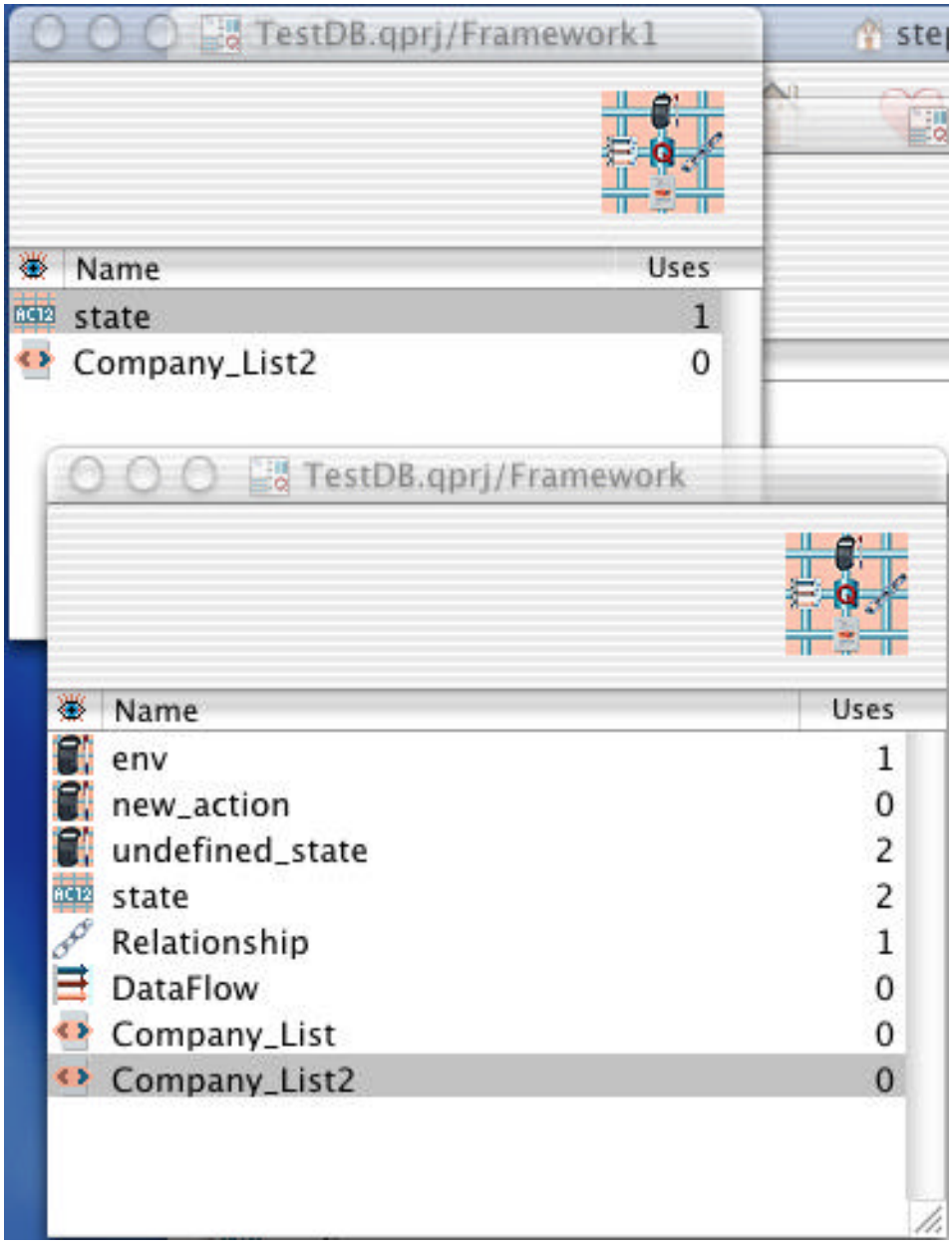
Dragging an icon has the same meaning as copy, then paste.

To drag an icon, select it by highlighting it with the mouse. Then, continuing to hold the mouse button down, drag it to the target window.



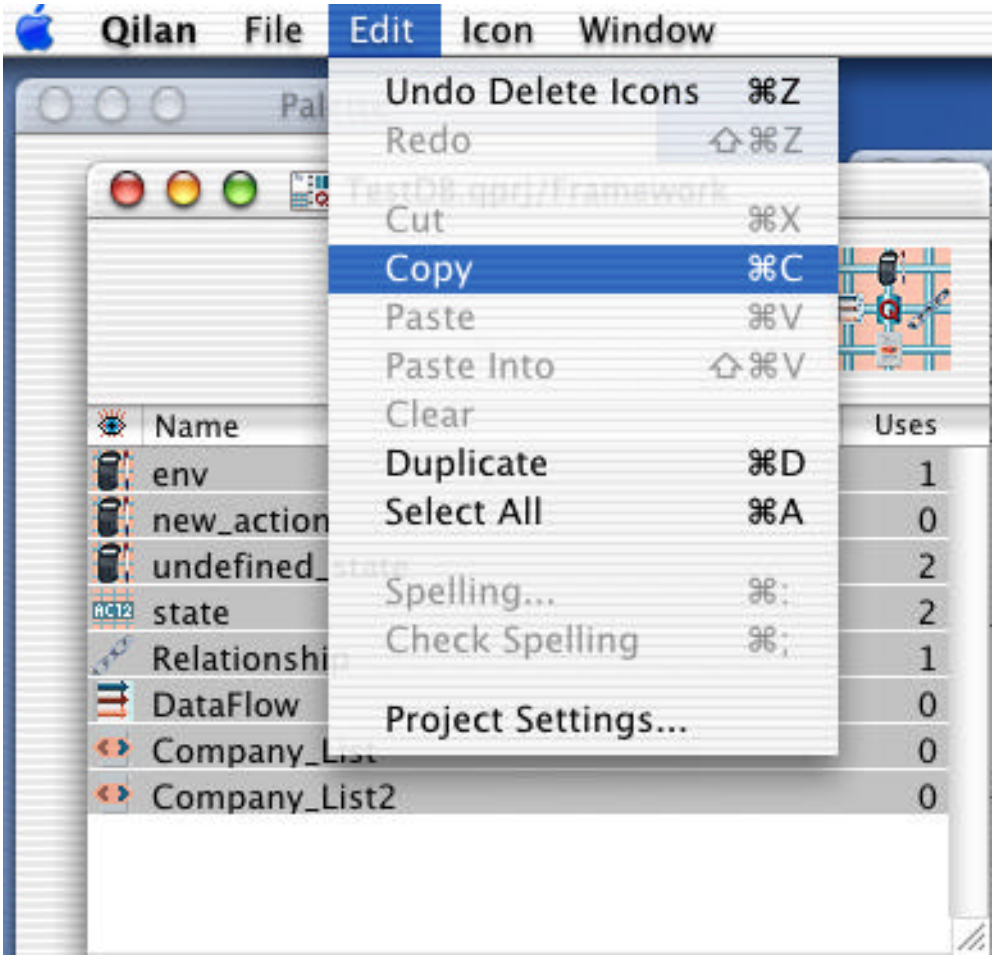
If the target window is ‘eligible’, a red square will appear. Release the mouse button and the icon will be copied.

Immediately after the drag is completed, any associated icons will be created. In the example below, the field, “state”, contained or referenced the WebTemplate is automatically created”.

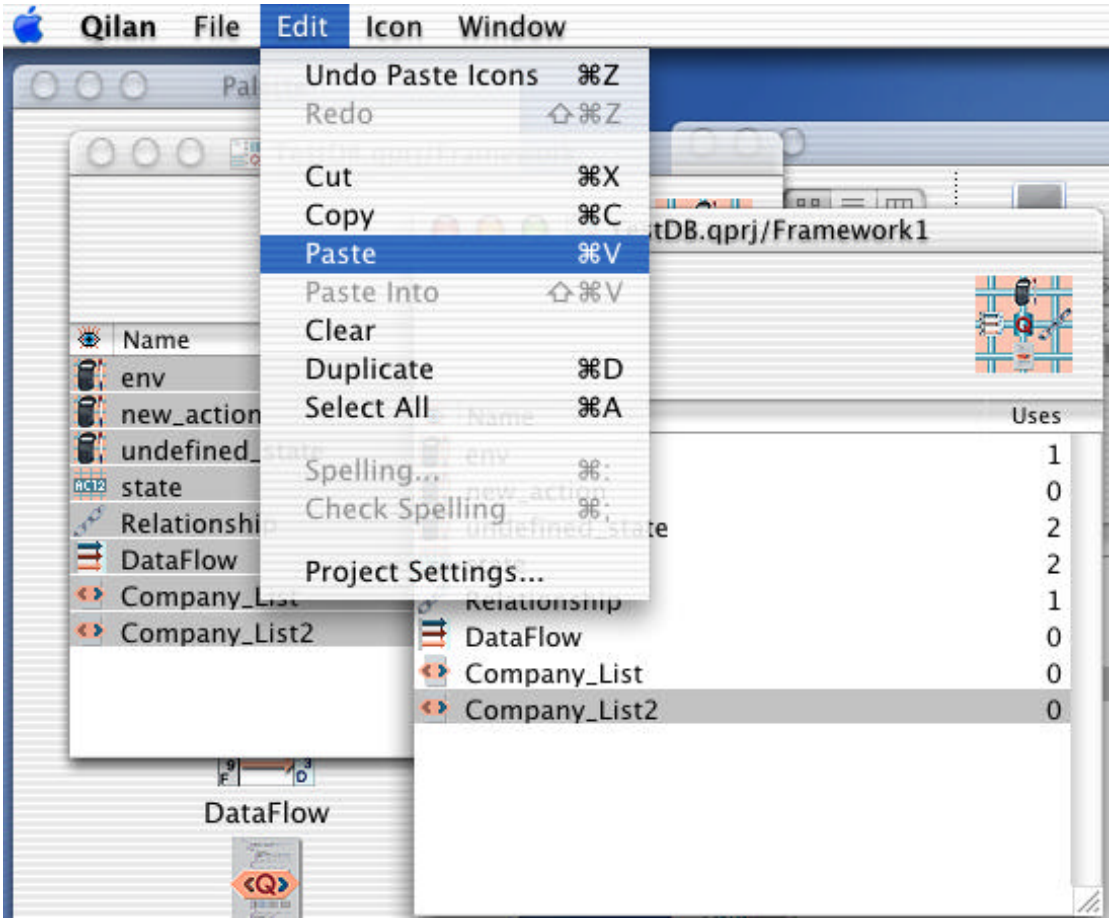


Copying an icon places it in temporary memory. If an eligible Qilan window is then opened or activated, the paste command will be available. Multiple icons (within the same window) can be copied by holding down the Shift or Command key, then using the mouse to select the desired icons.

If you want or need to share project 'snippets', open an empty Project window then drag or paste the desired icons. The new Project should then be saved and distributed.



If the icon being pasted has the same name as an existing icon, the pasted icon will be renamed. The new name will be appended by a numerical increment.

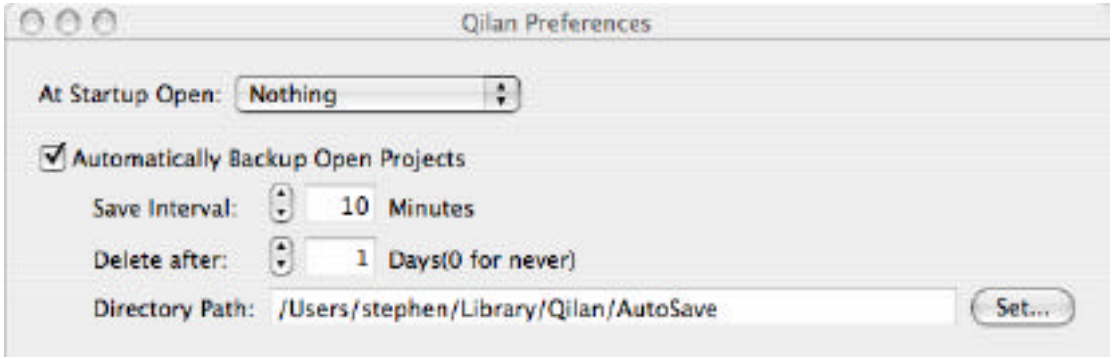


When naming field, use caution appending icon names with numbers. Here's what can happen… Assume you name an icon "field1". Duplicating "field1" will result in "field2". But what happens if you already have an icon named "field2"? Qilan will create "field3". The short story is that it can get very confusing.

Global Preferences

The Qilan Developer maintains global preferences that apply to the way the developer behaves. This is in contrast to Project Settings (following section) that are applicable only to the open project.

Global preferences can be accessed from the Qilan menu by selecting, "Preferences…"



At Startup Open refers to what project the Qilan developer will open when the application is launched. There are three options:

> Open last projects: The last saved project will be opened.
>
> New project: An empty project will be opened.
>
> Nothing: The application will be launched, but no project will be opened.

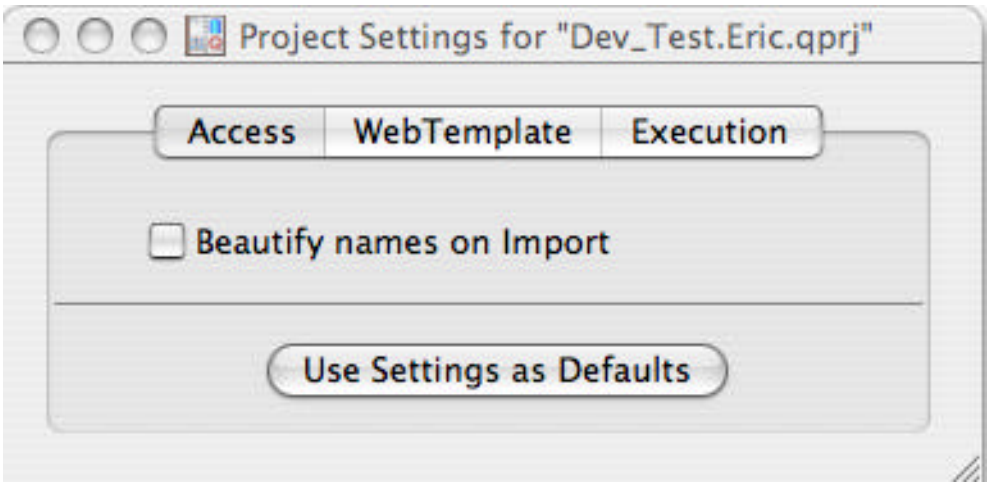
Checking 'Automatically Backup Open Projects' will create a copy of the open Project in the specified location at the selected interval. The option to, 'Delete After', refers to how long an interval backup will be retained. If this option is set to '0', the interval backups will never be deleted, thus resulting in a new backup each time the project is opened. We recommend this setting be greater than 0, depending upon the number of interval backups you desire and available disk space.

Uncheck this option to prevent backup copies from being created.

Project Settings

Project settings are accessed from Edit > Project Settings… Preferences are divided into three panes: Access, WebTemplate and Execution. Clicking on the tab to opens the pane.

The **Access** pane refers to those settings that effect the importation of the data base schema.



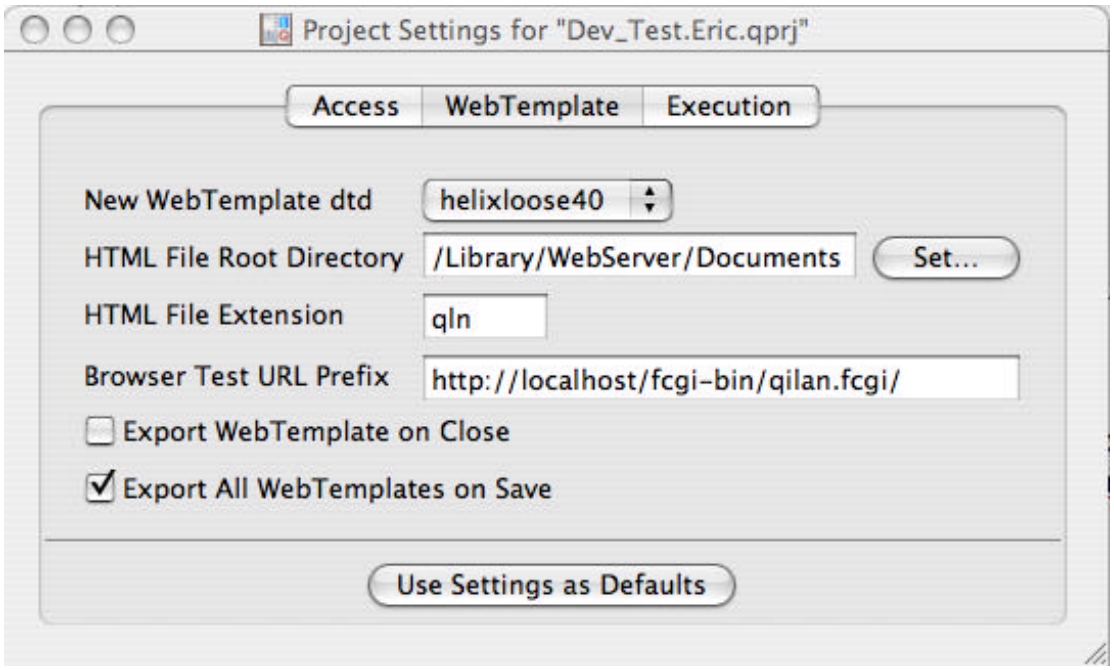
Beautify Names on Import

> When the database schema is imported, Qilan can adjust the names so that they are more readable. Designers of SQL databases sometimes use cryptic names for tables and fields. Checking this option attempts to make them more 'English like'. Double clicking in the Value column will display a check mark. This means the option is selected.

Use Settings as Defaults

> Click this button to use the settings made under this tab when new projects are created.

The **WebTemplate** pane refers to those settings that effect webtempate defaults and overall operation.



New WebTemplate DTD (Document Type Definition)

Qilan ships with four standard html DTDs: 3.2, 4.0 loose, 4.0 strict, and frameset 4.0. Each time a new WebTemplate is created, Qilan refers to this setting to determine what type DTD will be used.

DTDs for existing WebTemplates cannot be changed.

A project can have a mixture of DTDs. Choose the DTD that best matches your HTML design, client browser software and desired functionality.

Four additional DTDs are available containing Helix specific ‘Q’ tags corresponding to the traditional HTML DTDs. If you plan on accessing Helix, you *must* use a Helix DTD.

HTML File Root Directory

When a WebTemplate is exported for qilan.cgi execution, it must be placed into a specific directory. This option allows you to specify that directory. Click the Set… button to choose a directory or type the directory path in the space provided.

Important Security Considerations

Exported WebTemplates are specially designed for the Qilan Engine. These files contain sensitive data, such as login information, and must be protected from unauthorized access. If you are not familiar with designing access controls, we suggest you consult with a web server specialist.

Regardless of the access restrictions you implement, we strongly urge you to test and verify your configurations. Site security is a highly complex issue and should be professionally reviewed on a regular basis.

HTML File Extension

The default suffix used for HTML documents, when exporting WebTemplates. The use of the suffix is optional, but may be useful in identifying files of a certain type. Some applications, notably web servers and browsers, use suffixes for MIME type extensions. Entering ‘qln’, for example, will appear as, “myfile.qln”.

The extension is used in conjunction the WebTemplate icon name when the WebTemplate is not given a specific HTML name.

Browser Test URL Prefix

This is the URL string (preceding the webtemplate name) that will be used by the browser when previewing the webtemplate in a web browser. Your default internet browser will be used as set in System Preferences. Refer to the Apple System documentation for details.

Export WebTemplate on Close

A WebTemplate is designed within the development environment, but cannot be used unless it is exported. Check this option to automatically export the WebTemplate each time the WebTemplate window is closed. See the WebTemplate documentation for more details.

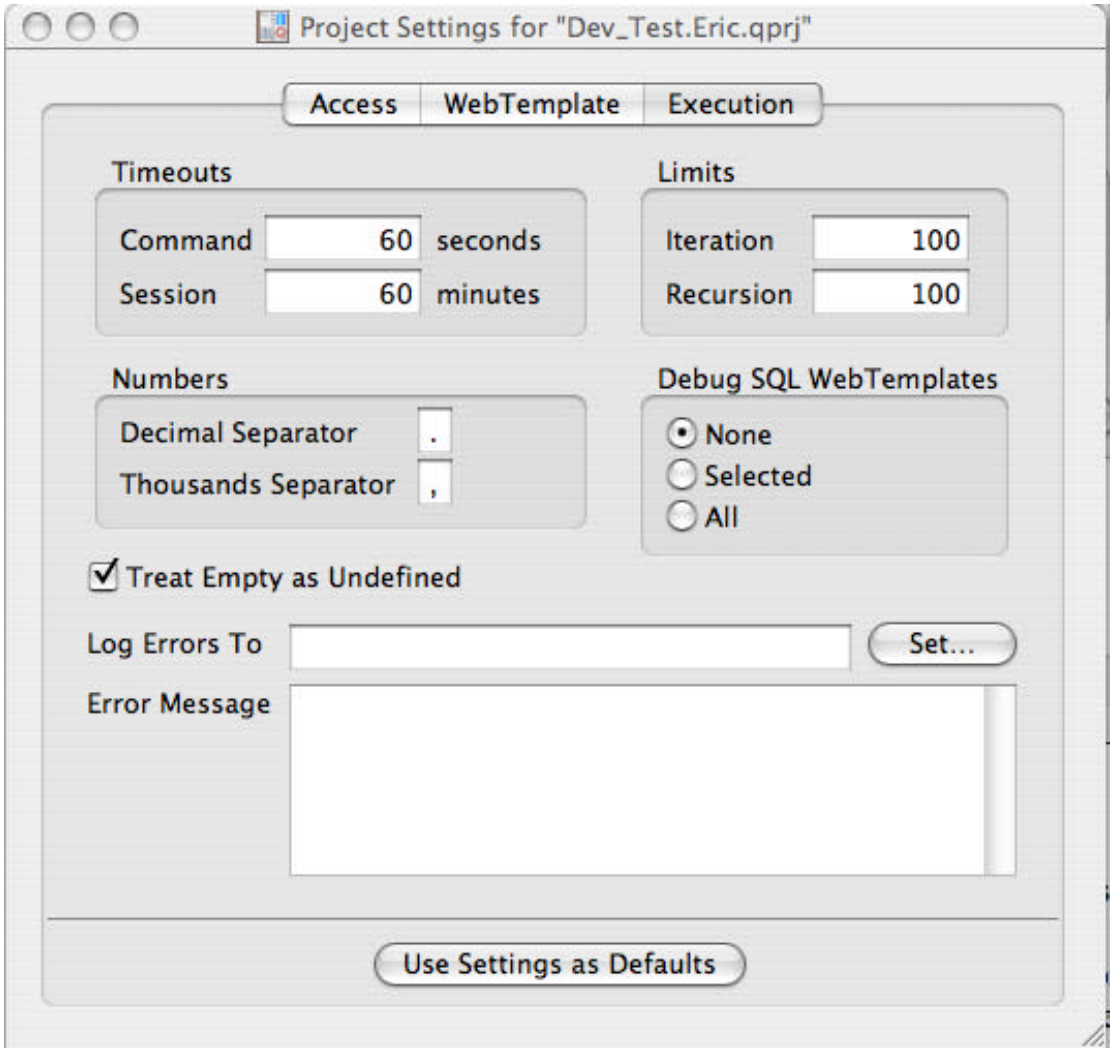
Export All WebTemplates on Save

A WebTemplate is designed within the development environment, but cannot be used unless it is exported. Check this option to automatically export the WebTemplate each time the project is saved. See the WebTemplate documentation for more details.

Use Settings as Defaults

Click this button to use the settings made under this tab when new projects are created.

The **Execution** pane refers to those settings that affect the Qilan engine and how the WebTemplate is processed.



Session Timeout

> This value determines how long Qilan will consider session files valid, starting from the time the webtemplate was last accessed. When a timeout is reached, the content of session files (framework variables) will be marked as 'expired' and therefore no longer be retrievable by the browser client. The default value is 60 minutes.

Command Timeout

This value determines how long Qilan will wait for the script, as identified by the QRUN commandline attribute, to return an output. When this limit is reached (in the absence of any output), qilan.cgi will terminate the requested script, and then continue with the remainder of the WebTemplate. If the value is undefined or 0, qilan.cgi will wait for an output (possibly forever) and is therefore not recommended. The default value is 60 seconds.

The Command Timeout value, as set in Project Settings, will be used only when a timeout is not selected as QRUN attribute.

Iteration Limit

The number of times the 'Q' tag, QWHILE, will be allowed to cycle. If this value is undefined or 0, there will be no limit. When this limit is reached, qilan.cgi will stop the QWHILE execution and report an error to the user (via the browser interface). The default value is 200.

This value is also used as the abacus depth limit. For extremely complex abacus constructions where many abacus expressions are used within others, consider a moderate increase in this value.

Recursion Limit

This value limits the depth of abacus icons that can be invoked. If this value is undefined or 0, there will be no limit. When this limit is reached, qilan.cgi will stop the abacus execution and report an error to the user (via the browser interface). The default value is 100.

Designers of recursive routines should be aware of this setting. Extensive routines may fail if this value is set too low; on the other hand, if set to 0, undefined or an extremely high value, poorly constructed routines may consume processor cycles and appear to freeze the CPU.

Decimal Separator

The character used when numbers are formatted as decimals. Qilan will default to a period (.) if no other character is identified. See the abacus operator, "formatted by" for more details.

Thousands Separator

The character used when numbers exceed 999. Qilan will default to a comma (,) if no other character is identified. See the abacus operator, "formatted by" for more details.

If you are planning on installing Qilan on a machine located in a foreign locale, the decimal and Thousand Separators can be used so that numbers are formatted properly. You can always format numbers individually using the “formatted by” abacus operator, but its much easier just to set them once in the Project Settings.

Debug SQL

This setting is used to log SQL submissions (via qilan.cgi) and messages returned by the database. It may be helpful in tracking down database errors, malformed SQL queries or other anomalies resulting from communication failures. Logging is directed to the Apache Error Log, located in `/var/log/httpd/error_log`.

There are three settings which operate in conjunction with the DEBUG attribute of the QLOGIN tag as follows:

ALL: Every SQL statement will be logged regardless of the QLOGIN DEBUG attribute.

NONE: No SQL statements will be logged regardless of the QLOGIN DEBUG attribute.

SELECTED: SQL statements will be logged only for QLOGIN tags that include the DEBUG attribute.

Logging SQL can create very large files; judicious use is advised.

Treat Empty as Undefined

This setting automatically converts empty values to undefined. An empty value is normally considered defined (but valueless) and is associated with HTML inputs. The conversion performed by checking this preference occurs before abacus evaluations, therefore expressions that test for empty (such as ‘Empty __’) will always return FALSE.

If you are familiar with how databases handle data and expect Qilan to function in a similar fashion, you should enable this preference setting. Data inputs will automatically be interpreted as defined or undefined.

Log Errors

The name and location of the log file used to store error messages output by qilan.cgi. When valid information is entered (existing path and filename), the error output will be written to a log file; otherwise the error output will appear on the client browser.

Valid log file information refers to a path, filename and correct file name permissions. A sample approach is as follows:

• Create a folder named, “QilanLogs”, in `/Library`.
• Highlight the folder, and then choose ‘Get Info’. Set the permissions so that any user can write to this file.
• Open Project Settings and type (or select) the path and name of the log file: `/Library/QilanLogs/QilanErrorLog`.

When an error occurs, qilan.cgi will create a log filed named, “QilanErrorLog”. As this file is created by qilan.cgi, it will automatically have correct access permissions. If desired, you may then remove write permissions from the folder.

Use Settings as Defaults

Click this button to use the settings made under this tab when new projects are created.

Errors written to the log file will be formatted as follows:

Each error will contain two or three lines. The first line starts with a "+", followed by the date, time, IP address, WebTemplated accesed, and ends with the error message.

```
+Jul 16 12:35:06 192.168.1.5 /Library/WebServer/Documents/EnterData
JDBCException The database specified has not been started or can not be found.
Your action has been aborted.
```

The second line starts with a "-", followed by the date, time, and IP address, and ends with details about the error.

```
-Jul 16 12:35:06 192.168.1.5 {\n    attributes = (\n        "BORDER=\"1\"", \n
"TABLE=\"OpenBase/newtable\"", \n        "FIELDS=\"<OpenBase/newtable/number1>
<OpenBase/newtable/text1> <OpenBase/newtable/_rowid>\""\n    ); \n    errLineNo
= 44; \n    tag = QTABLE; \n}
```

The third line is present only if there is some form input or a query string. It starts with a "?", followed by the date, time, and IP address, and ends with the inputs.

```
?Jul 16 12:35:06 192.168.1.5 {NumericalData = <null>; TextData = <null>; }
```

The second and third lines both have all CR characters replaced by the sequence "\n".

You can view the error log remotely using system tools (remote terminal, ftp, etc.), or use Qilan's QRUN with 'cat' or 'tail'.

Error Message

Typically, when qilan.cgi generates an error, a detailed output is shown on the client browser (errors are produced by Qilan, Mac OS X, JDBC or the database). This is the default for development. However, when a project is deployed, an error message may be added to or shown in place of the error output. The error message may contain any character, including HTML codes.

****

General Note Regarding Execution Settings

Changes to execution project settings should immediately be followed by, “Export All HTML”. This will insure your changes are properly written to all WebTemplates.

****

Web Server Configuration

The following section discusses web server configuration; a basic knowledge of the UNIX file system may be helpful. Refer to an easily readable UNIX primer for details.

Qilan is fully compatible with the Apache web server, versions 1.x and 2.x. We strongly suggest however, you use the most recent version of 1.3.x or 2.0.48 or higher. There are slight configuration differences between version 1.x and 2.x for FastCGI. Please refer to the Appendix for details and complete instructions.

The Apache webserver, version 2.x, is a relatively new release. It is included with OS X Server as an optional install, but is also available from versiontracker.com (Faby distribution) as a complete installation package. Which one should you use? I suggest you use the installation package from Apple, as this is included with Mac OS X. The package available from versiontracker.com includes a nice GUI and is frequestly updated. By the way, Apache 1.3.x works great!

The default Apache web server configuration (standard OS X install) is to identify the Documents directory as the ‘root’ folder. Therefore, the Documents directory is set to allow unrestricted access. Any document placed into Documents is retrievable, including exported WebTemplates.

Avoid exporting WebTemplates to the ‘Documents’ directory.

We suggest you create a directory that can only be accessed by qilan.cgi. All other attempts to access an exported WebTemplate directly will be denied.

The technique to be employed is simple, but does require some configuration. We will be creating a directory and then restricting access to non-http requests. This effectively blocks all access except that originating from the server itself.

The Apache web server is controlled by a series of files, the main one being the ‘`apache.conf`’ file. On Mac OS X, this file has been renamed to ‘`httpd.conf`’. It is located in

```
/private/etc/httpd/
```

It is very important you maintain a backup of this file in the event you make a mistake or need to revert back to the original. If you are unsure or uneasy about making changes to the `apache.conf` file, please consult with a specialist.

Most systems contain two configuration files: one used by Apache and a backup. Make sure you edit the correct one. It’s not a good idea to edit the configuration file when the web server is in use.The first step is to create a directory inside Documents. Let’s call this new folder, “Qilan”. Once this is done, change or create the Qilan preference, “HTML File Root Directory”, to:

```
/Library/WebServer/Documents/Qilan
```

The second step is to create a small text ‘access file’ and place it within the Qilan folder. The file’s name is important, “`.htaccess`”. Note the name begins with a period. Its contents are as follows:

```
<Limit GET POST>
order deny,allow
deny from all
</Limit>
```

The file instructs the web server to ‘Limit’ (meaning prevent) access to the files contained within the Qilan folder to all GET and POST requests via http.

The final step is to tell the web server to recognize and use `.htaccess`. This requires changing the `apache.conf` file.

The following segment is taken from apache.conf. Again, if you are uneasy about changing a configuration, consult with a specialist. Note that lines beginning with the pound sign (#) are comments and not executed by Apache.

Accessing the `apache.config (httpd.config)` on Mac OS X can be a bit difficult as the path to the file starts with 'private', which is an invisible folder. Launch TextEdit then select, "Open" from the File menu. Type in the path to the error log (/private/etc/httpd/) then press the enter key.

This section of the configuration file tells Apache what can be overridden for directories located within the root folder.

Locate this line - do not change it.

```
<Directory "/Library/WebServer/Documents">
```

The following lines are present in the apache.conf file, and are shown for reference. Do not change anything here.

```
#
# This may also be "None", "All", or any combination of "Indexes",
# "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
    Options Indexes FollowSymLinks

#
# This controls which options the.htaccess files in directories can
# override. Can also be "All" or any combination of "Options",
# "FileInfo", "AuthConfig", and "Limit"
```

Here's where we want to make our changes. First, the existing line:

```
  AllowOverride None
```

This line instructs the web server to ignore `.htaccess` files. In other words, do not allow anything to override the default settings.

Change the line to:

```
  AllowOverride Limit
```

This line instructs the web server to recognize an htaccess file containing Limit parameters, if the file is located inside the root directory. Note that spelling and case are important.

Close `apache.conf` and save your changes, then restart the web server.

Clients will now access your exported Qilan documents using the URL:

`Your_domain/cgi-bin/qilan.cgi/Qilan/HTML_document_name`

Client attempts to access:

`"Your_domain/Qilan/HTML_document_name",` will be forbidden.

Accessing a Database

An Access window represents a User's entrée to the database. It contains information necessary for login, a representation of database tables and fields, and logical structures necessary for database manipulation.

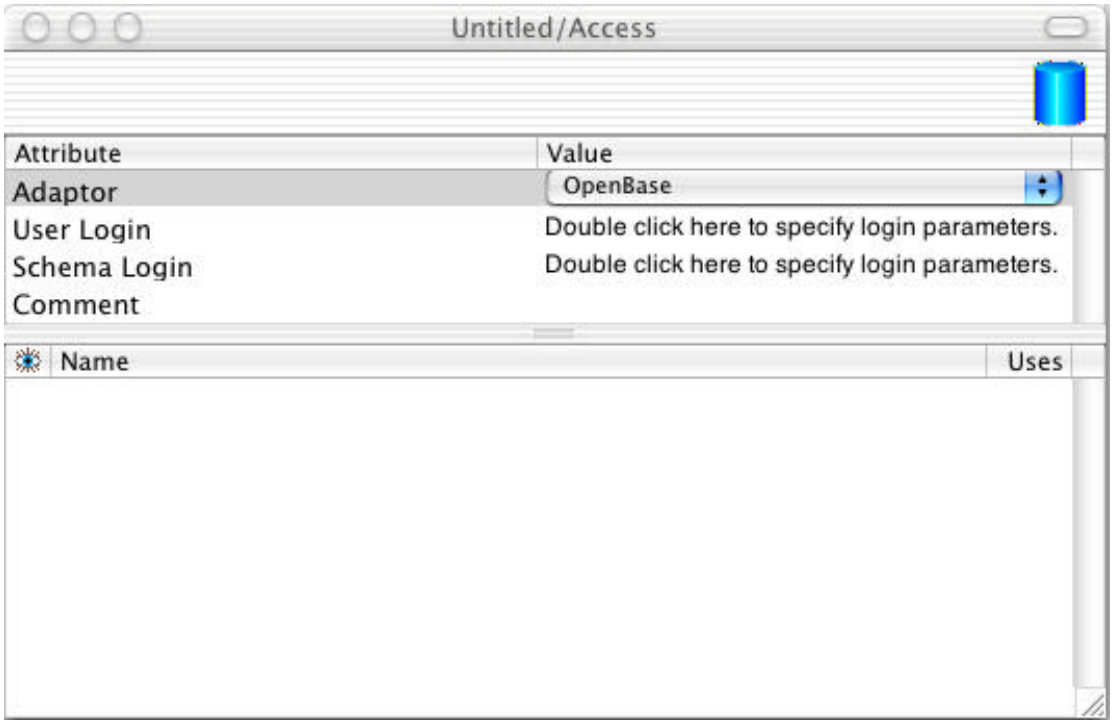
Bring the project window to the front, and then drag an access icon from the palette. Double click to open it. An access contains two parts: the attributes (top) and the schema (bottom).

Qilan employs JDBC (Java DataBase Connectivity) technology to interact with database software. With appropriate adapters installed, Qilan will import/export database schemas, pass SQL commands, retrieve data and obtain information about the database.

Most JDBC adapters supplied by data base manufacturers are written in JAVA and will run without modification on OS X. Qilan will recognize them when they are placed in /Library/Java/Extensions.

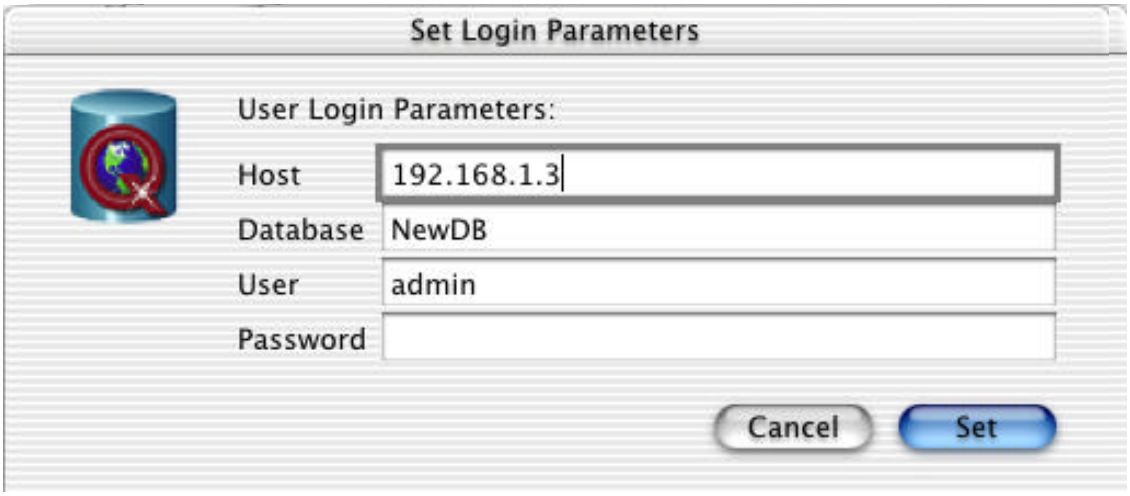
Refer to the web site: http://java.sun.com/products/jdbc/ for more information about JDBC and how to obtain adapters.

Login attributes allow the user to select a database and configure the login parameters for the user and administrator (schema user).

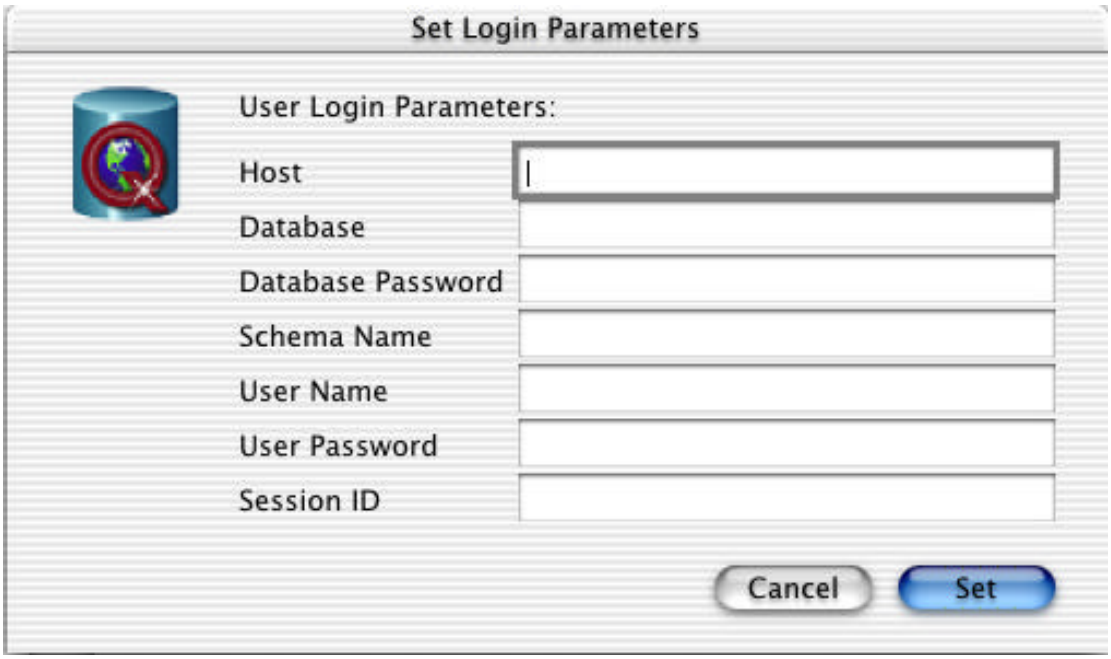


Choose a database from the pop-up menu, and then double click on the line, “User Login” or “Schema Login” to open the database’s login panel.

A basic login panel (used by OpenBase):



A more complex login panel (used by Front Base):



Databases, as defined by their JDBC drivers, may enable various options. When options are available, Qilan will change the appearance of the Login panel to support them.

Before you login, insure the database is running and you are permitted to access it. Attempts to login to a non-existent database or one, which is not running, will result in an error message.

For some databases, which use proprietary drivers, the JDBC database driver is not automatically installed by Qilan and must be obtained separately from the manufacturer. Attempting to access a database without the requisite driver installed will result in an error message.

Complete the login information (note that names may be case sensitive depending upon the database), including the name of the database, its location (host name), user name and password. Then click, “Set”. This will complete the connection information and return you to the Access window.

The password entry will be echoed in bullets; insure the password is typed correctly. The nature of this security provision is to prevent the casual observer from viewing the password as typed by the developer.

The access window will then display the name of the database, host name and user/schema name. The password will not be shown. If you need to edit this information or re-open the Login window, just double click on the User or Schema login line.

Access User

The access defines two types of users: Schema and Login. The Schema user, used by Qilan, imports/exports database schema (tables and fields). The Login user is used by Qilan to perform data entry, retrieval, updates and deletion.

Schema User

This user, typically defined by the database as the 'administrator', has access to all tables and permissions. The schema user imports and/or exports database structure. It is necessary to define (or redefine) the schema user and password prior to using Qilan's import/export schema functions. This is performed using the database manager tools.

We strongly suggest you create at least one other user in addition to the Schema User. Avoid defining the Schema User as the Login User as this may create a security risk.

Login User

An access defines a single login user. This user must have permission to perform data operations called for by Qilan. For example, if a QDELETE is used on a web template, the user (as referenced by QLOGIN) must have permission to delete records; otherwise the delete will fail. User permissions are set in the database.

Qilan provides several methods to define user names and/or passwords:

Create multiple accesses, each with its own username and/or password. This creates a static username and/or password linked to a specific access.

Ask for a username and/or password when a client submits an HTML request. This latter approach can be used with QLOGIN attributes. See WebTemplate documentation for details.

Dynamically generate a username and/or password drawn from Qilan data (user data typed into an abacus) or from other databases.

Although a login username/password is defined within the access icon, it is considered the default. It is only used when no other username/password is submitted via the QLOGIN.

Importing the Database Schema

This function automatically builds a parallel representation of the database structure within Qilan. Database tables and fields will be imported. The table structure will appear at the bottom of the access window. Double clicking on a table icon will open its structure so that fields and relationships can be viewed.

To import a database schema bring the Access window to the front. Complete the Schema Login information and insure the database is running and available, then choose File > Import Schema. This function works for JDBC compliant databases only. If a schema already exists and you wish to update it based on a modified database, just choose File > Import Schema. The schema will update showing newly created database objects.

Note: Prior to updating a database schema, be careful not to delete database tables or fields that are being used by Qilan. If this becomes necessary, first remove all Qilan references to the database object(s) then update the schema. Failure to do this will result in errors.

Is it necessary to import a database schema to access tables and fields in the database? The answer is no. All that is necessary is that names and types (for fields) match. This little tidbit is useful when you are accessing corporate database, those where the system administrator does not want to give you schema access or databases that do not support importing the schema.

Some databases, notably FileMaker, create ‘pseudo’ fields. These are internal objects that are used by the database but are not treated as ‘real’ fields. Qilan can access these objects with parallel fields of the same name and type.

Exporting the Database Schema

This function automatically builds a parallel representation of the Qilan Access structure in the selected database. Access tables and fields will be exported.

To export a Qilan Access schema, bring the Access window to the front. Complete the Schema Login information and insure the target database is running and available, then choose File > Export Schema.

> Exporting a database schema works for most JDBC compliant databases. If a specific database does not support exporting schema, the Export Schema menu item will be disabled.

The export function compares database schemas. If you add a table or field to Qilan's schema, a corresponding object will be added to the database. Modified objects are updated. If you delete an object in Qilan, the corresponding data base object will be removed.

If an existing database table or field is deleted from the Qilan schema, data contained within the database table (fields) or the individual field will also be removed. Modifying field formats may also unexpectedly change or truncate existing data.

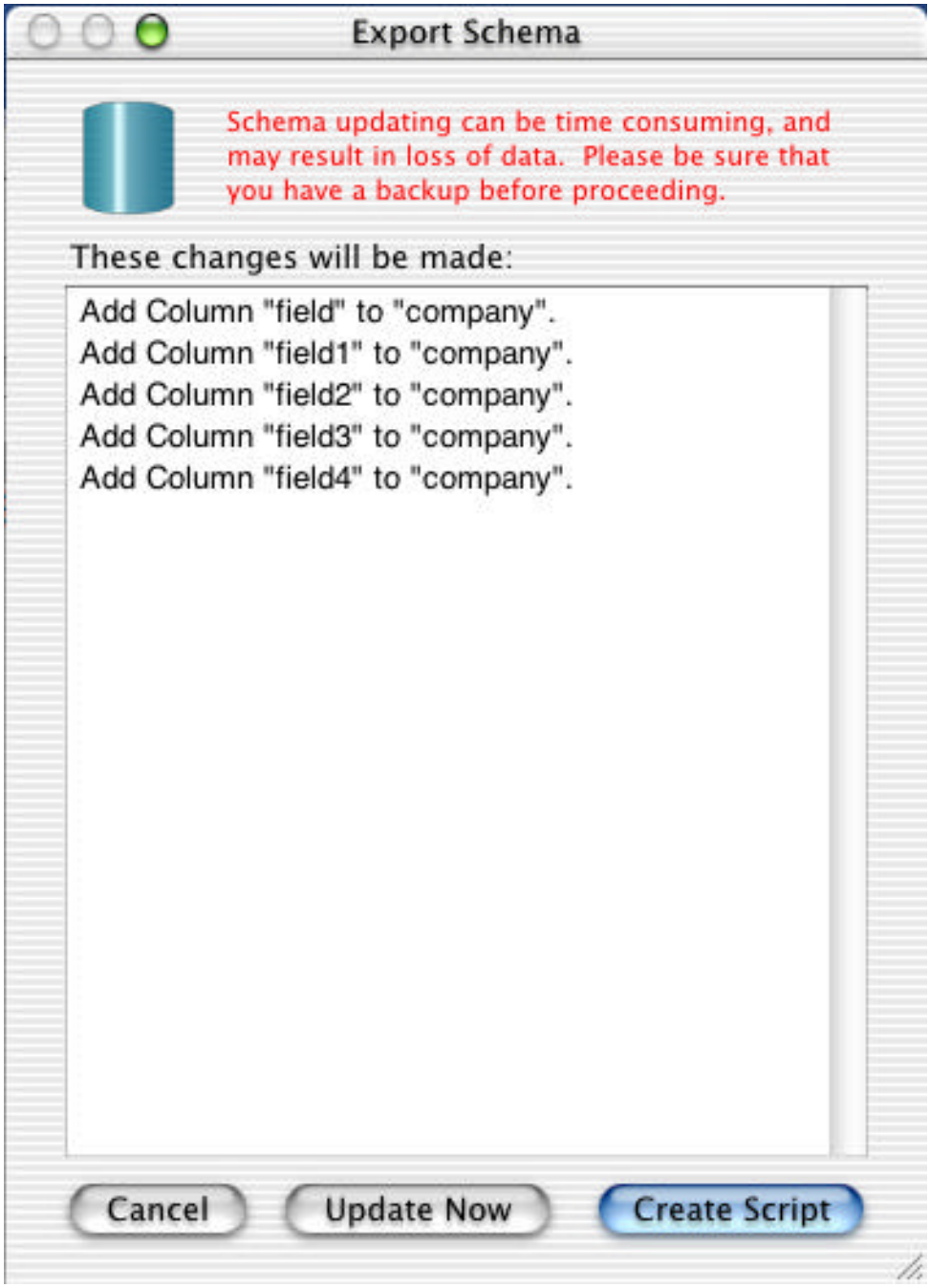
Be wise: back up your data prior to Export!

After choosing File > Export Schema, Qilan will present the "Export Schema" window. This window will display the Qilan schema. There are three options: Cancel, Update Now and Create Script.

Clicking Cancel will close the Export Schema window. Export Schema will be aborted.

Clicking Update Now will immediately begin the export process. It is suggested this be performed when there is low database activity. Updating may cause the database to build indexes or other structures causing temporary access interruptions.

Export Schema Dialog:



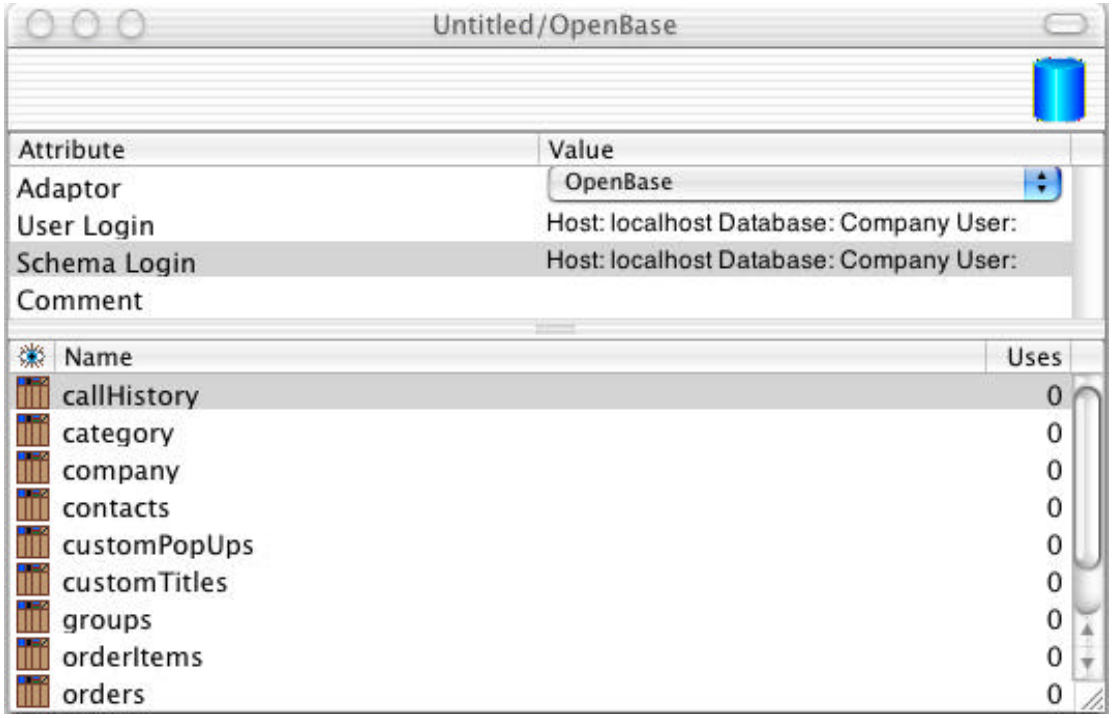
Clicking Create Script will write all the database changes to a text file. This file may be opened and edited (if necessary) or applied by the database manager at a later time.

Qilan will not export tables or fields whose external name is undefined. Insure an external name is defined for all database objects you are exporting.

Some databases automatically create internal fields whenever a table is created. For example, OpenBase creates "_rowid", as an internal unique identifier for each table. To insure these fields are not altered accidentally and to make them available in the Access, Qilan immediately performs an Import Schema following each Export Schema.

Tables

The lower portion of the access window displays the database table structure. If Import Schema is successful, the tables will appear automatically. You can manually create tables by choosing Icon > New Table. Only the tables you want to access are required. This permits you to exclude certain tables that may contain sensitive information. If a table is not present, Qilan cannot access its data.



Refer to the documentation on Qilan Icons for more information as to how to manually create fields and abaci within tables.

When a Table is created within Qilan, an External Name is required. This name will be used by the database to identify the Table. Table names within a database must be defined and unique.

After typing the External Name, press the tab key. If you immediately close the window after typing the External Name Qilan may not save your changes and import/export errors may result.

Data Base Compatibility

For users of FrontBase, please use version 2.22 or higher for OS X. Older versions may result in SQL errors.

Download from: www.fontbase.com

For users of OpenBase, please use version 7.0 or higher for OS X.

Download from: www.openbase.com

For users of FileMaker, please use version 5.5 or higher for OS X.

Download from: www.filemaker.com

For users of Helix, please use version 4.5.5 or higher. While older versions that support AppleEvents may work, we recommend downloading the most current version.

Download from: www.helixtech.com

Please visit www.qilan.com for additional information concerning newly added databases.

Fields

Fields represent a carrying mechanism for data. That is, if a user enters a piece of data (such as a name), it is placed into an object called a field. By referencing the field, the data itself becomes accessible.

Framework fields are transient, the data represented by them is maintained during qilan.cgi processing or with server session files; table fields, on the other hand, represent persistent or stored, data.

Table fields refer to the storage of data in the database. The database defines how data is actually stored. For example, if a user enters a number with no decimal points, but the storage defines four decimal points, the data will be stored with four decimal points. If this number is later retrieved from the database, it will contain four decimal points.

The manner, in which data is stored, character, number, date, etc., is known as the data type. Data types are important because they guard the *semantic integrity* of the data; that is, to ensure that it reflects reality in a sensible way. The integrity of the data is at risk if you can substitute a name for a telephone number or if you can enter a fraction where only integers are allowed.

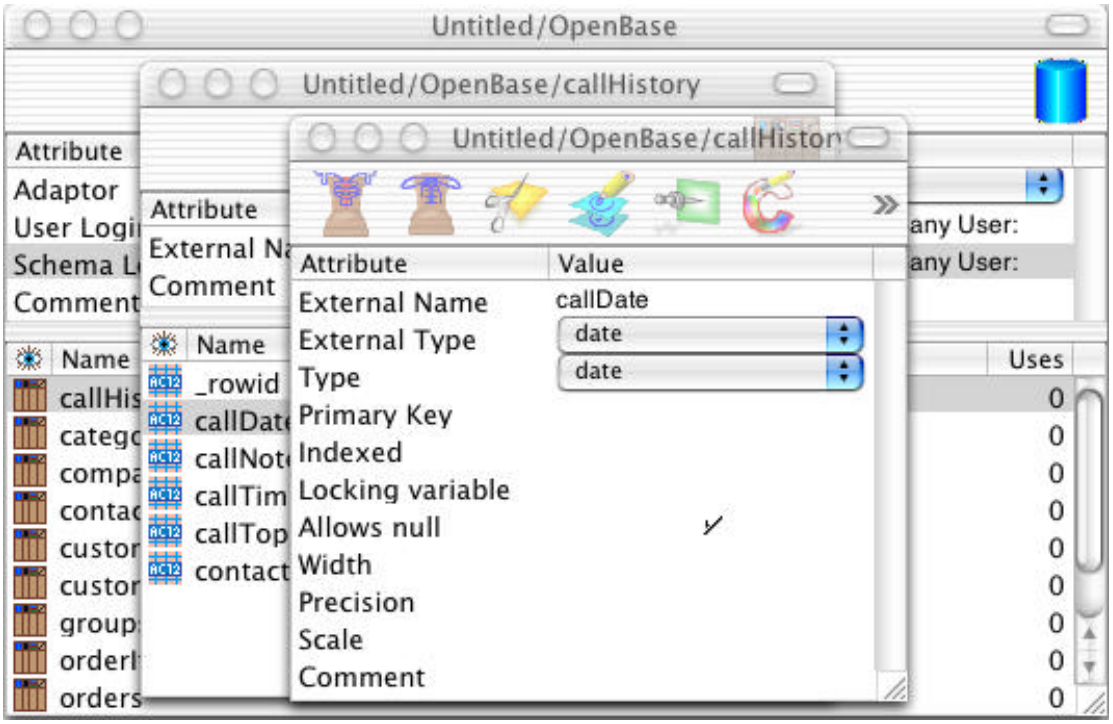
Qilan will coerce formatted values retrieved from the database to more generic forms. This is done so that data can be more easily manipulated or displayed. The data itself is not actually changed nor is the storage format as defined by the database.

Your choice of storage format will usually be based upon two criteria: interpretation and space allocation.

> Interpretation refers to the meaning of the data. Examples include literal character strings, numbers and dates. Databases apply different methods to data formats. Mathematical operations can be performed on numbers, but not character strings.
>
> Space allocation is performed automatically by the database, based upon a field's format. A database application is most efficient and transferable when data sizes are fixed and minimized.

Creating a New Table Field

With the Table window open (Project > Access > Table) drag a Field from the palette into the Table window. Double click the Field icon to open its window.



External Name

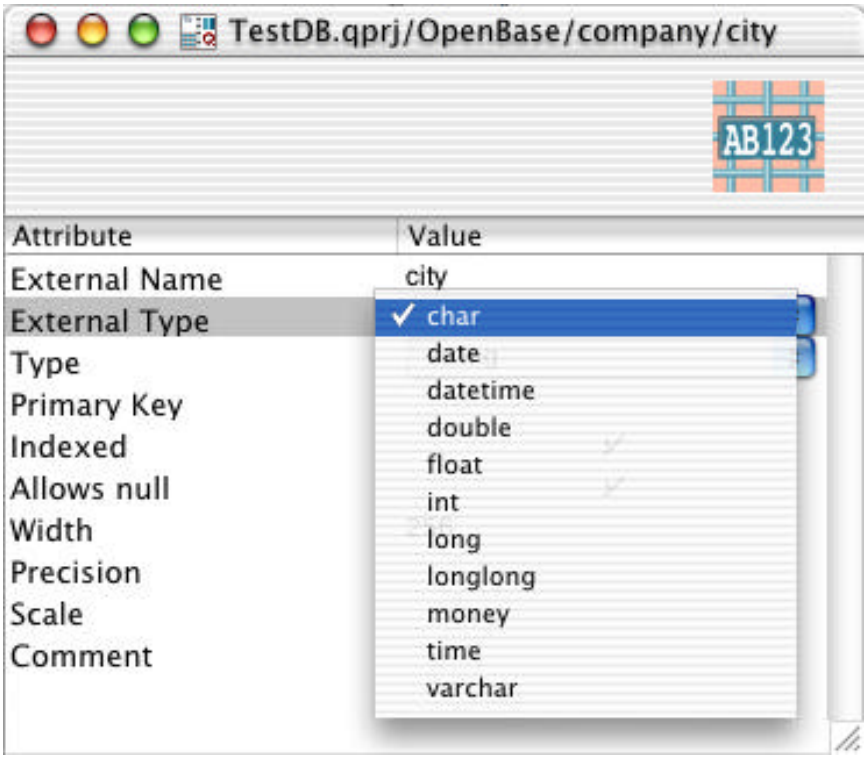
This is the name of the field as known by the database. If you import the schema, the database will complete this entry. If you are creating a new field, check with your database documentation for naming conventions or other restrictions on field names.

External Names for Fields must be defined and unique.

After typing the External Name, press the tab key. If you immediately close the window after typing the External Name Qilan may not save your changes and import/export errors may result.

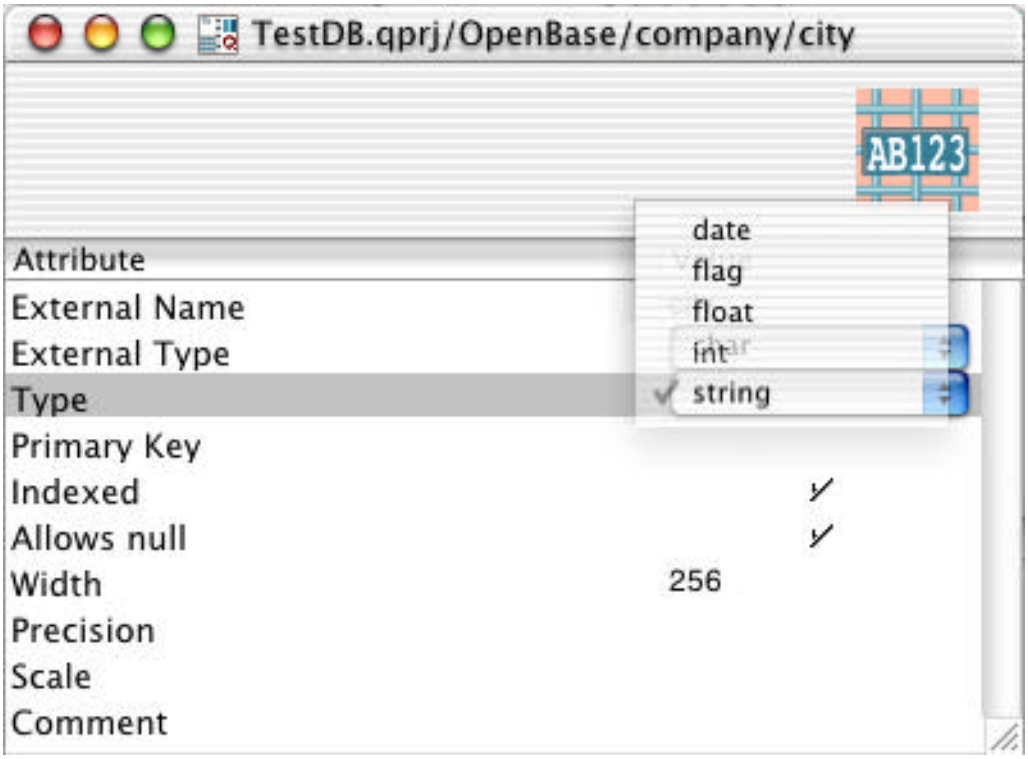
External Type

The field's format as known by the database. If you import the schema, the database will select the proper type. If you are creating a new field, select the type that most closely matches your storage requirements. Note that some databases may have varied options as well as refer to the same format with different wording. A reference tutorial, concerning field formats, is provided in the Appendix.



Type

Type refers to how Qilan will attempt to coerce the format as specified by the database. If you import the schema, Qilan will select the proper type. If you are creating a new field, select the type that most closely matches the type specified by the database. Refer to the Appendix for a coercion table.



Primary Key

A field is considered a primary key when it is unique for each and every record in a table. Databases typically index primary keys automatically. Double click on the line to add/remove a check. When a checked, the field will be a primary key; when unchecked, it will not be a primary key. If you check more than one field in a table as the Primary Key, the concatenated values of all the selected fields will be consider the Primary Key.

Some databases do not support more than one field as being identified as the Primary Key. Consult your database documentation.

One question that inevitably comes up is how to create primary keys. The best way to do this is not to create them at all, but let the database do it for you. OpenBase and FileMaker do this automatically; FrontBase does this by defaulting an integer field to UNIQUE (although this must be done with the Manager.app). Can Qilan perform this function? Actually, yes! A session ID is unique for the machine running Qilan and can be used for primary keys. They will always be character types and may be as long as 30+ characters.

Allows Null

This instructs the database to accept an undefined or empty field value. If disallowed, the database will return an SQL error whenever an undefined or empty value is passed and a record is created or updated. Double click on the line to allow/disallow nulls. When a checked, nulls will be permitted; when unchecked, nulls will be disallowed.

It is suggested that you use Qilan tools for evaluating undefined or null values and set the database to allow nulls. Informative error messages or other data operations can be created in Qilan, whereas the database may only return cryptic error messages.

Width

Double click on the line to enter a numeric value. Width refers to maximum length of a character string. Please consult your database documentation for maximum supported widths.

If you leave this blank for character data types, the database may report an error or set the field width to zero (0) or one (1). Be advised that changes to a field width, after one or more records is created, may result in the erasure of all data in the field.

Object type fields are treated as strings, however a width need not be entered as they typically support unlimited widths.

Precision

Double click on the line to enter a numeric value. Precision, used with numerical types, defines the number of digits to be stored. The exact definition of this setting is dependent upon which database and External Type are chosen. Please refer to the reference documentation.

Scale

Double click on the line to enter a numeric value. Scale refers to the fractional element of the data type. It is used with numerical and date/time data types. The exact definition of this setting is dependent upon which database and External Type are chosen. Please refer to the reference documentation.

Locking Variable

Double click the line to designate this field as a ‘locking variable’. When selected, a check mark will appear. Locking variables are used to create the record snapshot for optimistic record locking (see Record Locking for more information).

Indexed

Qilan will request an index be built on the selected field when the schema is exported. Depending upon number of records in the table, building an index may take time and temporarily limit data base access. When checked, the field will be indexed. If unchecked, the index will be dropped. There is no need to index fields identified as Primary Keys; databases automatically index these fields.

When should a field be indexed? In general, fields identified as relationship 'targets' should be indexed. Fields containing defined and varied data benefit from indexing especially when they are used for searching or sorting. Be careful though, excessive indexing may actually slow data base operations and/or substantially increase the size of the database itself. Consult your database documentation for additional information.

Qilan will automatically build indexes on fields identified as relationship 'targets'. This will automatically optimize database performance. If you drop an index used as a relationship target, Qilan will re-build the index when the schema is exported.

Indexes automatically built by Qilan, as the result of a relationship target, will not be checked as 'Indexed' in the Access > Table > Field window. The index will be built in the database.

Comment

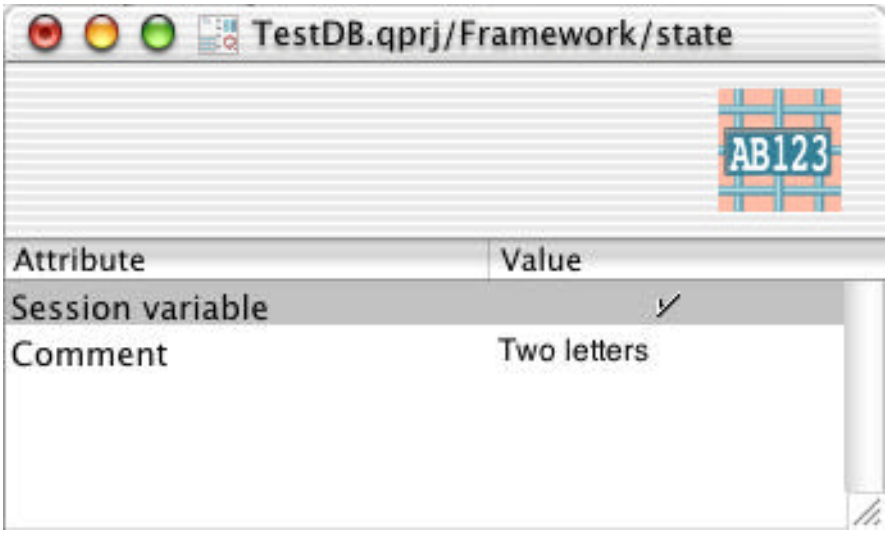
User entered text.

Framework Fields

Framework fields are transient, global values and, if designated, session values. They may be used for temporary storage, processing, display, calculations and maintaining session data.

To create a field, drag a field icon from the palette into framework window. Name the field by highlighting its default name and retyping. This name will be used whenever the field is referenced from within Qilan.

A Framework field that is set from the browser (either via Form data entry or from the "QUERY_STRING") is always of type string. A framework field that is set by a QASSIGN tag takes on the type of the value of the field or abacus assigned to it.



Session Variable: Double click on the line to maintain the value of this field throughout the session. A checkbox indicates the field value will be kept.

Session variables are stored by name. This can be very useful if you want to pass session values from framework to framework or project to project. All you have to do is name the fields the same. Remember that field names are case sensitive.

Comments: User text that can be added to any field by double clicking the field window.

Framework Icon Limits

Each Framework should be limited to 256 fields. This is not an absolute maximum, but pop-ups that allow for the selection of fields may only list the first 256. There is no limit as to the number of icons a Framework may contain.

Relationships

Relationships (joins) define a state of data equality. Databases store information in two-dimensional tables consisting of rows and columns. Tables contain rows of data consisting of columns of simple data types such as strings of characters, numeric data types, dates, etc. Databases usually consist of more than one table. Tables are related to each other through *keys*, columns that define uniqueness and references between tables. Access to data that spans more than one table is called a *join*. Tables within the same Access icon or from a Framework to any table can be joined.

Using the Relationship table1.link = table2.link, a record in table1 would reference one or more records in table2.

A source link may be singular element (field), such as an ID# = ID#, or compounded using multiple fields or abacus constructions.

When a link is compounded, all links (taken together) form the relationship. For example:

Framework -> to -> DBTable

Source	Target
ID#	ID#

This relationship is defined as: ID# = ID#.

If we add a new link:

Source	Target
ID#	ID#
LastName	LastName

the relationship is then defined as:

[IDValue = IDValue] AND [LastName = LastName]

A source value may be a field or derived value; target values are always table fields. Derived values can be the result of an abacus expression or framework input values.

Relationship Types

When a link refers to one, and only one, identical match in a target table, the relationship is said to be one-to-one.

When a link refers to one or more, identical matches in a target table, the relationship is said to be one-to-many.

> Relationships used as links for DataFlows are one-to-many;
>
> Relationships used by abacus expressions, where the abacus expression is built in an Access > Table, are one-to-many.
>
> Relationships used by abacus expressions, where the abacus expression is built in a Framework, are one-to-one. If the relationship finds more than one matching value, the first value (where, link = link) will be returned.

Relationship Semantics

Table linking, in addition to types, has two other characteristics: source/target and inner/outer.

The source table is that which establishes the link and the target, the respondent. If we were to write our logic on paper, we would begin with the source, then refer to the target. Therefore, the source is considered to be on the ‘left’ and the target, on the ‘right’.

When a link is composed of only existing values between two tables, it is referred to as being within or inside the table. This type of linkage is the most restrictive as a relationship must exist in both tables for a link to be considered valid.

When a link may not be present (in either table), it is referred to as being outside the table. This type of linkage is the most common. For example, values from the source table will be returned even though there are no matching values in the target table (left outer join).

We therefore have the following relationship semantics:

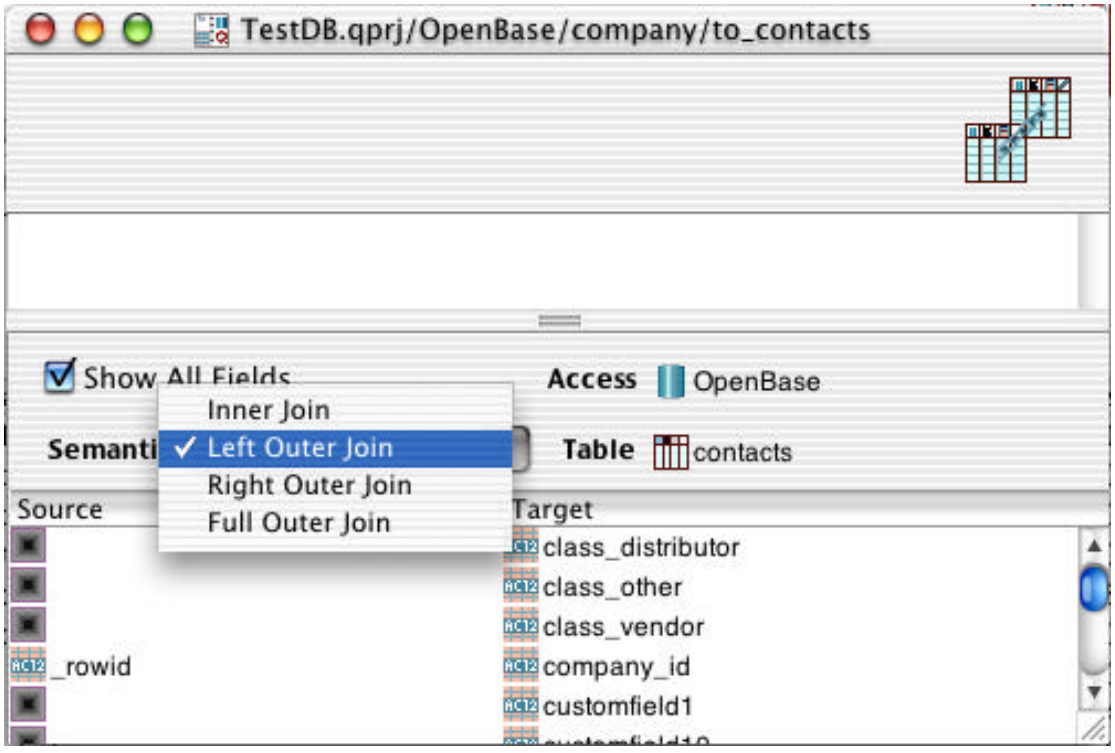
The **left outer join** selects rows from the source table, then matches values in the target table. A source row will be returned regardless of a matching value in the target table.

The **right outer join** selects rows from the target table, then matches values in the source table. A target row will be returned regardless of a matching value in the source table.

The **full outer join** combines a left outer join and a right outer join.

The **inner join** selects rows from the source table only when there is a matching value in the target table.

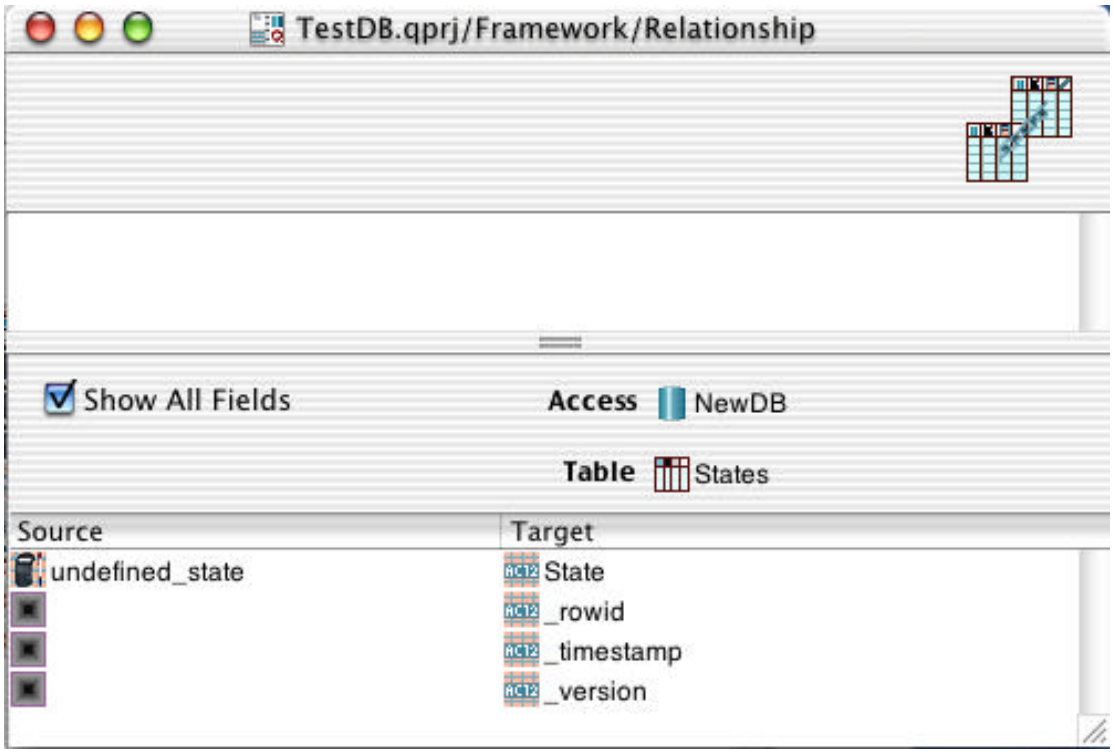
The Access Relationship Window



Access > Table relationships relate data between tables within the same database.

Relationships built in the Framework default to the semantic, ‘Left Outer Join’ method.

The Framework Relationship Window



Unlike relationships built in an Access > Table, Framework relationships can relate data between Framework values and a database table, tables within the same database or between tables in different databases.

Building Relationships

Relationships are very useful objects. They can be used to retrieve, create or update information based upon a link. Access Relationships link and/or locate information and can be used in conjunction with abacus expressions. When built in the Framework, Relationships can be used to locate data, or more importantly, update existing data or create new records.

The Access Relationship

To create a new Relationship, ensure the Table window is open and active (Project > Access > Table). Drag a relationship from the palette. Double click to open its window.

Relationship Settings

Relationship Semantic: Select the method of linkage between the source and target tables.

Relationship Access/Table: With the relationship window open, click the ‘Table’ tab from the palette. A list of target tables from the Access containing the current relationship will be shown. Drag a table from the list to the table ‘hole’. The Access will be defined automatically. Once this selection is made, fields from the target table will be shown.

Show All Fields (Checkbox): Initially checked, shows all target fields. When unchecked, suppresses target fields not used as part of the link.

Source/Target Objects: After a table is selected, the target column will list all fields in the target table. With the Relationship window open, click on the ‘Values’ tag from the palette. Fields/Abaci will be displayed from the table containing the relationship (source). Drag any field or abacus to the ‘hole’ in the source column to create a link.

Abacus constructions used for source values, when containing constants, will automatically use the semantic, ‘inner join’, regardless of the pop-up setting. A ‘constant’ is considered any typed or acquired value.

The target table must be in the same Access as the source table.

Comments: In the space at the top of the relationship window, type any user text. Text will be stored with the relationship object.

Relationships with defined source values ‘lock’ the target table. If you need to change the table, you must de-select all the source values. A target table will also be considered ‘locked’ when a DataFlow is using the relationship object.

OK, timeout! Relationships are useful, but how? This is indeed the $64,000 question. Relationships make data retrieval easier and reporting very sophisticated. Here are a couple of examples:

Let's assume you are storing inventory records in an inventory table. Each record has an ID reference to a part number. Now, you want to display inventory. How do you show the part's name if only the part number is in the inventory table? The answer is easy: use a relationship that links part number in the inventory table to the part number in the part's table. Then, use an abacus and get the part name based on this relationship. When you retrieve (QFIND) data from the part's table, select the abacus that gets the part's name. This is really very fast and works great.

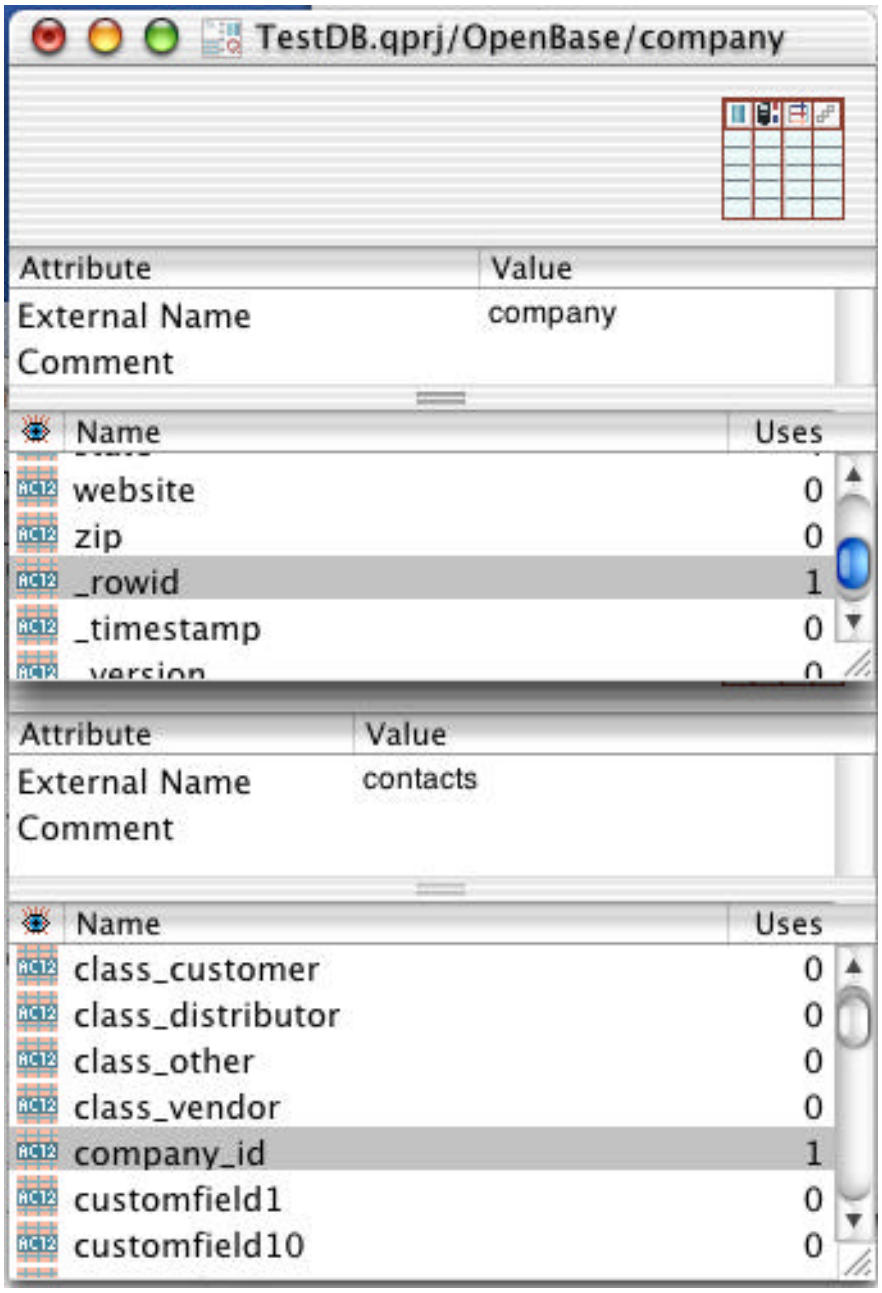
The second example will make you look good. Using a relationship semantic, let's find out how many customers really have orders and show them to boot. Assume that you have a table of customers and a table of orders. Create a relationship in the customer's table linking their customer number to a matching value in the order's table. Build an abacus in the customer table based on the relationship that gets order dates. Now, retrieve (QFIND) the list of customers along with the abacus that returns the order dates. If you use the 'left outer' semantic, all customers will be shown, regardless of the presence of orders. Change the semantic to 'inner' and watch what happens. Now only the customers who have orders will be shown.

Using relationships can save you time and produce reports without building complex queries.

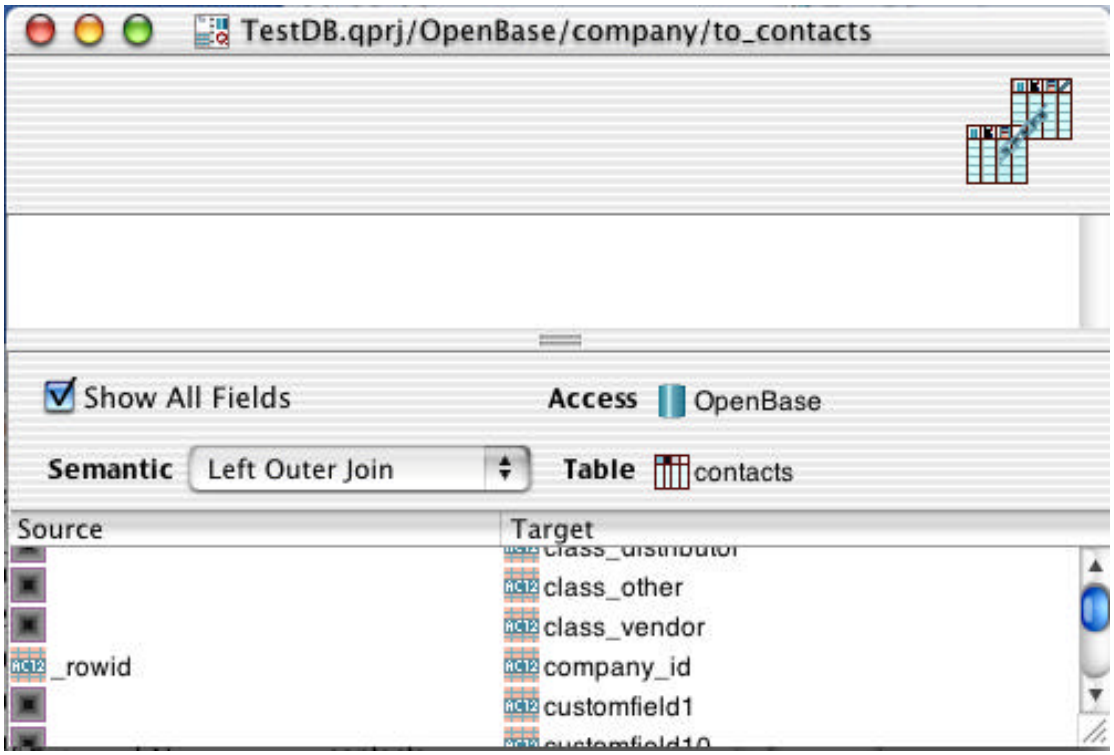
Using a Relationship in an Abacus Expression

Access > Table > Abaci (as well as Framework > Abaci) can use a relationship to locate information. Although Qilan can use the QFIND to retrieve data from a specific table, it is often easier; although not as efficient, to use a relationship built in Access > Table or Framework.

The following screen prints demonstrates how to create a join between two Access > Tables and then retrieve the result. We have two tables, “Company” and “Contacts”. For each company, there may be one or more contacts. In the Company table, the primary key is ‘_rowid’. In the Contact table, the foreign key is ‘company_id’. This is how we refer to a primary key in another table.

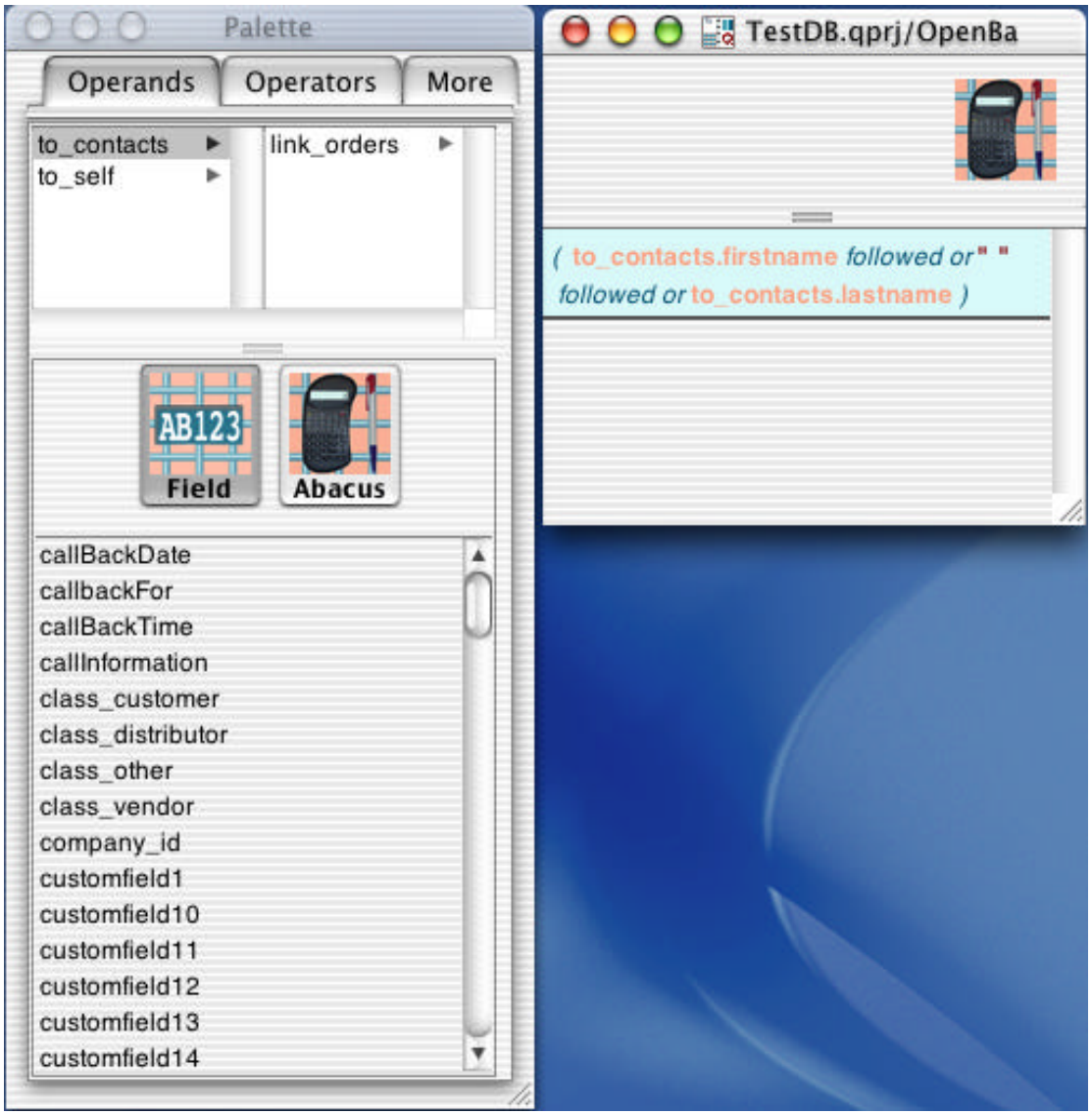


Shown above are the two Access > Tables we are ‘joining’. The relationship will be built in the Company Table because we want to list companies uniquely and show related information.

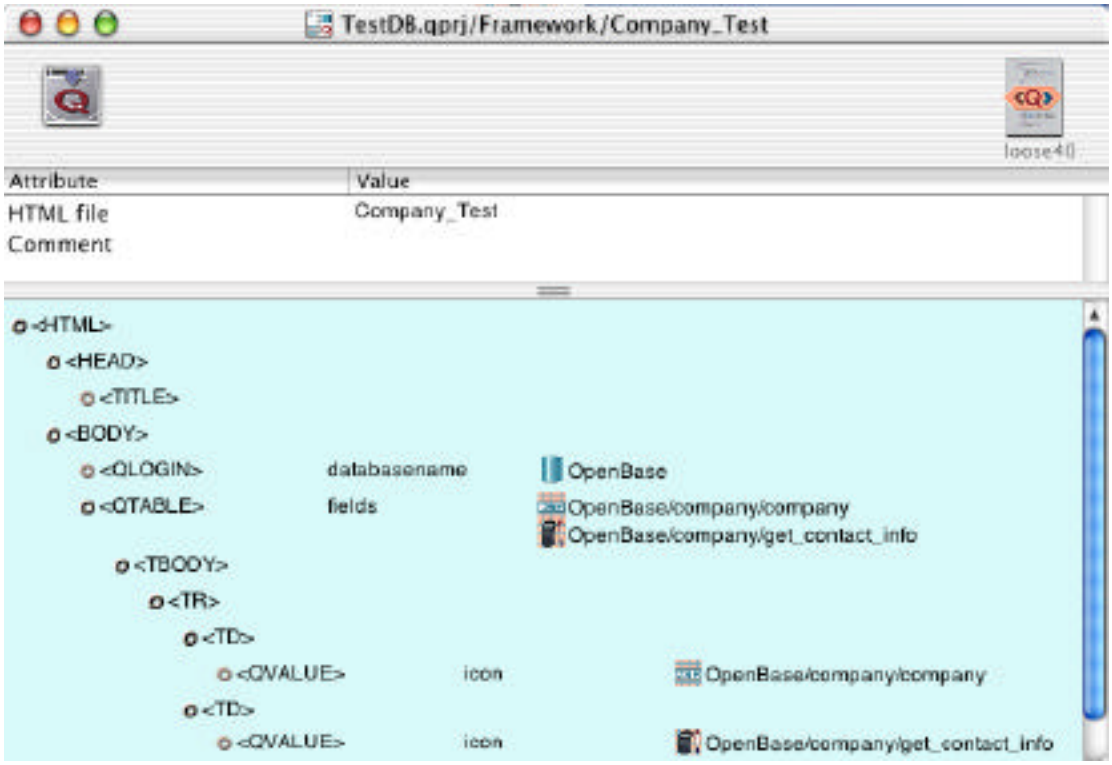


The relationship designates the Contacts table. After opening the relationship, the palette will display fields and abacus expressions from the Company table. We drag the field _rowid (as the source) to link the primary key to the foreign key. This establishes a state of equality.

The semantic default is ‘left outer join’. This will give us a complete list of companies, regardless of whether there is a link established.



Again, in the Company table, drag an abacus from the palette and double click to open it. Click on the 'Operands' palette tab. The relationship to the Contacts table will be shown. Highlight the name. The lower portion of the palette window will display all fields and abacus expressions in the Contacts Table. Drag the desired value to the abacus window. As shown, the first and last names are selected. Observe the syntax: [table.name]. This abacus can now be used as data, as if originating, from the Company table.



Finally, we display the company name along with the contact name using the QTABLE tag. The retrieval is from the Company table:

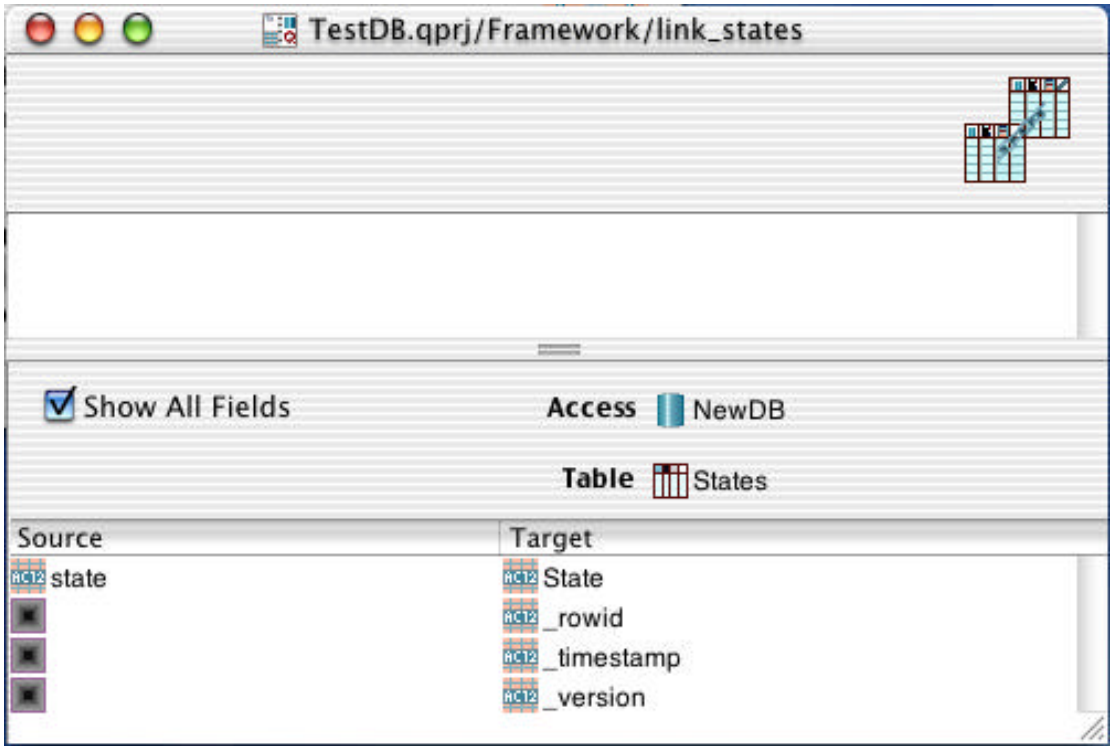
OpenBase/company/company
OpenBase/company/get_contact_info

We have not discussed the QTABLE tag yet, but it is essentially the same as a FIND, with integrated HTML table formatting.

The completed expression, including the abacus and relationship is shown below. The palette window reflects that of the relationship.

The Framework Relationship

To create a new Relationship, ensure the Framework window is open and active (Project > Framework). Drag a relationship from the palette. Double click to open its window. A window similar to the one shown below will open.



Relationship Settings

Relationship Access/Table: With the relationship window open, click the 'Table' tab from the palette. A list of target tables from a selected Access will be shown. Drag a table from the list to the table 'hole'. The Access will be defined automatically. Once this selection is made, fields from the target table will be shown.

Show All Fields (Checkbox): Initially checked, shows all target fields. When unchecked, suppresses target fields not used as part of the link.

Source/Target Objects: Upon the table selection, the target column will list all fields in the target table. With the Relationship window open, click on the 'Values' tag from the palette. Fields/Abaci will be displayed from the Framework containing the relationship (source). Drag any field or abacus to the 'hole' in the source column to create a link(s).

Comments: In the space at the top of the relationship window, type any user text. Text will be stored with the relationship object.

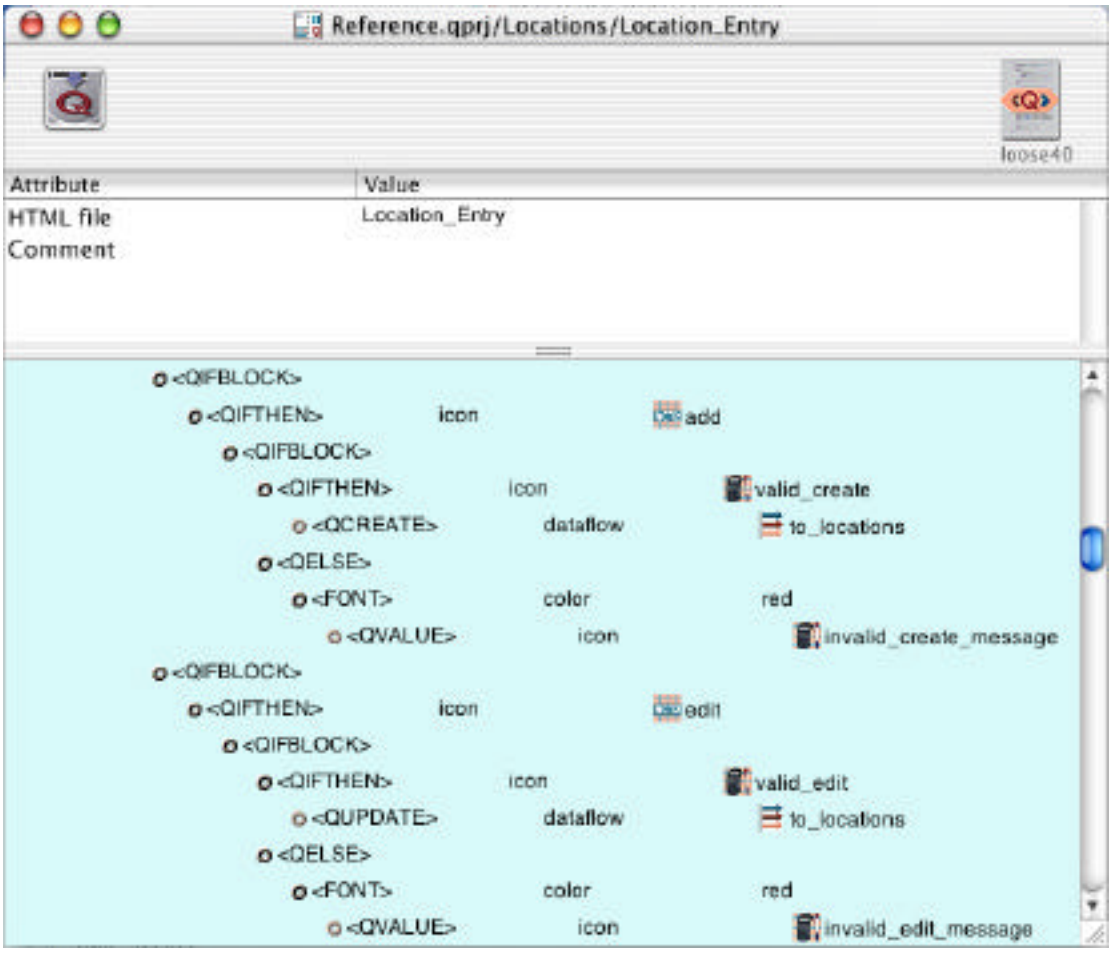
Unlike relationships in the Access, Framework Relationships can also specify links used for the creation and updating of existing database records. When used in this fashion, Relationships identify which record(s) are to modified; or in the case of creation, which Access > Table.

Refer to the DataFlow and WebTemplate documentation for specifics and additional information.

The screen prints on the following pages demonstrate how Relationships, DataFlows and 'Q' tags are used to create and update database records.

This screen print shows the relationship between the Relationship (top) and DataFlow (bottom) in the Framework window. Note how the Relationship establishes the link and the DataFlow, using the defined link, specifies how the data will be moved to individual fields.

The WebTemplate, as shown below, identifies and triggers the DataFlow and completes the action with either the QUPDATE or QCREATE tag.



A relationship used by a QUPDATE is considered VALID when a Table is defined **and** the source field(s) and/or abaci used as links are defined and non-empty.

A relationship used by a QCREATE is considered VALID when a Table is defined. Source fields and/or abaci may be defined, undefined, empty or missing.

Note how in the WebTemplate fragment shown above, QIFBLOCKs are used to test the input data and then trigger the proper tag (create/update).

DataFlow

DataFlows define the specific data elements (fields or abaci) that are to be inserted into database fields. DataFlows are always based upon relationships.

A DataFlow uses a relationship to say, “Move the data in these Framework icons into these database fields”, where the definition of the database, link and specific table come from the relationship icon.

DataFlows are triggered exclusively by the ‘Q’ tags, QUPDATE and QCREATE. QUPDATE will uses a Dataflow to update or create an existing record(s) (replace field values); QCREATE creates new record(s).

DataFlows created in an Access > Table can be used to move data from one table to another.

This is a very advanced feature and requires a bit of explanation. The typical use of a DataFlow is to move framework data into a database triggered by either the QUPDATE or QCREATE on a WebTemplate. When you select a DataFlow from a table, you must do so within the scope of a QFIND. Here’s a simple example:

```
QFIND       fields          table.rowid
      QUPDATE               table.dataflow
```

The QFIND locates one or more records from ‘table’. For each record found, the QUPDATE triggers the table.dataflow. Using a DataFlow in this manner makes is very easy to create or update in one table from a source table.

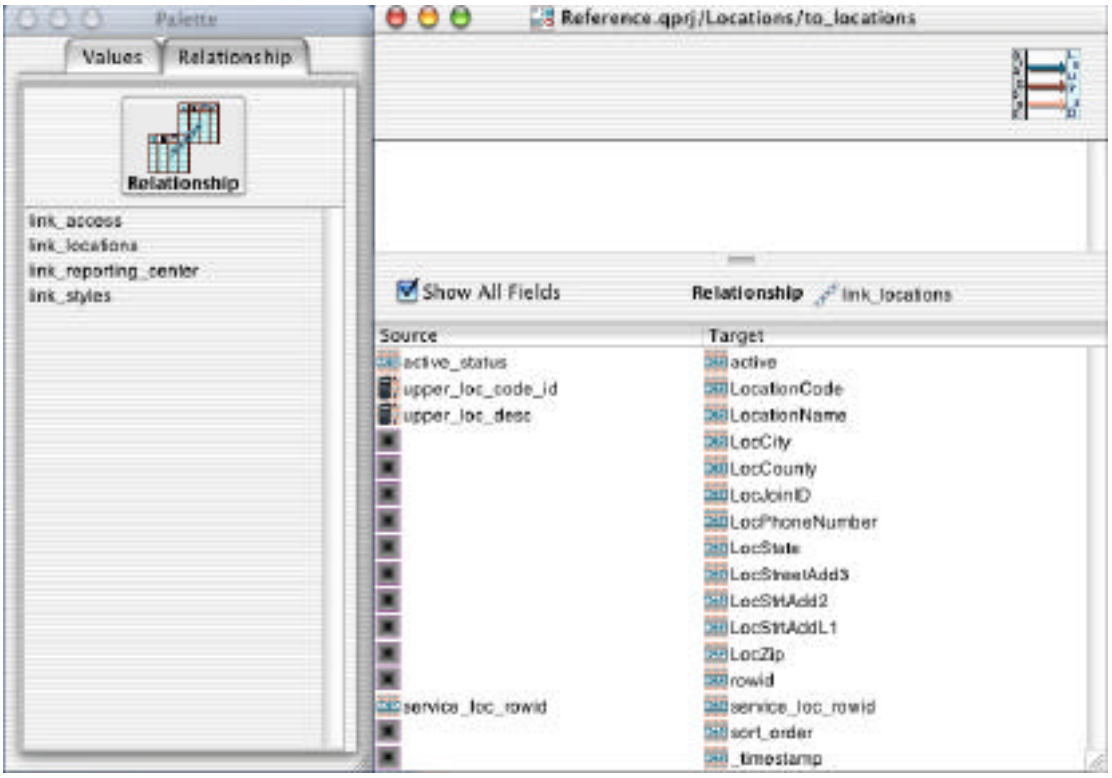
Building a DataFlow

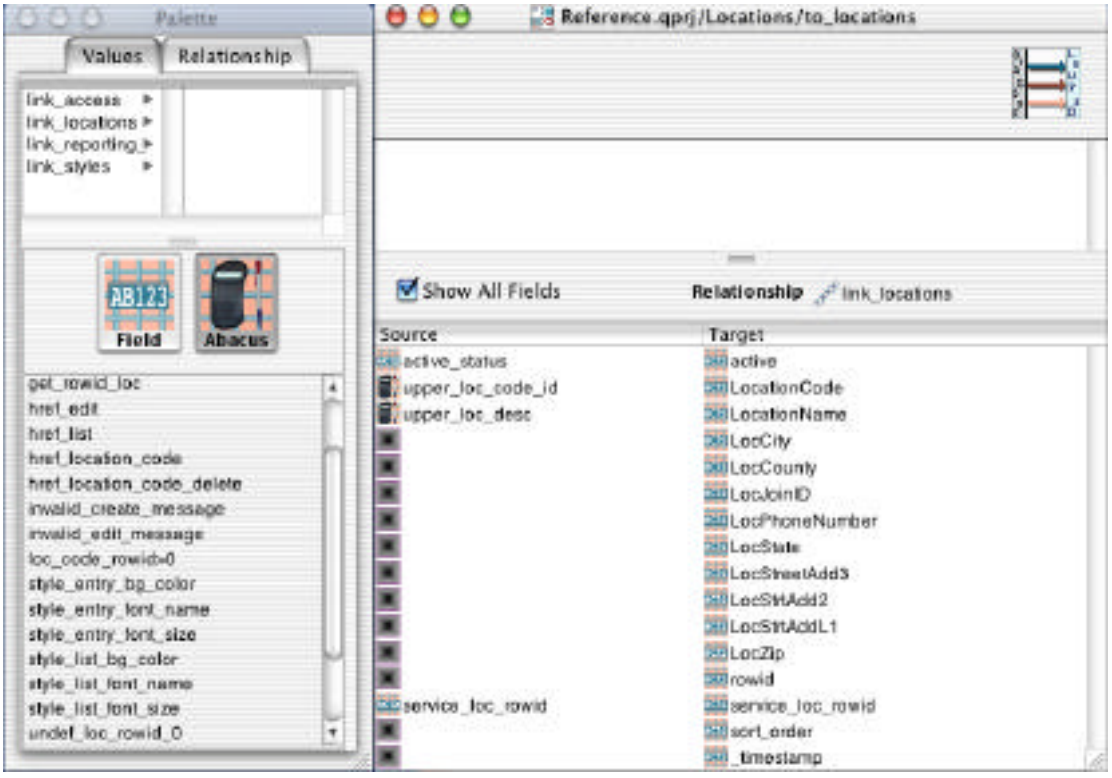
Open a Framework window then drag out a new Dataflow icon; double click the icon to open it. Like all other Qilan windows, the upper portion of the window can accept text comments.

The DataFlow window has three components: relationship, source values and target values.

The relationship refers to a Relationship icon in the same window as the DataFlow icon. The Relationship will define an Access > Table and Access > Table > Fields as the target values. Source values can be fields or abacus expressions located in the same window as the DataFlow icon.

The Relationship and source values are chosen from the palette. The DataFlow palette shows two tabs: Values and Relationships.





DataFlows will be considered invalid without the identification of a relationship. Click on the palette's relationship tab. Use the pop-up to select an access, then a relationship. Drag the relationship's name, from the list, to the 'hole' labeled, "Relationship".

Once a relationship is dragged into the 'hole', it cannot be removed. If you need to change a relationship from an existing dataflow, remove the dataflow from all QUPDATE or QCREATE tags, which are using it. Then Edit > Clear the DataFlow from the framework window.

To the left of relationship icon, a checkbox is shown, labeled "Show all fields". It is on by default. The purpose of this checkbox is to allow you to view available table fields (targets) specified by the relationship. If you do not specify a source field or abaci for a corresponding target, the target field will not be updated.

The next task is to determine which source fields (or abaci) will inserted. Return to the palette and click the Values tab. Drag the name of the field or abacus to the dataflow window. Release it over the source line, which corresponds to the target field. If you make a mistake, choose Edit > Undo or Edit > Cut or just drag a new field or abaci over the source value.

Abacus Expressions

What is an Abacus?

An abacus is a derived value computed from fields and/or other abacus expressions, environmental or system value, specific computational procedure or user defined entry. Abacus 'results' may be used in queries, displays or as part of other abacus formulations.

Frameworks and Tables contain abacus expressions. There is no limit on the number of abacus expressions that can be created, however Qilan does preset a recursive depth. Refer to the Project Settings for more information.

Abacus expressions using or referencing Access > Table > Fields are interpreted as values originating within the database. Whereas abacus expressions using or referencing Framework > Fields are transient values, specific to WebTemplate process.

The meaning of the preceding statement is subtle, but important. Consider the following abacus query built in an Access > Table:

([rowid] = (acquire [value])

"Rowid" is a database value. It has permanent properties, such as type, precision, scale, width, indexed, etc. "Value" on the other hand, is obtained from the Framework. Its properties are derived from values assigned to it from other calculations during the cgi execution. In other words, "value" is transient and changeable.

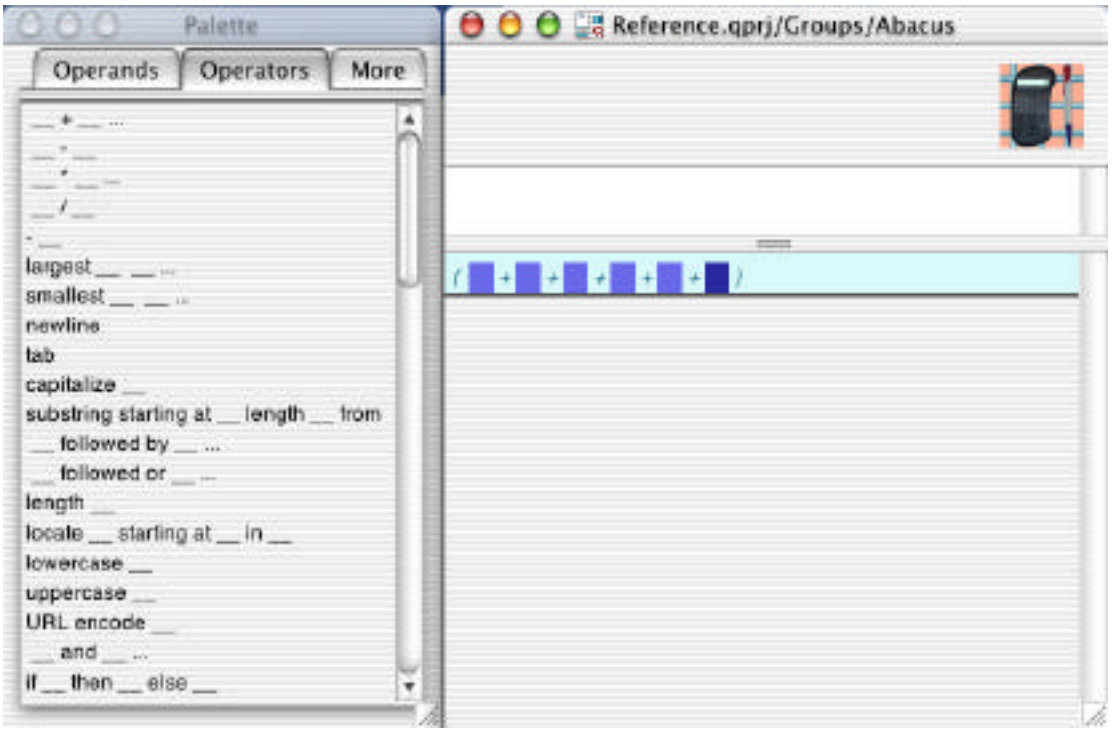
Operators

Qilan contains over 60 abacus operators. Each operator performs a specific operation. A completed abacus containing one or more operators is referred to as the abacus 'expression'. An expression will always return a result. Results may be undefined, defined and valued or defined and empty and be of the type: Text, Number, Date, or Flag.

Most operators consist of two parts: the 'parameter' and the function. The parameter is where the designer places a field, abacus, and other expressions or enters a value. The function describes what mathematical, display or manipulative action the operator performs. The parameter is represented by the underscore '__' in the operator list and a purple square when dragged to the abacus window. Purple squares are interpreted as undefined.

Qilan improves the efficiency of many operators by allowing them to 'expand'. This feature makes it easier to construct complex expressions as well as reduce operational redundancy.

Operators (see palette below) that end with an ellipsis (…) are expandable. After dragging the operator to the abacus window, highlight the right most purple square, then choose Icon > Expand.

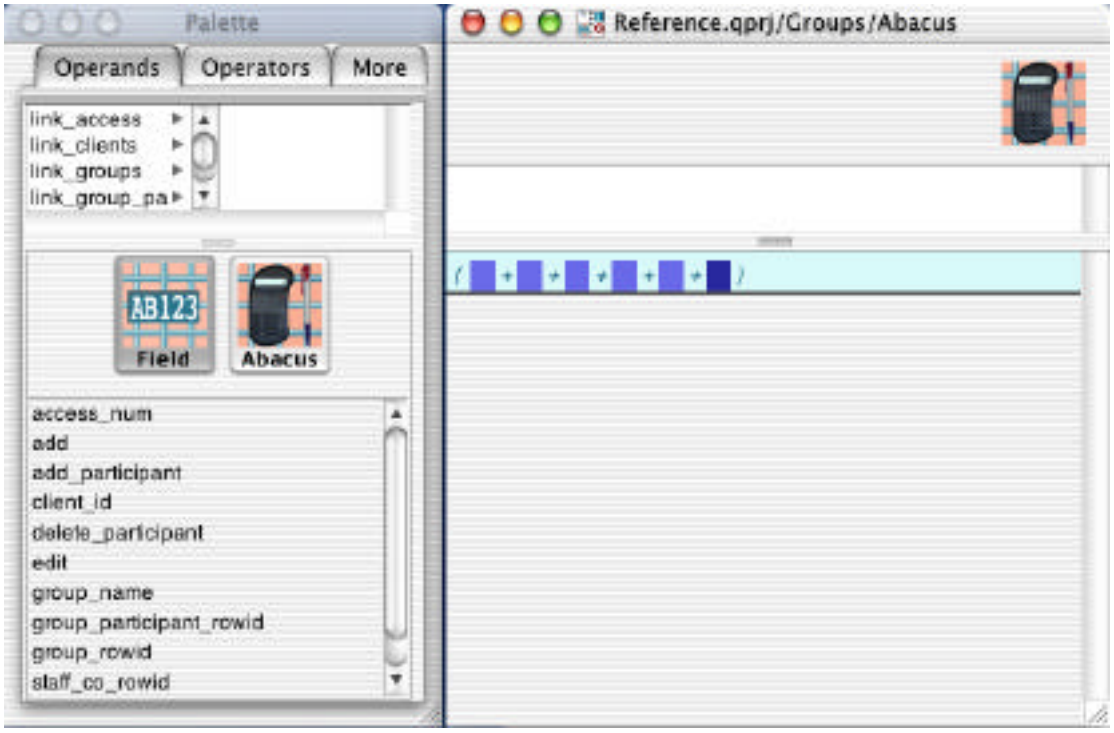


An ‘expanded’ operator can be contracted by highlighting the right most purple square, then choosing Icon > Contract.

Operands

Operands are fields, abacus expressions or constants, representing data, which are used by operators. Operands may be obtained from the same table containing the current abacus, WebTemplate objects, another table using a relationship or user entered text.

Operands may also be placed in an abacus widow without an operator. In this case, results will that of the operand.



Operator Listing

__ + __

Adds two numbers and returns a number. If the value placed into either side cannot be coerced into a number, the result will be undefined. {Expandable}

__ - __

Subtracts the right parameter (number) from the left parameter (number) and returns a number. If the value placed into either side cannot be coerced into a number, the result will be undefined.

__ * __

Multiplies two numbers and returns a number. If the value placed into either side cannot be coerced into a number, the result will be undefined. {Expandable}

__ / __

Divides the left parameter (number) by the right parameter (number) and returns a number. If the value placed into either side is cannot be coerced into a number, the result will be undefined.

- __

Returns the negative value of a positive number or the positive value of a negative number. If the initial value cannot be coerced into a number, the result will be undefined.

largest __ __

Compares two values of the same type and returns the largest value of that type. If the values are not of the same type, they will be converted to the same type, then evaluated. The output will then be formatted to that of the larger type. If either parameter is undefined, the result will be undefined. {Expandable}

smallest __ __

Compares two values of the same type and returns the smallest value of that type. If the values are not of the same type, they will be converted to the same type, then evaluated. The output will then be formatted to that of the smaller type. If either parameter is undefined, the result will be undefined. {Expandable}

newline

Accepts no values. Returns a Carriage Return/Line Feed.

tab

Accepts no values. Returns a tab character.

capitalize __

Accepts text input. Converts the first alpha character of each word to upper case.

substring starting at __ length __ from __

Extracts a text string using the relative position of text characters. Accepts two numbers and a text input. This operator returns a text string. The output will be undefined if any input is undefined or empty. Substrings that are partially outside of the source string return only the valid part.

__ followed by __

Concatenates any two values. If either value is undefined, the result will be undefined. Non-text inputs will be converted to text. The result will be text. {Expandable}

__ followed or __

Concatenates any two values. If either value is undefined, the other value will be passed. Non-text inputs will be converted to text. The result will be text. {Expandable}

length __

Accepts any text value and returns the number of characters in the string. Non-text inputs will be converted to text.

locate __ starting at __ in __

Returns the numerical starting position of a text string within a text string. Accepts a text input, number and text input.

unicode __

Unicode accepts a string and returns an integer. The first character of the string is the only one of significance. Dates or numbers are first converted to a string, and then first character of the conversion is used. The integer returned is the unicode equivalent of the first character.

character __

Character accepts a number and returns a string. The character is the Unicode equivalent of the number. Dates or strings are first converted to a number (if possible), before being converted to a string. If the number starts with "%", "U+", "0x" or "0X", it is interpreted as a hexidecimal number. Otherwise it is interpreted as a decimal number. Example, ‘13’ returns a carriage return, ‘10’ returns a line feed.

uppercase __

Converts text input to upper case.

URL encode __

Converts ASCII characters (decimal 0 through 126) and outputs their hexadecimal equivalent. Note, A - Z, a - z and 0 - 9 are not encoded. Example: ‘space’ becomes ‘%20’.

This operator comes to the rescue in situations when you want to submit data via the GET method. Use it in abacus constructions to encode field names or data.

lowercase __

Converts text input to lower case.

__ and __

Returns ‘0’ if any input is NO, ‘1’ if all inputs are YES, UNDEFINED if any input is undefined and there is no NO input. {Expandable}

if __ then __ else __

‘If’ accepts a flag parameter, then any two parameters. This operator outputs the ‘Then’ parameter if the result of ‘If’ flag is “YES” and outputs the ‘Else’ input if the result of the ‘If’ input is “NO”. If the ‘If’ input is undefined, all outputs will be undefined. The parameters for ‘Then’ and ‘Else’ do not have to be of the same data type.

__ or __

Returns ‘1’ if any input is YES, NO if all inputs are NO, UNDEFINED if any input is undefined and there is no YES input. {Expandable}

__ contains __

Compares the contents a text string with a value. Accepts two text inputs and returns a flag. Returns ‘1’ if the second parameter occurs anywhere within the first parameter. If either operand is undefined, the result is undefined.

__ ends with __

Compares the end of a text string with a value. Accepts two text inputs and returns a flag. Returns ‘1’ if the second parameter occurs at the end of the first parameter. If either operand is undefined, the result is undefined.

__ starts with __

Compares the start of a text string with a value. Accepts two text inputs and returns a flag. Returns ‘1’ if the second parameter occurs at the beginning of the first parameter. If either operand is undefined, the result is undefined.

__ like __

Compares a value with a pattern match. Accepts two inputs and returns a flag. Returns ‘1’ if the pattern specified by the second parameter matches the first parameter. If either the first or second parameter is undefined, the result is undefined.

Using ‘like’ without the use of wildcards is basically the same as ‘equals to’. Wildcards extend search functionality by enabling patterns to be specified, not merely text strings. There are two basic types of wildcards: ‘accept any’ and ‘accept one’.

	SQL-92 Standard	Variation
Accept Any	%	*
Accept One	_	?

When __ like __ is used in an Access > Table, wildcards are defined by the database. Although most use the SQL-92 Standard or the accepted variation, databases are free to define their own syntax, including additional wildcards. Refer to your database documentation for the appropriate wildcard character.

When __ like __ is used in the Framework, wildcards are defined by the SQL-92 Standard.

Examples:

Search for all occurrences where the second character is ‘e’.

[table.field] like “_e%” escape __

The underscore will accept any single character as the first letter and the percent sign will accept any number of characters to follow the letter ‘e’.

Validate an input to insure it is in the format of a social security number (###-##-###).

[field] like "___-__-___"

This will return true (1) when the format is in the form of a social security number, containing two dashes (-).

__ like __ escape __

Compares a value with a pattern match. Accepts three text inputs and returns a flag. Returns '1' if the pattern specified by the second parameter matches the first parameter. If either the first or second parameter is undefined, the result is undefined.

The third parameter defines a character preceding a wildcard character. It accepts a text input. When encountered, the wildcard character will be interpreted as part of the text string.

Example:

Search for all occurrences of the letter 'a' followed by an underscore followed by, somewhere else in the word, the letter, 'b'.

[table.field] like "%a_%b%" escape "\"

The escape parameter identifies the backlash (\) as the escape character. In the pattern, the escape character preceded the underscore (_). This causes the database to interpret the underscore literally

__ = __

Compares any value with a value for equality. Accepts two inputs and returns a flag.

__ not equal __

Compares any value with a value for inequality. Accepts two inputs and returns a flag.

__ > __

Compares two values for inequality, where the left value is greater than the right value. Accepts two inputs of the same type and returns a flag.

__ >= __

Compares two values for equality or where the left value is greater than the right value. Accepts two inputs of the same type and returns a flag.

__ < __

Compares two values for inequality, where the left value is less than the right value. Accepts two inputs of the same type and returns a flag.

__ <= __

Compares two values for equality or where the left value is less than the right value. Accepts two inputs of the same type and returns a flag.

defined __

Accepts any value and returns a flag. The flag value will be ‘1’ if the input is valued or empty; ‘0’ if the input value is undefined.

__ defined or __

Accepts any two values and returns the first defined, non-empty value, evaluating left to right. {Expandable}

empty becomes undefined __

Accepts any value and returns undefined when the input is defined, but contains no value. All other inputs are passed without modification.

undefined __

Accepts any value and returns a flag. The flag value will be ‘1’ if the input is undefined; ‘0’ if the input value is defined or empty.

empty __

Accepts any defined value and returns a flag. The flag value will be ‘1’ if the input is defined, but contains no value. This is the nominal state of a blank HTML text input. The flag value will be ‘0’ if the input is defined and contains a value. An undefined input will return an undefined result. Note, a “value” is any input whose length is greater than 0.

not __

Accepts a flag value and returns the opposite flag value. ‘1’ becomes ‘0’. If the flag is not defined, the result is undefined.

text __

Accepts any data type input and returns the same input as text.

integer __

Accepts any data type input and returns the same input as an integer. If the input data type cannot be interpreted as a number, the result will be undefined.

float __

Accepts any data type input and returns the same input as a floating point number. If the input data type cannot be interpreted as a number, the result will be undefined.

date __

Accepts any data type and returns a date [OS format]. If the input data type cannot be interpreted as a number, the result will be undefined.

flag __

Accepts any data type input and returns the same input as a flag. If the input data type cannot be interpreted as a flag, the result will be undefined.

__ formatted by __

Accepts a number or date data type in the first parameter, and a text input in the second parameter. It formats the date or number according to the instructions (format) in the second parameter, and returns the formatted result (text).

HTML input values are always interpreted as text. Convert these values to numbers or valid Unix dates before placing them into the first parameter. For more information concerning the format syntax for the second parameter, please refer to ***Formats*** at the end of this section.

Formatted by can be used to convert Unix dates to more traditional formats accepted by other applications. [Unix_date] formatted by “%m/%d/%y” will convert 2000-11-30 00:00:00 –0500 to 11/30/00.

round __

Accepts a number data type and rounds fractions to the nearest integer. The result is a number.

round __ to nearest __

Accepts a number data type in the first parameter. The first number will be rounded to the value in the second parameter. The result is a number.

now

Accepts no values. Returns a date (date, time [OS format] and the difference in the current timezone from GMT [±]). The system localization format will be used. The US default is: yyyy-mm-dd hh:mm:ss ±0000.

The operating system keeps track of time right down to the millionth of a second (and beyond). Converting Now into a number (Float), will return the number of seconds prior to 1/1/2001 with a large scale. From experience, we have found that this number is good way (although not perfect) to create unique entry values.

day of the week __

Accepts a date input and returns a number corresponding to the day of the week, zero through 6. (Sunday = 0)

year __

Accepts a date input and returns a four digit number corresponding to the year.

month __

Accepts a date input and returns a number corresponding to the month. (January=1)

day of the month __

Accepts a date input and returns a number corresponding to the day of the month.

hour __

Accepts a date input and returns a number corresponding to the hour.

minute __

Accepts a date input and returns a number corresponding to the minute.

second __

Accepts a date input and returns a number corresponding to the second.

timezone __

Accepts a date input and returns a number corresponding to the difference between GMT and the current timezone [±], in seconds.

month __ day __ year __

Accepts three numeric inputs then returns a date. The input values should be within the legal ranges for months (1-12), days (1-31) and years. The system will attempt to 'role over' values outside of legal ranges. Time will be output as 00:00:00; TimeZone as the number of hours ± GMT.

hour __ minute __ second __

Accepts three numeric inputs then returns a time. The input values should be within the legal ranges for hours (0-23), minutes (0-59) and seconds (0-59). The system will attempt to 'role over' values outside of legal ranges. Date will be output as 01-01-1970; TimeZone as the number of hours ± GMT.

date __ time __ timezone __

Accepts a date, time and number, then returns a localized date. The Timezone refers to the number of seconds ± GMT. For example, a New York City time would enter –18000 (five hours).

environment __

Accepts a CGI environmental keyword and returns the result. Keywords, definitions and examples are provided at the end of this section. Keywords may be entered by direct typing.

capture __

Obtains values from HTML inputs or table fields, by name.

For HTML inputs, the ‘name’ refers to the HTML input ‘name’ attribute. Accepts string value types and returns the HTML input ‘value’ attribute. The name parameter and the HTML input name must be matched exactly. Names are case sensitive.

Capture is used to obtain the value of HTML inputs when the input name attribute is dynamically assigned. For example, consider a QTABLE where each row contains one HTML input. The name attribute is the table’s rowid.

```
QTABLE    fields        table.rowid
      TR
          TD
                QASSIGN   icon        framework.rowid
                          value       table.rowid
                INPUT     type        text
                          name        framework.rowid
```

When this table is displayed, there will be an input on each row. As each row is retrieved, we assign table.rowid to framework.rowid.

Next, we use framework.rowid as the name attribute for the HTML input tag.

The name of the input will correspond to the rowid of the record being retrieved.

The value of the input is then obtained as follows:

Capture [framework.rowid]

This abacus can then be used in other calculations, relationships or dataflows.

Observe that a list of inputs can be submitted with a single HTML input. Refer to ‘Capturing Values’ for examples.

For table field names, the 'path' to the table field must be typed as follows: /[access]/[table]/field. For example, where the Access name is, "MIS", the table name, "Accounts", and the field name, "amount", the correct path is:

```
/MIS/Accounts/amount
```

- The leading and separating forward slashes are required.
- Table and field names are the Qilan names, not the external names.
- Names are case sensitive

Accessing a table field in this manner is intended to offer the designer a method to dynamically select field(s). The traditional manner is to use a QVALUE by placing the field icon in the icon attribute, however any variables passed to the Capture operator will be interpreted as the field name.

acquire __

Obtains values from an enclosing "scope". A scope could be a QFIND (or equivalent), or the Framework. Accepts any value type and returns the same value type.

Acquire becomes useful when requiring a framework value in table > abacus constructions. For example, a table > abacus query routinely evaluates table data against framework values. Acquire obtains this value.

```
( rowid = ( acquire <Status/sched_rowid> ) )
```

This abacus is constructed in the Access > table. 'Rowid is a field in the table which is compared to 'sched_rowid' in the Framework, 'Status'.

Another use for 'acquire __' is to obtain a framework values to complete data strings in the Access > Table. Suppose you want to create an 'href' abacus in a table which will be retrieved via a QFIND. This abacus needs a value from the Framework. Construct this as follows:

```
( "Client_Payments" followed by "?" followed by "access_num" followed by "="
 followed by ( acquire <Payments/access_num> ) followed by "&" followed by
"consumer_id" followed by "=" followed by client_id )
```

acquire __ from __

Obtains values from an enclosing "scope". A scope could be a QFIND (or equivalent), or the Framework. The second parameter is used to identify a QGROUP or QFIND tag by its <NAME> attribute.

'From' refers to the literal name of a WebTemplate object (QGROUP or any of the QFIND derivitives). It is case sensitive.

Abacus expressions containing or referencing the expression, Acquire __ From __, must be demoted (or within) the named object.

In the following example, the first QFIND queryicon is acquiring a value returned by a QGROUP. The QFIND is demoted within the QGROUP. Likewise, the second QFIND is also acquiring a value from the QGROUP.

```
QGROUP    Name          [Summary]
          Ave           [AvePrice]
     QFIND      Name [Find]
                QueryIcon   [Price ≥ AvePrice]
                Fields[CompanyName]
          QFIND      QueryIcon [BestCustomer]
                     Fields [CustomerName]
```

When all data returns are performed from the same database table, Qilan does not know which object to refer to. For instance, if the query for the innermost QFIND acquires a value from the QGROUP, the identification of the intended 'object' will be ambiguous, as the other QFIND is also an eligible object. When this situation arises, you may name the target object. This is the purpose of Acquire __ From __.

assign __ from __

Specifies a field, then a value. This operator is used to assign a value to a field. It is useful for setting control values for looping and recursive functions. The operator returns the value of the second parameter.

'From' refers to a value assigned to a field. It may be a literal value, such as, '1', or a reference to a field or another abacus.

Framework field values are established when the WebTemplate is processed. The values will be retained throughout the processing unless they are reused or explicitly changed using the Assign __ From __ expression or QASSIGN tag.

The form, 'Schedule_Service', uses a single date to set its entry. What we want to do is create a clickable link, that when clicked, changes the user's entry to the current day. Here is how this is done:

```
( ( assign day from ( ( day of the month sched_date_assigned ) defined or day ) )
defined or ( day of the month now ) )
```

The value of 'day' is derived from 'sched_date_assigned' (a date). The value of 'sched_date_assigned' (also a field) is created by a href link as follows:

```
( "Schedule_Service" followed by "?" followed by "access_num" followed by "="
followed by access_num followed by "&" followed by "staff_rowid" followed by "="
followed by staff_rowid followed by "&" followed by "sched_date_assigned"
followed by "=" followed by ( URL encode now ) )
```

Note how the value of 'sched_date_assigned' is set to the value of Now. We use URL encode __ because the Unix date has spaces in its string and therefore needs to be encoded.

The format of this construction, when used in conjunction with HTML A href values, will submit the form and return the same page ("Schedule_Service"). The value of 'day' will now be the current day.

The additional operators included with the 'assign __ from __' are used to set the value of 'day' when the page is initially retrieved. At this point, 'sched_date_assigned' will be undefined.

__ encrypt with __

Encrypt implements Twofish (128-bit block cipher) with a variable-length key up to 256 characters to securely encrypt string data. The first and second parameters accept strings as their input. The second parameter is interpreted as the 'key', and should be at least 8 characters in length. The time necessary to encrypt is dependent upon the key length.

Input 'strings' may consist of any combination of non-numeric or alphanumeric characters. Numbers alone (with or without decimal points) are not considered strings. Numbers should be converted to strings by using the '__ formatted by __' abacus operator. For example:

(("34.56" formatted by "##.##") encrypt with "aAc454vMcd")

Keys should consist of alphanumeric characters; the use of numbers alone should be avoided.

__ decrypt with __

Decrypt implements Twofish (128-bit block cipher) with a variable-length key up to 256 characters to decrypt string data encrypted with the abacus operator '__ encrypt with __'. The first and second parameters accept strings as their input. The second parameter is interpreted as the 'key', and must exactly match the key used with the encryption abacus. The time necessary to decrypt is dependent upon the key length.

Encryption/Decryption operators give you the ability to store data remotely without fear. Your data is transmitted and stored in a database totally encrypted. Some uses come immediately to mind… credit card information, storing information in databases over insecure lines, passing information between two versions of Qilan in different locations, etc. Now I'm happy!

function __ parameter __

This operator implements most of the mathematical functions as described by the C math library (libm). {Expandable}

To use this abacus operator, double click on the function ‘hole’, and then type the name of the desired function. Available functions are listed below (names are case insensitive). Entering an unlisted function name will return an error.

```
( function "sqrt" parameter number_x )
```

Other abacus operators or abacus constructions may be dragged in to the function ‘hole’ so long as they output a listed function as a text string.

```
( function ( if what_function then "sin" else "cos" ) parameter number_x )
```

The parameter ‘hole’ contains a single numeric value. This is the value upon which the function will act. Entering a non-numeric value will return an error.

When a function requires the entry of more than one parameter value, click once on the rightmost undefined parameter hole to highlight it. Choose, “Expand Selection”, from the Icon menu. The operator will expand so that a second, undefined, parameter value appears. The parameter value ordering is, ‘x, y’.

```
( function "remainder" parameter number_x parameter number_y )
```

Other abacus operators or abacus constructions may be dragged in to the parameter ‘hole’ so long as they output numeric values or values which can be coerced into numbers.

```
( function "log" parameter ( function "exp" parameter number_x ) )
```

Function Operators

`acos`	computes the principal value of the arc cosine of x in the range [0, pi].
`acosh`	computes the inverse hyperbolic cosine of the value x.
`asin`	computes the principal value of the arc sine of x in the range [-pi/2, +pi/2].
`asinh`	computes the inverse hyperbolic sine of the value.
`atan`	computes the principal value of the arc tangent of x in the range [-pi/2, +pi/2].
`atanh`	computes the inverse hyperbolic tangent of the value x.
`atan2`	computes the principal value of the arc tangent of y/x, using the signs of both values to determine the quadrant of the return value.
`cbrt`	computes the cube root of x.
`ceil`	returns the smallest integral value greater than or equal to x.
`copysign`	returns x with its sign changed to y's.
`cos`	computes the cosine of x (measured in radians).
`cosh`	computes the hyperbolic cosine of x.
`erf`	calculates the error function of x; where x = 2/sqrt(pi)*integral from 0 to x of `exp` (-t*t) dt.

`erfc`	calculates the complementary error function of x; that is, `erfc` subtracts the result of the `erf` from1.0.
`exp`	computes the exponential value of x.
`expm1`	computes the value of `exp`(x)-1.
`fabs`	computes the absolute value of a floating-point number x.
`floor`	returns the largest integral value less than or equal to x.
`hypot`	computes the `sqrt`(x*x+y*y)
`ilogb`	returns x's exponent n, in integer format.
`j0`	computes the Bessel function of the first kind of the order 0, for the real value x.
`j1`	computes the Bessel function of the first kind of the order 1, for the real value x.
`lgamma`	returns `log` || (x) |.
`log`	computes the value of the natural logarithm of the value x.
`log10`	computes the value of the logarithm for the value x to base 10.
`log1p`	computes the value of `log`(1+x) accurately even for a tiny value x.
`pow`	computes the value of x to the exponent y.
`remainder`	returns the remainder r := x - n*y where n is the integer nearest the exact value of x/y; moreover if |n - x/y| = 1/2 then n is even.

`rint`	returns the integral value (represented as a double precision number) to the nearest x.
`sin`	computes the sine of x (measured in radians).
`sinh`	computes the hyperbolic sine of x.
`sqrt`	computes the non-negative square root of x.
`tan`	computes the tangent of x (measured in radians).
`tanh`	computes the hyperbolic tangent of x.
`y0`	computes the linearly independent Bessel function of the second kind of the order 0 for the positive integer value x.
`y1`	computes the linearly independent Bessel function of the second kind of the order 1 for the positive integer value x.

So, what do these functions actually do? Well, if you're a math major, the answer can seem obvious from the description, but for the rest of us… If you want to know more, try using the Terminal application and typing, "man [function_name]". I suggest you first take a look at "man math" for a quick overview of the math functions. A bit 'geeky', but informative nevertheless.

Abacus Limits

When one abacus is used or referenced by another abacus, Qilan treats each abacus as if it were on a different level or depth. By default, a user configurable maximum depth has been set to 100. See Recursive Limits for more details.

Conversion

Calculations in Qilan are lenient with regards to proper format. If an operand is of the wrong type for an operator, an attempt is made to convert it to the right type. If the conversion cannot be done, an 'undefined' value is returned. The conversion rules are very lenient. For example, an empty string with no digits in it converts to the number 0.

Conversion Rules

Numeric calculations are all done with double precision floating point numbers. Integers are converted to their corresponding floating point.

If the required type is a Flag, the Text "YES", "Y", "TRUE" or "T" are interpreted as YES; "NO", "N", "FALSE" or "F" are interpreted as NO, all other text strings are undefined; the number '0' is NO, all other numbers are 'YES'.

If the required type is a Number, the Flag NO is interpreted as 0, the Flag YES is interpreted as 1. Text, as numerical representations, will not be treated as numbers unless explicitly converted using the Float or Integer operator. HTML input value are always interpreted as text.

If the required type is Text, the value is converted to Text, just as if it was being put into the HTML document.

If the required type is a date and the input value is text, only values that conform to the system format will be interpreted as a date, all other values will be undefined. Numbers and text representations of numbers will be interpreted as dates.

Some databases, notably FrontBase, require explicit data typing. If you receive errors that refer to incorrect data formats or other data incompatibilities convert your data. The following abacus operators are used to explicitly type data:

Integer __	Float __	
Date __	Flag __	(aka boolean)
Text __		

Manipulating Dates and Times

Qilan uses the localized system specification for dates and times. For Mac OS X, the numerical equivalent for the current date and time is the number of seconds ± 1/1/2001. For example, if the Now operator is converted to a number, it would output a very large negative value, if the current date is prior to 1/1/2001.

Dates and times are recorded in seconds. Therefore a day equals sixty seconds times 60 minutes times 24 hours or the numerical value of 86400. Fractions of seconds can also be expressed in milli, mirco, pico or nano seconds.

Mac OS X assumes all dates include time and locale. This is also the specification for most SQL databases. A date input, therefore, must contain all the necessary components. Attempting to extract the month number from the string, "3/24/00", will result in an undefined value. However, if the string, "2000-03-24 00:00:00 –0050" is used, the value will be "3". For HTML date submittals, you have three options:

Create separate fields for month, day and year, and then pass these values to the Month__Day__Year operator. Qilan will then create a legal date.

Use the 'locate' and 'substring' extraction operators to parse the date into the month, day and year. Then pass these values to the Month__Day__Year operator. Qilan will then create a legal date or type the date as text in any one of the following formats:

[yyyy-mm-dd]	omitting time
[yyyy-mm-dd] [hh:mm:ss]	omitting timezone
[yyyy-mm-dd] [hh:mm:ss] [[+|-]tt[:]zz]	fully formatted

Use the abacus operator, "Date __" to convert a text entry to a date data type. Otherwise, Qilan will interpret it as text.

When the Month__Day__Year operator is used, without a time specification, the time will be set to zero. The GMT difference will still be expressed. If the Hour__Minute__Second__ operator is used without a date specification, the date will be output as 1970-01-01.

To avoid all the potential difficulty dealing with dates, times, timezones and formatting specifics, consider storing dates as integers. Although this may not be suitable for all applications, it works well when the *day* is the storage value.

Using the 'formatted by' abacus operator, format a date (with or without time specification) as YYYYMMDD (%Y%m%d). This will return a fixed length (8) numerical value which increments for each passing day. Calculations performed on this value will refer to a day, rather than a daily time span.

One caution however, storing a day as an integer implies that queries must be based on the comparisons of integer values, not integers to text values. Consider these queries:

Framework:
([date] formatted by [%Y%m%d]) > [20030101]
(integer ([date] formatted by [%Y%m%d])) > (integer [20030101])

Table:
[day] > [20030101]
[day] > (integer [20030101])

These pairs are NOT the same and will yield different results. Each pair compares an integer to a text value, and an integer to an integer value.

Defined, Empty and Undefined Values

Qilan treats values as DEFINED, DEFINED and EMPTY or UNDEFINED. A defined value contains data or is designated as empty. An undefined value is neither empty nor defined.

Operators that accept numbers or dates will convert empty values to '0'. Operators that accept text strings will pass empty values as having no value.

> An input value is considered empty when user input is 'declared' but not entered. When an operator is dragged into the abacus window, purple squares replace operator parameters. The purple square is considered an undefined value. When a user double clicks the purple square (declares a value), a double quote will appear (""). Qilan now interprets this as 'empty'. To enter a value, the user merely has to type. To remove the (""), highlight the quotes and choose, Edit > Clear. The purple square will be restored and the value will be undefined.
>
> When an empty value is encountered in a text string, it will be passed as a 'defined, yet empty value'. For example, ABC""DEF, where the double quotes represent an empty value, the result will be ABCDEF.
>
> You can test for an empty value using the Empty __ operator.

Undefined database values will produce an undefined for calculations. Most operators produce an undefined result if any of its operands are undefined.

If you are used to the database concepts of defined and undefined (null), and do not wish to use 'empty', enable the project preference setting, "Treat Empty as Undefined". Setting this preference will automatically convert empty values to undefined.

Operator Input/Output Summary

Operator	Inputs	Output
And	N Flag	Flag
Capitalize	1 Text	Text
Capture	1 Text	Text
Contains	2 Text	Flag
Defined	1 Any	Flag
Divide	2 Number	Number
EndsWith	2 Text	Flag
EqualTo	2 Any	Flag
Substring	3 NNT	Text
Flag	1 Any	Flag
FollowedBy	N Text	Text
FollowedOr	N Text	Text
Function	T Number	Number
IfThenElse	3 FAA	Any
GreaterOrEqualTo	2 Same	Flag
GreaterThan	2 Same	Flag
Largest	N Any	Any
Length	1 Text	Number
LessOrEqualTo	2 Same	Flag
LessThan	2 Same	Flag
Like	2 Text	Flag
Locate	3 TNT	Number
LowerCase	1 Text	Text
Minus	2 Number	Number
Negate	1 Number	Number
Not	1 Flag	Flag
NotEqualTo	2 Any	Flag
Number	1 Any	Number
Or	N Flag	Flag
Plus	N Number	Number
Round	1 Number	Number
RoundTo	2 Number	Number
StartsWith	2 Text	Text

Smallest	N Number	Number	
Text	1 Any	Text	
Times	N Number	Number	
Undefined	1 Any	Flag	
UndefinedBecomes	N Any	Any	
UpperCase	1 Text	Text	
URLEncode	1 Any	Text	
NewLine	0	Text	
Tab	0	Text	
DefinedOr	N Any	Any	
FormattedBy	2 (ND) and T	Text	
Now	0	Date	
Today	0	Date	
DayOfWeek	1 Date	Number	
Year	1 Date	Number	
Month	1 Date	Number	
DayOfMonth	1 Date	Number	
Hour	1 Date	Number	
Minute	1 Date	Number	
Second	1 Date	Number	
TimeZone	1 Date	Number	
MonthDayYear	3 Number	Date	
HourMinuteSecond	3 Number	Date	
DateTimeZone	2 Date; Number	Date	
Capture	1 Any	Any	
Acquire	1 Any	Any	
AcquireFrom	2 Any	Text	Any
AssignFrom	2 Field	Any	Any

Codes for Input/Output Summary:

The first character gives the number of inputs the operator recognizes. Associative commutative operators allow N, meaning that any number of inputs may be present. "Any" means that all types are acceptable.

NNT expects two Numbers and a Text.
D is a date.
FAA expects a Flag and two more of Any (not necessarily matching) type.
TNT expects a Text, a Number, and a Text.

'Same' expects two inputs of the same type. If they are not of the same type, one of them will be converted to match the other.

If any input does not match the type expected by an operator, it is converted.

Formats

Introduction

Numerical and date data may be displayed in a variety of user defined formats. Qilan uses a special abacus operator for this purpose. The operator, __ formatted by __, will format the data when a number or date is placed into the first (leftmost) parameter and an acceptable syntax is typed into the second (rightmost) parameter.

If string data (text) is placed into the first parameter, it will be coerced into a number or date if it can be interpreted. For example, the text string ‘1234’ will be treated as a number. Strings that cannot be coerced, such as ‘A1B1’ will be passed through unchanged regardless of the format syntax.

Formatting Options: Numbers

Format strings, entered into the second parameter, can include numeric characters. Wherever you include a number in a format string, the number is displayed unless an input character in the same relative position overwrites it. For example, suppose you have the positive format string "9,990.00", and the value 53.88. The operator would display the value as 9,953.88.

Separators

Format strings can include the period character (.) as a decimal separator, and comma character (,) as a thousand separator.

If you want to use different characters as separators, you can set them in Project Settings or enter your own using the __ formatted by __ abacus operator.

Placeholders

You use the pound sign character (#) to represent numeric characters. For example, suppose you have the positive format "$#,##0.00". If the characters 76329 were entered, they would be displayed as $76,329.00. Strictly speaking, however, you don't need to use placeholders. The format strings ",0.00" , "#,#0.00", and "#,##0.00" are functionally equivalent.

In other words, including separator characters in a format string tells the operator to use the separators, regardless of whether you use (or where you put) placeholders. The placeholder character's chief virtue lies in its ability to make format strings more human-readable.

Spaces

To include a space in a format string, use the underscore character (_). This character inserts a space if no numeric character has been input to occupy that position.

Currency

The dollar sign character ($) is normally treated just like any other character that doesn't play a special role. However, when you enable localization, the dollar sign character is converted to the currency symbol appropriate for the environment in which the application is running.

All other characters specified in a format string are displayed as typed.

Specification of Multiple Formats

The syntax is a 'literal' expression of the desired format. If more than one numerical formats is desired a designer can use a semicolon to specify different formats. For example, a format can be specified for a positive, zero and negative value.

Positive Format:

$###,##0.00

Positive Format;Negative Format:

###,##0.00;(###,##0.00).

Positive Format;Zero Format;Negative Format:

$###,###.00;0.00;($###,##0.00). Note that zero formats are treated as string constants.

As implied in the above list, you're only required to specify a format for positive values. If you don't specify a format for negative and zero values, a default format based on the positive value format is used. For example, if your positive value format is #,##0.00, an input value of "0" will be displayed as 0.00. If you don't specify a format for negative values, the format specified for positive values is used, preceded by a minus sign (-). If you specify a separate format for negative values, its separators should be parallel to those specified in the positive format string. Separators are either enabled or disabled for all formats-both your negative and positive formats should therefore use the same approach.

Formatting Options: Dates

Date formats can include any characters (except %'s), which are copied as typed. They can also include %'s, which are interpreted as follows:

Specifier	Description
%%	a '%' character
%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%c	shorthand for " %X %x ", the locale format for date and time
%d	day of the month as a decimal number (01-31)
%e	same as %d but does not print the leading 0 for days 1 through 9
%F	milliseconds as a decimal number (000-999)
%H	hour based on a 24-hour clock as a decimal number (00-23)
%I	hour based on a 12-hour clock as a decimal number (01-12)
%j	day of the year as a decimal number (001-366)
%m	month as a decimal number (01-12)
%M	minute as a decimal number (00-59)

%p	AM/PM designation for the locale
%S	second as a decimal number (00-59)
%w	weekday as a decimal number (0-6), where Sunday is 0
%x	date using the date representation for the locale
%X	time using the time representation for the locale
%y	year without century (00-99)
%Y	year with century (such as 1990)
%Z	time zone abbreviation (such as PDT)
%z	time zone offset in hours and minutes from GMT (HHMM)

Date/Time Examples:

11/20/99	%m/%d/%y
11/20/1999	%m/%d/%Y
Nov 20, 1999	%b %d, %Y
November 20, 1999	%B %d, %Y
Sunday, November 20, 1999	%A, %B %d, %Y
1 PM, Sunday, November 20, 1999	%I %p, %A, %B %d, %Y
(today), as an integer	%Y%m%d

If you add a number to a date, say 86400 seconds (one day), the result will be a number, not a date. So, don't forget to use the date __ abacus operator to format the output as a date. For example,

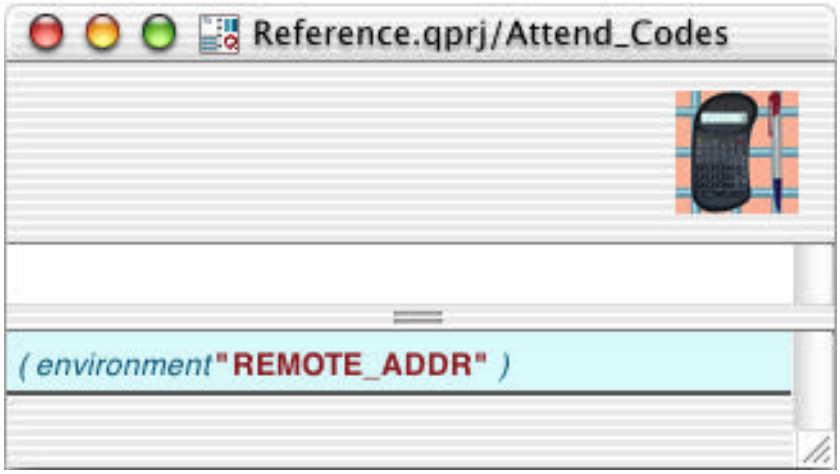
((date (now + "86400")) formatted by "%m/%d/%y")

will output tomorrow's date in a mm/dd/yy format.

CGI Environmental Keywords

In order to pass data about the information request from the server to the cgi script, the web server uses environment variables. Environmental variables are set when the web server executes qilan.cgi.

Entering the appropriate keyword into the abacus operator, environment __, will extract the variable information as described below. If the variable information is unknown or unavailable, the result of the abacus operator will be undefined.



Keywords are case sensitive.

The following environment variables are request-specific:

QUERY_STRING

> The information which follows the '?' in the URL which referenced the qilan document. It is interpreted by qilan.cgi to preset framework variables. It should not be used for other purposes.

PATH_INFO

> The extra path information, as given by the client. This is what follows cgi-bin/qilan.cgi/ in the URL that invokes the qilan document.

SESSION_ID

The session_id, as given by the client. This is what follows cgi-bin/qilan.cgi// in the URL that invokes the qilan document.

SESSION_ID is a Qilan specific environmental variable. It is created when one or more framework fields are specified as a 'session variable' and appears as an encrypted string in the URL.

The following environment variables are not request-specific and are set for all requests:

SERVER_SOFTWARE

The name and version of the information web server software answering the request (and running qilan.cgi). Format: name/version

SERVER_NAME

The server's hostname, DNS alias, or IP address, as it would appear in self-referencing URLs.

GATEWAY_INTERFACE

The version of qilan.cgi executed by the web server. Format: CGI/version.

The following environment variables are specific to the request being fulfilled by the web server:

SERVER_PROTOCOL

The name and version of the information protocol this request came in with. Format: protocol/version.

SERVER_PORT

The port number to which the request was sent.

REQUEST_METHOD

The method with which the request was made. For HTTP, this is "GET", "HEAD", "POST", etc.

PATH_INFO

The extra path information, as given by the client. In other words, scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO. This information should be decoded by the web server if it comes from a URL before it is passed to the CGI script.

PATH_TRANSLATED

The web server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.

SCRIPT_NAME

A virtual path to the script being executed used for self-referencing URLs.

REMOTE_HOST

The hostname making the request. If the web server does not have this information, it should set REMOTE_ADDR and leave this unset.

REMOTE_ADDR

The IP address of the remote host making the request.

AUTH_TYPE

If the server supports user authentication, and the script is protects, this is the protocol-specific authentication method used to validate the user.

REMOTE_USER

If the server supports user authentication, and the script is protected, this is the username they have authenticated as.

REMOTE_IDENT

If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.

CONTENT_TYPE

For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.

CONTENT_LENGTH

The length of the said content as given by the client.

In addition to these, the header lines received from the client, if any, are placed into the environment with the prefix HTTP_ followed by the header name. Any ‘-‘ characters in the header name are changed to ‘_’ characters. The server may exclude any headers that it has already processed, such as Authorization, Content-type, and Content-length. If necessary, the server may choose to exclude any or all of these headers if including them would exceed any system environment limits.

HTTP_ACCEPT

The MIME types which the client will accept, as given by HTTP headers. Other protocols may need to get this information from elsewhere. Each item in this list should be separated by commas as per the HTTP spec. Format: type/subtype, type/subtype.

HTTP_USER_AGENT

The browser the client is using to send the request. General format: software/version library/version.

CGI Process Environment

Variable	Value
SCRIPT_URL	/cgi-bin/printenv
SCRIPT_URI	http://www.qilan.com:16080/cgi-bin/printenv
HTTP_UA_CPU	PPC
HTTP_UA_OS	MacOS
SCRIPT_FILENAME	/Library/WebServer/CGI-Executables/printenv
SERVER_NAME	www.qilan.com
HTTP_ACCEPT	*/*
REMOTE_PORT	52914
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 5.23; Mac_PowerPC)
HTTP_HOST	www.qilan.com
SERVER_PORT	16080
SCRIPT_NAME	/cgi-bin/printenv
PATH	/bin:/sbin:/usr/bin:/usr/sbin:/usr/libexec
HTTP_EXTENSION	Security/Remote-Passphrase
nokeepalive	1
REMOTE_ADDR	127.0.0.1
REQUEST_METHOD	GET
GATEWAY_INTERFACE	CGI/1.1
HTTP_CONNECTION	Keep-Alive
HTTP_PC_REMOTE_ADDR	24.62.225.146
SERVER_PROTOCOL	HTTP/1.1
SERVER_ADMIN	admin@example.com
REQUEST_URI	/cgi-bin/printenv
SERVER_SOFTWARE	Apache/1.3.29 (Darwin) mod_fastcgi/2.4.0 mod_ssl/2.8.16 OpenSSL/0.9.7b
SERVER_ADDR	127.0.0.1
HTTP_ACCEPT_LANGUAGE	en
DOCUMENT_ROOT	/Library/WebServer/Documents/www.qilan.com

Session Server

What is a Session Server?

A session is the exchange of information between a client and a server. Because the HTTP/ HTTPS protocol is a stateless protocol, a session server maintains and stores state information between and amongst client requests.

Server-side connections can use the session_id to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Qilan. When returning an HTTP object to a client, a server may also send a session_id (reference to state information). The session_id, imbedded in the URL string, references a file for which that state is valid.

Got it? The concept of a Session Server can get really obscure, but it simply enables data can be passed from form to form without being sent to the client (as hidden input values) or stored in a database.

Let's try an example… Suppose you want to keep track of a customer's number. Your site has lots of pages the customer can visit. How do you know the customer number as they navigate throughout your site? One way is to retrieve then pass the customer number back as a hidden html input value for each form. This works, but requires additional web page construction and allows the customer number to be viewed in the html page source. A potential security risk. Another option is use 'cookies' to store the customer number. Again, additional web page construction is required and not all clients enable cookies or feel comfortable using them. Lastly, html 'meta' tags can be used to store some types of state information, but is often not reliable.

Qilan, on the other hand, keeps client values on the web server in a special file.

Creating Sessions

The session server is automatically invoked whenever any framework field is designated as a ‘variable’. Access to this setting is achieved by double clicking on the framework field icon.

Qilan then performs the following actions when a client accesses a WebTemplate, located in the same framework as the designated field(s):

qilan.cgi is invoked;
A file is created in /Library/Qilan/Sessions;
A timestamp is written to the file;
SESSION_ID is valued;
The html file is executed; and
Session variables (name/value) are written to the file;

Qilan makes the file name available as an environmental variable, “SESSION_ID”. Use of this environmental variable in the URL string will return a link to the file and its contents:

[IP_Addr]/cgi-bin/qilan.cgi/.[SESSION_ID]/[html_name]

When qilan.cgi executes the WebTemplate, framework fields of the same name as contained in the session file will be valued from the session file.

Note, the period (.) preceeding the session_id is required. This syntax is necessary to designate the following string as the SESSION_ID. Also note that session_ids may exceed 30 characters in length.

I recall saying that viewing a hidden input value may present a security risk, so why doesn’t the same apply to the session_id? There are three reasons. First, Qilan creates machine unique session_ids; second, the id itself is internally protected against attempted hacking; and third, session_ids expire. Don’t get me wrong, nothing is perfect.

Session files are created or updated when one or more Framework fields are denoted as ‘session variables’. If a Framework contains fields marked as ‘session variables’, the loading of any WebTemplate in that Framework will trigger the creation or updating of session files.

Qilan will create new session files if the session ID does not exist in the URL string, otherwise the existing session file will be updated.

Qilan will only update unexpired session files.

Session information pertains to the site, not just the page. By this, I mean that you will want to establish a session file as every user accesses your site, then keep updating that file until they leave or the file expires. To accomplish this, create a site ‘portal’ page. This page can be used for login, site introduction, and table of contents or just about anything else. The key is to funnel users through this page before any other page is accessed. This page is then used to create the session file. From here on, the browser will use a session ID and display it in the URL string.

On your portal page, use a BASE tag to set an absolute path. This path will then be prepended to all subsequent requests for WebTemplates. Note how the path incorporates the environmental variable, SESSION_ID.

```
( "http://192.168.1.3/cgi-bin/qilan.cgi/" followed by "." followed by env_session followed by "/" )
```

The abacus, ‘env_session’ gets the environmental variable, SESSION_ID.

```
( environment "SESSION_ID" )
```

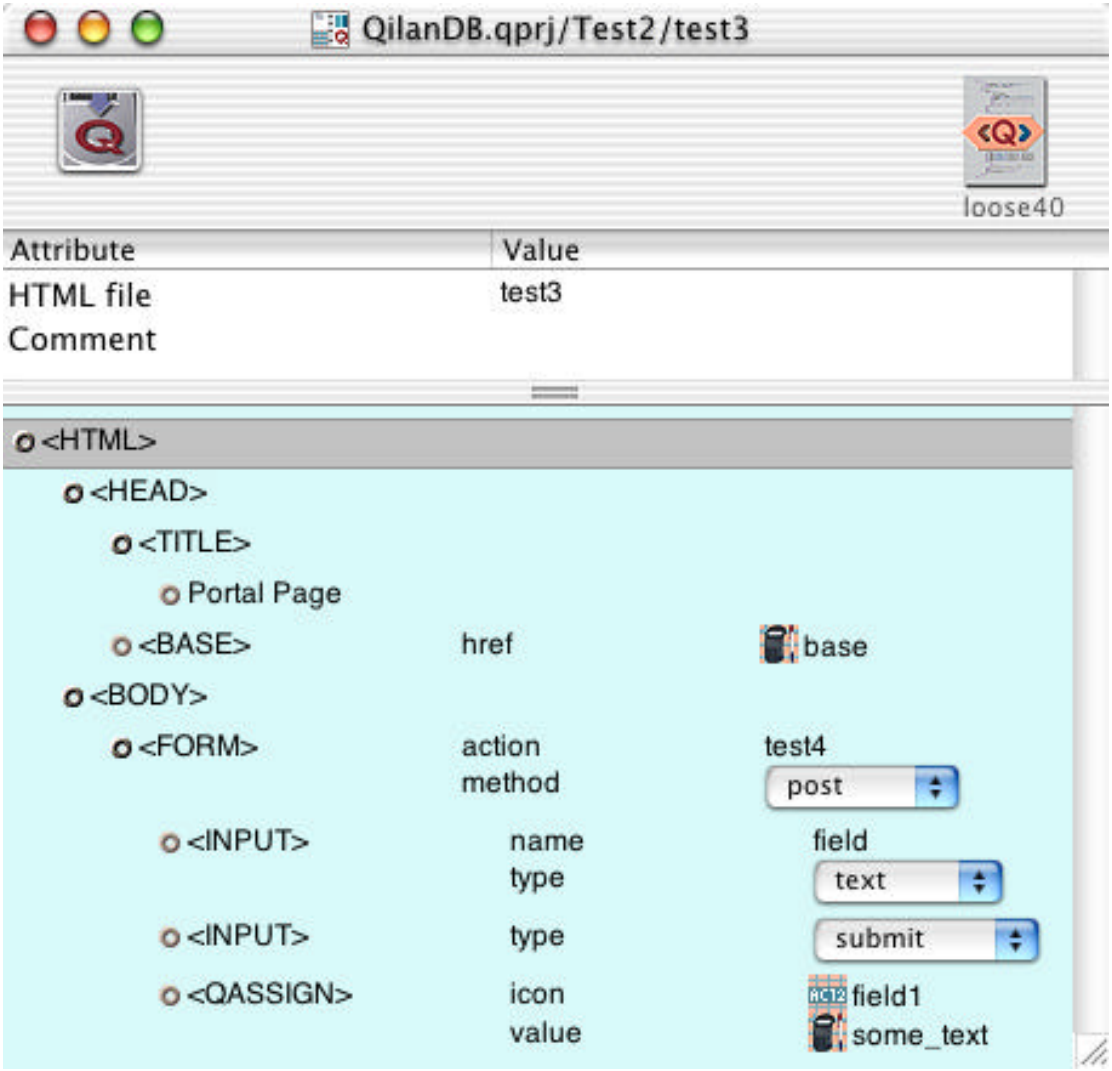
The BASE tag will then be generated as follows:

```
"192.168.1.3/cgi-bin/qilan.cgi/.V3PfjYPsa3EB22chGRuoYPfjYPsa3En/"
```

You might wonder why an absolute path and not a relative one? The reason is that the browser is responsible for using the BASE path, and not all browsers interpret the BASE path the same way. Your situation may be different, however.

Once you set the BASE, all future references to WebTemplates need only refer to their location relative to the BASE path. This applies to the FORM action attribute as well as links.

The following WebTemplate shows the basic elements of a 'portal' page. Note the BASE tag – the "base" abacus is used by the 'href' attribute. The FORM tag's action attribute will post the values from this page to the form, "test4".

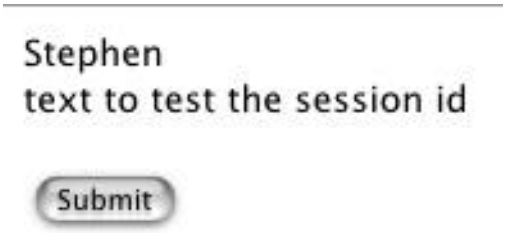


What's the QASSIGN for? Field1 is denoted as a 'session variable'. When this form is submitted, the value assigned to field1 (some_text) will be placed into the session file. The form 'test4' will retrieve this value even though the value of field1 is never actually submitted.

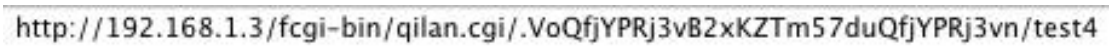
Test3 appears on the browser as follows:

A name has been entered, but the value assigned to field1 originates from the abacus, ‘some_text’.

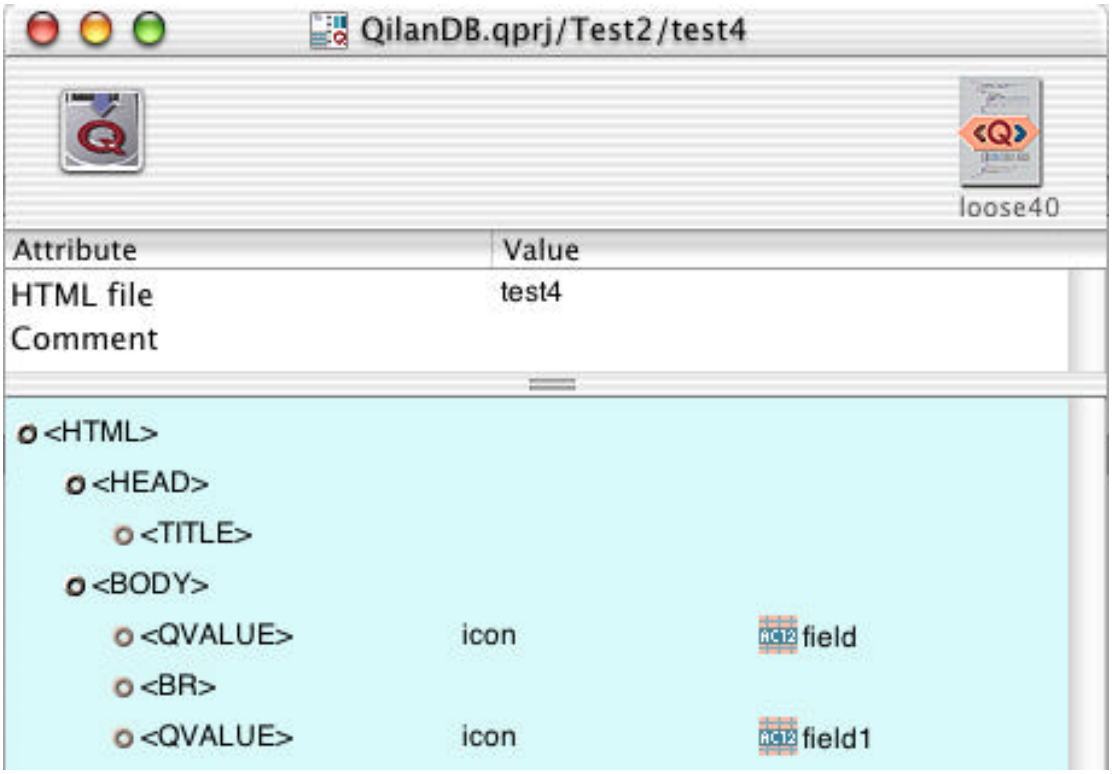
When we click the submit button, the following screen appears:



Note how both the input and assigned values are retrieved. Qilan knows where to get the values for the page, ‘test4’, because the URL contains the session ID (see below).



The WebTemplate, ‘test4’ follows. Only QVALUE tags are required to display the data.



Session files are accessible until they expire or are deleted. The expiration date/time is updated each time the session file is accessed. An automatic expiration length can be set in Project Settings; the default is 60 minutes.

Session files are NOT automatically deleted. They must be either manually deleted or removed by using a system script.

Expired session files must be deleted or they will quickly accumulate and cause system difficulties.

You might wonder why Qilan just doesn't delete session files for you automatically. The reason is really quite simple – it's your machine! Mac OS X uses a complex scheme of user permissions and file access mechanisms to maintain file integrity and system security. In order for Qilan to delete session files, Qilan would have to access to the root user, not a good thing.

Now, let's make this as painless as possible.

OS X incorporates an executable function known as 'cron'. Cron will run scripts on a timed basis, from every minute to once a year. Using a timed script to delete expired session files, let's say once a day, we can ask the system to perform session file maintenance.

QilanSessionReaper

Installation of Qilan installs the file, “QilanSessionReaper” (QSR). When this file is executed or run, expired session files will be removed. The file is placed in /Library/Qilan/bin/.

QSR has one optional commandline parameter, [-v]. If specified, a message is put into the log file documenting how many files were (a) successfully deleted, (b) unsuccessfully (not) deleted, or (c) not expired.

Available on the Qilan website is a small utility program, CronniX. CronniX is an easy to use scheduler that allows you to run scripts at certain times or intervals. It's a very powerful tool. Please refer to the enclosed ReadMe for installation and operational instructions or visit the web site at: http://www.koch-schmidt.de/cronnix.

CronniX will request you type the script you want to run. Type the following:

/Library/Qilan/bin/QilanSessionReaper

Using the CronniX interface, set how often you want QSR to be run, then save your settings. A recommended interval is daily for low volume sites or hourly for more active ones.

Record Locking

"Locking" is a term that denotes a technique for preventing users from overwriting other users' work. Qilan supports three ways to implement locking: optimistic, pessimistic and locking on a column. All have different strengths and weaknesses. You may use a combination of these approaches in your designs.

Optimistic Locking

Utilizing optimistic locking, database records are never actually locked. When the record is first read from the database, Qilan makes a snapshot of all the fields checked as ‘locking variables’. Before the data is updated, the snapshot and fields comprising the snapshot are compared and if they are not the same, the update fails.

The advantages of optimistic locking are the following:

- All databases support optimistic locking.
- Optimistic locking is easy to use.
- Optimistic locking does not use any extra database resources.

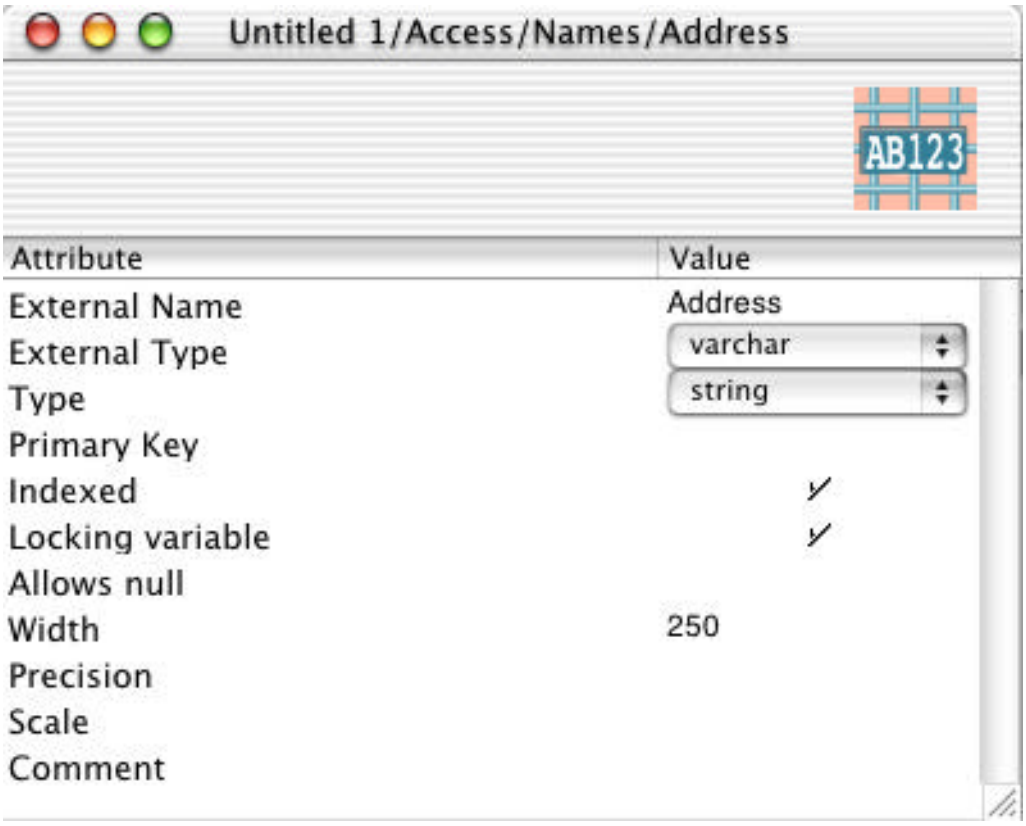
The disadvantages of optimistic locking are:

- Users are not notified that someone else modified a record until they try to update it.
- Optimistic locking slows down updates.
- All applications or multiple projects that update a database table must agree on the fields to lock on.

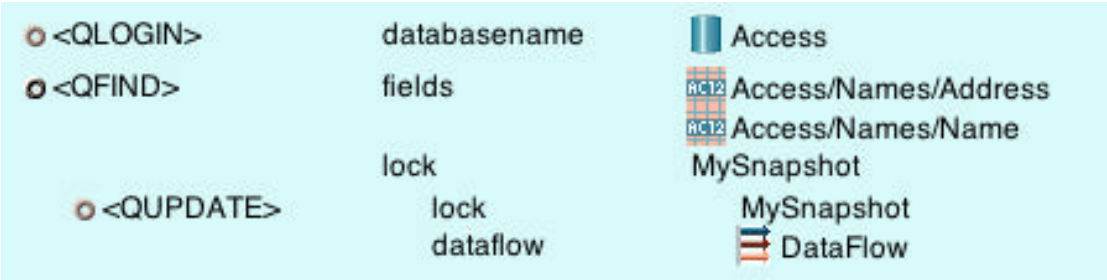
When records are retrieved using a QFIND with the ‘locked’ attribute, a snapshot is named, created and maintained locally on the server. QUPDATEs can reference these snapshots, thereby determining whether intervening modifications have occurred.

Qilan stores the ‘snapshot’ in the session sever. Insure you have correctly configured and are using the session server before implementing record locking. An incorrectly configured session server will result in undefined values following the QUPDATE.

The first step is to denote one or more fields as a ‘locking variable’. This is performed in the Access > Table > Field dialog as shown.



You may include or exclude as many fields as you desire for the record snapshot. Snapshots are created when the record is retrieved (with a QFIND). They are referenced (to insure they have not changed) by the QUPDATE.



The QFIND lock attribute names the snapshot. The UPDATE lock attribute retrieves a snapshot for a given name. In this example, the field, “Name”, is used to create a snapshot (MySnapshot) when each record is retrieved.

Pessimistic Locking

Qilan implements pessimistic locking at the table level. Locking occurs when the SQL isolation level, 'serializable' is chosen (See QPROCESS). Depending upon the database executing a QFIND, QUPDATE, QDELETE, etc., on a locked table, will fail or be blocked until the table is unlocked.

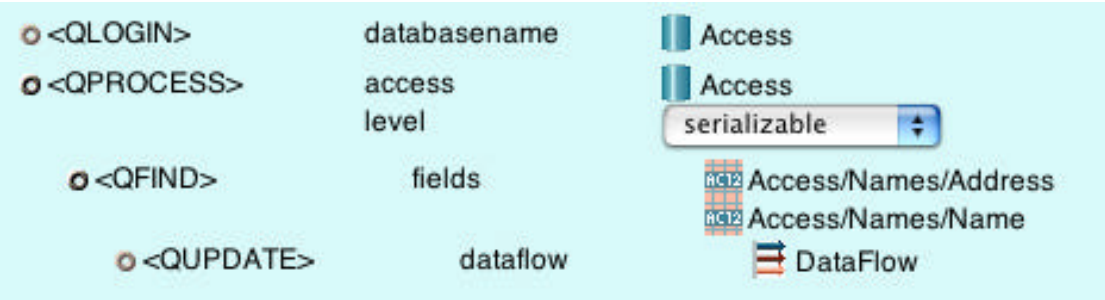
The advantages of pessimistic locking are:

• Pessimistic locking can prevent users and other applications from reading data that is being changed.
• Users are notified immediately if they cannot access a database row.
• Pessimistic locking is easy to use.

The disadvantages of pessimistic locking are:

• Not all databases support SQL serialization and the databases that do support it may do so differently.
• Pessimistic locking uses extra database resources.
• Pessimistic locking can prevent other users from having read-only access to data.
• Pessimistic locking can cause deadlocks.
• Pessimistic locking can cause excessive locking.

The implementation Pessimistic record locking is shown below. Tables referenced by QFIND, QUPDATE or any other tags will be locked until all tags enclosed by QPROCESS have been executed.



Please note, not all databases support the 'serializable' SQL isolation level. Refer to your database documentation.

Locking on a Column

One effective way to implement on-demand locking is to add a locked column to a database. The Qilan designer can add a single column to a table and then change a value in the column to indicate whether a record is locked. This locked column can hold anything from a simple boolean value to a user's name and a time stamp.

The advantages of locking on a column are:

- This method lets the user or logical design determine when a record should be locked.
- This method enables users to see who locked the record and when it happened.
- You can give authorized users the capability to override the lock.

The disadvantages of locking on a column are:

- This method allows users to lock a record and keep it locked indefinitely.
- If the database connection is lost, the lock is still enabled.
- Extra database space is needed for the column.
- An extra transaction may needed to lock/unlock the record.

SQL-92 Isolation Levels

A corollary to the concept of record locking is the determination of the access method, in other words, the *read* mechanism. When using an access method, it is important that the user know the degree of isolation supported between processes within the same WebTemplate or amongst different users attempting to access the same record data. This is commonly defined by referring to the SQL-92 Isolation levels. Qilan supports all four SQL isolation levels for processes within the scope of a QPROCESS tag.

SQL-92 defines four Isolation Levels:

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable

Isolations levels are stated in terms of three prohibited operation sequences, called *Phenomena*:

- Dirty Read
- Non-Repeatable Read
- Phantom Read

If a database does not support Serializable, then it is possible that the record data may change within a transaction because of an update by another transaction. Phenomena define the ways in which the record data may change during the transaction.

When an SQL Isolation level is not defined as a QPROCESS attribute, the database default will be used. Refer to your database documentation for more details.

Dirty Read

This occurs if one transaction can see the result of the action of another transaction before it commits. Consider the following example:

```
T.1       Write                          Rollback
T.2                    Read(a)
```

If T.2 reads the value written by T.1, then a DIRTY READ has occurred.

Non-Repeatable Read (also called Fuzzy Read)

This occurs if the result of one transaction can be modified or deleted by another transaction before it commits. This is illustrated below:

```
T.1        Read                             Read(a)
T.2              Write/Delete/Commit
```

If T.1 gets a different result from the each Read, then a NON-REPEATABLE READ has occurred

Phantom Read

This occurs if the result of a query in one transaction can be changed by another transaction before it commits. This is illustrated below:

```
T.1       Select                           Select
T.2             Update/Create/Commit
```

If T.1 gets a different result from each Select, then a PHANTOM READ has occurred

Why is this important? For the most part, this stuff is very esoteric, but when your application starts doing weird stuff like returning wrong data you might be more interested. Isolation levels apply to the time between a request for data and the retrieval. If your application has a small number of accesses, isolation levels may be moot, but as the number of accesses increase, you should try to use *repeatable read* as often as possible. Data integrity is a good thing.

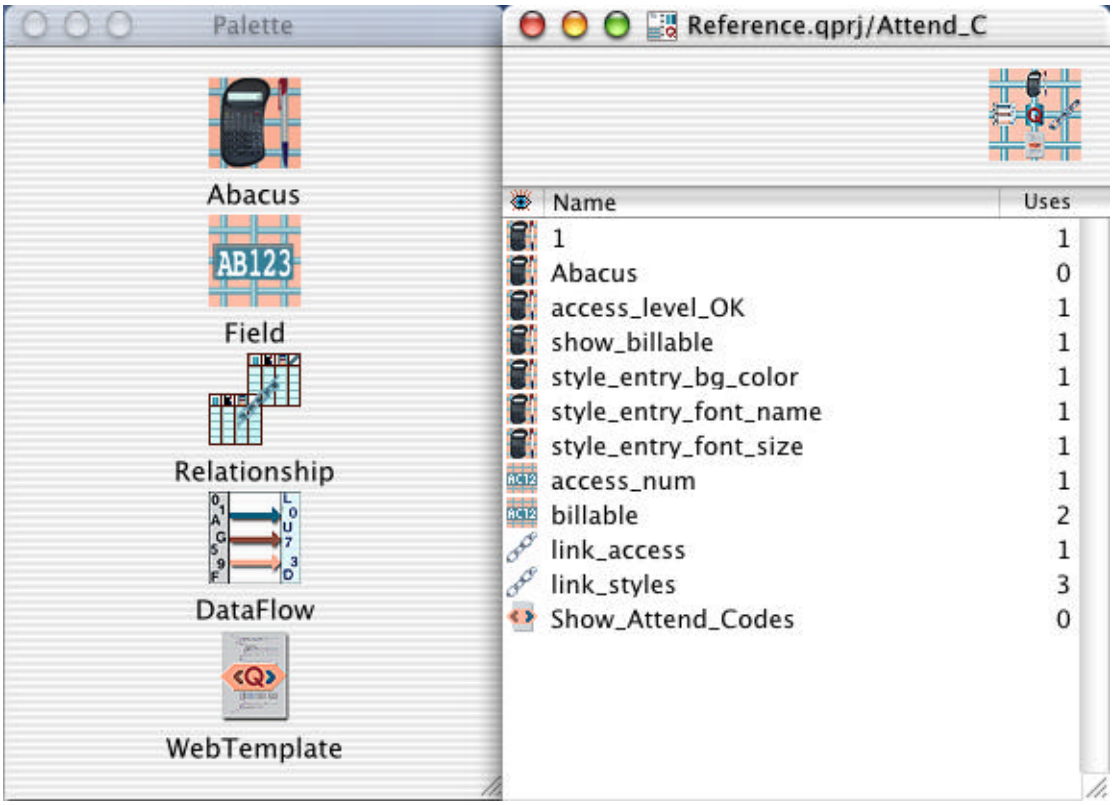
Designing Abacus Expressions

The abacus translates user logic into a form databases can understand. In most cases, this is SQL (Structured Query Language). Qilan uses the SQL-92 standard.

Databases which are not fully SQL-92 compliant may not work properly or return SQL errors with some abacus constructions. If this occurs, you can write your own SQL expression using the 'query' attribute for the QGROUP or QFIND tags. Otherwise, we encourage you to use the 'queryicon' attribute, which references abacus expressions directly.

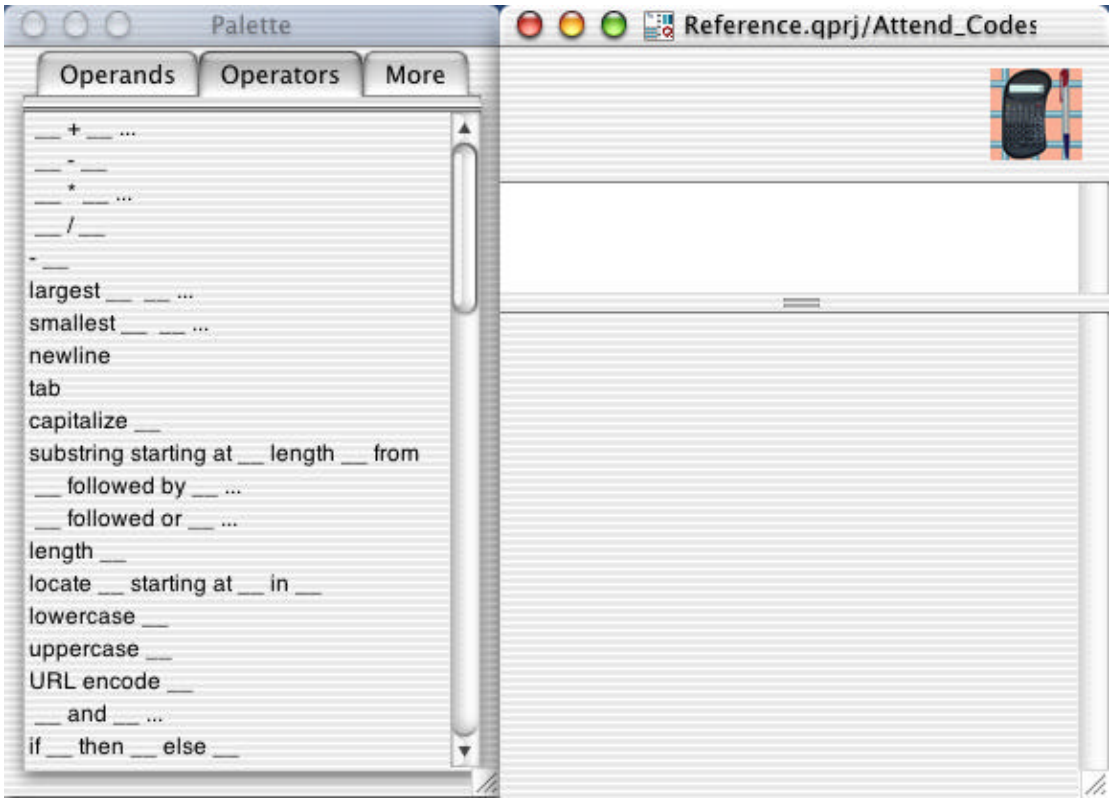
Building Abacus Expressions

Abacus icons are available in the Framework or Access > Table window. Drag an abacus from the palette, press the option-command-A keys or select Icon > New > Abacus from the Icon menu.



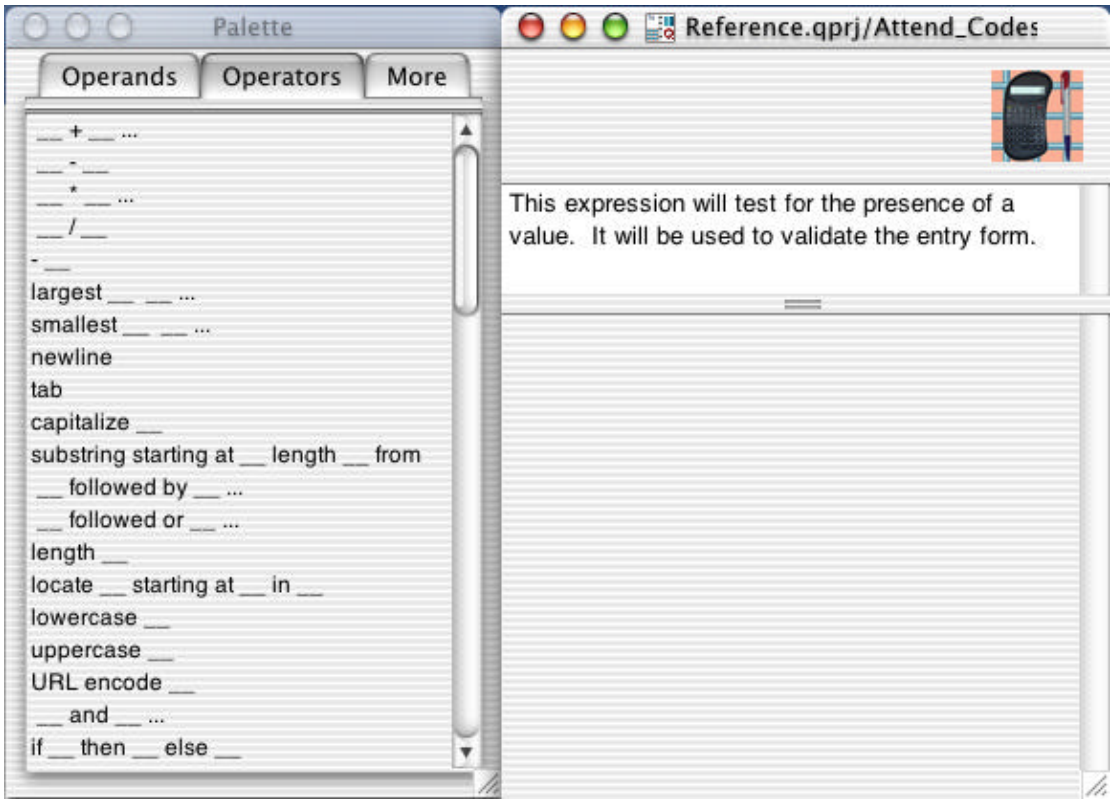
Name the abacus by highlighting its default name, then retyping. Within a Framework or Table window, abacus expressions must have case sensitive unique names. Abacus expressions will be grouped together, ordered alphabetically.

Abacus expressions are built inside an abacus window. To open the abacus window, double click on the abacus icon. This will open the abacus window and display related palette options. If the palette is not open, select Window > Open Palette.



The abacus window (above, right) is composed of two panes: Comments and Expressions.

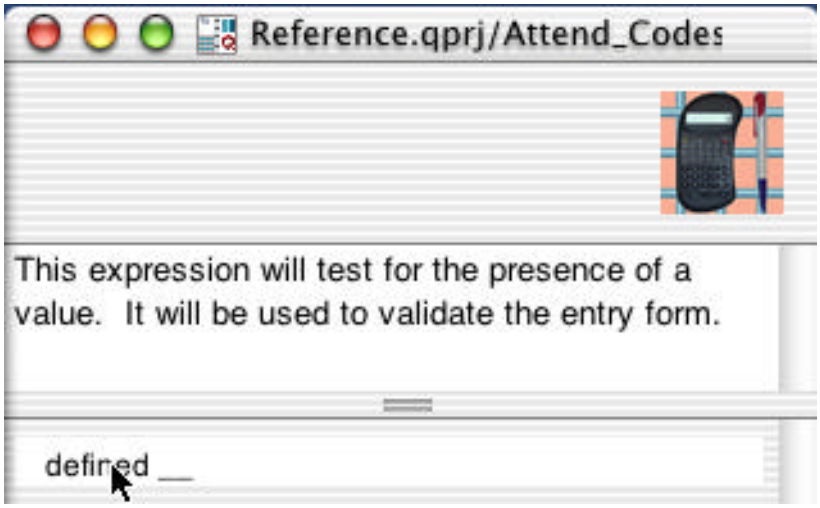
The upper pane may be used for text comments. Place the cursor in the upper pane and click. The text insertion cursor will appear. You may enter as much text as necessary. When the abacus window is closed, the text will be saved.

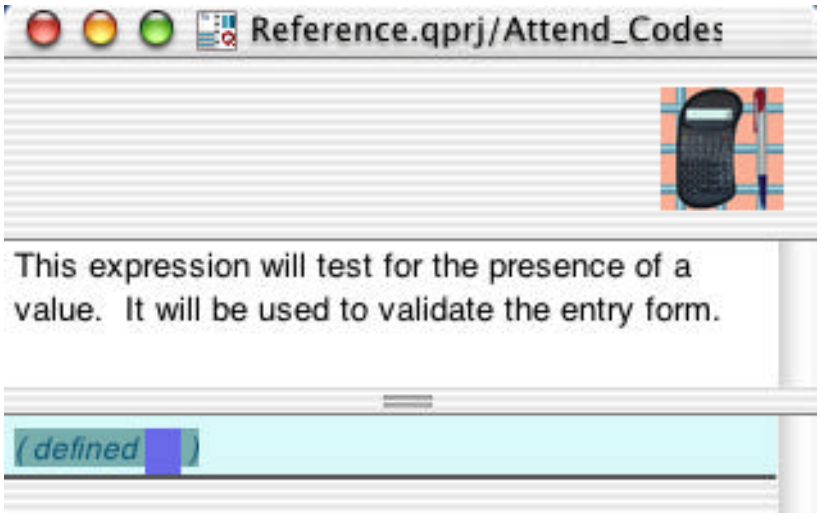


The lower pane, initially appearing empty, displays and allows manipulation of the expression.

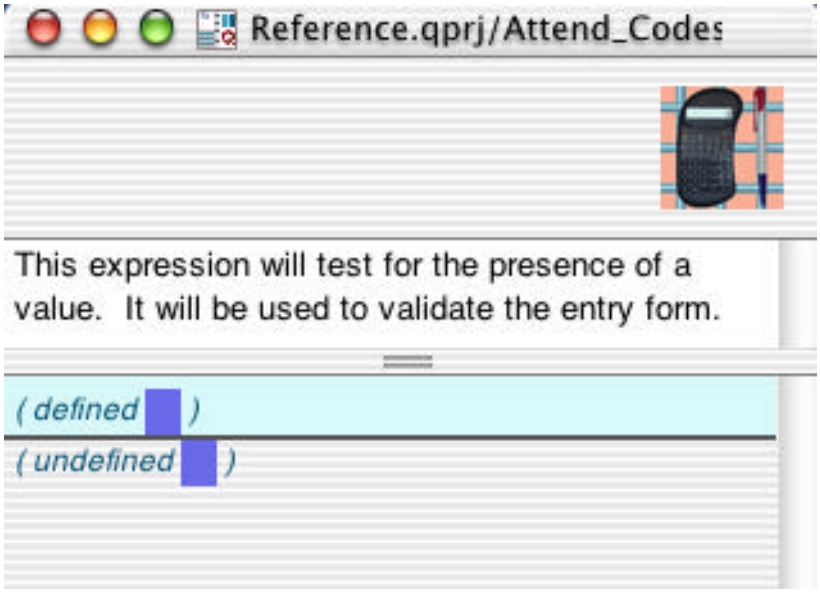
To create an expression, click the ‘Operators’ palette tab, then drag an operator from the adjacent palette to the upper portion of the lower pane.

The line will turn white when it is properly positioned. Release the mouse button and the operator will appear in the window.

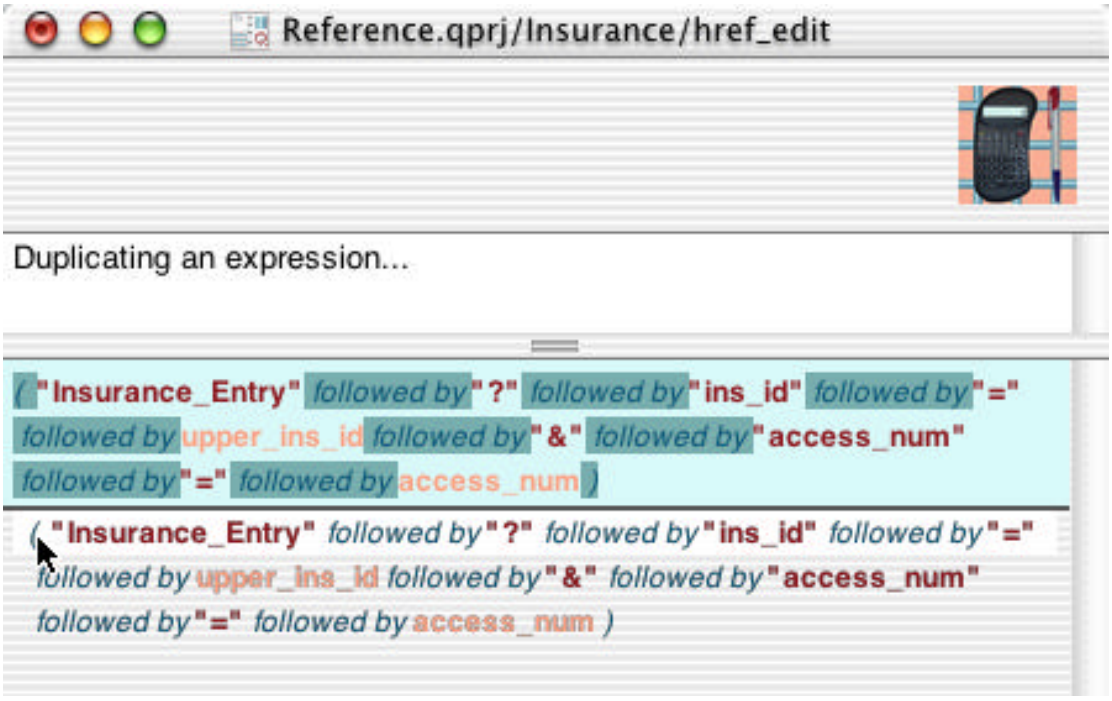




Following the placement of an operator, a bold line will appear. This line separates active (top) and inactive (bottom) expressions. The active expression outputs the result; the inactive expression is retained for reference, but not used. This special Qilan feature allows expressions to be tested and manipulated without changing the original design.



To create an inactive expression, highlight the active expression, then press command – D from the Icon menu. A copy of the expression will appear in the inactive area. You can also highlight any section of the active expression and drag it to the inactive area.

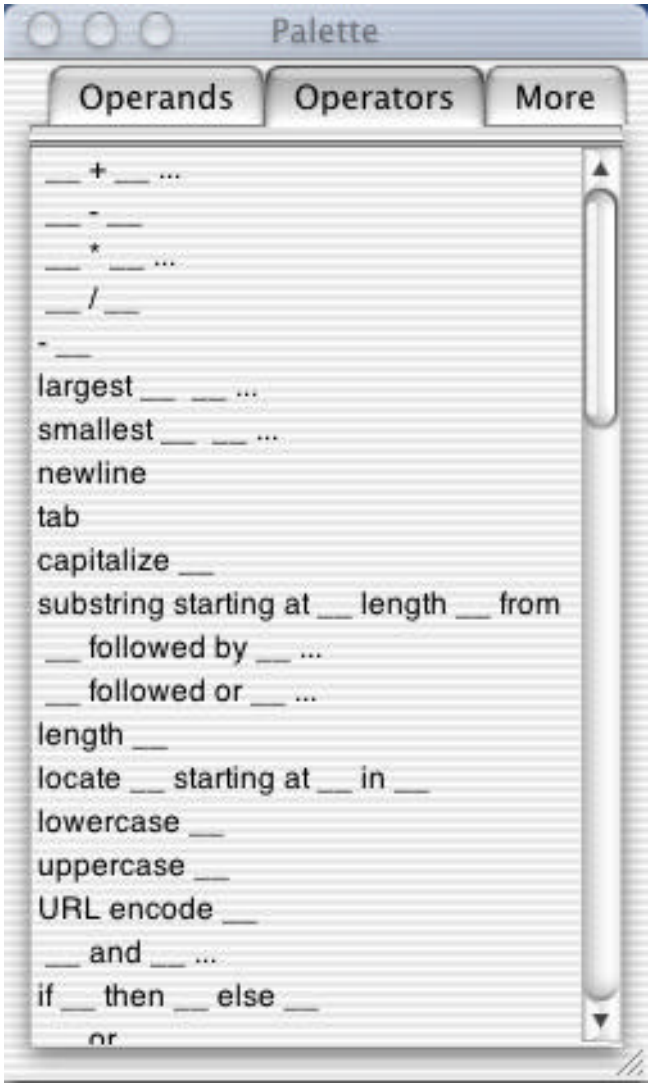


The Abacus Palette

The abacus palette, used to construct abacus expressions, contains three tabs: Operands, Operators and More.

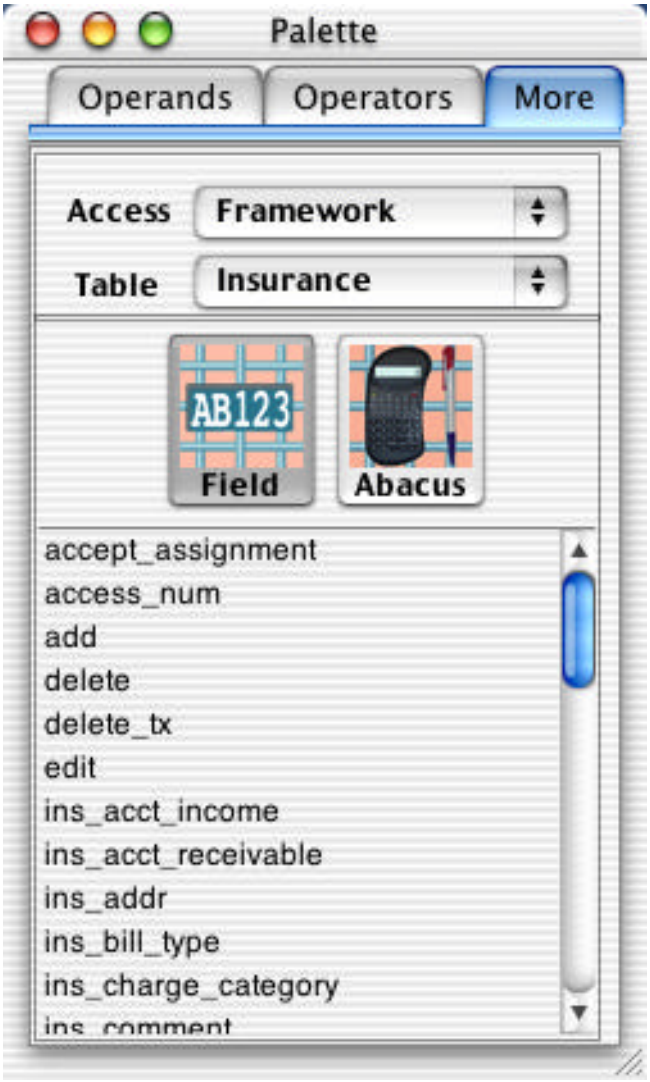
Operands consist of fields, abacus expressions and relationships. Operands shown on the palette will be those from the same Framework or Table as the abacus itself.

Operators consist of functions (Please refer to the Abacus Reference for complete details).

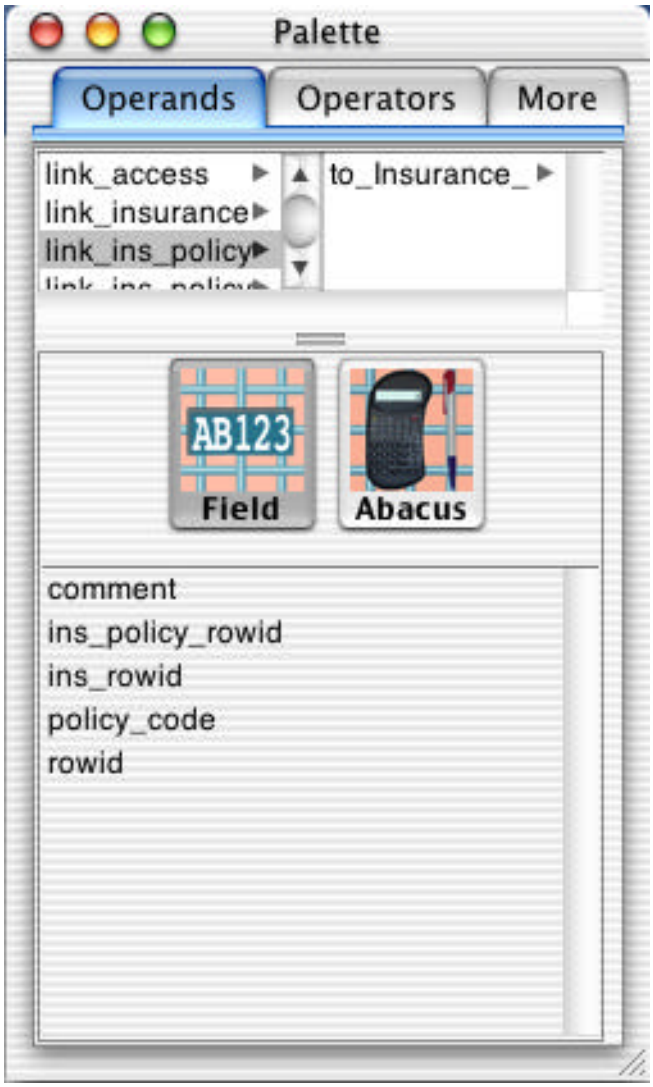


The ‘More’ tab is used for the operators, “Acquire __” and “Acquire __ From __”. Fields and abacus expressions can be from any Framework or Table in the Project.

Using a value from outside the current window is a bit difficult to understand. Qilan values icons when a WebTemplate is processed, but there is no restriction on where values are obtained during this processing. Therefore, an abacus may ‘acquire’ the value of another field (source value from another Table or Framework), *so long as the source is valued on the WebTemplate*.



The Operands tab allows the designer to choose a relationship. Existing relationships are displayed in the upper portion of the palette window. Selecting a relationship changes the fields and abacus expressions to that of the relationship's target table.



The selection of a field or abacus is then interpreted as, "get me this value based on the relationship link (source = target)".

The syntax uses a dot notation, "relationship.value".

Relationships can be cascaded through many tables. If you select a target table, which contains relationships, those relationships will become available in the viewer at the top of the palette.

How far can you go? If you attempt to perform circular relationships for example, Qilan will likely report an error ("too complex"). The more relationships that need to transversed, the slower the data retrieval process will become. If you find yourself going beyond three or four relationships, you should carefully review your design.

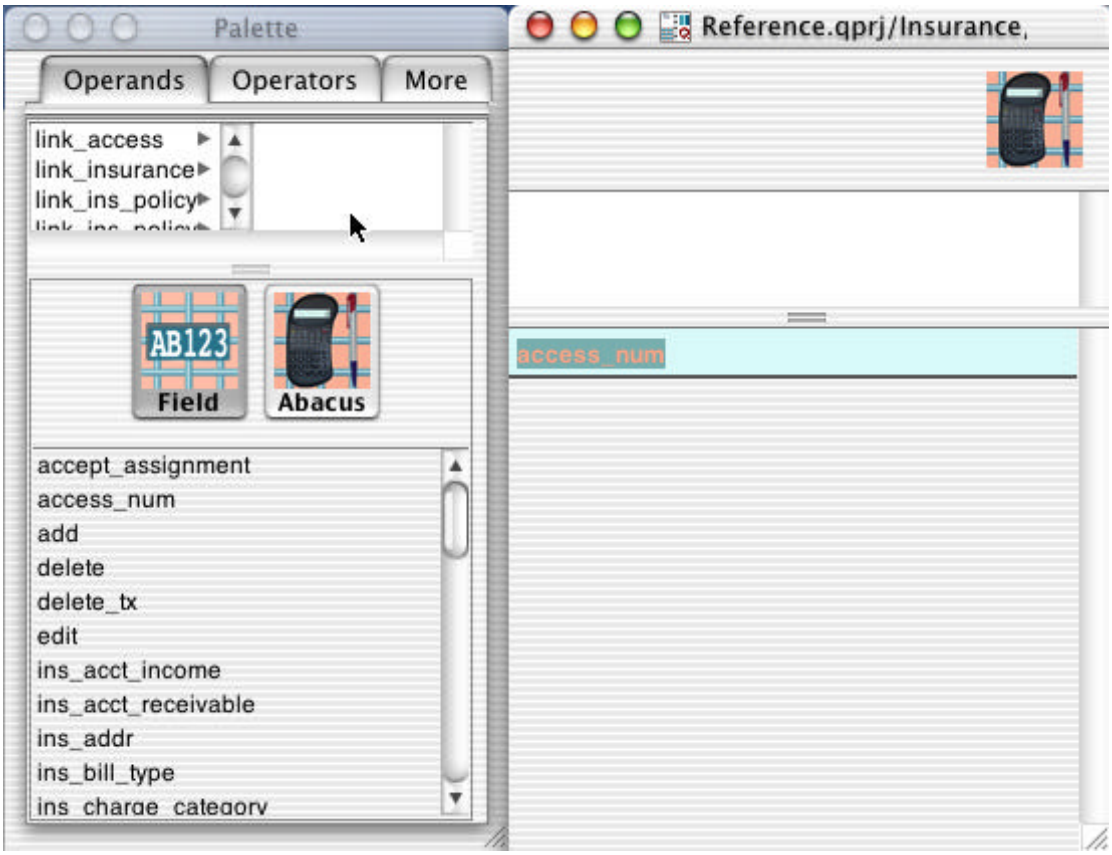
Two points to consider… the relationship semantic and type. If the relationship is located in a Table, the semantic can have dramatic consequences. Always try to limit relationships to 'inner joins' when possible.

Relationship types that are one-to-one will return a single value. Relationship types that are one-to-many will return the first value found. Designers need to be aware of the one-to-many relationship because unexpected results may occur.

You can deselect a chosen relationship by dragging the highlighted relationship name down to the bottom of the list and then releasing the mouse button. This will change the Operand list to the current Framework or Table window.

Creating an Expression

Dragging operators and/or operands into the abacus window automatically creates an expression with a result. For example, suppose the operand field, “access_num” was dragged into the abacus window, the result would be the value of “access_num”.



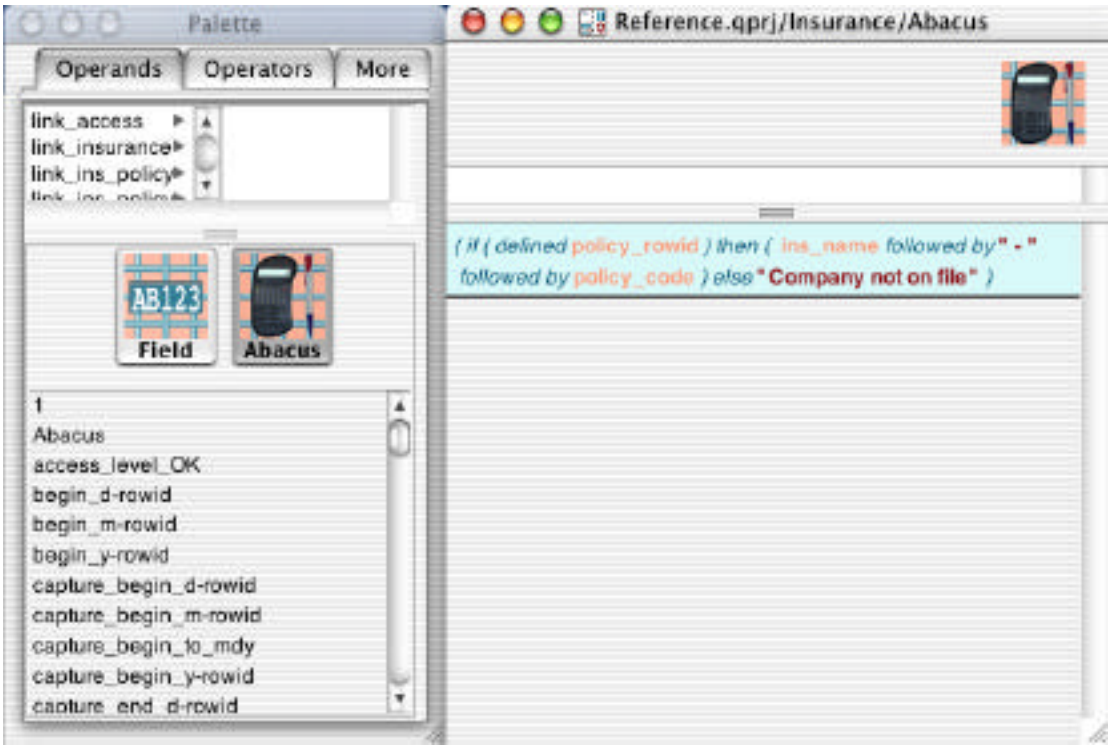
The selection of operators allow data to be manipulated or derived. For example, the operator “__ followed by __” concatenates values. The use of “__ and __”, “__ or __” and “if __ then __ else __” operators enable logical comparisons. Operators may be used within other operators.

In the example below, the output would be, “Steve’s Insurance – SI” when policy_rowid is defined, otherwise the message, “Company not on file” will be shown.

```
( if ( defined policy_rowid ) then ( ins_name followed by " - " followed by policy_code )
 else "Company not on file" )
```

Field values or the result of other abacus expressions may be used in the creation of new abacus expressions. When the abacus window is first opened, the listed icons available will be from the table in which the abacus was created (source).

The name of the abacus you are creating will be available for selection in the abacus list. This allows recursion. Be careful not to select this abacus unless you intend to create a recursive operation.



Note how the name of the abacus icon, “Reference.qprj/Insurance/Abacus” also appears in the Operand list of abacus icons. If you mistakenly select the Operand, “Abacus” and use it in itself, Qilan will report a ‘recursion error’.

Some icon names, when viewed in the abacus window, have funky names. Icon names containing a '/' or '.', will have those characters replaced with their hexadecimal equivalent when shown in the abacus window.

Period (.)	%2E
Slash (/)	%2F

The reason is simple; these characters have special meaning. Recall that Qilan is designed for use on the internet and therefore must be compliant with special protocols and character interpretations.

Let's build an expression that evaluates a user's input and returns a result to be stored in the database. Our logic must first determine if the value is valid, and then test for the correct format. If this is checks out, then we want to encrypt the input with a user defined key.

We will begin by showing the final result:

```
if ( ( defined input ) and ( input like "___-__-____" ) and ( defined key ) and
( ( length key ) >="8" ) ) then ( ( text input ) encrypt with key ) else
"You must enter you data and enter a key" )
```

Taking this apart, we start with the 'if then else' operator dragged from the palette list of Operators.

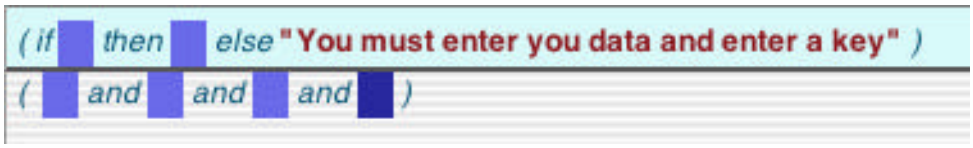
```
( if  then  else  )
```

Note how the parameters are empty and darkened. It's easy to know what to say when something is wrong, so let's type the error message first. Double click in the 'else' parameter'.

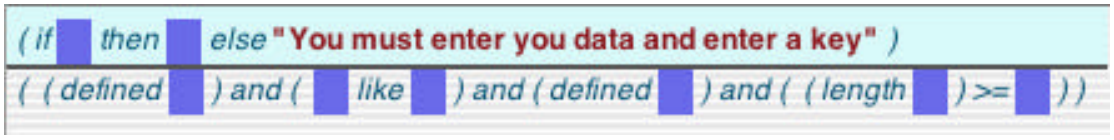
The parameter ‘hole’ now displays a double quote. A flashing cursor will appear between the quotes. This is your indication to begin typing. When you are finished, just click anywhere else in the abacus window.

The quotes surrounding entered text is not part of the literal output. For example, the abacus appearing as, (text “steve”) would actually output, steve – without the quotes. If you wanted to quote the text itself, then insure the text is quoted, as follows: (text ““steve””)

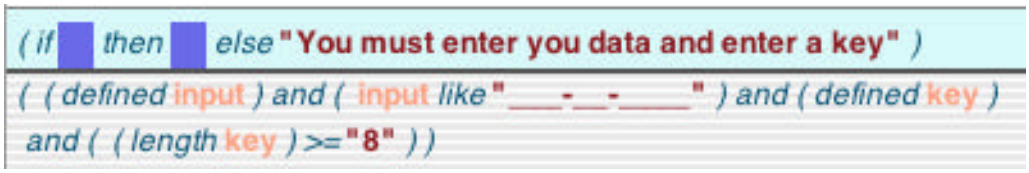
Our next step is to actually create the validation logic. There are four items we will need to validate, so let’s create an expanded ‘and’ operator to accommodate them. Drag the ‘and’ operator to the inactive area until the line turns white, then release the mouse button. It is often more convenient to use the inactive area for composition and then move the completed expression.



Now, we can drag in the abacus operators we will be using to evaluate our data. Drag an abacus operator over an empty parameter hole until it highlights, then release the mouse button.

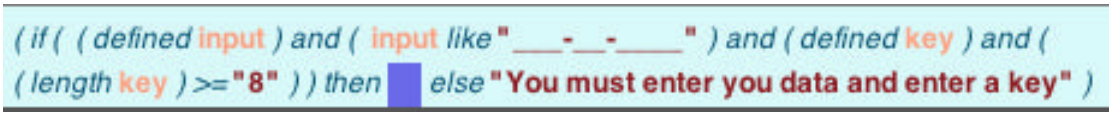


Let’s complete this logic by dragging in the fields we will want to evaluate. Then complete the parameter logic as required by ‘like’ and ‘length’.

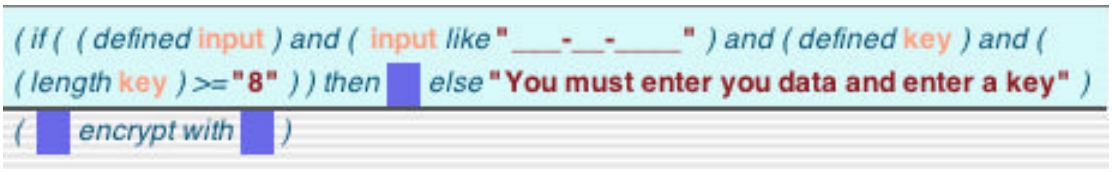


What we want to insure is that the input data is defined and that it is in the pattern of ###-##-####. We also want to insure the key is defined and at least 8 characters in length.

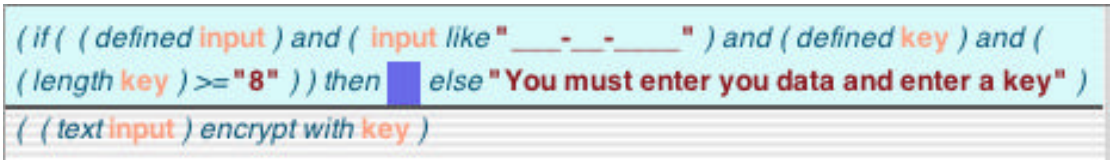
Now that the logic is complete, highlight the outermost parenthesis and drag the entire statement to the ‘if’ hole until it highlights; then release the mouse button.



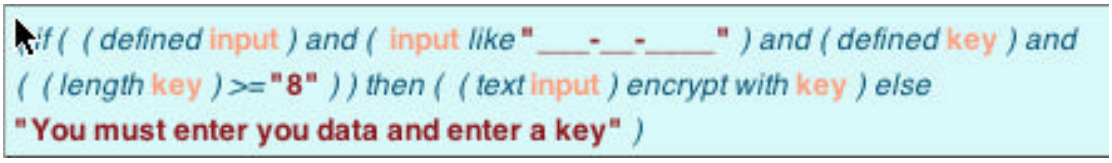
Almost done. The last thing we have to do is to deal with the input data and key when everything is done correctly. Always the hardest part! We want to encrypt the data. Return to the palette and drag the ‘encrypt’ operator to the inactive area.



The ‘encrypt’ operator accepts strings, so to avoid the potential for errors, let’s convert the input field value to text. We do this by dragging the ‘text’ operator into the first parameter of the ‘encrypt’ operator. Now we are ready for the user data.



Finally, highlight the outermost parenthesis of the inactive expression. Click and hold the mouse button. Drag the expression over the 'then' parameter until it highlights, then release the mouse button. Now our expression is complete.



At any time during the construction of the expression, you can remove unwanted operators or operands by highlighting the element then pressing the delete key or selecting 'Cut' or 'Clear' from the Edit menu. Furthermore, expression elements can just as easily be duplicated or copied.

Transferring Expressions between Abacus Windows

Expressions created in the abacus window can be dragged to other abacus windows. To do this, open a new abacus window while the window containing the abacus expression is open. Highlight the outer parenthesis then drag the expression to the new abacus window.

Empty Abacus Expressions

An empty abacus (one without any operands or operators) can be saved and used in other abacus expressions or constructions. However, if an empty abacus expression is used or referenced by a WebTemplate 'Q' tag, qilan.cgi will report the following error, "Attribute Required".

Recursion Limit

Abacus expressions can use abacus results produced by other abacus expressions. This design feature allows the designer to reuse abacus expressions and greatly simplifies constructions as well as permits considerable flexibility. When one abacus uses another abacus, Qilan must evaluate the first abacus before the result of the second abacus is determined. The number of abacus expressions that must be evaluated before the result is output is referred to as the recursive 'depth'.

Generally, a depth of '100' should be considered a reasonable design maximum. If abacus expressions do not refer to themselves (directly or indirectly), this limit will not likely be reached. However, when an abacus refers to itself (directly or indirectly) or has been inappropriately designed (circular logic), it is possible to reach this limit.

Qilan allows the designer to set a maximum recursive depth. When this value is reached, the engine, qilan.cgi will report an error to the user (via the browser interface).

To set a maximum recursion depth, select Edit > Project Preferences… Double click on the line labeled, "Recursion Limit". If no value or 0 is entered, there will be no limit. Enter any value you want, then tab. The default value is 100. The setting is saved when the project is saved.

If you set a recursive depth of '0', and you make a design error, Qilan (and possibly your server) will appear to hang and may even make it impossible for you to quit the process. We strongly suggest you set a recursive depth.

The WebTemplate

WebTemplates are used to create user interface forms and/or data processing mechanisms. They are created in a Framework.

WebTemplates have four primary functions:

WebTemplates format user and database data (input/output). Qilan defaults to using HTML, but potentially, any DTD can be used.

WebTemplates retrieve data base data. Data can be queried using a flag formatted abacus icons or with standard SQL. Fields or derived values are accessible as well as standard summary functions.

WebTemplates trigger the storage, updating or deletion of database data. Using relationships and dataflows, data can be created, updated, deleted or transferred from one database to another.

WebTemplates logically manipulate data and program execution. Using the wide variety of ‘Q’ tags, logical steps can be created to test, evaluate, loop or control procedural constructs.

Choosing a WebTemplate DTD

The multifunction design of the WebTemplate requires a bit of planning before we 'just create one'. WebTemplates are defined by a very specific format, which governs the legal placement of HTML tags as well as what tags and attributes are available. This format is known as the DTD (Document Type Definition).

When a WebTemplate is created, it conforms to the DTD as specified in the Project Settings. Once created, a WebTemplate DTD cannot be changed, although each WebTemplate can have different DTDs.

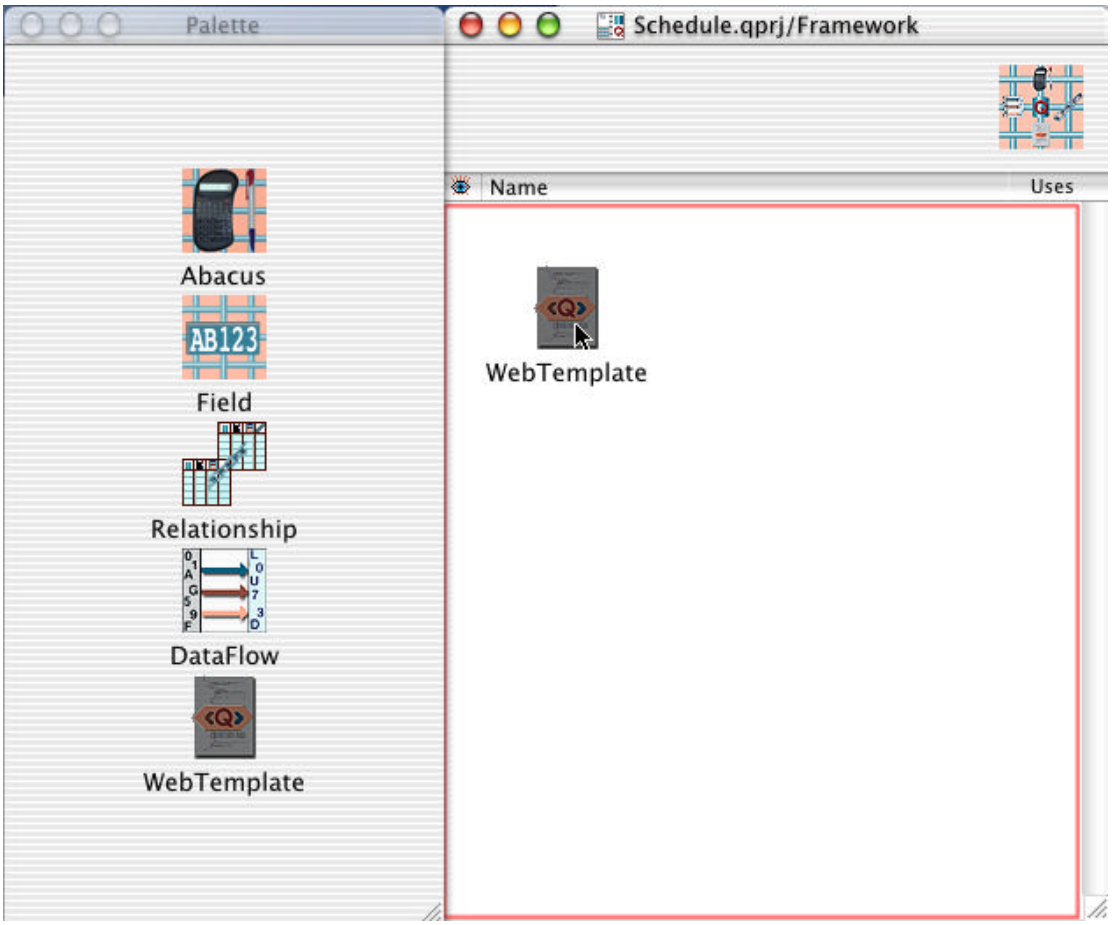
Qilan supports copy/paste WebTemplate elements from one WebTemplate to another, regardless of the DTD type.

How do you choose the correct DTD? Ah, the question of the ages! (Not really). DTDs define a set of tags, attributes and legal structure for the web designer. If you need to use frames, for example, then use the 'Frameset 4.0 DTD'. What you need to watch out for is that the browser (rendering application in technical terms) may not know what to do if it encounters specialized tags. Almost all browsers today know how to handle DTD 3.2. This is your common denominator. As time progresses however, the 4.0 DTDs are becoming the accepted norm. DTD 4.0 loose is my choice because of its backward compatibility with the 3.2 specification and inclusion of many new features.

Some databases, notably Helix, actually have their own DTDs. This is because specialized tags are necessary to handle data transport. If you are communicating with Helix, you will need to use a Helix DTD.

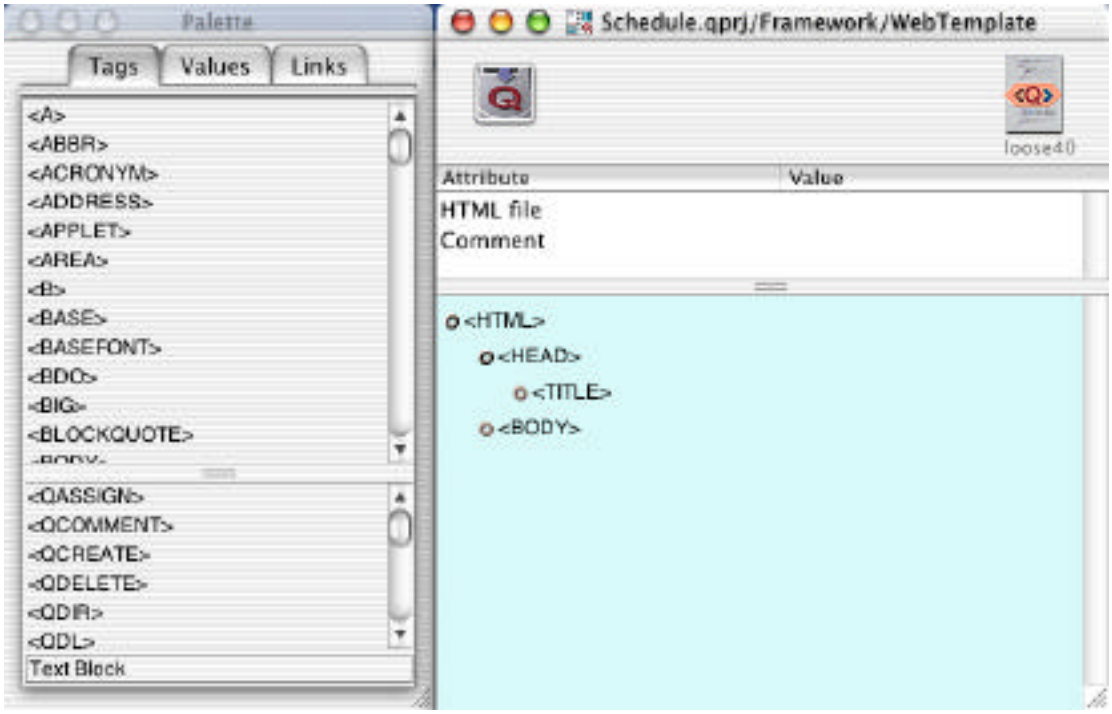
So, consider your data needs, design and likely audience when choosing a DTD.

WebTemplates are available in the Framework window. Open a Framework by double clicking on its icon from the Project window then drag a WebTemplate icon into the Framework window from the palette. Alternately, you can press command-option-W. Rename the WebTemplate icon by highlighting the default name and retype.



Frameworks may have an unlimited number of WebTemplates as well as share common icons, such as fields, abacus expressions, relationships and dataflows.

Double click the WebTemplate icon to open the WebTemplate window and display its palette. Note, if the palette does not appear, choose Window > Show Palette.



The WebTemplate window consists of an upper and lower pane. The upper pane has two lines for text entry: the first gives the WebTemplate an exported name and the second to enter text comments.

WebTemplates are designed and maintained using the Qilan developer, but not available for web access until they have been exported. The ‘export’ process creates a special file that is executed by qilan.cgi when requested by a web user.

The location of the exported file is defined by the ‘HTML File Root Directory’, in the Project Settings. So, if the root directory is /Library/Documents/, all the exported WebTemplates will be located in /Library/Documents/.

The name of the WebTemplate can be anything you want; suffixes are optional and need only be used if the web server requires a specific MIME type or the WebTemplate requires special processing. If you do not name the WebTemplate and choose, Export HTML, the WebTemplate icon name will be used with a .qprj suffix.

A word of caution, if all the WebTemplates are exported to the same directory, insure they all have unique names otherwise you may accidentally overwrite existing WebTemplates.

When you are ready to export the WebTemplate, choose Export HTML from the File menu. The WebTemplate window must be the active window for this command to be available.

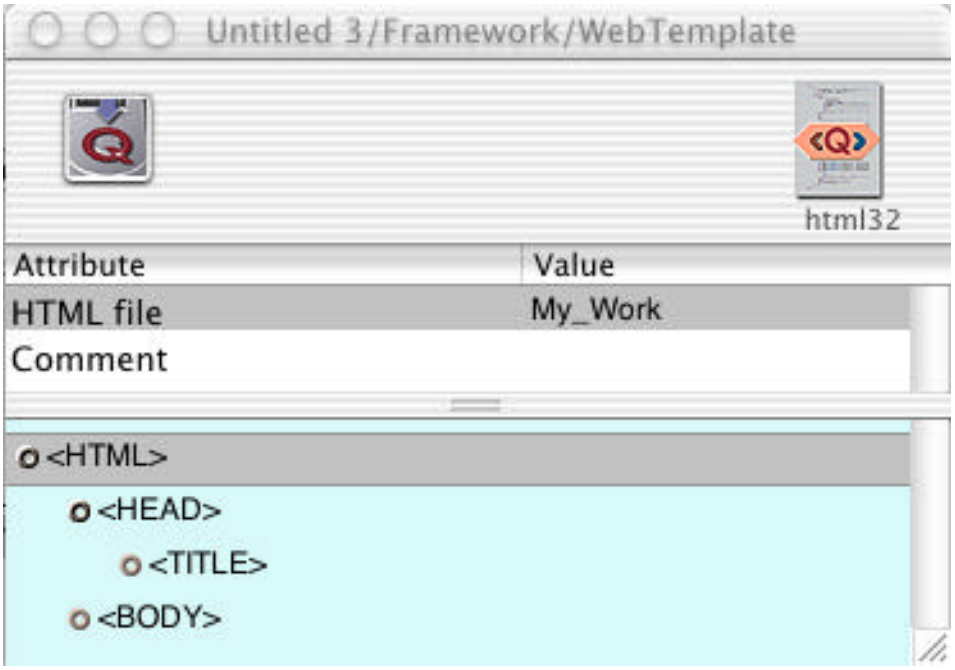
If you want to export all your WebTemplates at the same time, choose Export All HTML from the File menu. This command is always available.

Here's a hint… If your project gets really large, choosing Export All HTML can take a long time. Also, if you set the Project Settings to 'Export All WebTemplates on Save', the same time delay can occur each time you save the project. To help, only export the WebTemplate you are working on and deselect the option to 'Export All WebTemplates on Save'. At the top of each WebTemplate is an 'export' icon. The one with the 'Q' pictured on the hard drive. Click this to export.

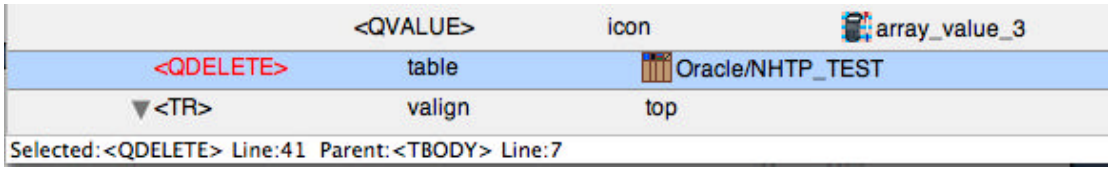
The second line in the upper pane is used to enter text comments. Comments will be saved when the WebTemplate window is closed.

The central portion of the WebTemplate window is where you layout your page. The section organizes all your 'background' construction (abacus expressions, fields, relationships, etc.) and creates a process. It is this 'process' that is executed by qilan.cgi. HTML formatting is fully integrated and/or controlled by this process.

When a new WebTemplate is opened, Qilan will automatically insert required HTML tags for the selected DTD.



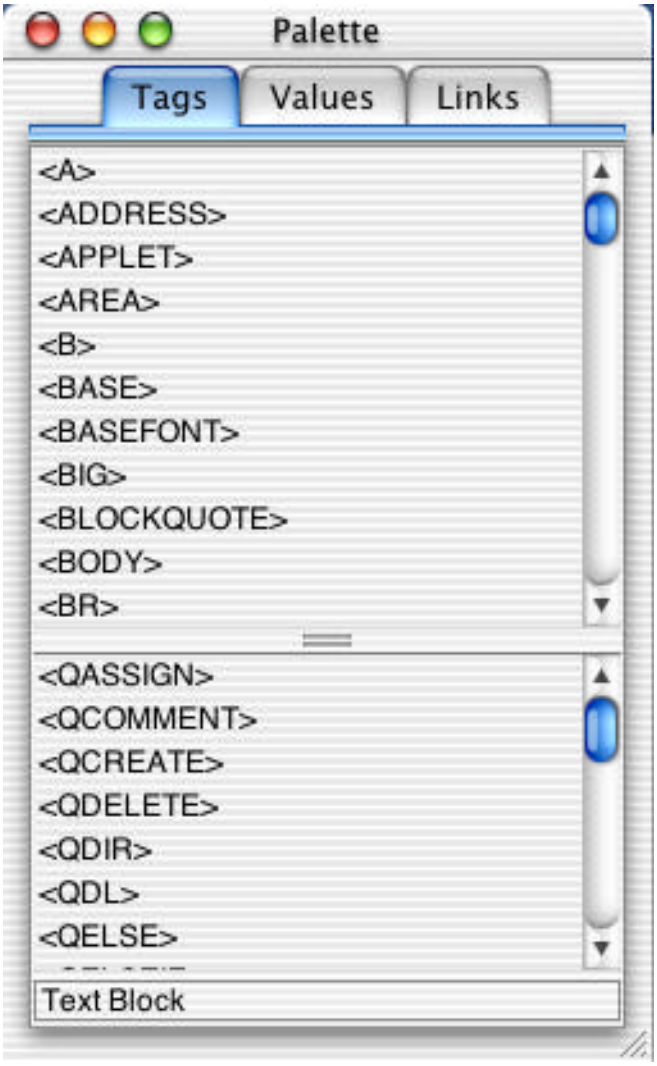
At the bottom of the webtemplate window is a status bar. When a tag hightlighted, the status bar will display the tag's name and line number as well as the name and line number of its parent tag. If the tag is dragged to a new location, the status bar will update to inform you of its new location and parent tag.



The WebTemplate Palette

When the WebTemplate window is opened, the palette will display three tabs: Tags, Values and Links.

The Tags grouping contains all the HTML tags valid for the DTD (upper portion), Qilan 'Q' tags, and a special tag for entering text, "Text Block" (lower portion).



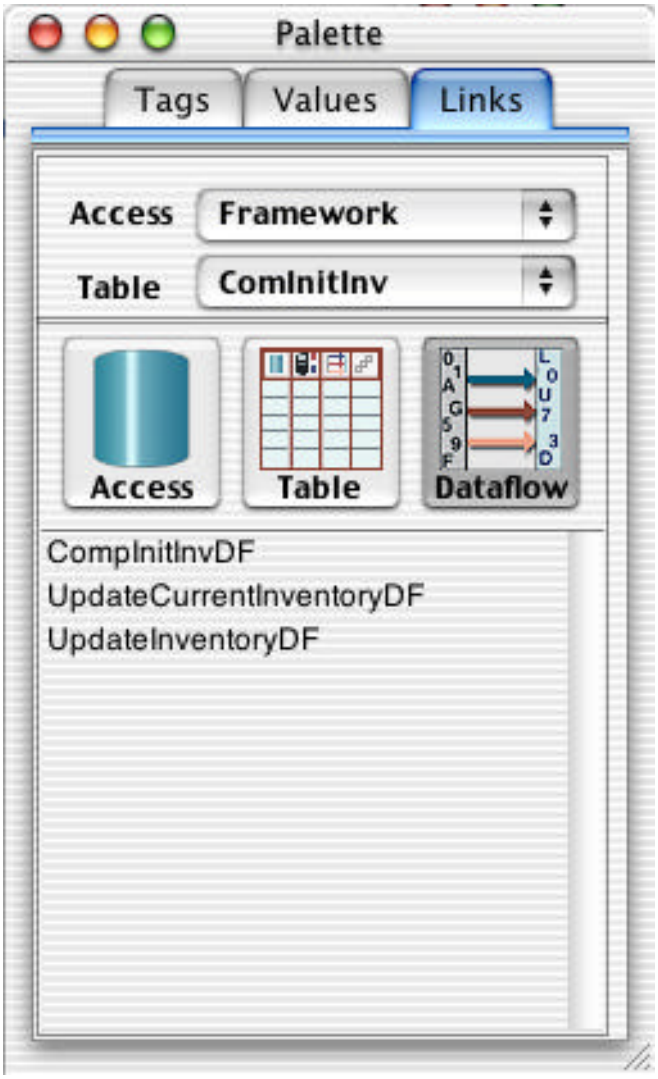
To select a tag, click on its name and press and hold the mouse button. Drag the tag onto the WebTemplate until you see a small insertion arrow, then release the mouse button.

The Values grouping displays fields and abacus icons from the Frameworks and Tables. All Frameworks and Tables are accessible.

To select a field or abacus expression, first choose a Framework or Access Table from the Access popup, then a specific Table. The fields and abacus expressions will then appear.

To view the field or abacus icons, click on the respective icon in the middle portion of the palette window. To select an icon, click on its name and press and hold the mouse button. Drag the icon name onto the WebTemplate to a 'black hole' then release the mouse button.

The Links grouping displays Access, Table and DataFlow icons from the Frameworks and Accesses. All Frameworks and Accesses are accessible.



The Links tab enables the designer to connect database elements. To select an Access, Table or DataFlow, first choose a Framework or Access Table from the Access popup, then a specific Table.

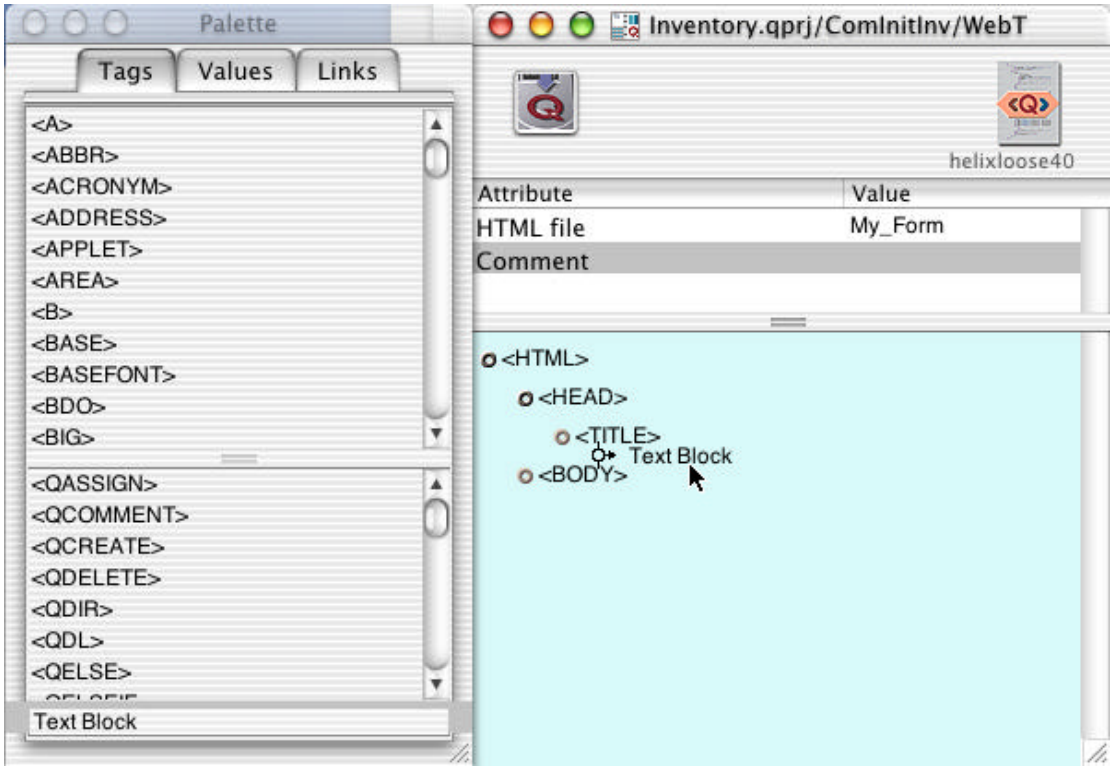
To view Access, Table or DataFlow icons, click on the respective icon in the middle portion of the palette window. To select an icon, click on its name and press and hold the mouse button. Drag the icon name onto the WebTemplate to a 'black hole' then release the mouse button.

Building a WebTemplate

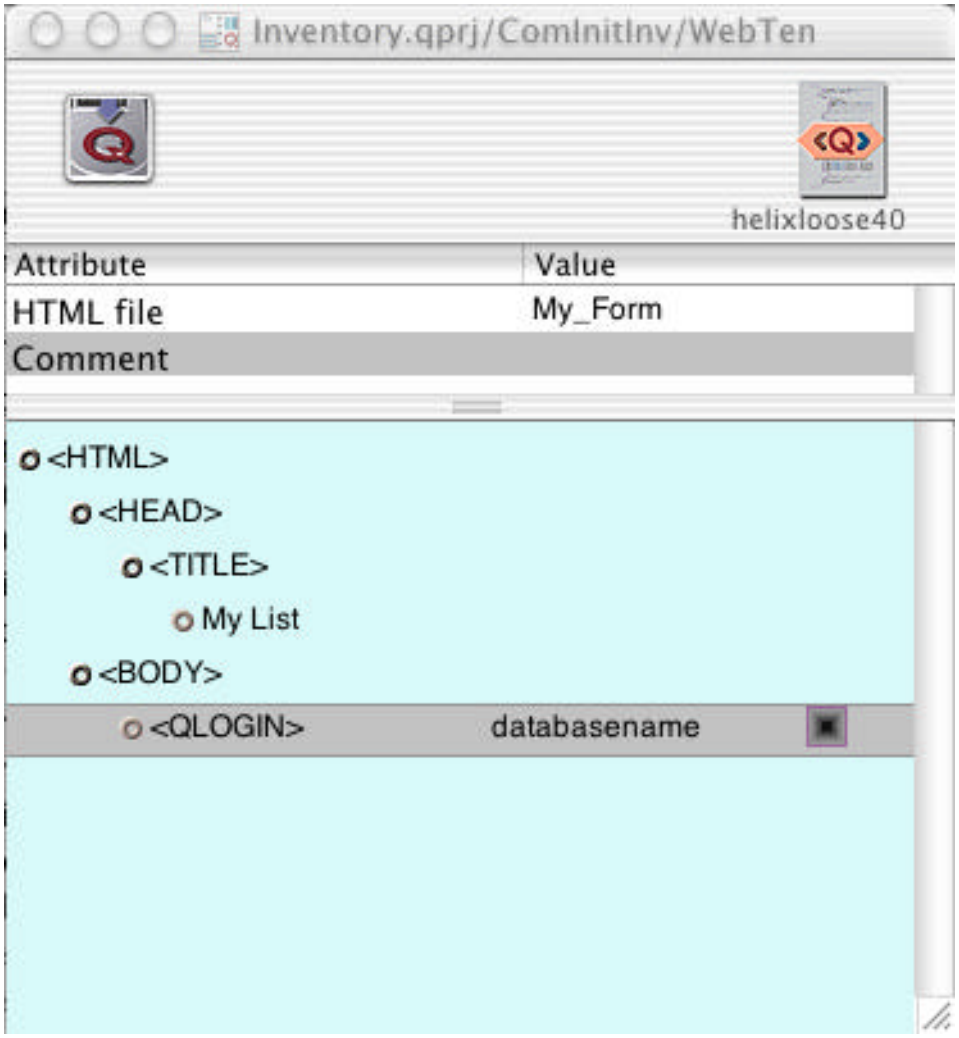
Fundamentally, a WebTemplate is HTML document, composed of formatting tags that define the layout structure, links and form controls. Qilan adds special 'Q' tags that are used integrate data, control processing and link to external database systems.

While the vast majority of your WebTemplates will be accessed by a web user, WebTemplates can be used or 'called' by external scripting programs. When used in this way, HTML formatting is unnecessary.

Let's start by displaying a list of records from an Access > Table. Open a new WebTemplate window and drag out a Text Body tag. Drag this tag into the TITLE tag until the insertion arrow appears below and indented, then release the mouse button.



After the tag is 'placed', double click the Text Block to begin typing. Type the page's title, "My List". When you are finished, click anywhere outside the Text Block.



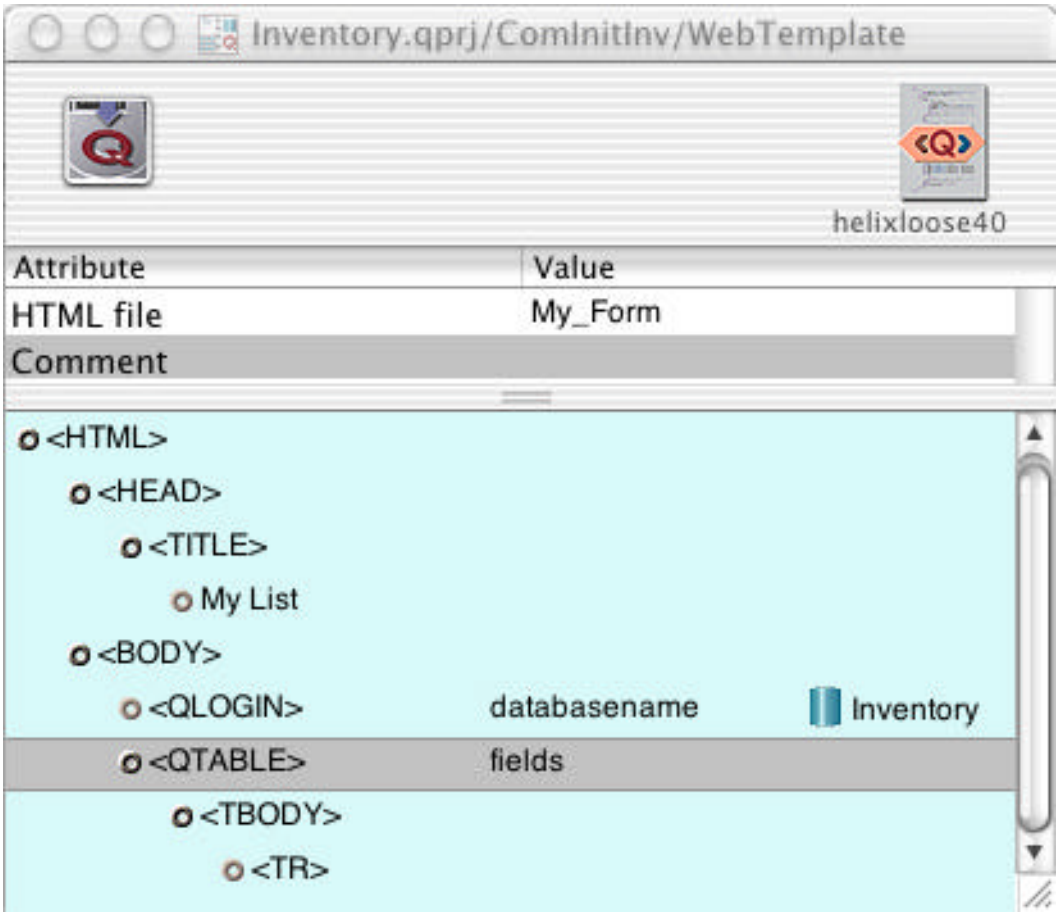
Qilan uses an outline structure to format HTML and 'Q' tags. Indents are used to indicate the closing of tags and processes. Therefore, closing tags are *NEVER* used.

To begin the retrieval of list data from the database, we first have to identify the database, using its Access icon. Recall that the Access icon contains the location of the database and logon information. Qilan's QLOGIN tag is used to link the Access. Click the palette's Tags tab then drag QLOGIN onto the WebTemplate. Although a QLOGIN can be placed anywhere on the WebTemplate, it must be placed prior to any other 'Q' tag that accesses the database. To complete the QLOGIN, we will select an Access, from the palette and drag it into the black hole labeled, "database".

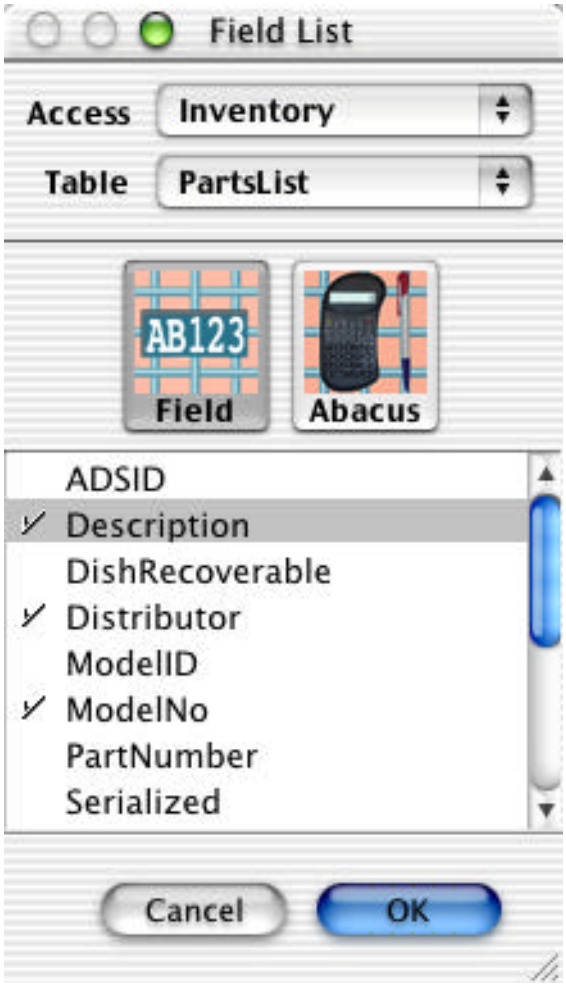
This is a good time to point out how Qilan reads the WebTemplate. Simply, top to bottom. As tags are encountered, they are processed. This is the reason a QLOGIN must occur before other 'Q' tags that access the database. As you become more sophisticated in your designs, you will find that tag placement can be used as a tool in controlling processes.

The next task will be to retrieve the database data. We will break this into two parts: format and retrieval. Qilan has several 'Q' tags that retrieve data, but for simplicity sake, let's use a Table.

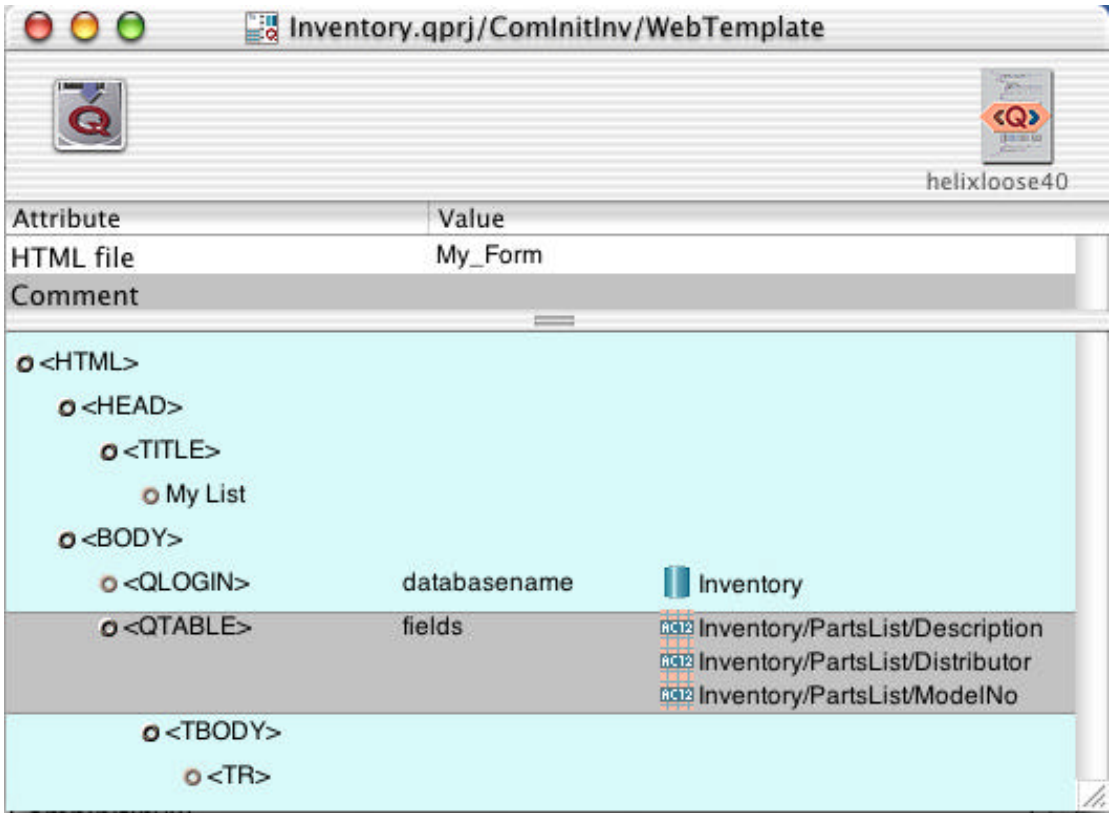
An HTML Table is merely a format structure, however a QTABLE integrates a Table format with data retrieval options. Continuing with our construction, drag a QTABLE from the palette so that it is placed just below the QLOGIN.



QTABLE appears just like an HTML table, but contains a special element, ‘fields’. Double click on this line and the following dialog opens:



Field List allows for the selection of fields and/or abacus expressions from a specific Access > Table. Those items checked (by double clicking), will be retrieved. Note that ‘retrieved’ doesn’t mean ‘displayed’. That comes next. When you finished, click ‘OK’.

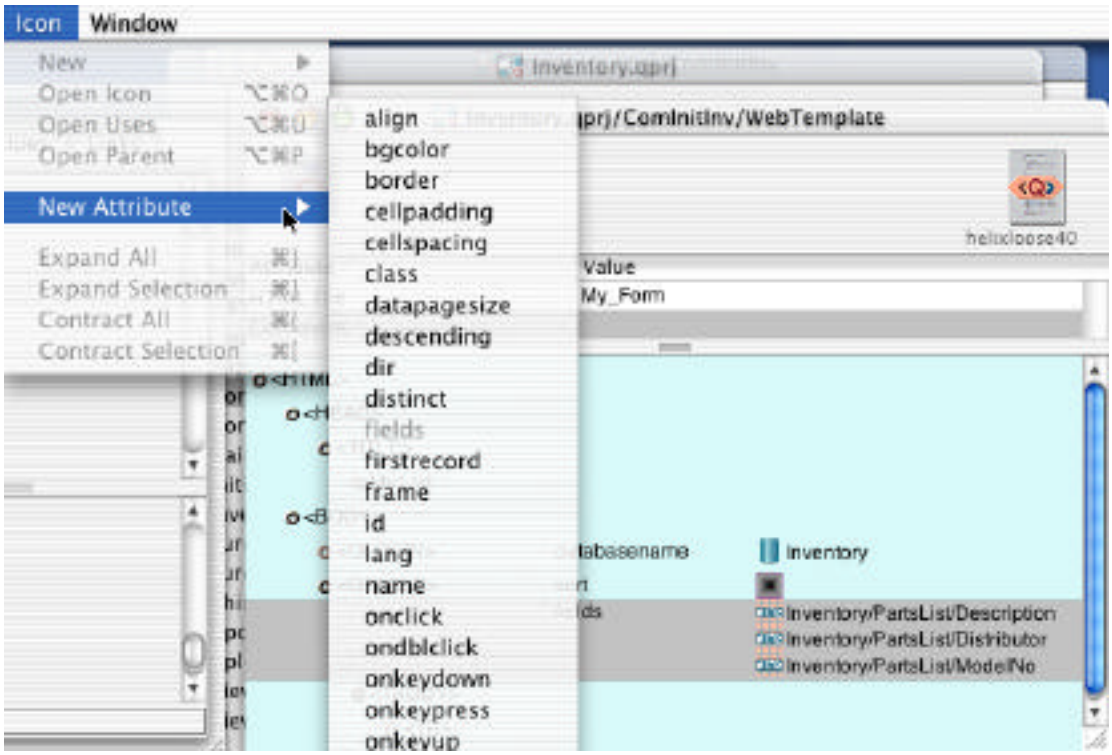


The WebTemplate now shows our selected fields. While the QTABLE tag is highlighted, select New Attribute from the Icon menu.

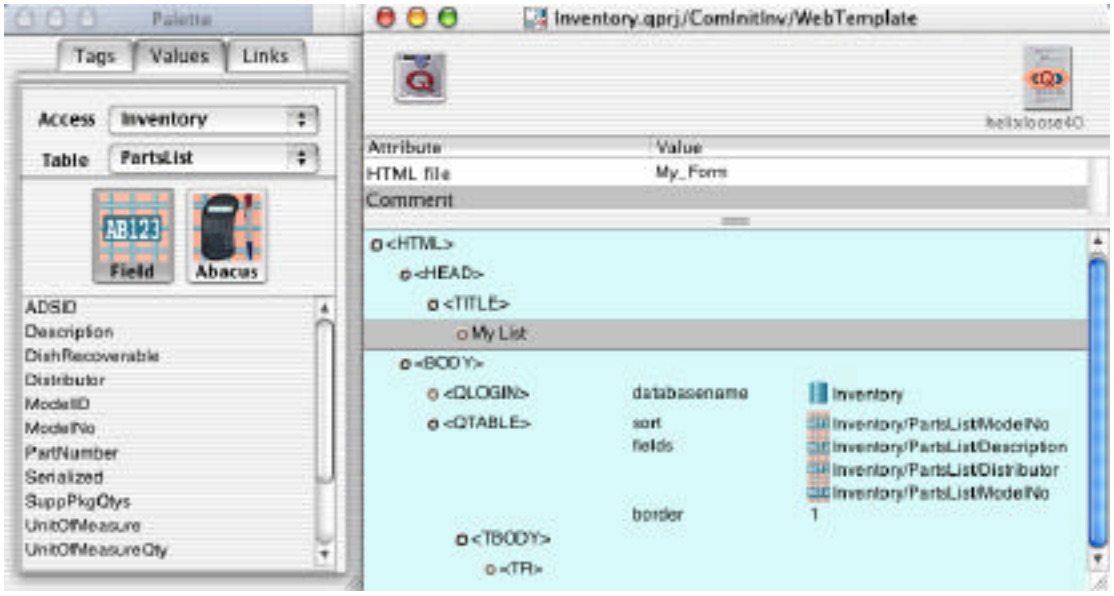
All the HTML attributes will be shown along with Qilan data options. For example, we will want the list to be sorted, so the 'sort' option is selected. By default, the sort order is in ascending order. We also want a border, so we choose this attribute as well.

Some attributes, such as 'border', require a specification. Double click on the line labeled, 'border' and enter the number '1'.

How do you know what attributes require specification and what values are acceptable? If you are unfamiliar with HTML, this can be a bit difficult. We suggest you consider using a graphical layout program (and import the HTML source code), consult with an expert, or grab yourself a good book on HTML. A bookstore with a large computer section is an excellent resource. The internet also has many web sites that can be very helpful.



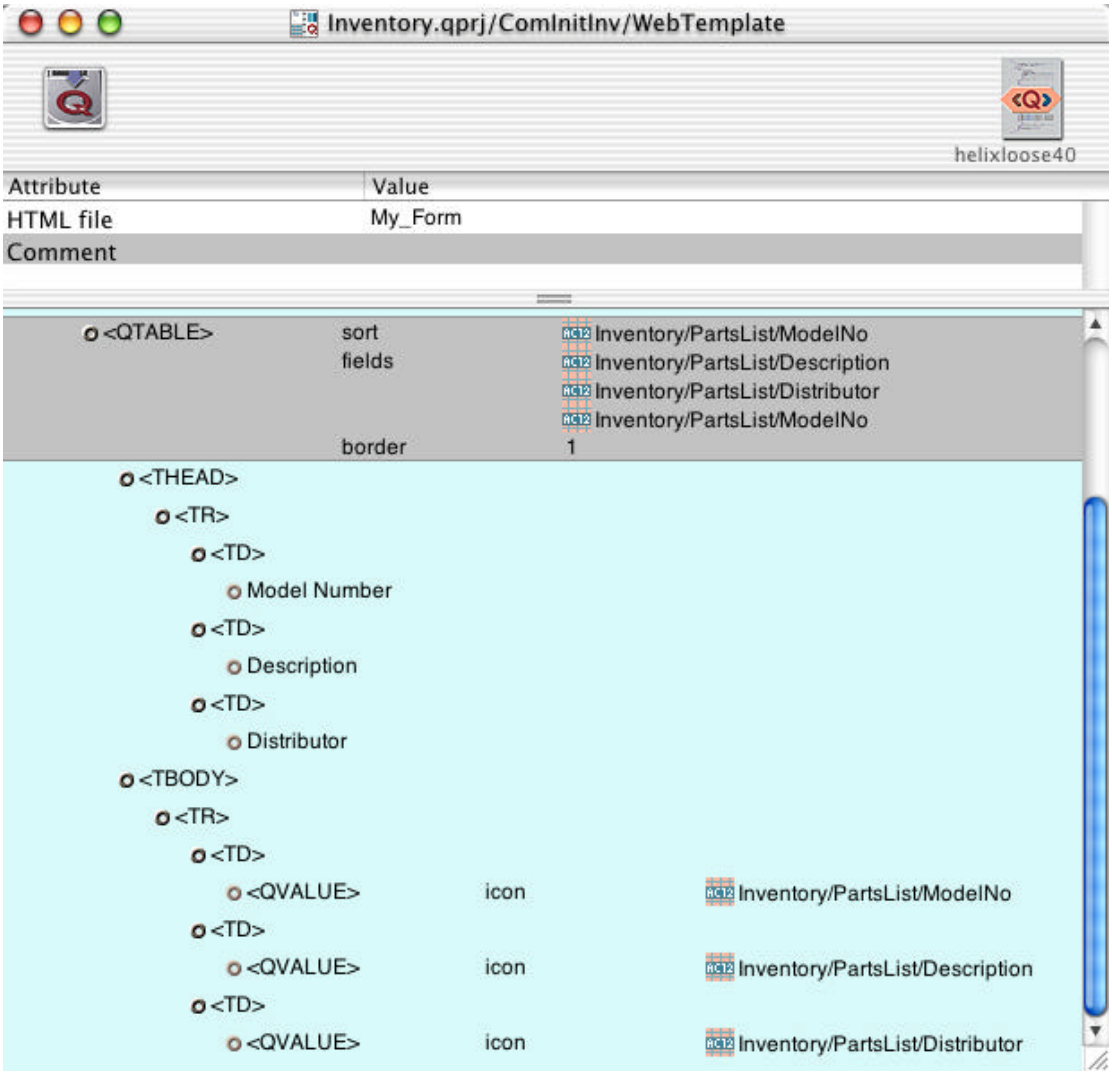
To choose a sort order, we need to select a Table field from the palette. Select the same Access and Table chosen in the Field List dialog, then drag the desired field to the black hole adjacent to the sort attribute.



Now its time to layout and display the data. A nice table has column labels. This table will have three columns. Let's add the tag, THEAD. This tag will be used to show column labels and will not iterate.

What's iteration? Qilan's QTABLE tag repeats the TBODY section for each record retrieved. This is a very cool feature. In effect, the TBODY section is built for one record (or row), then just repeats itself until the list is complete. Other table tags, THEAD and TFOOT (known as 'sections') do not repeat.

TD tags are added to the TBODY section to layout the data display.



QVALUE tags display data; the QTABLE tag retrieved the data. Drag QVALUE tags from the palette so that they are indented inside each of the three TD tags. Note how each QVALUE has a black hole. This is where the field or abacus icon goes.

To get the field or abacus icons, select them from the palette (Values Tab). Select the same Access and Table chosen in the Field List dialog, then drag the desired field or abacus icon to the black hole(s).

This WebTemplate can now be accessed from the internet:

http://[IP]/cgi-bin/qilan.cgi/My_Form

Previewing a WebTemplate in an Internet Browser

On the webtemplate toolbar, clicking the 'preview in browser' icon will open the webtemplate in your default internet browser.

If have made changes since the last time you exported the webtemplate and want to preview them, make sure you re-export the webtemplate prior to previewing them in the browser.

Note that the default URL string (See Project Settings) is set for qilan.fcgi. If you have not installed FastCGI, you should change the URL path so that qilan.cgi is used. Failure to do this may result in the Apache error, 'Page Not Found'.

Webtemplate Printing

To aid in the development process as well as document your work, webtemplates can be printed in a structured format. The layout is very similar to the way webtemplates are presented in the lower portion of the Uses window.

At the top of each webtemplate, on the toolbar, you will find an icon for printing and page setup.

> If they are not present, select ‘Customize Toolbar’ from the Icon menu, while a webtemplate is open and the active window. Drag the print and page set up icons to the webtemplate toolbar.

To print the webtemplate, click the print icon. The webtemplate will be printed immediately. At the top of page 1, the following information will appear:

```
Qilan Version          Registration Info       Date/Time
Project_Name/Framework_Name/Webtemplate_Name
```

Inserting Tags

HTML and ‘Q’ tags are dragged onto the WebTemplate to construct the page. Typically, HTML tags are used for formatting and layout while ‘Q’ tags are used for data processing and form control.

As the tag is dragged from the palette onto the WebTemplate, a small arrow will appear. This will indicate a ‘legal’ placement. If you do not see an arrow, the tag cannot be legally placed.

Tags dragged from the palette onto the WebTemplate window will be checked against the DTD for legal placement.

Qilan strictly enforces tag placement according to the selected DTD. While this is desirable in most circumstances, it can be limiting for the intrepid designer. Consider the situation where a database value needs to be obtained for use inside a HEAD tag. DTDs will prevent the placement of most Q tags.

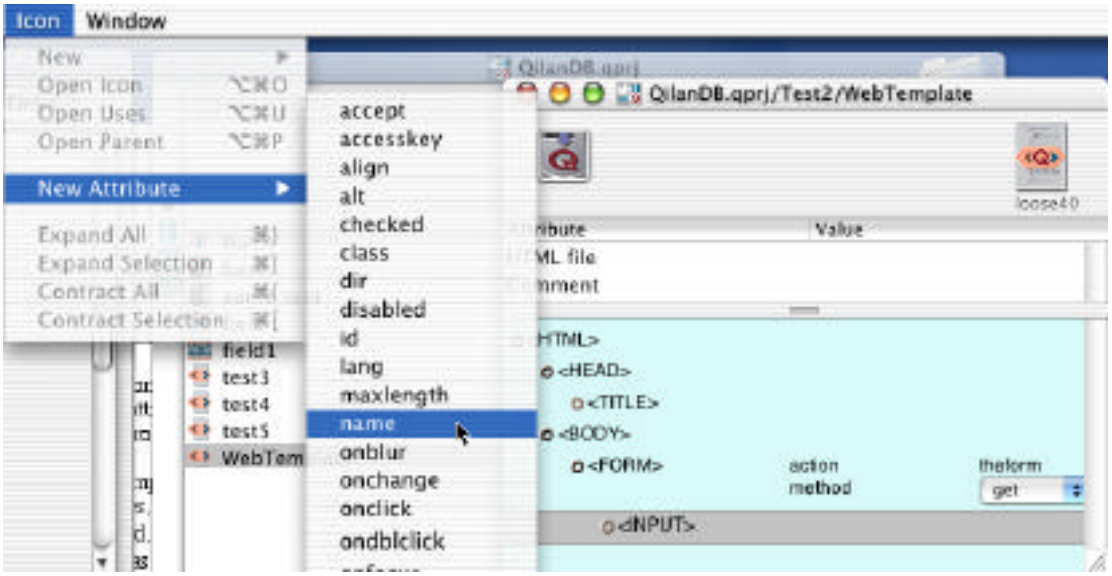
To get around this limitation, you can directly modify the DTD file (for the fearless developer) or merely hold down the option key when you drag a tag. Qilan will allow a tag to be placed anywhere a child tag is permitted. *Placing tags where a browser may interpret them as being 'illegally placed', may result in display errors or unexpected anomalies*. Always test your designs.

Selecting Tag Attributes

Most tags have 'attributes'. These are special settings or extensions that are used to modify or enhance the tag's format or functionality.

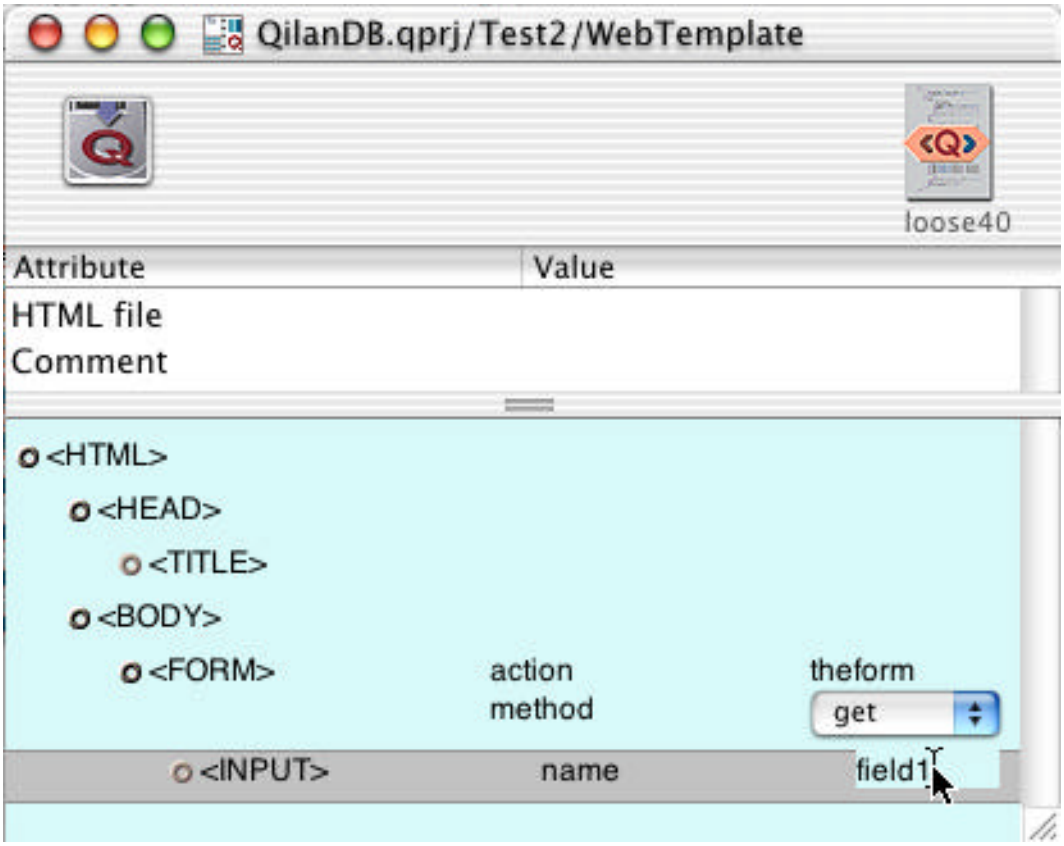
Some tags, especially INPUT tags, will not function properly without 'attributes'. Please refer to an HTML reference or qualified professional for assistance with HTML attribute settings.

Attributes may be associated with a tag by first highlighting the tag in the WebTemplate window, then choosing the attribute from Icon > New Attribute submenu.



You can add as many attributes to a tag as necessary. As you add new attributes, they will be listed vertically adjacent to the tag.

Some attributes require additional specification, which must be typed directly onto the WebTemplate. Double clicking on the attribute line will display a cursor indicating where the 'value' is entered. After you have finished typing, click anywhere else on the WebTemplate.



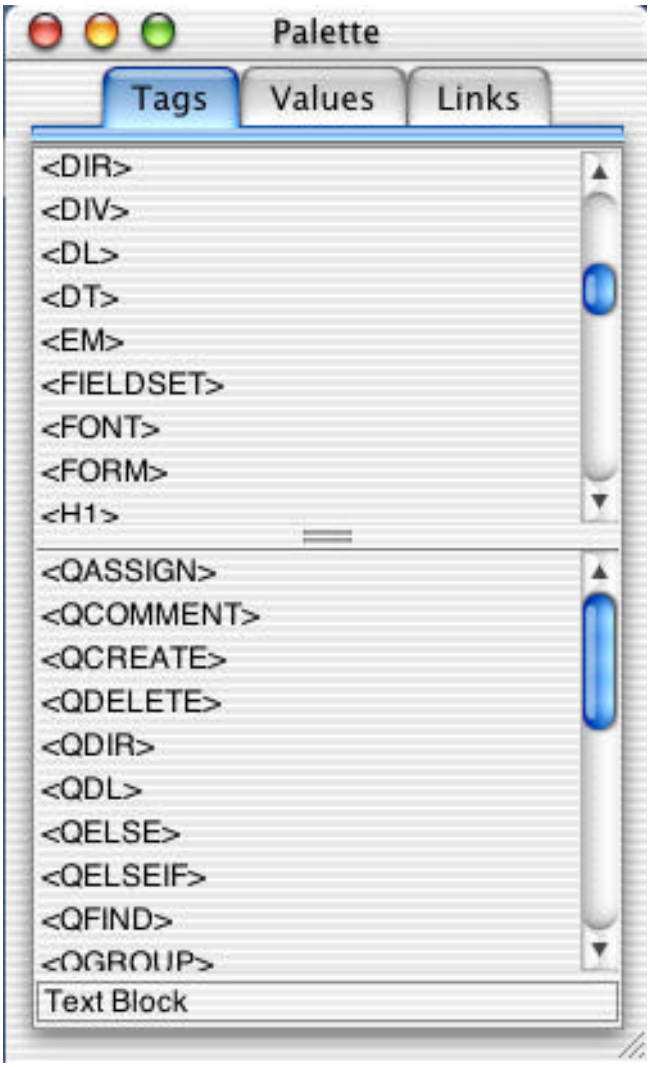
If you want to remove an attribute, highlight the attribute line, then select Edit > Clear. Be careful not to select Edit > Cut, or the entire tag may be removed.

You can also remove any WebTemplate element by highlighting the element then pressing the DELETE key.

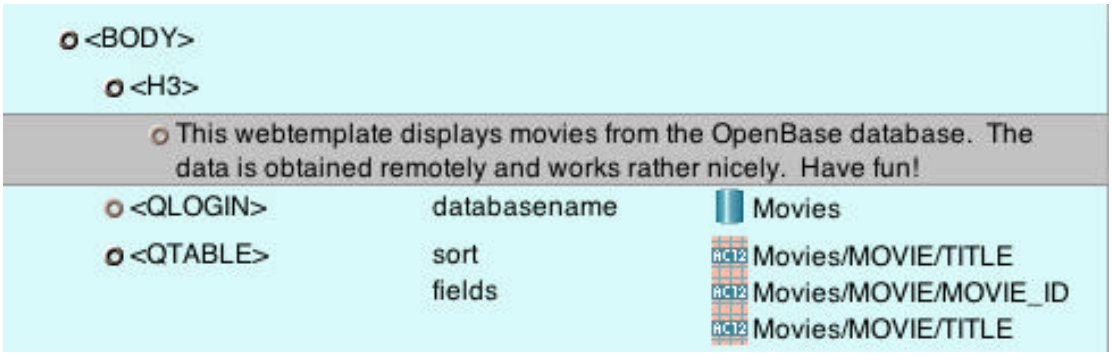
As with tags, attributes are also defined by the DTD. If you need to use an attribute not available in the pop-up menu, you will have to add it to the DTD. Please refer to the section at the end of this manual that explains how this is done. It is not difficult, but does require you to be rather precise with reference to your typing.

Text Blocks

Text blocks are available for insertion anywhere allowed by the DTD. They are used to display user text, as typed. Drag a text block from the palette by clicking on the Tags tab. Text block is located at the bottom beneath the 'Q' tags.



A text block indented within the H3 header tag. The text will formatted as header text when shown on the client browser.



When Qilan serves the exported WebTemplate page to the client browser, 'Q' tags and other non-HTML tags are removed. Text blocks are considered part of the HTML and will be passed without parsing. Text blocks may be used for JavaScript or text display, but HTML code should be avoided. Unexpected errors may result if improper HTML syntax is encountered

WebTemplate Editing

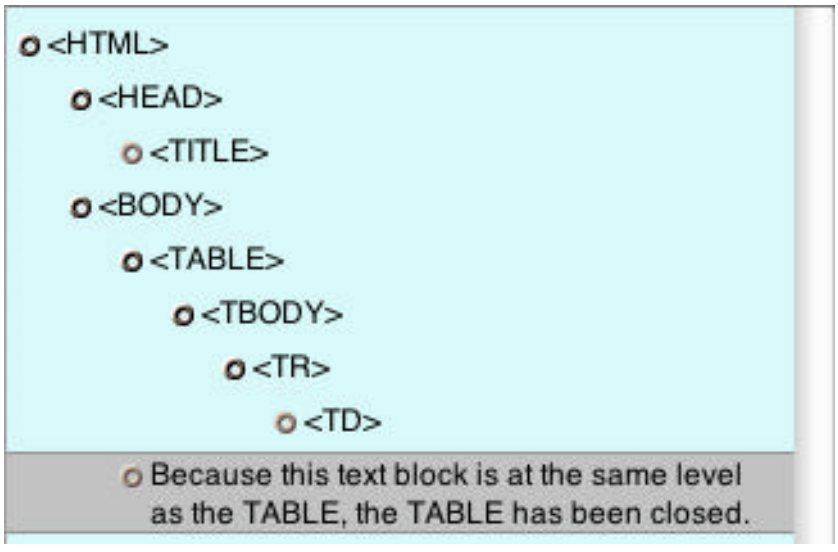
Tags placed on the WebTemplate are automatically formatted as an outline. This structure is used to format the HTML and as well as control the assignment of variables during processing.

Most HTML tags require explicit termination. This means that when a format is 'declared' or started, the browser needs to know where it ends. For example, here is the TABLE syntax:

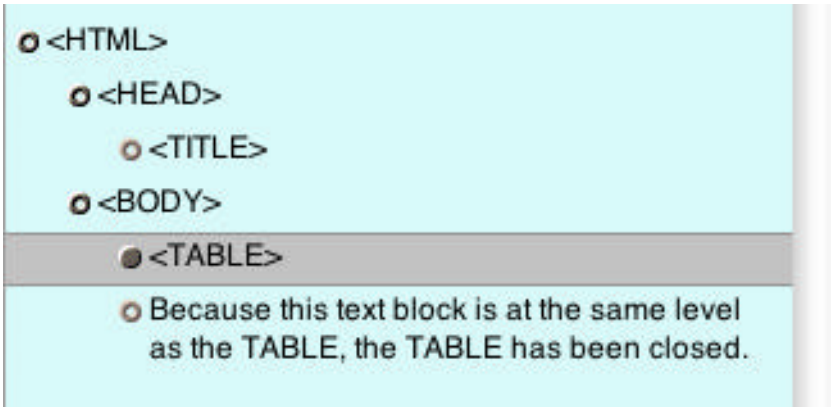
```
<TABLE>
    <TBODY>
        <TR>
            <TD>
            </TD>
        </TR>
    </TBODY>
</TABLE>
```

Tags starting with '</' are closing tags.

Qilan uses tags from the palette to start a format, such as a TABLE, but does not require an explicit closing tag. A closing tag is inferred as soon as another tag is encountered at the same outline level as the opening tag.



Those little disks in front of the tags, what do they do? Tags containing indented tags (think of a parent/child relationship) can be collapsed so that only the parents are displayed. This makes editing the WebTemplate easier and, for very large WebTemplates, helps to speed up redrawing. To collapse a parent tag, just click on the disk. A collapsed tag will display a solid disk otherwise the disk will have a hole in the middle. Note that each time a WebTemplate is opened, all tags will be expanded.



WebTemplate tags and outline sections can be dragged/copied/pasted between or within WebTemplates. To copy an outline section, highlight the tag or outline section, then choose Edit > Copy. If you first collapse the outline section, large sections can be copied at one time.

Open a new or existing WebTemplate, highlighting a valid parent HTML tag, then Edit > Paste.

What's a valid parent? A parent is a tag can legally contain children, and a child tag can legally be 'owned' by a parent. For example, you cannot paste a TR tag into a TD tag, but you can paste a TD tag into a TR tag.

Now that's clear, what's the story with the BODY tag? If you copy a BODY tag (with all its contents), how do you paste this into a new WebTemplate? What's the parent? Because Qilan creates new WebTemplates with a defaulted BODY tag, you must first remove the defaulted BODY tag before a new one will be allowed. This is due to the fact that only an HTML tag can be a parent for the BODY tag.

WebTemplate tags or outline sections copied to other WebTemplates within the same Framework will maintain all field, abacus and dataflow references. If you paste into a different Framework, Qilan will create needed fields, abacus icons and dataflows. Carefully note the following:

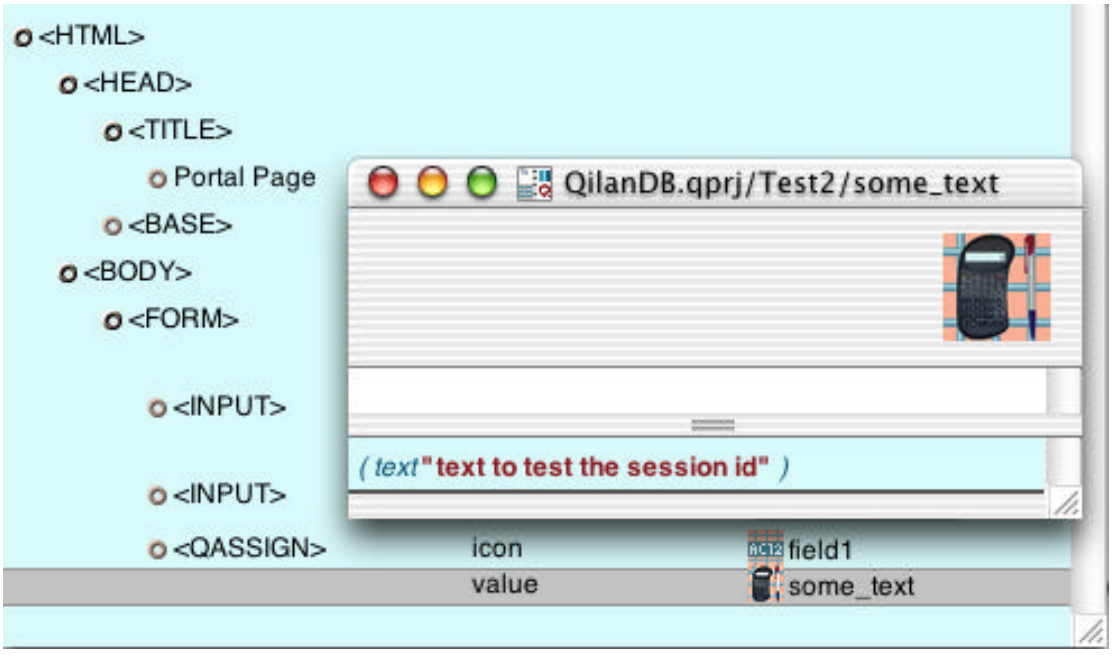
Target fields of the same name will be replaced. If a target field has been designated as a session variable, the value will become that of the source field.

Target abacus icons of the same name will not be replaced. New abacus icons will be created with their name appended by the number '1'. New abacus icons created in the target Framework will be undefined.

Relationships and DataFlows will be created using replaced fields and/or new abacus icons, as appropriate.

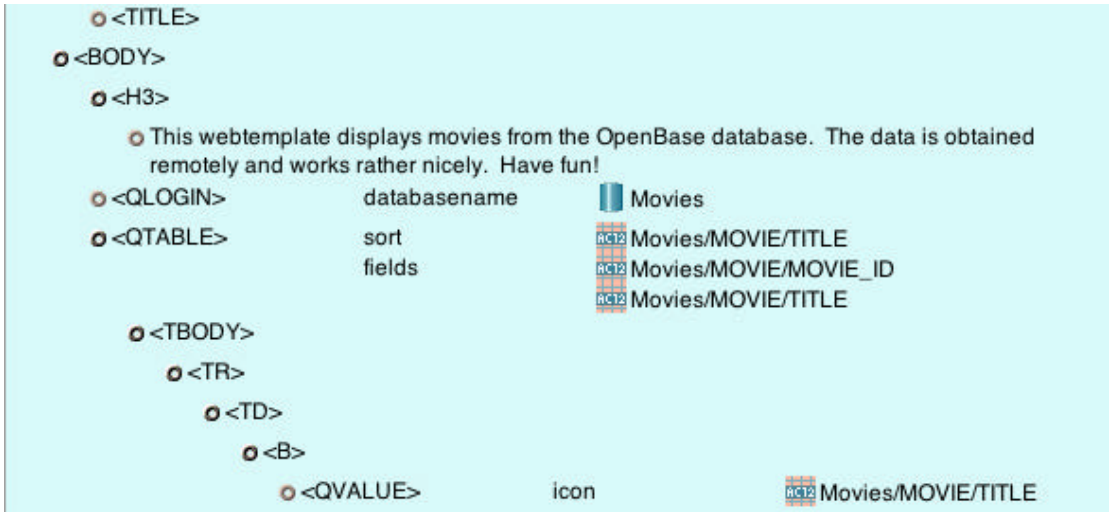
Copied sections referring to Tables or their contents will not alter Table elements.

Double clicking on a WebTemplate icon will open the icon widow. This permits direct editing. Some attributes can accept user text or an icon. When both are permissible, select, Icon > Open Icon from the menu.

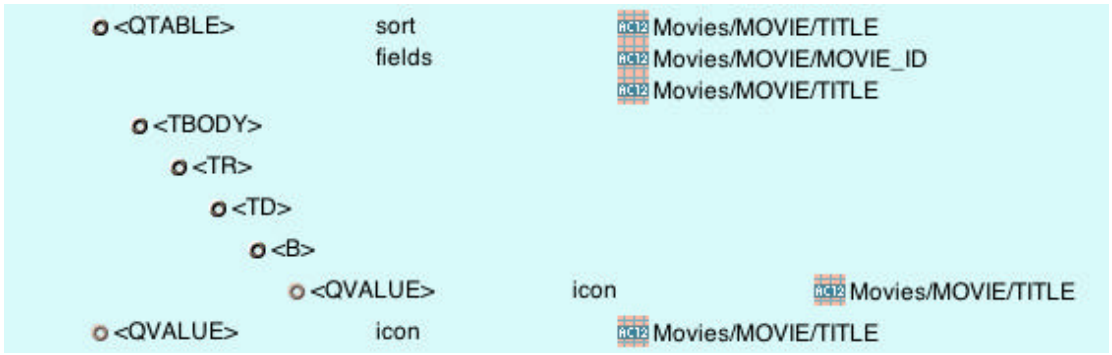


A few words about Variable Assignment…

Qilan processes (executes) a WebTemplate with respect to a 'Q' tag's relative outline position. For example, if data is located using a QTABLE or other find derivative, the data is only accessible within the scope of the QTABLE.

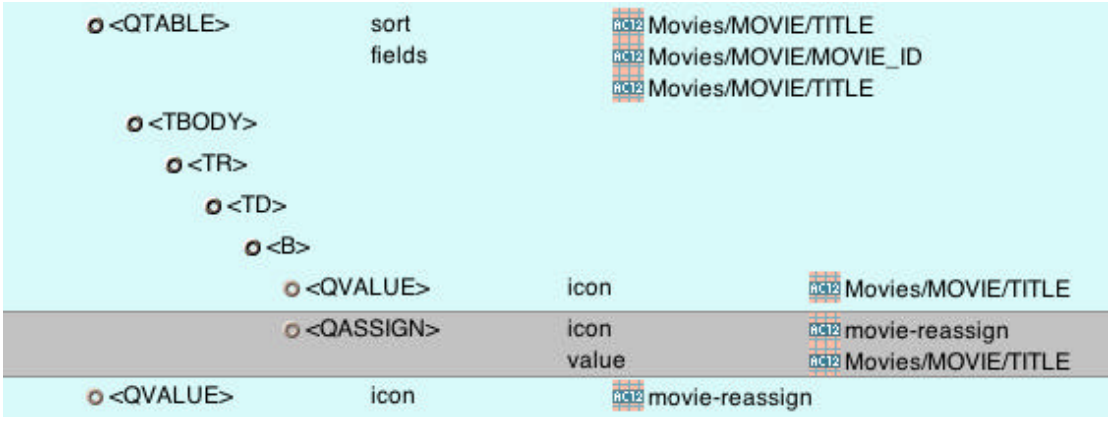


In the above example, QVALUE is valued by the QTABLE retrieval of 'Movies/MOVIE/TITLE'. The value is within the scope of QTABLE.

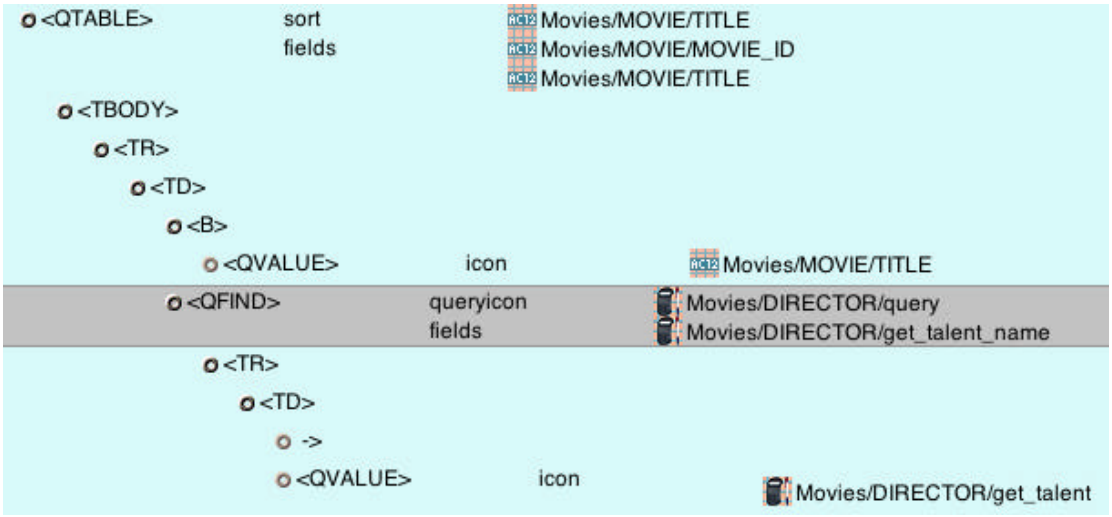


If we attempt to display the value of 'Movies/MOVIE/TITLE' outside of the QTABLE scope, as shown above, the value will be undefined.

We can re-assign the value of ‘Movies/MOVIE/TITLE’ to a Framework field, in which case it can be displayed or reused elsewhere on the WebTemplate.



In the following example, a search is performed for each record returned by the QTABLE. The queryicon (shown below) used by the QFIND refers to a QTABLE value. The relative position of the QFIND to the QTABLE allows the query to retrieve the proper value.

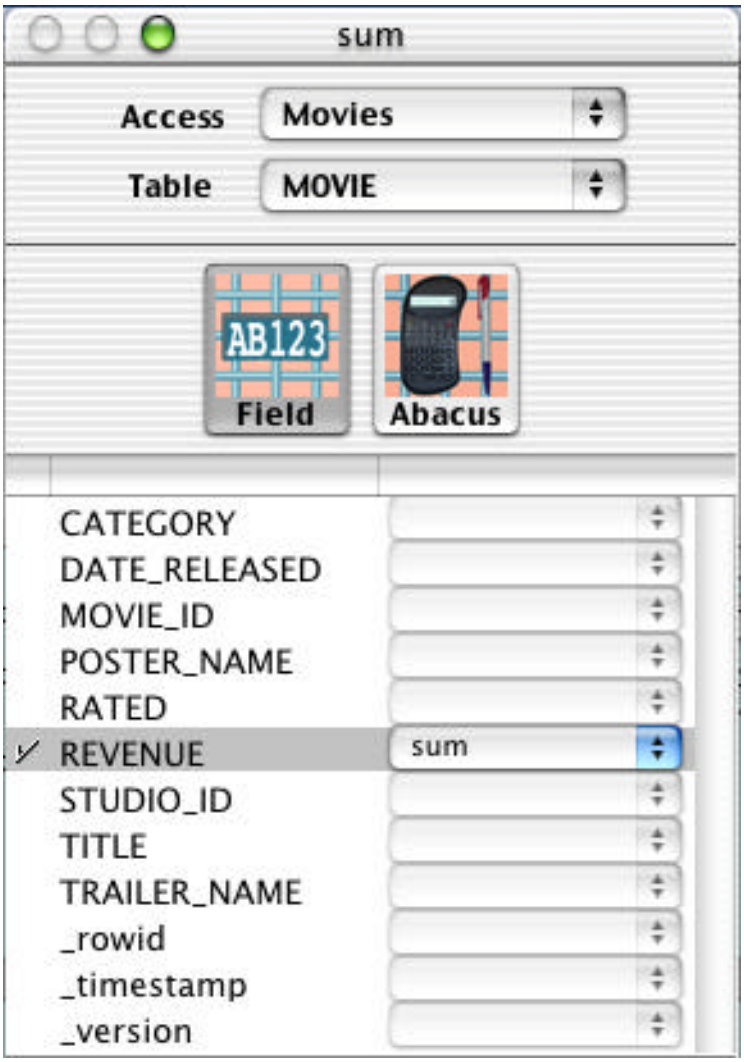



```
( MOVIE_ID = ( acquire <Movies/MOVIE/MOVIE_ID> ) )
```

Values assigned to Framework fields will be retained during WebTemplate processing until and unless they are re-assigned.

In this example we use the same framework field to display three different summary totals. Each QGROUP contains the same QVALUE icon, “sum”. As each QGROUP is calculated, its retrieval is assigned to “sum”. Qilan will re-assign the value of “sum” each time a new QGROUP is encountered.

```
<TABLE>
    <TBODY>
        <TR>
            <TD>
                <QGROUP>        sum         Movies/MOVIE/REVENUE
                    <QVALUE>        icon        sum
            <TD>
                <QGROUP>        sum         Movies/VOTING/NUMBER_OF_VOTES
                    <QVALUE>        icon        sum
            <TD>
                <QGROUP>        sum         Movies/STUDIO/BUDGET
                    <QVALUE>        icon        sum
```

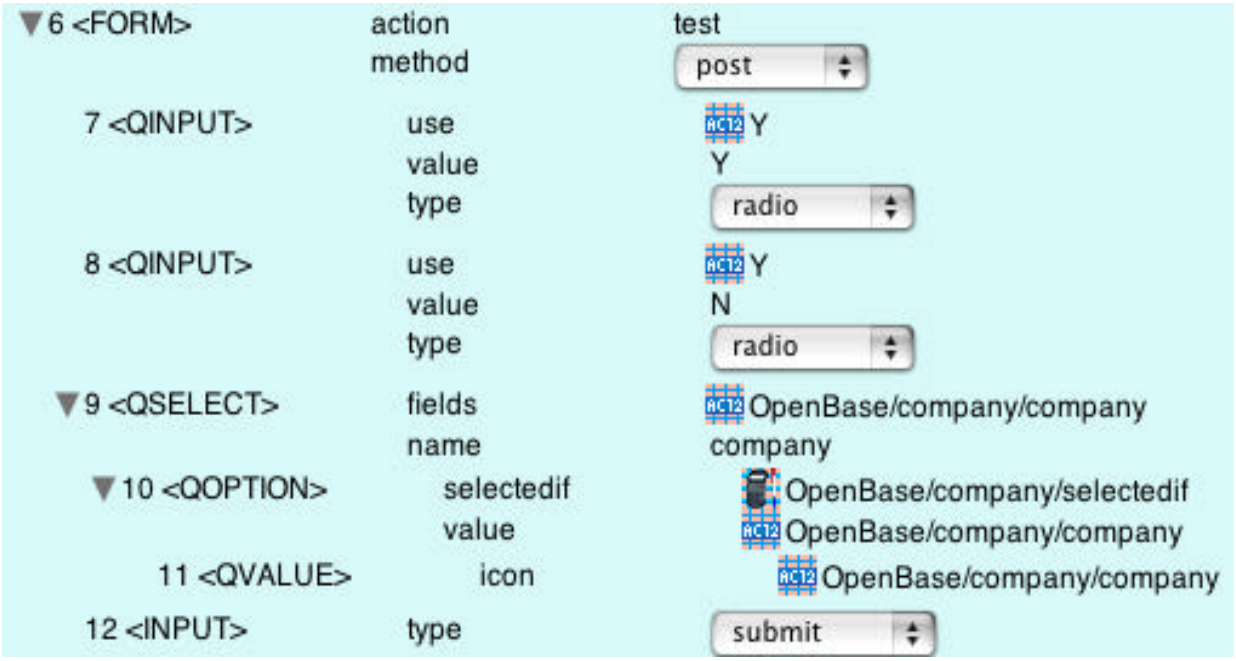


WebTemplate Line Numbers

When a WebTemplate error occurs due to a design error, SQL problem or other anomaly, Qilan will refer to a specific line number on the WebTemplate. For example here is an error shown by the browser:

```
java/sql/SQLException
Database Company not started on 192.168.1.2..
{
    "__JavaBridge Stack Trace" = "java.sql.SQLExc
    attributes = (
        "FIELDS=\"<OpenBase/company/company>\"",
        "NAME=\"company\"",
        "TABLE=\"OpenBase/company\""
    );
    errLineNo = 10;
    tag = QFIND;
}
```

Note that line 10 is referenced.



By default, line numbers are not shown. To display line numbers, select, 'Line Numbers' from the Icon menu with the WebTemplate window open.

Importing Existing HTML

In order to facilitate HTML design, Qilan allows a designer to import existing HTML documents. An imported HTML document will be converted then displayed in Qilan's outline structure. You may then add 'Q' tags to perform data processing.

Before you import an HTML document, check its HTML version. You can locate this information at the top of the document when viewed as source code. If you cannot find this information, use the DTD for the HTML version 4.0 loose.

Open Edit > Project Preferences. Select the HTML DTD type. Close the Project Preferences window.

Drag a new WebTemplate icon (from the palette) into the framework window. Double click on its icon to open the WebTemplate window.

Choose 'Import HTML…" from the File menu.

HTML comments <! -- through -- > will automatically be placed into Text Blocks. Insure comments are properly formatted prior to import.

When JavaScripts are imported, they will appear in Text Blocks within the <SCRIPT> tag. If you are using a DTD that requires a 'type' attribute, check to insure it is properly declared, otherwise your JavaScirpt may not function properly. JavaScript authors do not always define this attribute.

Also be aware that an HTML comment is not the same as a Qilan QCOMMENT tag. HTML comments will be passed to the browser, but not shown to the user (except as 'source'). Items placed within a QCOMMENT tag will not be passed to the browser in any form. JavaScripts should not be placed within a QCOMMENT unless you do not want them to be processed by the browser.

Import considerations…

Importing an html file into an existing WebTemplate, will replace its contents. If you do this by accident, choose Edit > Undo.

Importing a Qilan WebTemplate will result in an error unless it is imported into the same framework from which it was exported.

It is suggested you rename the imported HTML document after it has been imported, otherwise the original document will be overwritten.

Qilan will import HTML without regard for the DTD used to create the source file. Importing a 3.2 DTD into a 4.0 DTD may result in errors or unexpected behaviors. Specifically, some tags may not be able to be moved, relocated or deleted. If this happens, try to correct the source code or add the necessary legal HTML tags. A typical problem occurs when a 3.2 DTD table is imported into a 4.0 DTD.

Although the 4.0 DTD specifications make the use of <TBODY> tag optional, Qilan requires it. Before you import, insure this tag is properly inserted in your HTML source code. If a <TBODY> is required, but not present <TR> tags cannot be moved or removed. The solution is to add a <TBODY> (to the table) as the parent tag to all the <TR>s.

As an artifact of importing an HTML file, Qilan will place a text block whenever a carriage return is encountered. You can either remove text blocks after import or remove carriage returns from the HTML source file prior to import. Either way, they will not affect the display of the HTML page.

The ‘Q’ Tags

Qilan provides the designer a variety of unique processing tags. These tags are specific to Qilan and do not appear in a readable form when accessed from a client browser. Most tags have attributes that are necessary for proper processing or offer additional functionality. Please review each tag carefully.

‘Q’ tags may be used as often as necessary within a WebTemplate. They may be fully integrated into the HTML or used separately outside of standard HTML formatting.

‘Q’ tag syntax is not passed to the browser as ‘source code’, except when they are used to display data. For example, QUPDATE, QWHILE, QFIND, QLOGIN, QIFBLOCK, etc. are parsed, whereas data results of the QVALUE tag are sent to the browser.

Qilan processes ‘Q’ tags as they are encountered on the WebTemplate. You can control the order of processing by the placement of ‘Q’ tags on the WebTemplate. Tags appearing at the top will be processed first; those appearing at the bottom, processed last.

If you have been reading the manual from the beginning, you have heard this before, “processing is top to bottom”. Why does this really matter? As you construct your WebTemplate you will want to want to control *when* an action occurs. For example, before a record is created or updated, you may want to evaluate its contents, change another record or even delete it. This type of timing control is also known as creating a procedure. Don’t be put off by this, it is a wonderful programming construct that is very nicely implemented by Qilan. Visually, you can see exactly when actions are going to be performed. Yes, it can get complex, especially when combined with logical operators (if/then/else).

QASSIGN

This tag assigns any value to a Framework field. It is used to create or update an icon value so that its new value can be used elsewhere on the WebTemplate. The value of a field icon set by the 'QASSIGN' is discarded (except when designated as a session variable) after the WebTemplate process is completed.

Attributes

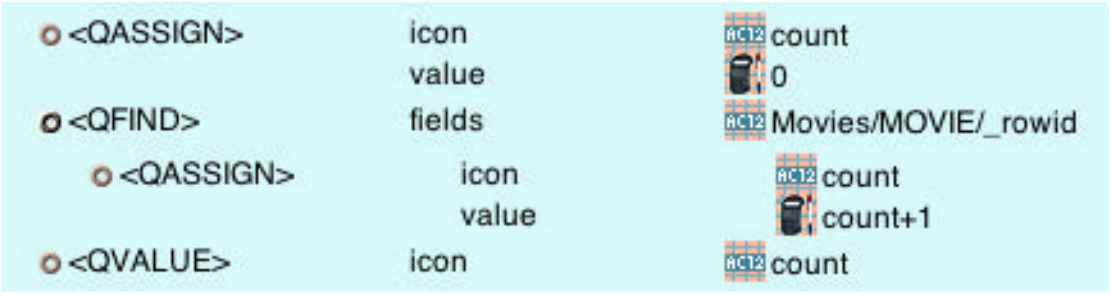
ICON: A framework field dragged from the palette (Values Tab).

VALUE: A field or abacus dragged from the palette (Values Tab), whose value has been previously obtained from a QFIND, QGROUP, QASSIGN, QVALUE or another framework field or abacus.

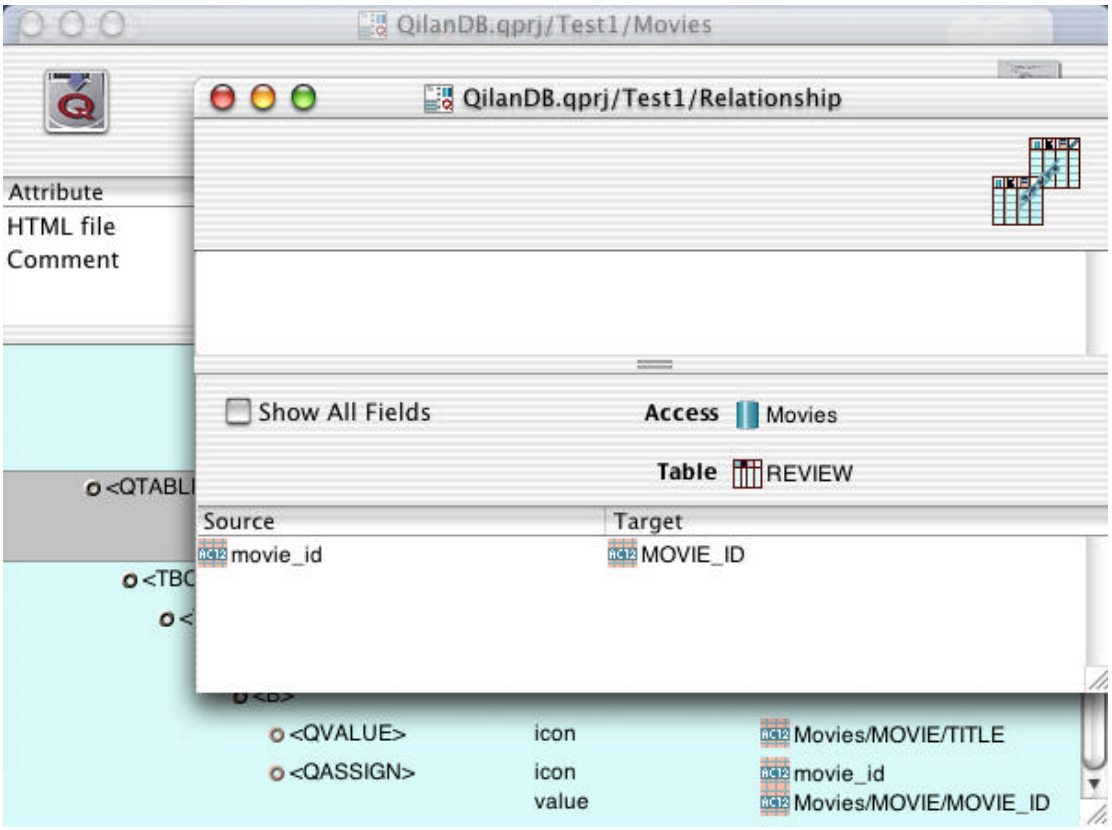
For those of you with some programming experience, QASSIGN works just like 'variable' assignment. Because Qilan only retains values during its execution, assignment is always 'local'. If you want to create a global variable, you must store the value on the web server as a session variable or in a database.

In this construction, we want to find how many records are retrieved from the database, but rather than performing a summary count, we just want to increment a counter. Faster and easier.

We start with a setting the value of 'count' at zero. Each time the QFIND iterates the icon count is set/reset to the abacus calculation of (count+1). When the last record is found, the value of count will be equal to the number of times the QFIND iterates (the number of records retrieved).



Many times you will want to retrieve a value from a database, then use that value in a Framework calculation. Here's an example:



The value Movies/MOVIE/MOVIE_ID, obtained using a QTABLE, is assigned to the Framework field, 'movie_id'. This field is then used in a relationship. This construction can be used to obtain single values from different databases based on a common link. An abacus expression, based on the relationship link is used to retrieve the data.

QCOMMENT

This tag is used to display or store WebTemplate structure (text, HTML tags and 'Q' tags) without processing by qilan.cgi or being passed to the browser. Its most common use will be to document WebTemplate constructions or just add comments to the WebTemplate.

Attributes

TITLE: Any user defined text entry. Entries will only appear on the WebTemplate and are not passed to the browser.

QCOMMENT can be placed almost anywhere on the WebTemplate. Any tags placed within its outline level will be excluded from processing. For example:

The QFIND and and iteration sequence will appear on the WebTemplate, but will not be processed. QCOMMENT and its subordinate tags will not appear on the web page.

This tag is very useful when there are portions of your WebTemplate that need debugging. It functions just like the comment characters in JavaScript (//). You can also use it to store WebTemplate fragments.

QDELETE

This tag deletes queried records in a specified table. If a query is not specified, all records in the specified table will be deleted. The use of this tag requires a QLOGIN on the same WebTemplate as the QDELETE. The user defined in Access, or entered as an attribute of the QLOGIN, must have delete permissions for the database table (in the database), otherwise the QDELETE will fail.

QDELETE is a very powerful tag that instantly (and quietly) removes data rows from a specified table. By default, QDELETE will remove *all* table records. Using a query or queryicon, you can selectively delete one or more table records. All records, which meet query criteria, will be removed when the tag is encountered on the WebTemplate.

Attributes

TABLE: A database table icon dragged from the palette (Links Tab). The selection of the table will automatically choose the database.

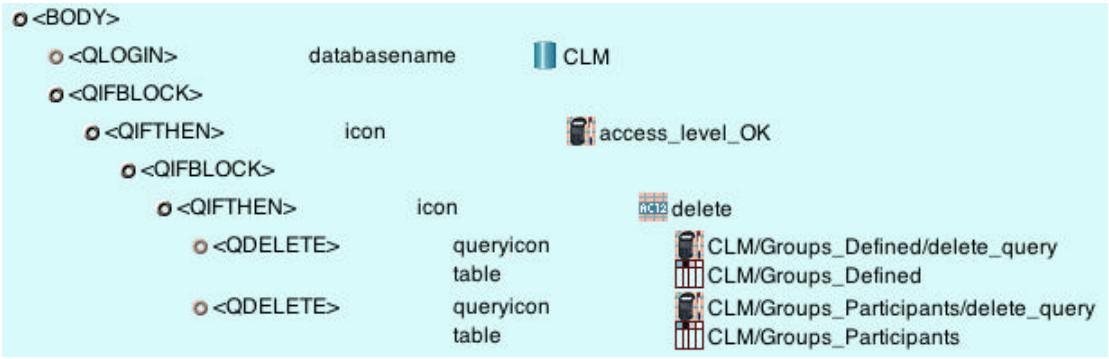
QUERYICON: A flag type abacus dragged from the palette (Values Tab), built to query the records in the specified table.

Queryicons need to be built in the same table as represented by the data. The field dialog window selects the table; the palette values tag is where you will select the corresponding table and abacus expression.

QUERY: A SQL query statement. A QUERY will take precedence over the QUERYICON.

A QUERY may be constructed in a database table, a framework abacus expression or typed directly on the WebTemplate itself. A QUERY is the portion of the sql Select statement following the 'Where' clause. When referencing a table field, be sure you use the field's 'external name', not the Qilan name.

The following example deletes records in two tables. Queryicons (shown below) are used for each QDELETE. Additionally, the field, "delete", valued when this form is submitted, is used to trigger the QDELETE tags.




```
( rowid = ( acquire <Delete/group_rowid> ) )
```

```
( group_rowid = ( acquire <Delete/group_rowid> ) )
```

QIFBLOCK

This tag is used as the parent tag for logical evaluations. All subsequent logical tags must be within the scope of a QIFBLOCK. Tags used to construct logical statements are QIFTHEN, QIFELSE and QELSE.

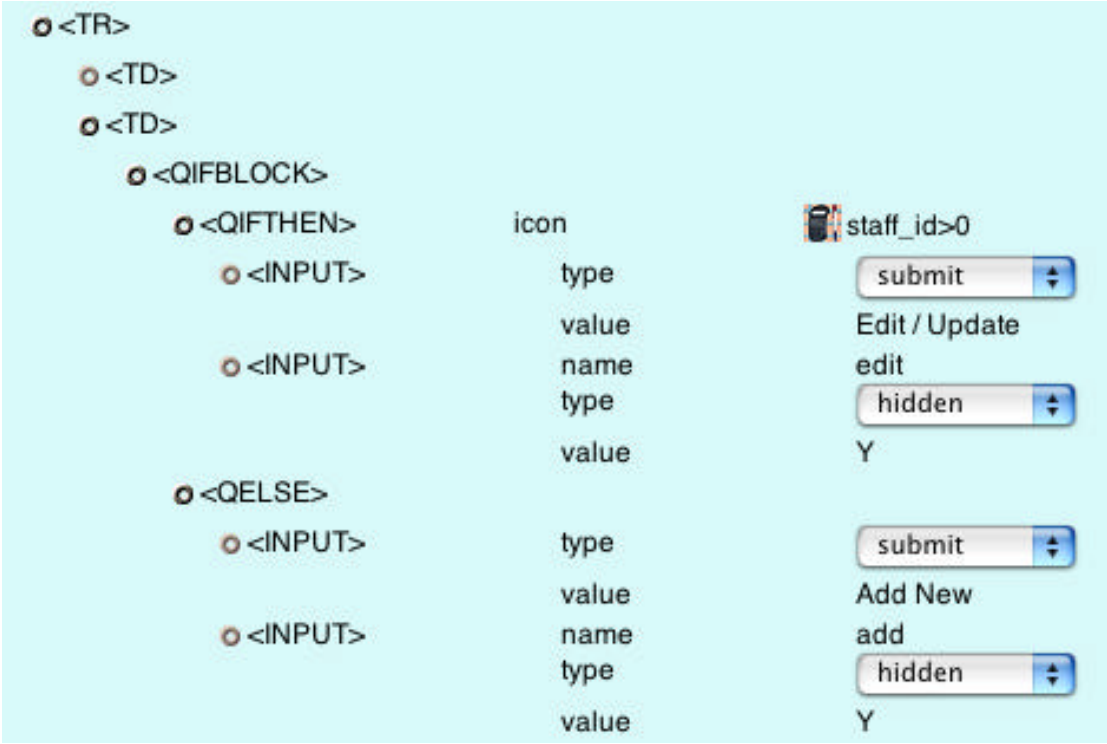
A WebTemplate may contain as many QIFBLOCKs as desired. Nested QIFBLOCKs are permissible.

What's a logical evaluation? Simply, this is choosing an action based on some condition. The 'condition', is a boolean value (Yes, No, True, False, 1, 0); the 'action' can be just about anything, including another logical evaluation.

Attributes

NONE. When a QIFBLOCK is placed on the WebTemplate, a QIFTHEN is automatically inserted.

Here, the QIFBLOCK contains a logical QIFTHEN/QELSE. The entire block is placed located inside a TD tag.



QIFTHEN

This tag specifies a condition, where tags placed below and indented under the QIFTHEN, will be processed. One QIFTHEN tag is required for each QIFBLOCK, although its icon may output an undefined value, in which case it will be ignored.

Attributes

ICON: A framework flag type abacus or field dragged from the palette (Values Tab). If the abacus or field result cannot be coerced to a flag type or is undefined, the QIFTHEN will be ignored.

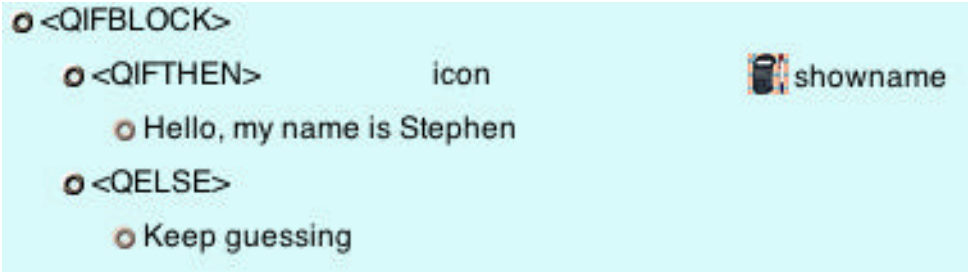
QELSE

Outputs tags placed beneath and indented when the condition specified for the QIFTHEN are not met.

Attributes

NONE

QELSE is an optional tag. When a QIFTHEN is used without a QELSE, nothing will be processed when a condition is unmet.



Here an abacus outputting a flag is used to control the display. Based on the flag output, a text display will result.

This construction will output nothing if the flag is positive. The text will be output if the flag is negative or undefined.

This construction will output the text only if the flag is positive. Note that the QELSE is omitted.

QELSEIF

This tag may be used instead of a QELSE to build more extensive logic within a single QIFBLOCK.

Attributes

ICON: A framework flag type abacus dragged from the palette (Values Tab). If the abacus or field result cannot be coerced to a flag type or is undefined, the QELSEIF and all indented tags below it will be ignored.

Here we have created a single QIFBLOCK with branching logic. One output is established for a positive result, but three outputs are used for a negative result.

Qilan processes QIF logic more efficiently when branching logic is used within a single QIFBLOCK, rather that using multiple QIFBLOCKs to accomplish the same thing.

QFIND

This tag retrieves records from a database table. Fields and/or abaci can be specified.

If a query is not specified, all records will be returned. The use of this tag requires a QLOGIN on the same WebTemplate and *before* the QFIND. The user defined in Access, or entered as an attribute of the QLOGIN, must have access permissions for the database table (in the database), otherwise the QFIND will fail. If you design your WebTemplate with multiple QFINDs (all accessing the same Access), only one QLOGIN is required.

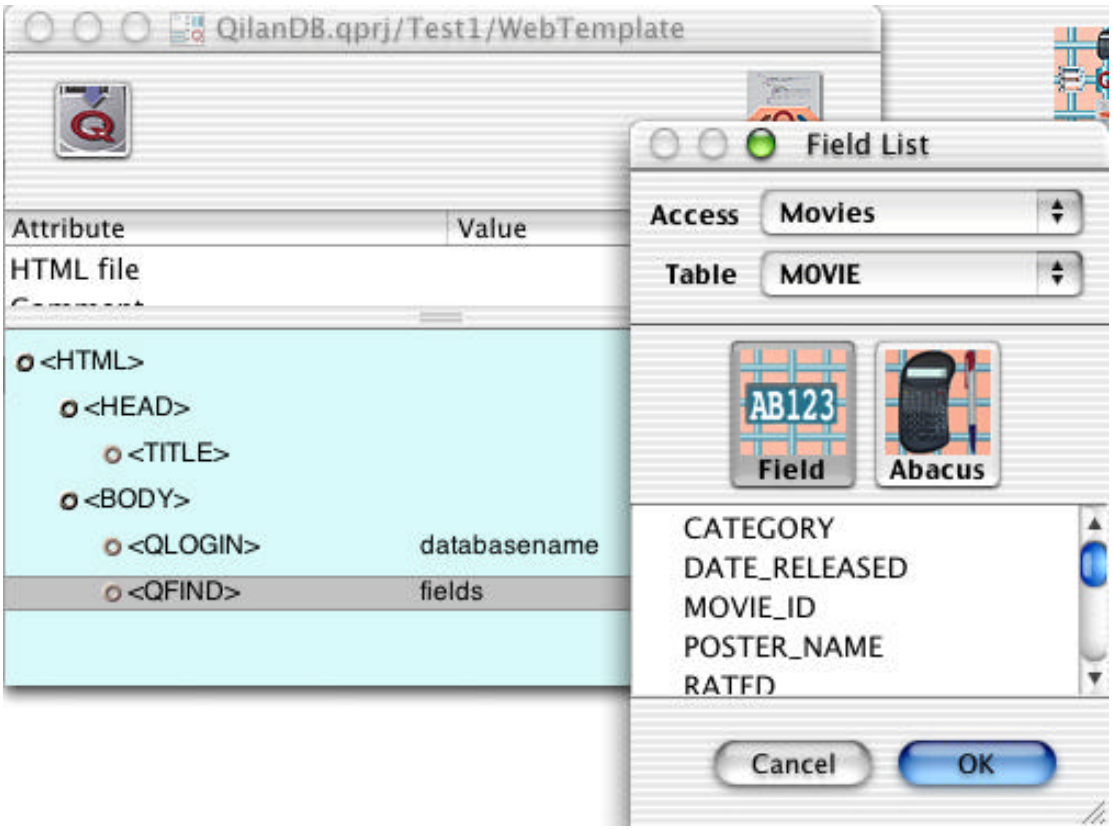
For each record retrieved, all tags within the scope of the QFIND tag will be repeated. This behavior is known as iteration.

Attributes

FIELDS: A specified field and/or abacus to be retrieved from a database table.

Selecting a field or abacus will make it available for use by other 'Q' tags (i.e., QVALUE, QASSIGN) and/or abacus calculations (i.e., acquire __).

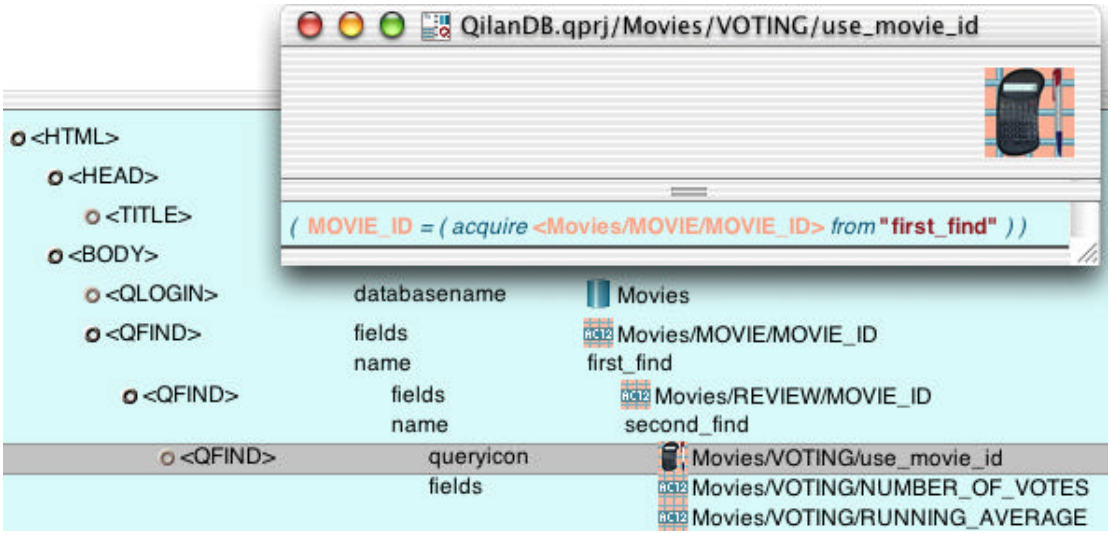
When a QFIND is placed on the web template, the field attribute will appear automatically. Double click on this line to open the Field List window. Use the pop-ups at the top of the window to select the proper table, then select as many fields or abaci as necessary by double clicking in the checkbox column.



NAME: An identifying value that references QFIND as an object.

Naming a QFIND tag enables Qilan to reference fields or abacus expressions retrieved by a specific QFIND when the target tag is ambiguous. A name may be a user defined text entry or a value generated by a field or abacus expression. When a name is referenced by an abacus, it must match exactly. See ‘acquire __ from __ and assign __ from __ for more details.

Wow, ambiguous tags! Yes, this really can happen. Consider three nested QFINDs. If your queryicon from the inner most QFIND ‘looks out’, how will Qilan know which of the two enclosing QFINDs you want to get a value from? By default, Qilan will use the enclosing QFIND, but to change this behavior, we need to use the name attribute.



In this example, we have set up three QFINDs, two of which are nested. Logically, Qilan will find the first record in Movies/MOVIE, then return all records in Movies/REVIEW, and then finally return all records in Movies/VOTING that match the MOVIE_ID from ‘first_find’. This retrieval will be repeated for each record in Movies/MOVIE.

Note that MOVIE_ID is returned by *both* ‘first_find’ and ‘second_find’. In order to differentiate the values, we specify the name of the QFIND in the query using acquire __ from __.

The use of a QFIND name attribute in this manner is the exception, normally just using an acquire __ is all that’s required.

QUERYICON: A flag type abacus dragged from the palette (Values Tab), built to query the records in the specified table. Qilan will attempt to coerce abacus results to a flag type, when possible.

Queryicons need to be built in the same table as represented by the data. The field dialog window selects the table; the palette values tag is where you will select the corresponding table and abacus expression.

Qilan attempts to parse the abacus expression and pass the result (as sql) to the database for processing. This avoids syntax errors; in fact it avoids syntax altogether – a very good thing!

QUERY: A sql query statement. QUERY supercedes QUERYICON, however if both are selected, QUERYICON will be used if QUERY is undefined.

A QUERY may be constructed in a database table, a framework abacus expression or typed directly on the WebTemplate itself. A QUERY is the portion of the sql Select statement following the 'Where' clause. When referencing a table field, be sure you use the field's 'external name', not the Qilan name.

SORT: An abacus or field dragged from the palette (Values Tab). Records retrieved will be sorted, in ascending order, on this value.

SORT icons need to be chosen from the same table as represented by the data. Use the palette values tag to select the corresponding table and icon. If you are familiar with SQL, SORT is the same as 'ordered by'.

If you desire the SORT to be in descending order, choose the attribute, DESCENDING along with the SORT attribute. Selecting DESCENDING without SORT has no effect.

FIRSTRECORD: The first record (starting at 1) to be extracted from the database. Accepts a field, abacus or any numeric value.

RECORDLIMIT: The maximum number of records to be extracted from the database, starting at FIRSTRECORD. Accepts a field, abacus or any numeric value.

Use of this attribute can help speed up the display of data on HTML forms. Although the database will process the query or queryicon regardless of the RECORDLIMIT (a sql limitation), only the number of records specified by RECORDLIMIT will be passed to the browser.

DISTINCT: A directive to return only unique records. Uniqueness is defined as the concatenation of the FIELDS specification. In other words, if you ask for Last Name and First Name, only one record will be returned with the same last and first name.

LOCK: A text entry used to create and name a record snapshot. The snapshot will be composed of all fields denoted as 'locked variables'. Names are case sensitive.

HTML / QFIND Derivative Tags

The behavior of the QFIND tag is find first, find next, find next and so forth until there are no more records to find. This action is typical of lists. HTML does not define a 'default' list format, but rather provides several styles for the designer to choose. Each style is a bit different, but they all share the same list functionality.

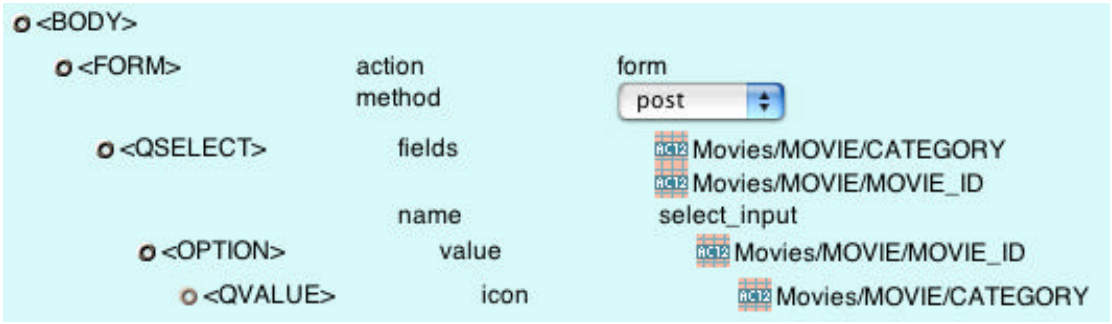
The approach taken by Qilan organizes HTML lists by their native style, then integrates the functionality of the QFIND. We suggest you refer to a reference book on HTML to understand the type of list each tag produces as well as required attributes and accepted values.

Each tag Qilan HTML list tag iterates, just like the QFIND. Tags appearing within the scope of each list tag (except for QTABLE) will be repeated for each record retrieved.

Briefly, here are the Qilan HTML list tags:

QSELECT

This tag is used when a multiple select list or pop-up is desired to display database data. It usually requires the specification of option and display values. It is the only list style that can be used as a FORM input.

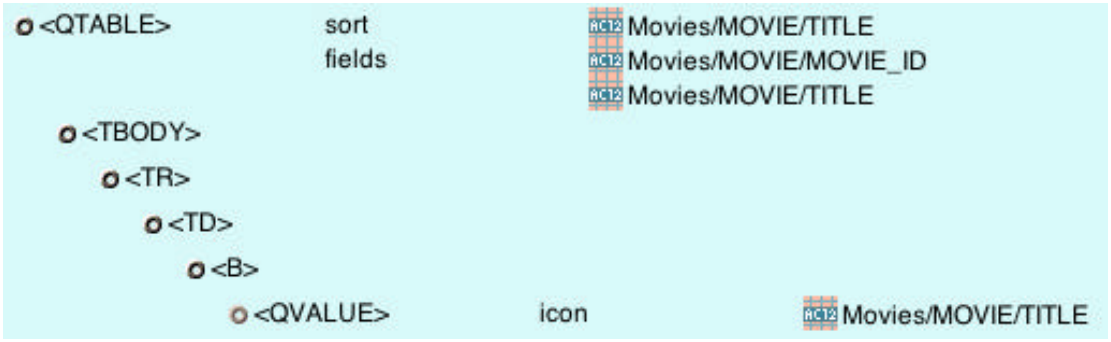


QTABLE

This tag is used when a table format is desired to display database data. This is very common format and useful for displaying limited amounts of data.

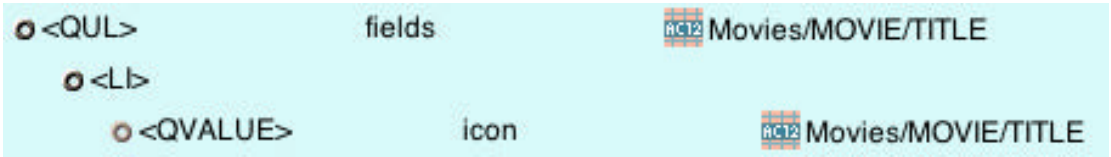
Unlike the other Qilan HTML list tags, subordinate tags within the scope of the QTABLE are not treated equally, in so far as iteration is concerned. If a QTABLE has TBODY, only the TBODY will iterate; otherwise iteration will begin with the first TR.

Browsers typically do not display table data until *all* the data is received. Lists of hundreds of records may take a long time or fail to display due to browser memory constraints. Consider using QUL, QDL or QOL as alternatives, as the browser will start displaying data as it is received.



QUL

This is the style of an ‘unordered’ list. It creates a simple, bulleted list. It requires the specification of a LI tag as each ‘list item’.



QDL

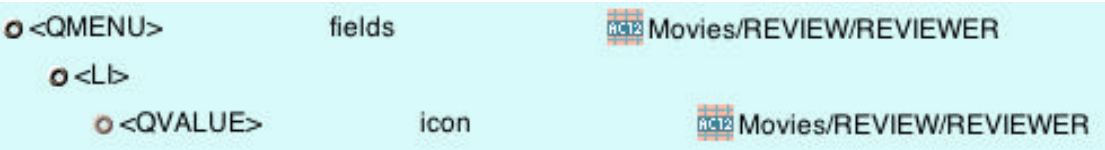
This tag creates a list of items with related definitions. Each item in the list is enclosed with a DT tag with the corresponding definition enclosed within a DD tag immediately afterwards. The definition will appear indented under each item.

QOL

This is the style of an ‘ordered list. It creates a simple, numbered list of items. It requires the specification of a LI tag as each ‘list item’.

QMENU

This tag creates simple, streamlined list of items. It requires the specification of a LI tag as each ‘list’ item. Most browsers will place a bullet in front of each list item.



QINPUT

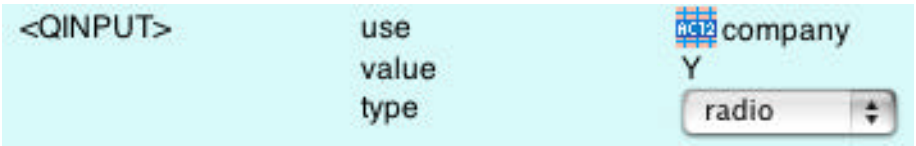
This tag is performs the same basic functionality of an HTML INPUT tag while adding specific Qilan enhancements.

Attributes

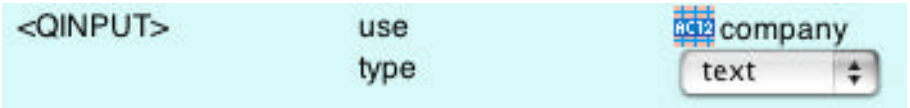
USE: Accepts field icons. The name of the icon (as it appears in the Framework window) becomes the NAME attribute and the icon's value becomes the VALUE attribute.

Qilan treats the input types, "checkbox" and "radio button" as exceptions to the USE rule. For checkboxes and radio buttons, USE will extract the icon's name, but not the value. The value attribute must be selected separately. The checked attribute is set automatically when the value of the field specified by the USE attribute matches that of the value attribute.

NOTE: NAME and CHECKED attributes are not available for QINPUT although other Qilan and HTML attributes can be used.

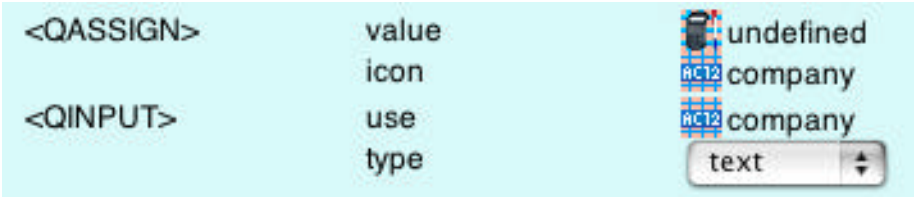


In this example, the HTML NAME attribute is set by the USE attribute: using the name of the field, 'company'. When the value of the field, 'company' equals the value attribute, 'Y', the radio button will be selected.



In this example, the HTML NAME and VALUE attributes are set by the USE attribute. The NAME attribute becomes the name of the field, 'company'; the VALUE attribute is determined by the value assigned to the company field.

A QINPUT may cause unexpected behavior for forms that reference themselves. The QINPUT value (as derived from the use attribute) will be that of the icon. In other words, the HTML input will appear to return the same value as was submitted. This may be desirable in some circumstances, however to 'clear' the value, you must QASSIGN the icon to the value, 'undefined'.



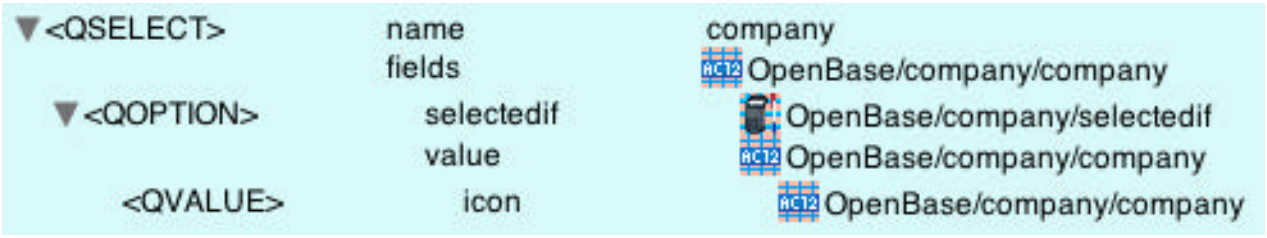
And, to create an undefined value…

QOPTION

This tag is a variant of the HTML OPTION tag and incorporates conditional functionality for the SELECTED attribute. All other HTML attributes associated with the OPTION tag are supported.

Attributes

SELECTEDIF : Accepts a field or abacus expression whose output is a value of a flag type. Positive values will produce the 'selected' attribute.




```
( company = ( acquire <Framework1/company> ) )
```

In this example, company name is retrieved from the Company table and formatted so that it appears as a select list (pop-up). A framework field (not shown) is created and named, 'company'. When this form element is submitted, the framework field will be valued with whatever item is selected from the list. If nothing were done to modify the selected attribute, the list would be redrawn, starting with the first item. To default the list to our chosen company, we create an abacus and use it in our 'selectedif' attribute. This abacus is built in the Company table. The abacus acquires the value of the framework field and compares it to all the items in the list. When an equality is found, a positive value is returned and the company name selected.

QGROUP

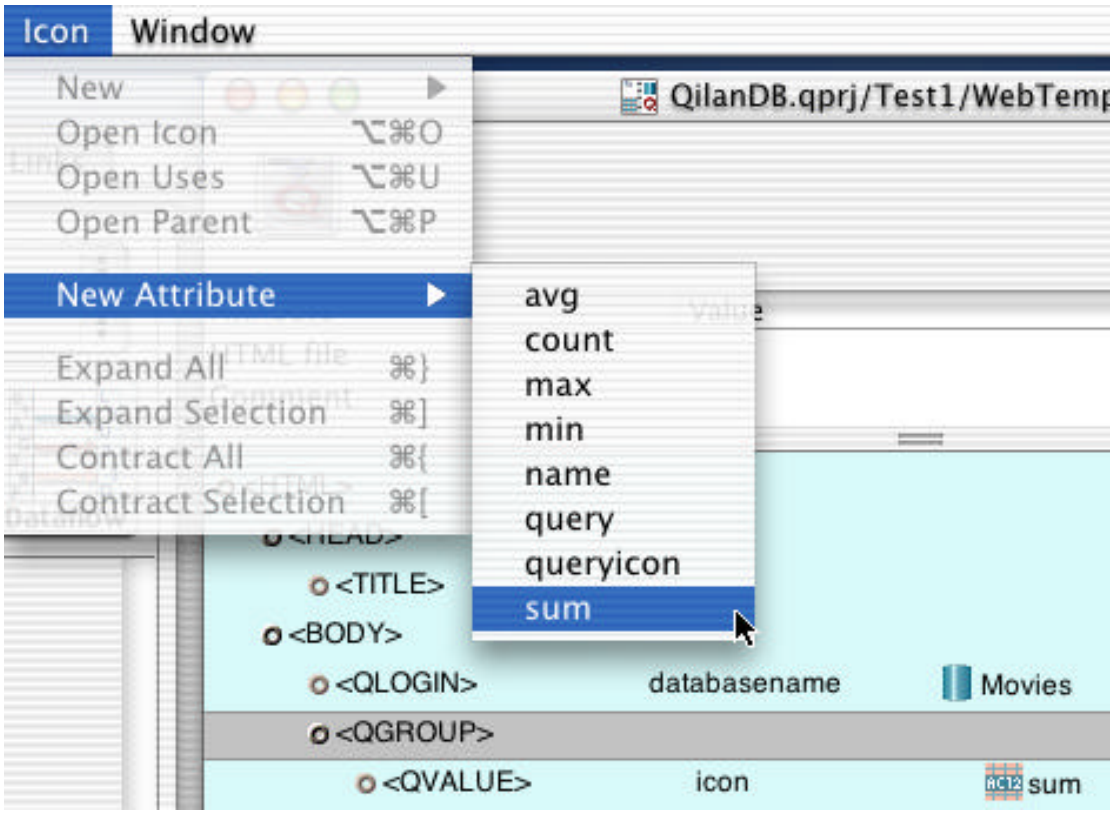
This tag returns summary values. The use of this tag requires a QLOGIN on the same WebTemplate and *before* the QGROUP. The user defined in Access, or entered as an attribute of the QLOGIN, must have access permissions for the database table (in the database), otherwise the QGROUP will fail. If you design your WebTemplate with multiple QFINDs or QGROUPs (all accessing the same Access), only one QLOGIN is required.

In this example, we want to sum the number of votes received by each movie. With the WebTemplate open, click the Tags tab on the palette, then drag a QGROUP tag onto the

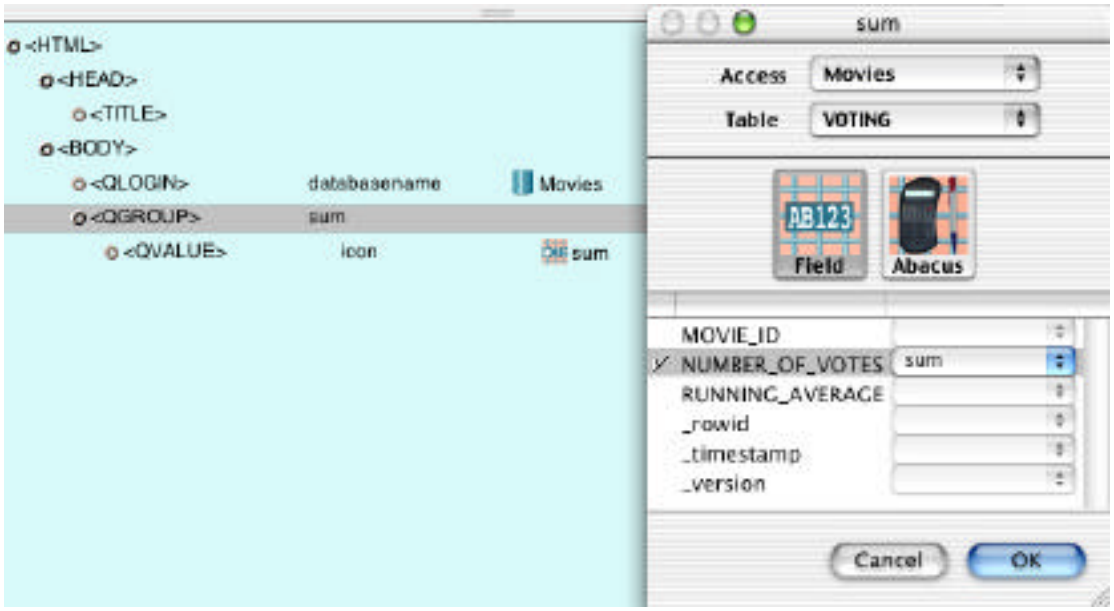
WebTemplate.

We will need to display the summary total, so place a QVALUE under the QGROUP tag. A framework field is used to hold the value. This may be a bit confusing, so let me explain what's really going on. Like the QFIND, there is a difference between data retrieval and data display. The QGROUP retrieves the data, or more correctly, aggregates it. You, as the designer, can choose to use the result in another calculation and/or show it to the user. Qilan gives you this choice. In this example, let's just show it. This is why we need to use a QVALUE.

The next task is to tell the QGROUP what type of aggregate value is to be performed.



Highlight the QGROUP tag then choose New Attribute, from the Icon menu. Choose ‘sum’.



Sum now appears on the same line as the QGROUP.

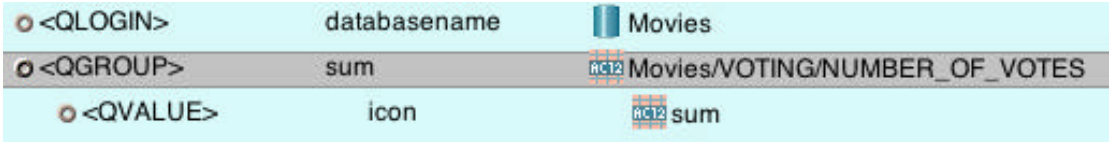
The question is now, “sum what from where”? To answer these two questions, double click on the ‘sum’ line. The sum dialog will open. Use the two popups at the top of the window to select the Access (database) and table. Once this is performed, the list of fields or abacus expressions will be displayed. Click the field icon to display fields - abacus icon to display abacus expressions.

The right portion of the list shows the fields or abacus expressions in the chosen table. Next to each is a pop-up. The popup lists all fields in the current Framework. The pop-up will remain dimmed until you select a table field or abacus expression. Double click to select. A check will appear when selected.

The Framework field is automatically assigned to the value of the summary total.

This is really a neat feature. Qilan automatically assigns a Framework field to the summary value. If you take a look at the sum dialog carefully, you might notice that you could summarize lots of table fields, all at the same time. Of course, you will want to assign them to different Framework fields (but you knew that!).

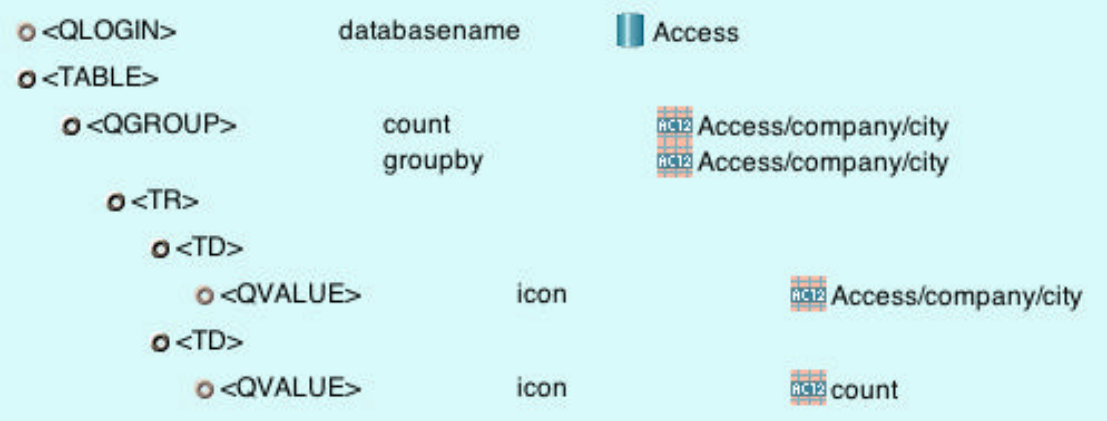
In this example, we have just used the QGROUP attribute, sum, but other aggregate calculations can be added. One QGROUP can perform summaries, averages, maximums, minimums and counts, all at the same time.



After closing the sum dialog, notice the table field being summed is shown, but the assigned Framework field is not shown.

Summarizing by a field (or abacus) is performed using the attribute, GROUPBY. When this attribute is defined, QGROUP will perform an automatic DISTINCT on the GROUPBY selection outputting an aggregate value for each record found.

The construction shown below will output one unique record for each city along with the number of times the city name is found amongst all the table records. Note how the QGROUP is placed between the TABLE and TR tag. The insertion is performed while holding the 'option' key. As each city is found, QGROUP will iterate. This effectively creates a new table row.



Attributes

NAME: An identifying text value that references the QGROUP as an object. Used by the abacus operators, acquire __ from __ and assign __ from __, where 'from' is the name of the QGROUP object.

QUERYICON: A flag type abacus dragged from the palette (Values Tab), built to query the records in the specified table.

QUERY: A SQL query statement. QUERY supercedes QUERYICON, however if both are selected, QUERYICON will be used if QUERY is undefined.

COUNT: Count on a field will return the number of records where the field is defined and meeting the query criteria, if any.

SUM: The total of a numeric field, or a value that can be coerced into a numeric field.

AVE: The average value of a numeric field, or a value that can be coerced into a numeric field.

MAX: The maximum value of a numeric field, or a value that can be coerced into a numeric field.

MIN: The minimum value of a numeric field, or a value that can be coerced into a numeric field.

GROUPBY: A field dragged from the palette (Values Tab), used to group records in a specified table. If GROUPBY is specified, the summary information will be returned in multiple records, one record for each unique value in the GROUPBY icon. They will be returned in ascending order of the GROUPBY icon, unless DESCENDING is specified.

FIRSTRECORD: The first record (starting at 1) to be extracted from the database. Accepts a field, abacus or any numeric value. This attribute will only be used when 'GROUPBY' is selected and defined.

RECORDLIMIT: The maximum number of records to be extracted from the database, starting at FIRSTRECORD. Accepts a field, abacus or any numeric value. This attribute will only be used when 'GROUPBY' is selected and defined.

DISTINCT: A directive to aggregate only unique records based on the field(s) selected by aggregate attributes. The aggregate attributes include SUM, COUNT, MIN, MAX and AVE. This attribute is inferred when GROUPBY is selected.

QLOGIN

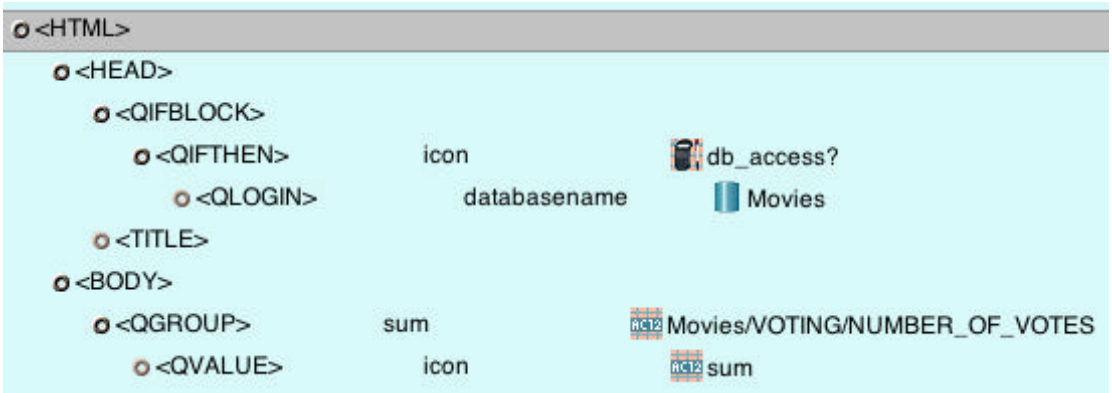
This tag contains information necessary to access the database. When this tag is used without attributes, it will use the log on information stored in the Access icon. WebTemplate attributes, if used, will take precedence over access icon parameters.

The QLOGIN initiates a database request for access and therefore needs to open a connection. This process may take time depending upon the location of the database, number of concurrent connections, configuration of the database nameserver as well as other factors.

Each WebTemplate should only have one QLOGIN for each database that will be accessed, although there is no limit on how many databases can be accessed by a WebTemplate. It is recommended that QLOGIN tags be located just after the BODY tag or within the HEAD tag.

Never place a QLOGIN inside a QFIND or HTML list derivative.

Consider placing the QLOGIN inside a QIFBLOCK when database access is dependent upon data validation or other logical operators.



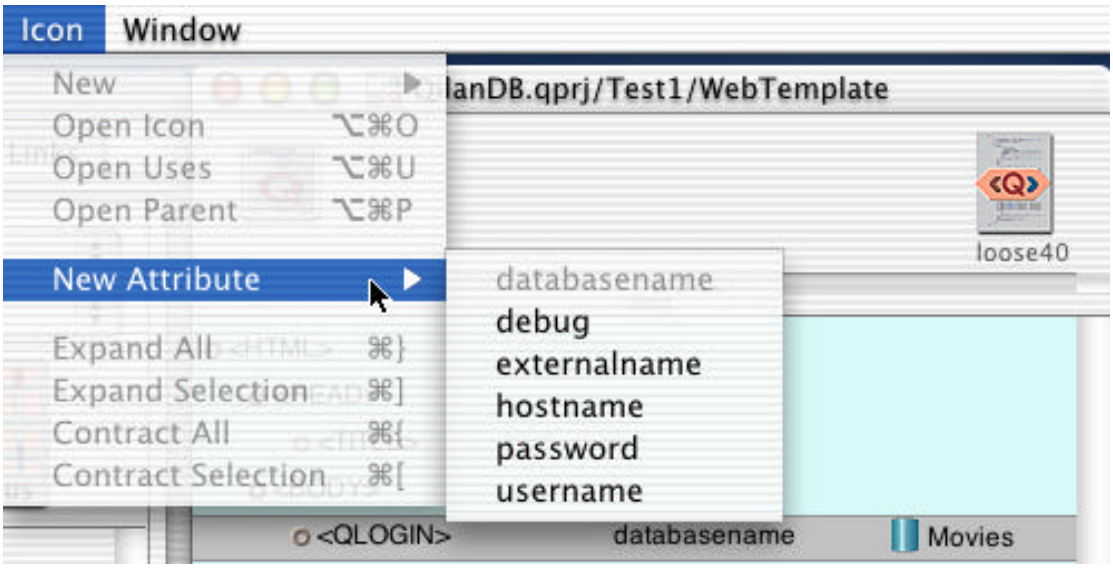
The screen shown above uses a flag abacus expression to determine whether a login is required. Note that however, if the abacus expression itself must access the database, this construction will fail.

Attributes

DATABASENAME: The name of the Access icon.

OVERRIDE: Accepts a framework icon formatted as a 'dictionary'. The dictionary keys/values will be used (when they exist) in place of the login panels' key words and input values.

DEBUG: Logs SQL sent to the database (when checked). See Project Settings for details.



QWHILE

This tag causes all tags (‘Q’ tags and HTML) within it to repeat (or loop).

Attributes:

ICON: A field or abacus icon, which returns a flag. When the flag is evaluated in the affirmative (Y, Yes, T, True or 1) QWHILE will repeat all tags contained within it, otherwise QWHILE will be ignored.

Qilan prevents loop constructions from ‘running amok’. In the Project Preferences, there is an iteration limit. A default setting (200) is automatically invoked in the event your loop does not behave properly.

In this example, 10 records are automatically updated. We start by assigning the value of zero to the field, ‘count’. Next we enter our loop. QWHILE is evaluated and returns, Yes. At this point, we re-assign the value of count to ‘count +1’, then perform our update. Now the loop is re-evaluated. The loop will continue 10 times until the value of count is greater than 10.

<QLOGIN>	databasename	Movies
<QASSIGN>	icon	count
	value	0
<QWHILE>	icon	count≤10
<QASSIGN>	icon	count
	value	count+1
<QUPDATE>	dataflow	DataFlow

QWHILE is very useful modifying blocks of text. For example, let's say you want to remove linefeeds (LF) from a block of text. Shown below is the final WebTemplate construction. Two Framework fields, 'export_layout' and 'export_layout_temp' are used as assignment variables.

The first action is to assign the text block, '10_export_layout' to the field 'export_layout'. This gives us a starting point. The abacus outputting the text block is shown below.

Next, we enter our loop and evaluate 'export_layout'. Does the text contain a line feed? We use the abacus expression, Character __ to make this determination. ASCII 10 represents a line feed.

Knowing the text block contains a LF we extract all the text preceding the LF. At this point, we need to hold this text in temporary limbo until the next assignment is performed. Therefore this abacus expression is assigned to 'export_layout_temp'. The reason is simple, it will be used to determine the value of the re-assigned 'export_layout'.

Now we are ready to re-assign ‘export_layout’.

```
( export_layout_temp followed or ( substring starting at ( locate_10 +"1" ) length ( length export_layout
) from export_layout ) )
```

Note how ‘export_layout_temp’ is used to re-write the text block by circumventing the line feed. What results is a new text block that is ready for re-evaluation.

If you find this difficult to grasp, just imagine peeling an apple. What’s going on is that a section of text, from the first character to the line feed (minus one letter), is put aside. Next, the rest of the text, from the line feed (plus one character) to the last character is also put aside. Lastly, the two text strings are combined. We keep doing this until there are no more line feeds left.

How fast is this? Actually, quite speedy. On blocks of text of 1000 characters or so, you probably won’t notice anything.

QLOOP

This tag is used to iterate through the contents of an array or dictionary. Its icon attribute must be a framework field assigned as an array or dictionary, or else Qilan will return an error.

The VALUE, KEY and INDEX attributes are used to extract specific elements for display and/or additional data manipulations.

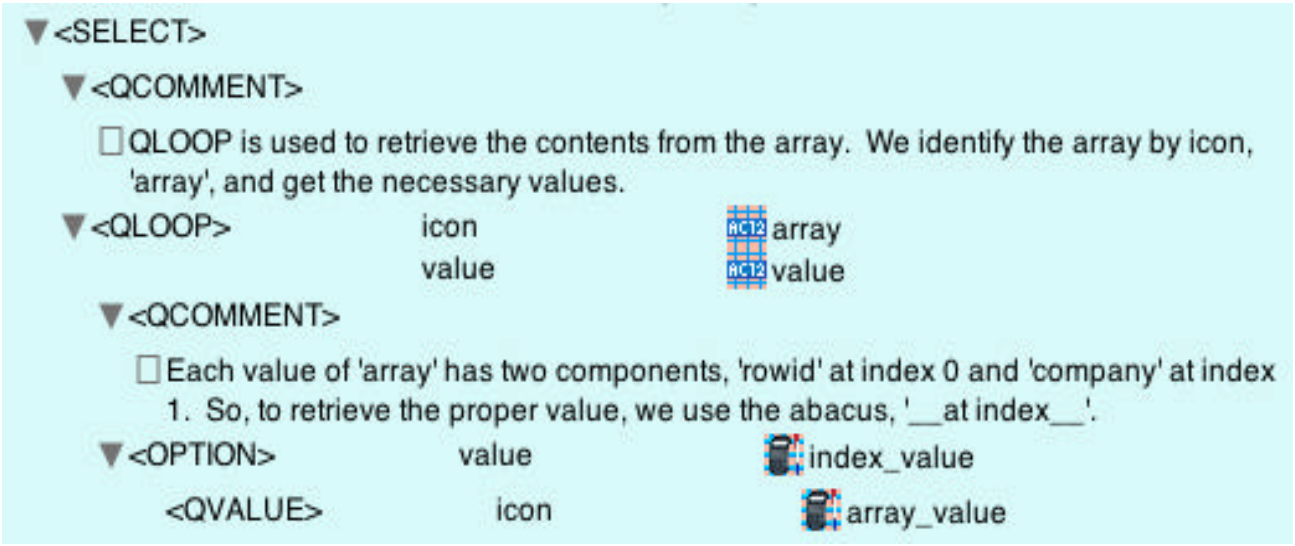
Attributes

ICON: A framework field assigned as an array or dictionary. A field assigned or valued as any other type of structure (e.g. text, number, date, etc.) will return an error. This is the structure that will be iterated.

VALUE: A framework field to which the value element of the array or dictionary is to be assigned.

KEY: A framework field to which the key element of the dictionary is to be assigned. This attribute cannot be used with the INDEX attribute.

INDEX: A framework field to which the index element of the array is to be assigned. This attribute cannot be used with the KEY attribute.



In the example shown above, QLOOP is used to iterate through an array. Note that QLOOP is placed within a SELECT and an OPTION tag is within QLOOP. For each index entry, a new option value is shown. Observe how the index and value elements are extracted as follows:

```
( value at index "0" )
```

In this example, the array value is composed of two elements, the index and its associated value. The abacus, '__ at index __' is used to extract the appropriate element.

```
( value at index "1" )
```

QCREATE

This tag unconditionally creates a new database record in a specified table. The tag uses a DataFlow icon to insert field or abacus values into fields in the target table.

Attributes

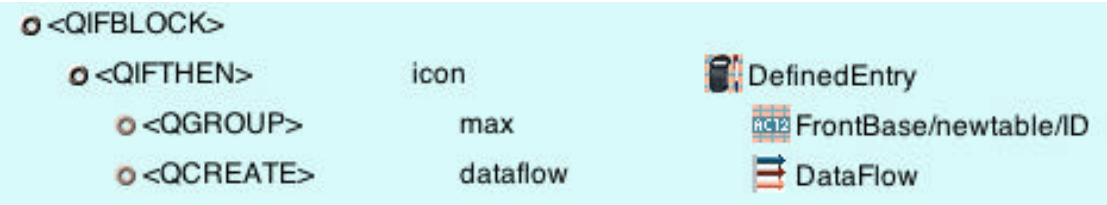
DATAFLOW: A dataflow icon dragged from the palette (Links Tab).

Notes

The identification of the database and table are specified by the Relationship, as used by the DataFlow. Source <–> Target relationship links are ignored.

At least one source field/abacus in the DataFlow is required otherwise the QCREATE tag will be ignored.

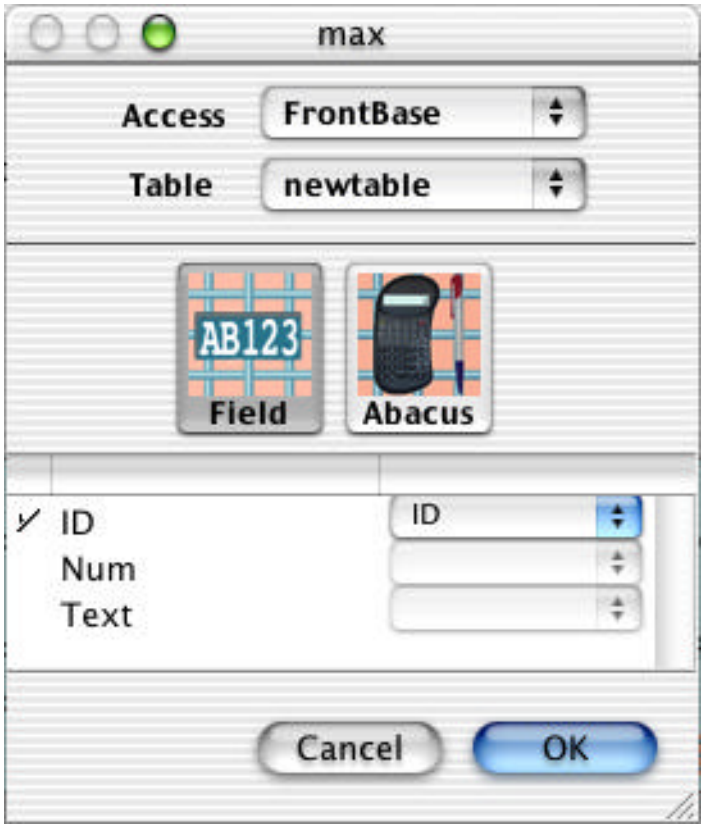
In the following example, a new record will be created with a unique record id (key). This is akin to the traditional approach of looking up the existing maximum record id then incrementing the count by one.



The WebTemplate section contains an IFBLOCK to determine whether a new record should be created. This is our 'validation' and is constructed to evaluate the presence of data in two fields, 'TextData' and 'NumericalData':

```
( ( ( length TextData ) > "0" ) and ( ( length NumericalData ) > "0" ) )
```

If the result of the validation is ‘Yes’, then we proceed to obtain the maximum value of the current record id. This is performed with a QGROUP with a max attribute. The value is then assigned to the Framework field, ID.

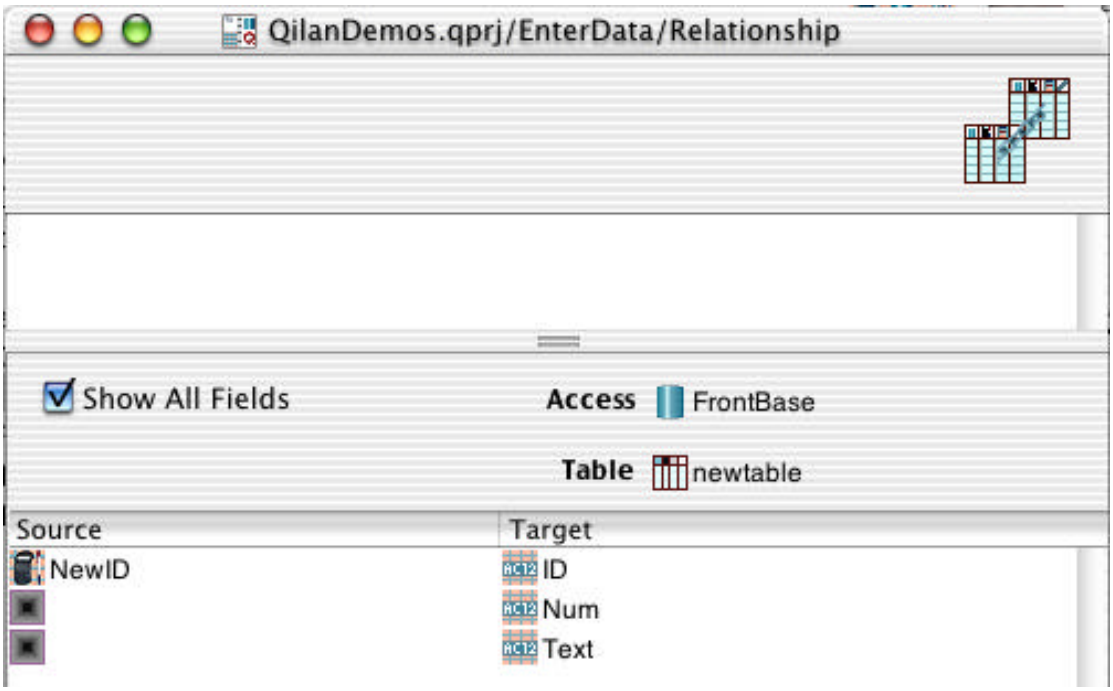


The next step is to increment the record id. This is done in the Framework using the following abacus expression.

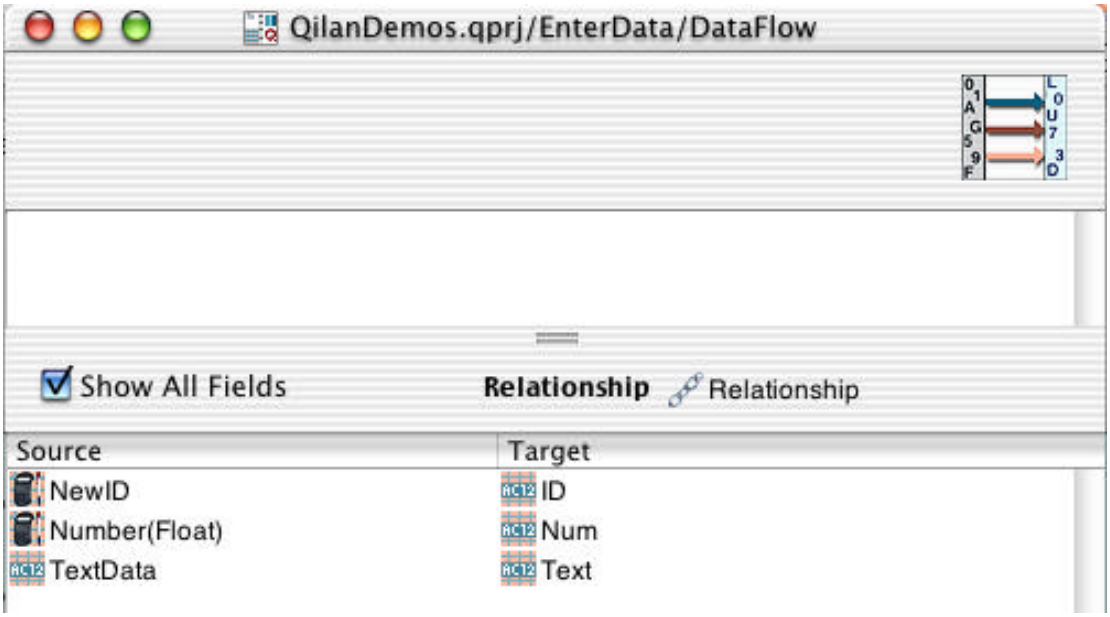
```
( integer ( ( ID defined or "0" ) + "1" ) )
```

Note how the ID is converted to an integer. Some databases require explicit data typing or the attempt to insert this value will fail. We are now ready for the relationship and dataflow specification necessary for the QCREATE.

QCREATE uses a DataFlow, but each DataFlow must be based on a relationship. The relationship shown is used for other linking tasks, but it will suffice for a QCREATE as only the identification of the Access and Table is used.



Finally, the DataFlow inserts our data into the new record. Note how the newly incremented record ID abacus is used.



Designing web pages can be a challenge, but that's another story. QCREATE will insert new database records each time a web page is loaded, or re-loaded. Unlike a traditional database, when a web page is re-loaded, all the Qilan 'Q' tags are re–processed. Unless the QCREATE is properly constructed inside QIFBLOCKs, you might find yourself with lots of unexpected records. An easy way to avoid this problem is to use QUPDATE with a create attribute. The next section discusses this very useful 'Q' tag.

QUPDATE

This tag modifies existing database records based on relationship link(s) or creates records when relationship link(s) does not exist in the target table. A DataFlow icon is used to insert field or abacus values into fields in the target table. The identification of the database and table are specified by the Relationship, as used by the DataFlow.

Attributes

DATAFLOW: A dataflow icon dragged from the palette (Links Tab).

CREATE: This attribute causes the QUPDATE to create a record when the relationship target link(s) do not exist in the target table.

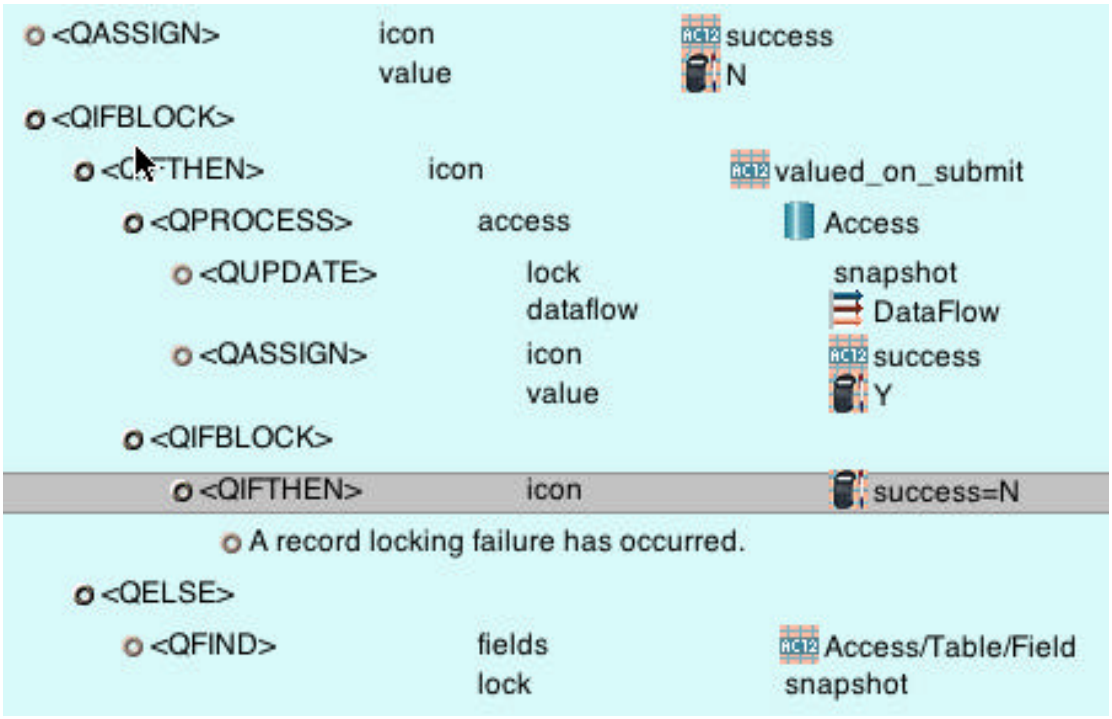
LOCK: The name (text) used to retrieve a snapshot created by an enclosing QFIND. Names are case sensitive.

Specifying a lock attribute instructs Qilan to use optimistic record locking. The QUPDATE lock name and snapshot will be compared to the QFIND's lock name and snapshot, if they are not the same, the update will fail. By default, a failure (error) will abort the WebTemplate without an error message. Refer to QTRY and QCATCH for more details as to how to produce custom error messages and/or other output strategies.

A field is valued upon submit and the FORM action returns the same page. The QFIND lock attribute snapshot is then available for use by the QUPDATE.

The field, 'success' is assigned 'N' prior to starting the QPROCESS. This value is maintained unless the QUPDATE is successful.

An example of how to design optimistic record locking:



Notes

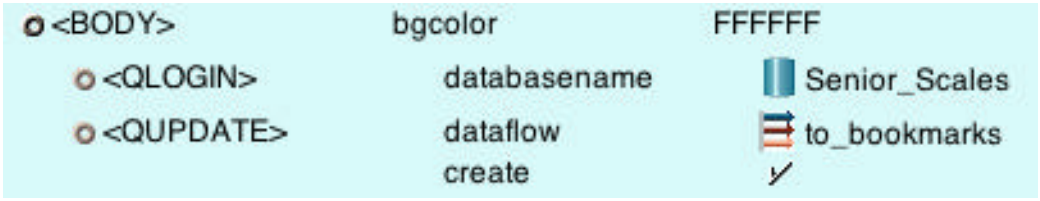
QUPDATE requires that all source values used as links in a relationship be defined. If any values are undefined, the QUPDATE tag is ignored and an error message will be returned.

Unexpected results may occur if any of the defined relationship source links are empty. In this case, the QUPDATE will be triggered and the DataFlow will be linked by the remaining defined values.

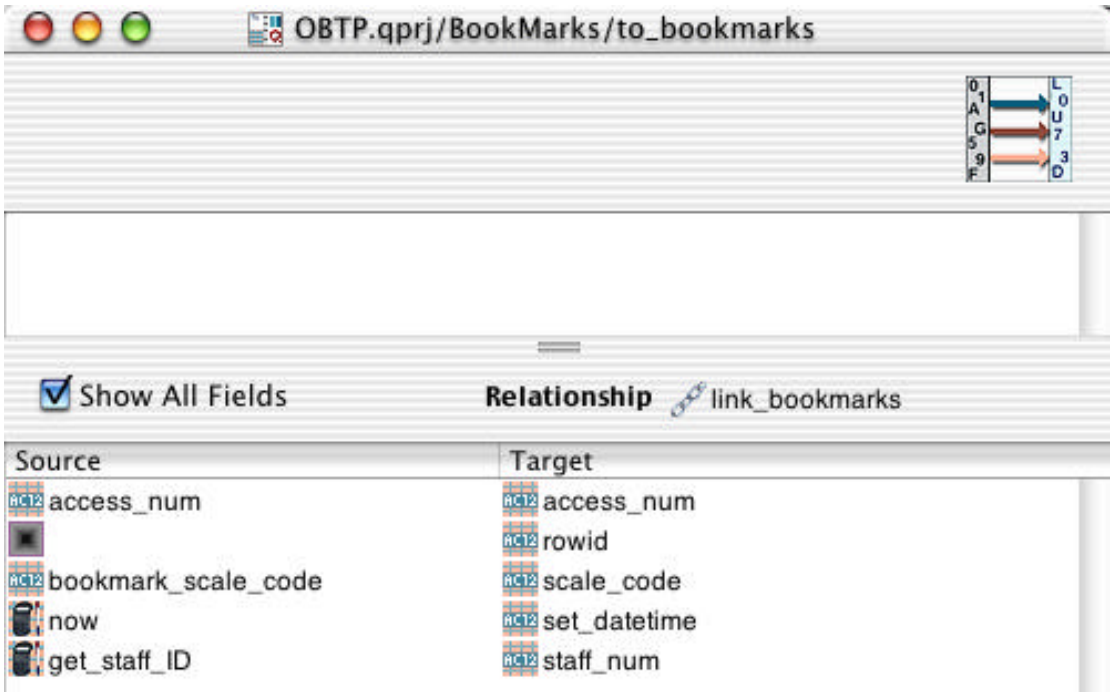
At least one source field/abacus in the DataFlow is required otherwise the QUPDATE tag will be ignored.

A QUPDATE may be used without a DataFlow when modifying the current record. See 'Special Case' as the end of this section.

In the following example, we will use data obtained from the database to create a link, then use a QUPDATE to create a new record in another table if one does not exist. Existing records will be updated. Here's the final construction:



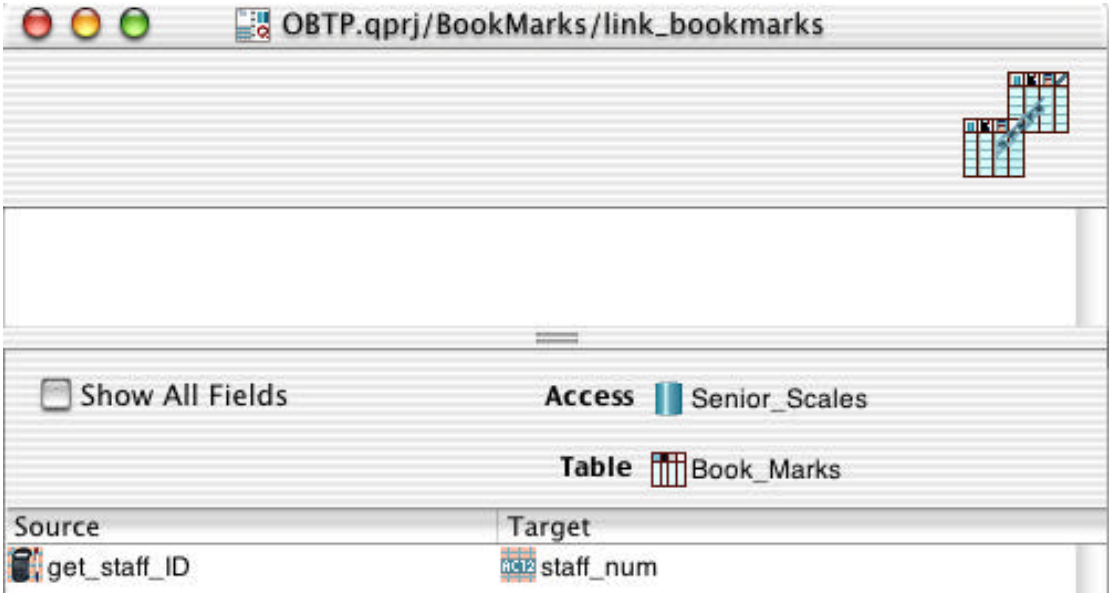
The BODY attribute, 'bgcolor', sets a background color. In this case, the value, 'FFFFFF' means white. Let's take a look at the DataFlow and see what this does.



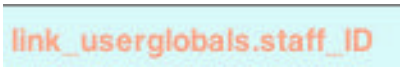
As it turns out, the DataFlow does quite a bit. Each time the QUPDATE is processed, Qilan will create/update four fields. The field, 'access_num' is posted to this form, so it will always be defined. Take special note of the abacus, 'get_staff_ID'. This data is not present on this form, but is obtained from the database before the update is performed. Now, let's take a look at the relationship, 'link_bookmarks'.

The relationship links the value, ‘get_staff_ID’ with the ‘staff_num’ in the ‘Book_Marks’ table. With reference to the DataFlow and the QUPDATE, the meaning is to create a new staff bookmark if the staff_num does not exist, otherwise update the staff bookmark.

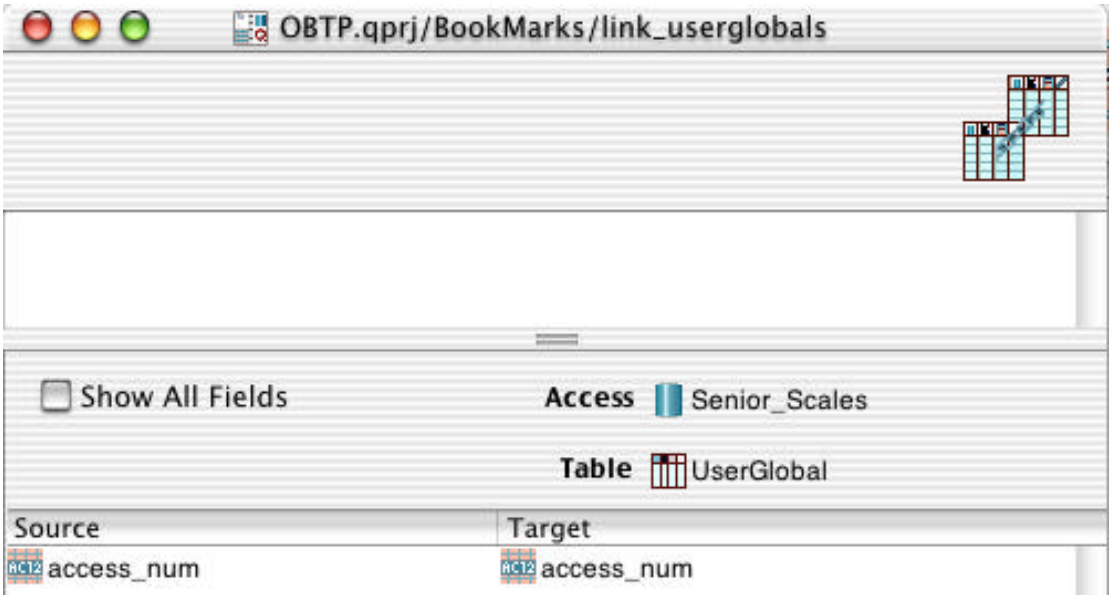
Observe that the value used for the relationship link must also be used in the DataFlow.



Where did the value of ‘get_staff_ID’ come from?



Staff_ID is a value obtained from a database table. A relationship has been created that uses a form value (in this case, access_num), to retrieve the value.



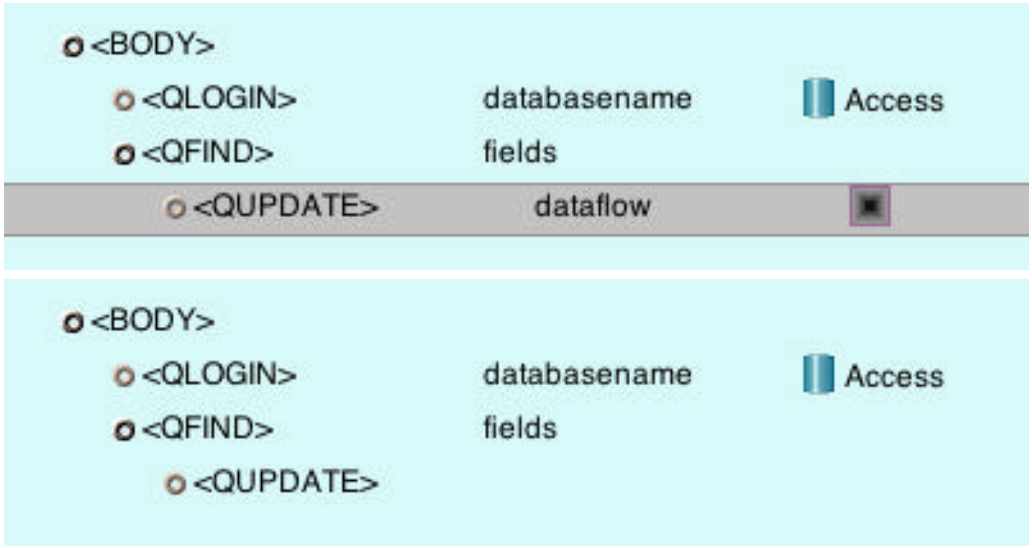
In summary then, when this form is accessed, the field value of ‘’access_num’ is valued. Qilan then retrieves the value of ‘staff_num’ based on the link, ‘assess_num <-> access_num’ in the table, ‘UserGlobal’. Now, the link to BookMarks can be established and the QUPDATE performed.

QUPDATE, with a create attribute, causes a record to be created when a link does not exist, but will only update the record if a link is found. This feature is very useful for updating forms on the web. Now, no matter how often a user clicks, “re-load”, the same record is merely updated.

Modifying the Current Record (Special Case)

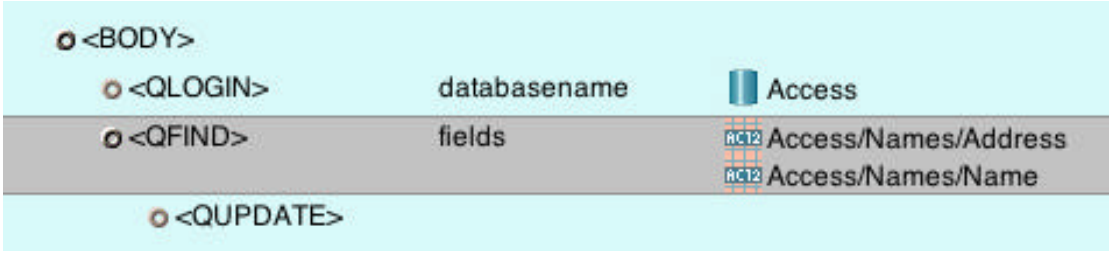
Qilan does not require the specification of a DataFlow when modifying the current record. This is a special case circumstance where the identification of the target table and field(s) is inferred by the enclosing QFIND.

To prepare a QUPDATE for use without a DataFlow, the DataFlow attribute must be cleared. Leaving the DataFlow attribute undefined (black hole) will result in an error. Drag the QUPDATE tag to the WebTemplate, the line will be highlighted as shown below. Click once on the line to select the attribute, then choose Clear from the Edit menu.

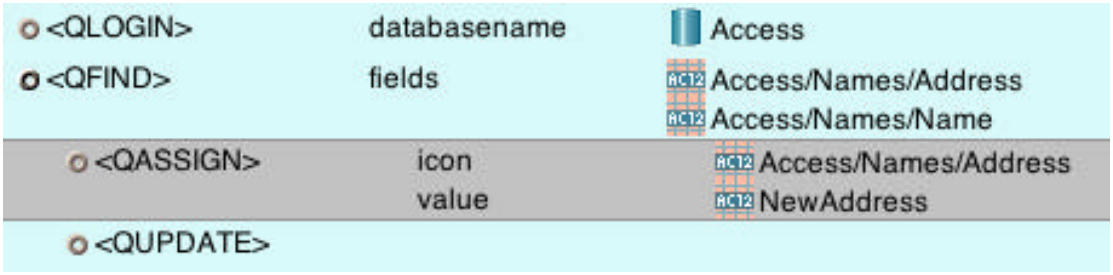


The next step is to select the table fields to be retrieved. Only fields retrieved can be modified.

Insure you select all fields checked as Primary Keys (Table -> Field specification window), otherwise the update will not occur.



To modify a retrieved field, its value must be reassigned. This is performed with the QASSIGN tag.



Note how the table value is used as the assigned ‘icon’ and the new value (NewAddress) is assigned to ‘value’. Qilan reads this as, ressign the icon (Access/Names/Address) to the value (NewAddress), then update the record. This will be performed for each record retrieved.

Only fields retrieved and reassigned are modified.

QPROCESS

The QPROCESS tag groups a transaction. Processes that generate SQL within the QPROCESS will be committed at the end of the process. If there is any failure within the process, all SQL that has queued will be rolled back.

A transaction is defined as any ‘Q’ tag that interacts with the database. Typically however, you will want to do several things and group them as a single transaction. For example, delete, find and update data. To maintain database integrity, you will want to insert that all three actions are performed. If the update fails for any reason, you will want the delete rolled back. This is where QPROCESS comes to the rescue.

Attributes

ACCESS: This attribute indicates which database is will get the commit at the end of the process.

Note that nesting QPROCESS tags to handle multiple databases is specifically NOT recommended. The "two-phase commit" which is required to handle this is not implemented. Multiple QPROCESS tags should be sequential.

LEVEL: This is the SQL isolation level for the transaction, and it can be set to read_uncommitted, read_committed, repeatable_read, or serializable. Refer to SQL Isolation Levels elsewhere in this manual.

QTRYBLOCK

This tag is used to isolate errors generated by the system, Qilan, Osmosis Gateway, webserver, JDBC driver or database as the result of enclosed tags.

QTRYBLOCK is used as the parent tag for error evaluations. All subsequent tags must be within the scope of a QTRYBLOCK. Tags used to construct logical error evaluation statements are QTRY and QCATCH.

A WebTemplate may contain as many QTRYBLOCKs as desired. Nested QTRYBLOCKs are permissible.

Under normal circumstances, when Qilan encounters an error generated by the system or database, the remainder of the WebTemplate is aborted and the user receives an error message. This message is usually cryptic and often not very helpful (except to the exceptionally informed). QTRYBLOCK allows these errors to be trapped and affords the designer the opportunity to display alternative messages, stop processing in a controlled manner, continue with the WebTemplate or even redirect processing.

Attributes

NONE. When a QIFBLOCK is placed on the WebTemplate, a QTRY is automatically inserted.

QTRY

This tag is used to evaluate enclosed executable processs (tags). If an error is encountered, the attributes associated with this tag will be defined; otherwise they will be undefined. QTRY tags must be enclosed by a QTRYBLOCK.

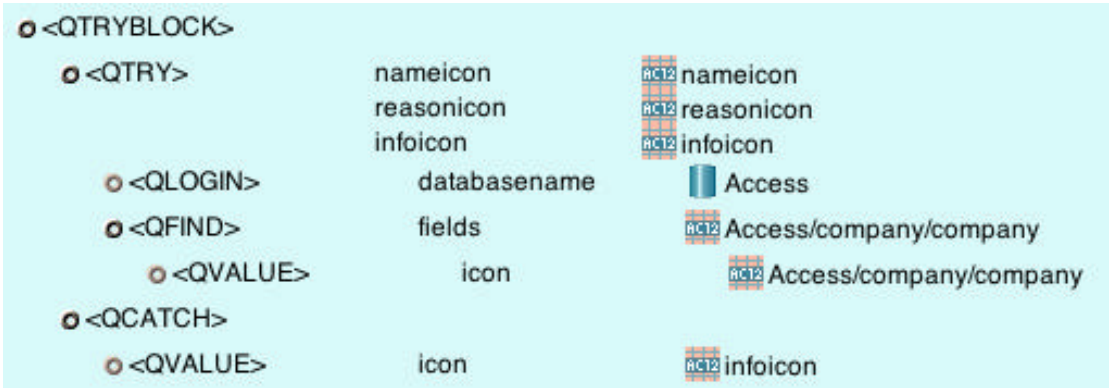
Attributes

INFOICON: A detailed error message usually containing a 'stack trace'. This output is most often helpful to designers or programmers as it provides extensive information regarding the error.

NAMEICON: The name of the error type. Many errors are the same type, so this output may not be helpful for debugging. This output may be helpful when the application prompts users to report errors.

REASONICON: A concise statement as to the actual cause of the error. Where possible, this message will be in plain English. If your application prompts users to take some action in response to error conditions, this output will likely be the most helpful.

In this example, we want to output any errors that are produced when we logon to the database and attempt to retrieve records. Note how a QCATCH is used to obtain the error output.



For an example, if the database were not available, the following error messages would be produced:

Infoicon: `{ "__JavaBridge Stack Trace" = "java.sql.SQLException: Database Company not started on 192.168.1.3.\n\tat com.openbase.net.OB_NameServer getPortNumber(Unknown Source)\n\tat com.openbase.net.b.a(Unknown Source)\n\tat com.openbase.net.b.a(Unknown Source)\n\tat com.openbase.jdbc.e.(Unknown Source)\n\tat com.openbase.jdbc.f.(Unknown Source)\n\tat com.openbase.jdbc.j.connect(Unknown Source)\n\tat java.sql.DriverManager.getConnection(DriverManager.java:517)\n\tat java.sql.DriverManager.getConnection (DriverManager.java:177)\n\tat com.commongrnd.jdbc.CSJDBCAdaptor.openChannel (CSJDBCAdaptor.java:187)\n\tat com.commongrnd.jdbc.CSJDBCAdaptorContext.openChannel (CSJDBCAdaptorContext.java:74)\n\tat com.commongrnd.jdbc.CSJDBCAdaptorChannel.openChannel (CSJDBCAdaptorChannel.java:118)\n"; attributes = ("FIELDS=\"\"", "TABLE=\"Access/company\""); errLineNo = 269; tag = QFIND; }`

Nameicon: `java/sql/SQLException`

Reasonicon: `Database Company not started on 192.168.1.3.`

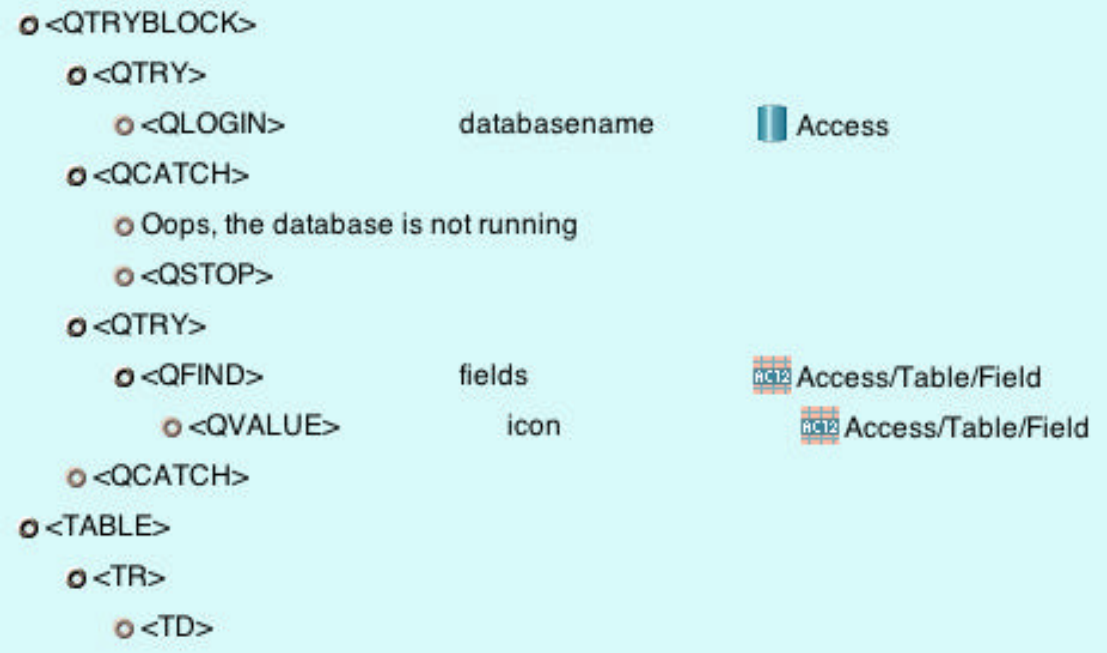
By ‘catching’ the error, Qilan will output the error message then continue with WebTemplate processing. Read on to the next section to learn more about QCATCH.

Error attributes are assigned to a framework field(s), which can then be further evaluated by a subsequent QCATCH.

A QTRYBLOCK may contain as many QTRY tags as desired, however only QCATCH tags that immediately follow a QTRY will only reference the preceding QTRY. The example on the following page demonstrates this.

In the following example, we are testing for errors inside two QTRY tags. The first tag contains a QLOGIN. If this tag results in an error, the following QCATCH tag will be executed. Note that when no QCATCH attribute is selected the QCATCH will always be evaluated. In our example, we output some text, then stop processing. By default, QCATCH will replace system errors with user output, then continue processing the WebTemplate. To stop processing (as shown here), we must use a QSTOP tag.

The second QTRY tag retrieves database records. Should the QFIND error, the error will be skipped and the remainder of the WebTemplate executed. An empty QCATCH is used without an attribute.



QCATCH

QCATCH, when used in conjunction with QTRY, evaluates error icons. The default behavior of a QCATCH is to suppress system errors and continue with WebTemplate processing. QCATCH tags must be enclosed by a QTRYBLOCK and preceded by at least one QTRY.

QTRY tags may be followed by one or more QCATCH tags, each testing for different error conditions.

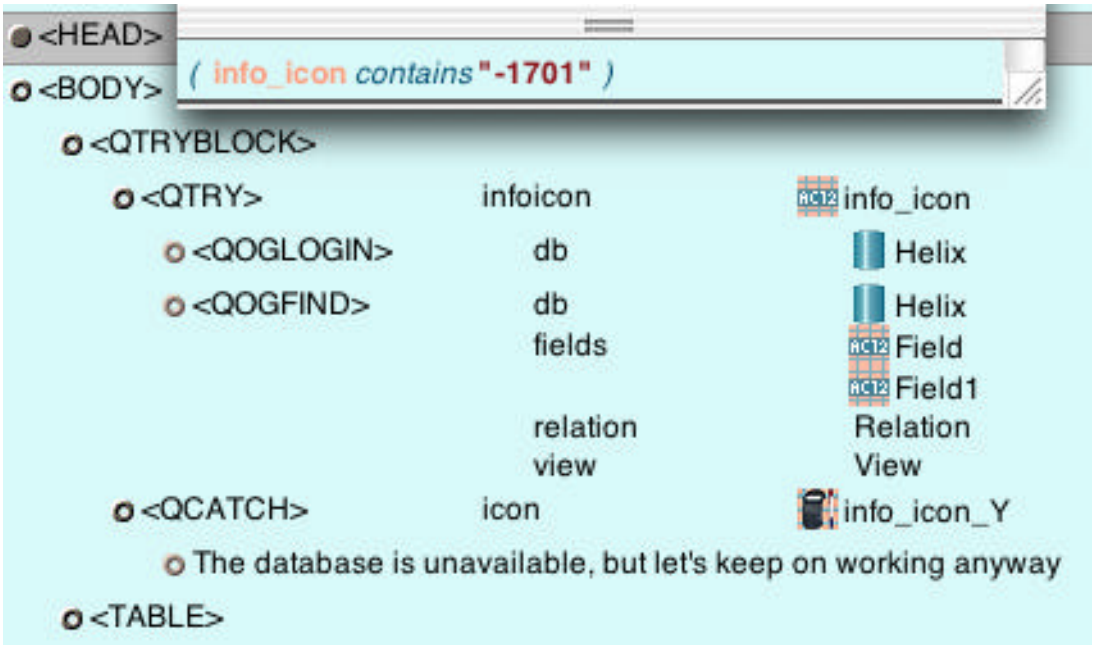
Attributes

ICON: A boolean abacus icon used to evaluate a previous QTRY attribute.

When a system error is detected from a preceding QTRY, tags enclosed by QCATCH will be output. Using the icon attribute, QCATCH can be selectively controlled.

The QCATCH icon attribute only evaluates attributes from the QTRY.

Using the Osmosis Gateway, sometimes AppleEvent errors are returned. One such error is a –1701, which roughly translates to the database not being available. Here's how to trap for this error:



The text enclosed by QCATCH will only be output when the info_icon contains –1701. When this occurs, the remainder of the WebTemplate will be processed. If any other error occurs, the WebTemplate will be aborted.

QSTOP

This tag is used to stop WebTemplate processing with or without an error output.

Attributes

REASON: A user defined icon value that will be displayed along with a system error. Note that QSTOP will generate a system error by itself.

When a QSTOP is used without the reason attribute, the WebTemplate will abort quietly. Processing tags and/or HTML which follow a QSTOP will be ignored.

QVALUE

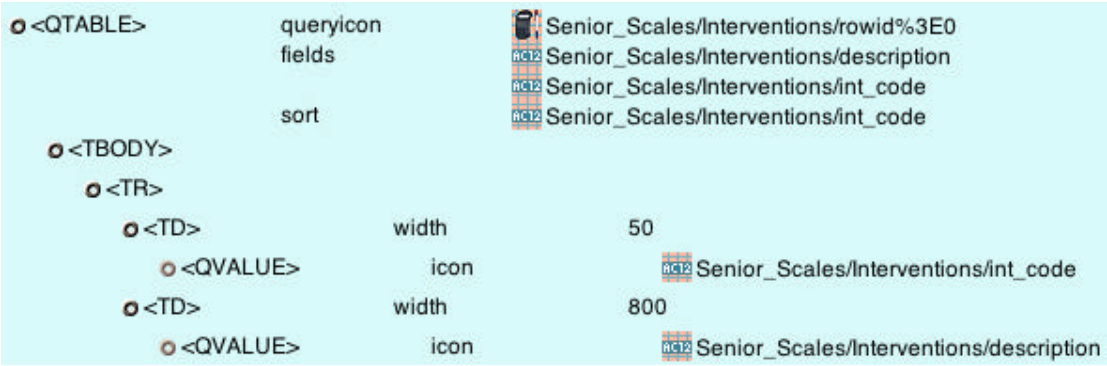
This tag displays the value of an icon.

Attributes

ICON: Any field or abacus icon may selected from the palette (Values).

This is probably the most used ‘Q’ tag. Its sole function is to display icon values. When an abacus or field icon is placed into its ICON attribute, the value will be passed to the browser for display.

An undefined ICON attribute (black hole), will result in the error, ‘Attribute Required’. Undefined icon values will be passed as undefined.



In the example shown above, field values are retrieved from the database using a QTABLE. QVALUEs are used to display these values within the table structure (TD).

QRUN

This tag passes commands, parameters and standard input (stdin) to the system and returns standard output (stdout), and optionally, standard errors (stderr). Commands to the system may be simple requests, such as ‘ping’, mailto, ftp, etc., or more complex routines executed by programmed scripts written in Unix, Perl or other low-level languages.

Attributes

COMMANDLINE: The location of the script to be run and optional arguments to be passed to the script. You may directly type the CommandLine on the WebTemplate, or use the output of an abacus or field.

The commandline consists of a series of "words", separated by whitespace. Whitespace is spaces, tabs, carriage returns or line feeds.

Each word either:

(a) contains no whitespace; or
(b) is appropriately "quoted".

There are three ways to quote:

- Enclose the entire word inside a pair of double quote (") characters;
- Enclose the entire word inside a pair of single quote (') characters; or
- Preceed each whitespace character by a backslash (\l) character. Note, the backslash character *always* quotes the next character, so be sure to double it if you want a backslash in your command line.

The first word in the command line is the program (or script) to execute. It must be a *complete* path name. For example, "ls" will not work; it must be "/bin/ls".

The second and subsequent words are the arguments to the command, as defined by the command. Empty arguments (eg. "") are illegal.

INPUTICON: Input data to be passed to the executable script. The system will interpret this as standard input (stdin). INPUTICON may be omitted if no standard input is to pass to the executable. INPUTICON may be a field or abacus.

STATUSICON: The script status returned by the system. The status will be placed into a Framework field icon. Most scripts return '0' if successful. If the timeout is reached before the script output finishes, STATUSICON will return '60'.

TIMEOUT: The length of time Qilan will wait for the script to return an output, in seconds. If a TimeOut attribute is not selected, the default timeout setting in the Project Settings will be used. An undefined or zero TimeOut setting will instruct Qilan to wait forever and therefore should only be used when necessary.

PATIENCE: The length of time Qilan will wait for the system to execute the script, in seconds. The default value is five seconds.

ERRORICON: The value of standard error (stderr) as returned by the system. The error will be placed into a Framework field icon. If this parameter is omitted (not placed into Framework field), stderr will be passed directly to the browser.

OUTPUTICON: When a script is configured to output a response, Qilan will accept that response and either place it directly into the HTML or into a Framework field icon. To have the response placed into the HTML, omit this attribute; otherwise use a Framework field icon.

Notes

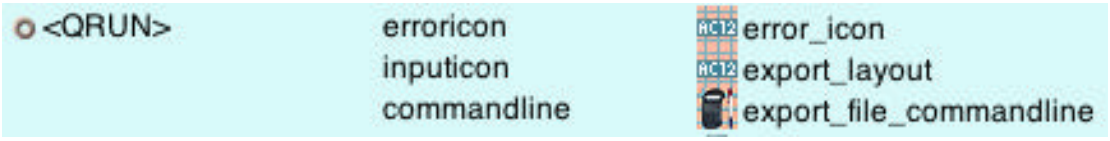
All scripts will be executed as the same user as the webserver, typically ‘www’. Applicable permissions and access levels apply.

QRUN is exceptionally powerful as it provides the ability to run system scripts and other executables directly by internet users. Thoughtful design, thorough testing and a complete security review should be mandatory prerequisites before deployment. Be especially cautious of passing standard input on the command line as a parameter. Always use INPUTICON for standard input.

Do you know what a ‘thread’ is? This is not a rhetorical question. When you initiate an action via QRUN, all subsequent actions are connected or linked. Why does this matter? Let’s say you start a script that takes 600 seconds to finish - a big print job or extensive search. You set QRUN’s timeout to 600 seconds. After 300 seconds however, the web server’s default timeout, the browser reports an error. Qilan.cgi is terminated along with all the scripts initiated by QRUN. What to do? Fortunately, the answer is rather simple. Rather than starting an action directly via QRUN, use the Unix command ‘batch’. This command places your script in a queue for execution as system levels allow. Once this is done, control is immediately passed back to Qilan.

More information concerning the use of ‘batch’ can be found in the Qilan Extras folder or by typing, “man at” using the Mac OS X Terminal application.

In this example, we will use QRUN to name and create a file. The file will contain data from the database and be located on an ‘exported’ volume. This will make it retrievable by a foreign system.



The attribute ‘erroricon’ is assigned to the Framework field, ‘error_icon’. This is done to capture any error, thus preventing the browser from displaying the error. Of course, we could put the error in a database for later analysis or test for the presence of an error. Typically, error responses are numeric, where zero means no error.

Commandline is probably the most important parameter. Let’s see what this looks like:

```
( "/Public/MakeFile" followed by" " followed by"/Network/Servers/Notes/" followed by (
( uppercase link_op_note.last_name_helix ) defined or get_mrn_unit_number ) followed by (
link_op_note.surgery_date formatted by"%m%d%y%H%M" ) followed by".opn" )
```

Lots of stuff here. “/Public/MakeFile” tells the system to locate an executable file in the directory, “/Public”. The file’s name is “MakeFile”. What does this do? It merely tells the system to create an empty file. The file is a simple Unix shell script that looks like this:

```
#!/bin/tcsh
#Invoke 'cat', create a file and pass it standard input.
/bin/cat - > $1
#Be sure to end this file with a carriage return.
```

Writing Unix shell scripts requires knowledge of Unix commands and specialized formatting. A recommended reference book is Unix Complete, by Dyson, Kelly-Bootle and Hellborn.

The only line that does anything is the one without the pound character (#). Lines preceded by the pound sign are interpreted as comments. The line '`/bin/cat - > $1`' means to start the Unix command 'cat', located in '/bin', create an empty file in a specified location and whose name is passed in the parameter '$1', then finally write stdin (inputicon) to the file.

Where's the location?

```
( "/Public/MakeFile" followed by" " followed by"/Network/Servers/Notes/" followed by (
( uppercase link_op_note.last_name_helix ) defined or get_mrn_unit_number ) followed by (
link_op_note.surgery_date formatted by"%m%d%y%H%M" ) followed by".opn" )
```

Unix treats spaces as separators. Therefore the next section of text is where the file is going to be located, /Network/Servers/Notes/. This path actually denotes a mounted volume of another machine. The file's name follows. The name is derived from the database table and therefore makes it unique for each file created. A date (properly formatted) and suffix are included.

Now for the file's contents. The value of the Framework field, 'export_layout' will be passed. The QRUN attribute, 'inputicon' is interpreted as standard in (stdin) by the system.

QHTTP

This tag is functionally equivalent to the header tag as described in RFC 2068 (Hypertext Transfer Protocol -- HTTP/1.1).

QHTTP header attributes specify metainformation: that is, information about the page (webtemplate), not information that is contained on the page. For example, http header attributes might be used to instruct the browser how to interpret page data (e.g., an application document) or modify web server caching.

QHTTP can be used with or without a HTML tag. We strongly urge you to familiarize yourself with the large number of attributes and extensive capabilities of the http header. More information can be obtained at http://rfc-2068.rfcindex.net/.

Attributes

The number and variety of header fields associated with QHTTP are extensive and are detailed by RFC 2068. Before you attempt to use a specific field or enter an attribute value, consult the documentation.

Note that QHTTP header fields specify metainformation: that is, information about an object, not the information that is contained in the object.

Webtemplates that define QHTTP header fields/values will replace those associated with the webserver and/or the HTML tag. By default, Qilan will automatically set the header attribute, “content-type”, to the value, “text/html”.

What follows is an edited excerpt from AOL.Webmaster.info that helps to explain the concept and usage of the http header. We hope you find it useful.

HTTP defines how web pages are handled and in what way they are delivered over the Internet. It also includes any information about the objects that are needed by proxy servers or a user's web browser. Currently, HTTP 1.1 is an IETF Standard.

The process of calling a web page from a server involves several steps. First, a client makes the request of a web server. The web server receives the request and checks to see if the page exists. If the page does not exist, an error message is returned to the client. If the page does exist, the web server then determines if there are any special processing needs, as would be the case with Qilan. The web server then prepares to send the file by adding any default or custom header information to the beginning of the requested page. Finally, the page is served to the client.

HTTP headers are used to define a variety of qualities about a web object. For instance, if a developer sets the cachability of a web site, HTTP headers such as Cache-Control, Expires, and Last-Modified would be used. HTTP headers can also include custom header information. By adding custom headers, a developer can create an HTTP header named Copyright and add site-specific copyright information to it. Because headers can be set in a variety of ways, the developer must choose the combination of headers and values that are appropriate to the needs of the web site.

It is important to note that an HTTP header is not the same as the META tag. META tags send information within the HTML page, while HTTP headers send information about the HTML page.

There are two ways to set the HTTP header information. The first method is to set an HTTP header via the web server. All outgoing pages will have a default set of headers attached to them that are readable by other web servers, web caches, and web browsers. The second method is to set an HTTP header for each specific page. With this method, a web page is created with Qilan, and then the page is scripted so that it will send the appropriate header information.

An Example: The VARY Header Field

Developers who wish to serve different content for different browsers should use the VARY header field to specify the set of request-header fields used in making the determination.

If you are using Qilan to serve different code to different useragents (browsers), try setting your web server to respond with the VARY header. This will ensure that all HTTP/1.1 servers will serve the proper content to your users. If you are simply using JavaScript after the page has been downloaded then the actual page downloaded will be the same for each useragent.

If, for example, you wish to serve content in different languages, use VARY: Accept-Language. When serving content based on web browser, use VARY: User-Agent.

Building Queries with SQL

The QGROUP and QFIND tags allow the designer to create and pass SQL queries directly to the database. This feature enables complex statements to be submitted. It should be used when the nature of the SQL statement is beyond the built-in functions provided for by Qilan or if a special or unique SQL syntax is called for by the database.

To submit a SQL statement use the 'query' attribute associated with the QGROUP or QFIND tag. Note that all the QFIND derivatives also allow for SQL.

You may enter SQL directly on the WebTemplate, as a text string, in an abacus or as data from the database.

Building 'raw' SQL expressions with abacus operators is governed by a set of special syntax rules. First, use of boolean or comparison operators that do not output text (e.g., AND, OR, Equals, Contains, etc.) will result in errors and should be avoided, however other string manipulators, such as Followed by, If Then Else and Locate Substring may be used. For example, where literal text is bracketed {},

({externalname} followed by {=} followed by {"city"}) works, but
({externalname} = {'city'}) does not.

({externalname} followed by { LIKE } followed by {"} followed by (acquire [icon]) followed by {"}) works, but

({externalname} starts with (acquire [icon])) does not

Note that the quotation mark is typed, even though the abacus parameter will appear to enclose the value in quotation marks.

For assistance in forming SQL statements, we suggest you refer to Chapter 12 of, "A Guide to The SQL Standard, Fourth Edition" by Chris J. Date and Hugh Darwen.

Please note that some databases may not support all database keywords and functions. For example, OpenBase supports 'sounds like', using the syntax 'A? B', however use of this syntax with FrontBase will result in errors.

The keywords SELECT and FROM are omitted, as the query is integrated into the QGROUP and QFIND tags. Your query statement should be WHERE {your statement}. The keyword, WHERE, is automatically submitted.

When a database field name is referenced, you must use the EXTERNAL name. External names can be located by double clicking on the field icon inside a Table window. Some databases enforce case sensitivity for external names. Check the database documentation.

Text strings must be quoted, but numbers cannot be quoted. The placement of parentheses may be database specific. We urge you to refer to your database documentation for specific SQL variations as well as database specific functions.

Building Queries with Qilan

Qilan abacus operators are used to query database data by linking together abacus operators and program variables so as to create logical statements. A query result is always a boolean value. Most abacus operators can be used in query expressions, however not all SQL databases understand how to interpret abacus operators. When a database cannot discern how to handle a specific operator, an error will be returned.

The Qilan translation of abacus expressions is not literal SQL. Qilan attempts to interpret the intent of the designer before creating the SQL statement. This allows queries to be built based on the logical needs of the designer or project rather than conform to an SQL syntax.

```
( zip ="03106" )
```

This is a very simple query. It is built in the Access > Table and compares a table field to a constant (typed value). The abacus, in which this expression is built, should be selected for the QFIND queryicon attribute. In this way, all records whose zip code is '03106' will be retrieved. Note that the zip code field does not have to be chosen by the QFIND.

```
( zip = ( acquire <SQLTest/zip> ) )
```

This query is similar to the previous one, except that the constant is replaced with a framework field. We must 'acquire' the value, as it not present in the Access > Table. 'SQLTest' is the name of the framework. When the WebTemplate is executed, the value assigned to SQLTest/zip will be used in the query.

```
( if ( acquire <SQLTest/choice> ) then ( zip = ( acquire <SQLTest/zip> ) ) else ( zip ="031096" ) )
```

This query combines the two previous statements using an IF THEN ELSE expression. SQL cannot process this type of statement, but Qilan can. The logical IF statement is evaluated first, then the result is converted into SQL and passed to the database.

```
( zip like ( ( acquire <SQLTest/zip> ) followed by "%" ) )
```

In this query, the ‘like’ expression is used to emulate a ‘starts with’. When the user types the first portion of a zip code, zip codes are searched with the user’s entry followed by ‘any other character’. This is what the ‘%’ sign denotes. It is often better to use the abacus operator ‘like’ than ‘starts with’. The reason is twofold: First, ‘starts with’ is case sensitive and ‘like’ is case insensitive; and two, ‘like’ is optimized by database engines.

```
( company defined or "%" )
```

```
( ( company like ( acquire <SQLTest/company_undefined> ) ) and
( zip like ( ( acquire <SQLTest/zip> ) followed by "%" ) ) )
```

This is a two-part query. The upper section is built in the framework (SQLTest). When the framework field, ‘company’ is defined, the value assigned to ‘company’ will be returned. On the other hand, when it is undefined, ‘%’, will be returned. In Access > Table, the second portion of the query statement uses an AND conjunction. This operator will only return records when both sides are true. The goal is allow the user to query company and zip codes; company may be left undefined. To accommodate this eventuality, an undefined company search criteria will default to ‘%’, the ‘like’ notation for, ‘select anything’.

```
( ( integer zip ) > "55555" )
```

This query first converts zip (a typical string data type) to an integer before performing the comparison to the constant, “55555”. Note that the table field is being converted. Qilan does not actually perform this function, but rather passes off this responsibility to the database. Databases may perform this function differently or fail completely. Please refer to your database documentation for specific capabilities and functionality.

```
( ( defined zip ) or Y )
```

```
( ( defined zip ) or ( Y ="Y" ) )
```

These two queries are very similar, yet the first one will fail and the second one succeeds. SQL databases do not understand a field alone in an operator even though the output is boolean. In this case, the field, 'Y', is used in the OR operator. In order to this to be processed correctly, we must create the expression, "Y=Y"; where the second 'Y' is a typed constant. Note that the operators, 'defined__' and 'undefined__' must be used alone and cannot be used as a comparison operator. The following construction will fail.

```
( ( defined zip ) ="Y" )
```

```
( company starts with ( substring starting at "1" length "3" from ( acquire <SQLTest/company> ) ) )
```

When building queries, use of substring extractions (and other text manipulation operators) built in the Access > Table may not work as expected or fail with SQL errors. To avoid these problems, build the query as a two-part statement with the text manipulation operator in the framework.

```
( company starts with ( acquire <SQLTest/company_1_3> ) )
```

HTML Input Forms

Unless you just intend to serve database data to web users, you will likely encounter the need for forms. Forms allow web users to submit data to the web server, which passes the data on to Qilan for processing. The HTML DTD defines forms and associated elements, not Qilan. On the other hand, Qilan icons integrate with forms in a prescribed manner. The purpose of this section is to review form basics from the 'Qilan' point of view.

The FORM Tag

To create a form, use the FORM tag. Inside this tag are each of the individual form elements plus any other HTML content to create a layout. You can include as many different forms on a page as you want to, but you cannot nest forms – that is, placing on FORM tag inside another.

A FORM usually requires two attributes: method and action. Method refers to how the data will be sent to the webserver while action is a pointer to the script that will process the form.

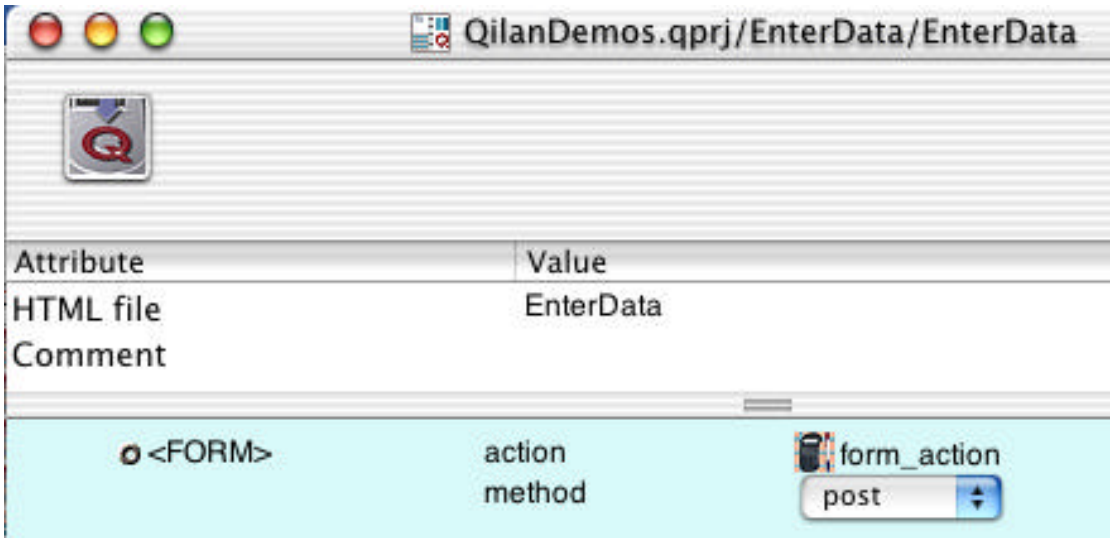
The method attribute can be 'post' or 'get'. What's the difference? Basically, 'get' sends the data to the web server in the environmental variable, QUERY_STRING. To the user, the data appears in the URL string. From a design standpoint, the 'get' method limits the amount of data (to approximately 1K), but it does enable the environmental variable to be used. On the other hand, the 'post' method allows for an unlimited amount of data to be sent as it is sent as stdin (standard in). Unless you need to use QUERY_STRING, I recommend the 'post' method.

A relative path or full URL indicates the action. The way most web browsers work (thank goodness), is that once you type in a full path, subsequent actions are relative to that path. So, if you put all your WebTemplates in the same folder, you need only to type the name of the exported WebTemplate. Neat!

One more thought, an action can reload the same page. In other words, you don't always have to go to a new page. I'll get to this a little later.

The action attribute can be result of a Qilan icon or database value. You could, for example, create a database table with a field called, “actions”. Based on WebTemplate values or some other criteria, dynamically change the content of the action attribute. Unfortunately, you will need to obtain values before the form is returned to the web user. If you need to change the form action attribute after the form has been served to the web user, consider JavaScript.

Here the form action is the result of an abacus expression.

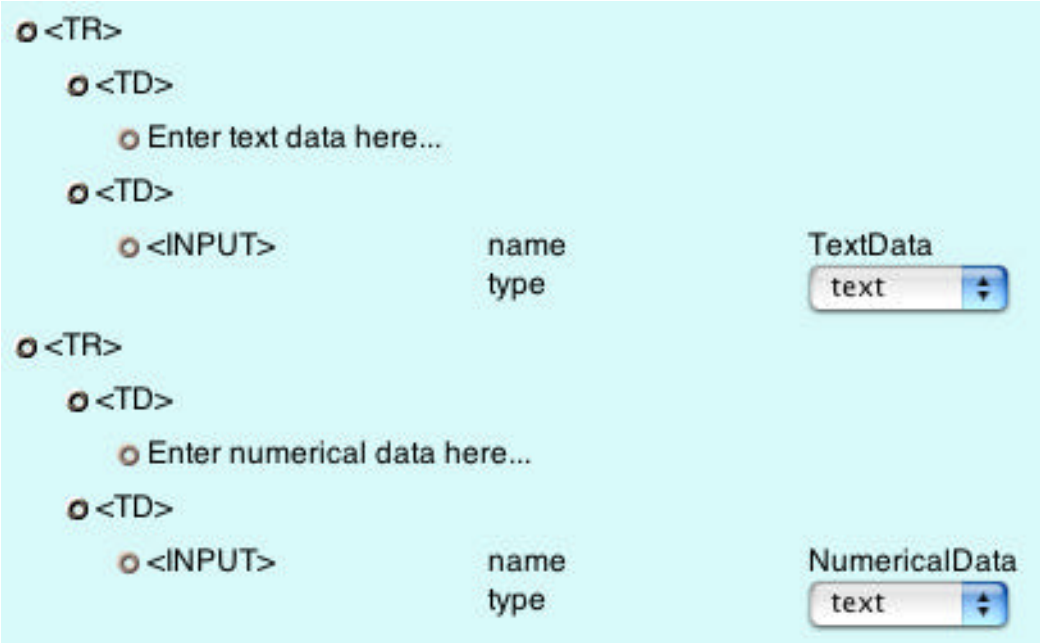


FORM Elements

Form elements comprise several tags that are designed to accept, display or control data. These tags include INPUT, SELECT and TEXTAREA.

The INPUT tag has several types, including text, radio button, check box, submit, hidden, etc. SELECT is used to display a scrolling list or pop-up. TEXTAREA is used when many lines of text are required. What you decide to use should be based on the type of data to be submitted and your target audience.

What's important to understand is how data is actually sent from a web page to Qilan. Consider the following WebTemplate:



This template contains two inputs, each with a 'name' attribute. The name attribute is used by Qilan to differentiate the data associated with each input. In other words, inputs are defined by a name – value pair.

TextData = [data] & NumericalData = [data]

Form input names must be unique when used on the same WebTemplate. Input name attributes are case sensitive.

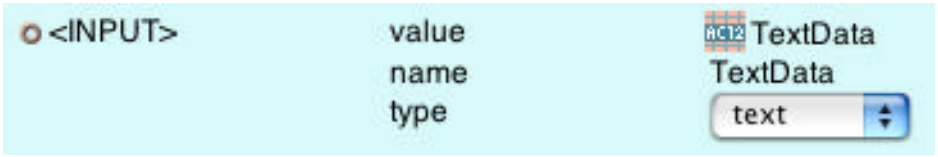
When input data is received by Qilan, Qilan 'values' a field of the same name. Qilan will search for a field, in the same Framework as the WebTemplate that has the same name as the input. If a field is located, the field will be valued.

In the previous example, we create two fields to accept the data:

This approach allows the designer to process data by referring to it using Qilan fields. For instance, the abacus that tests for a DefinedEntry uses NumericalData and TextData as follows:

A point to remember, a Qilan field passes a value, not identity. Although you can drag a field into a name attribute, the value of the field will result, not the name of the icon itself. This is a very common mistake. Actually, I made it once or twice, but that's another story.

Another useful attribute of inputs and other form elements is the 'value' attribute. Think of this attribute as the default value of the form element. For example, suppose you want a text type input to show the value of the data submitted by the user. Here's the basic construction:

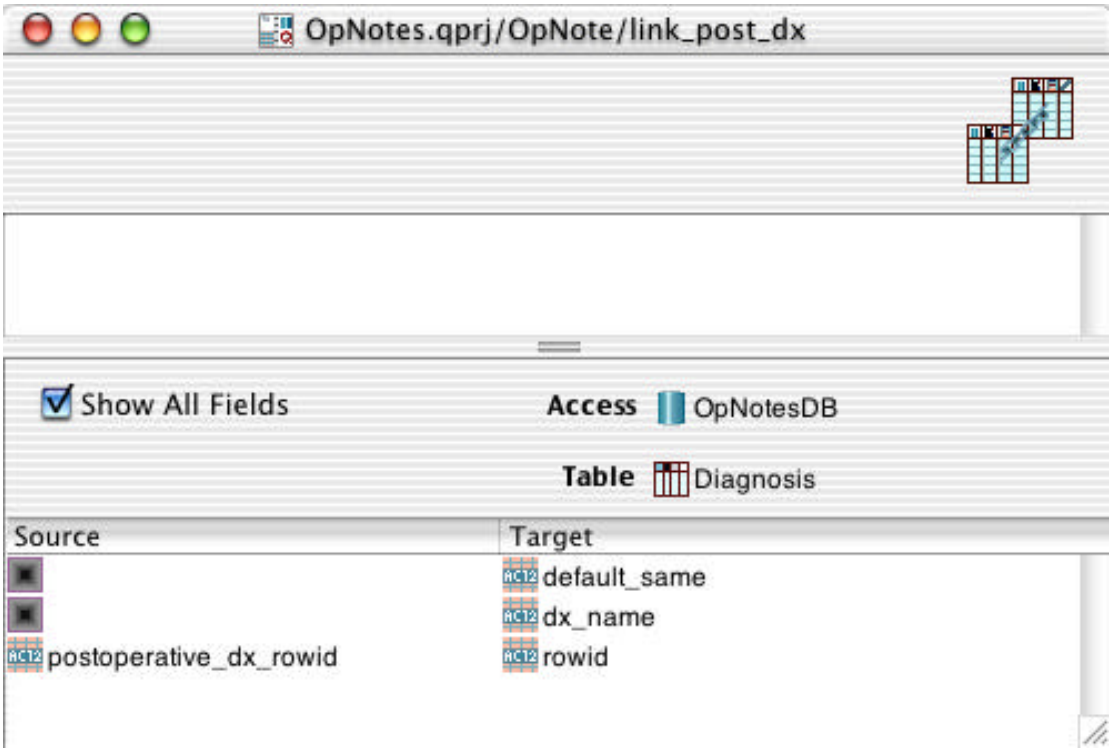


The value of 'TextData' is submitted to Qilan. Qilan values the field of the same name. The field is then used to set the value for the input.

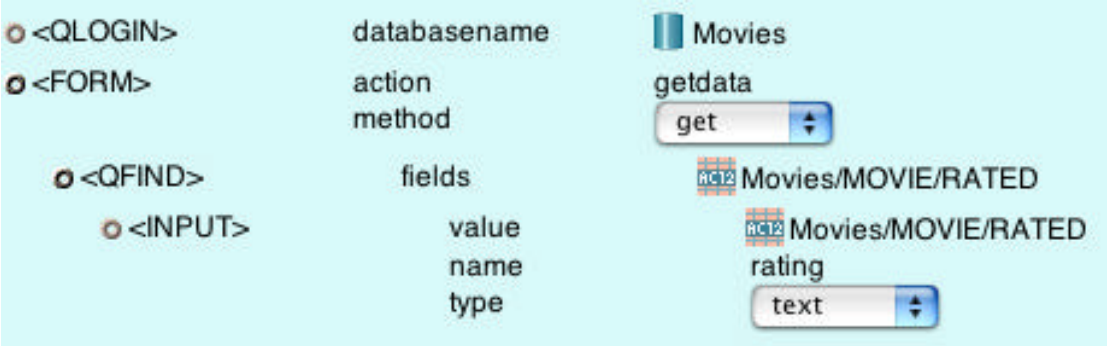
Now, whenever a user submits a value for the input, 'TextData', the value will be returned (echoed). Let's extend this example and return a different kind of value.

This is a TEXTAREA. Text areas allow for multiple lines of input with scroll bars. Note that a TEXTAREA does use a value attribute per se, but can be defaulted using a QVALUE. What is 'get_postoperative_dx'? This is really a database value that gets looked up when the form is processed.

The abacus expression uses a relationship link to obtain the value. In this case, we might not have a defined field, so we use 'defined or' to get another in the event the first choice is undefined.



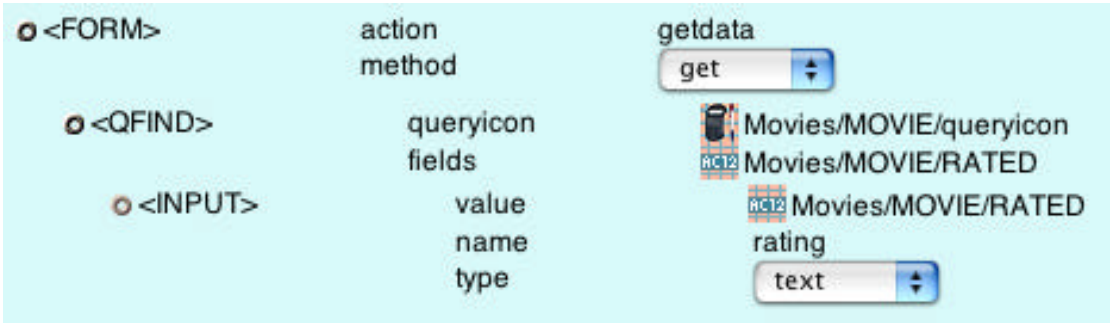
Using a relationship link to default a field is one technique, here's another.



This WebTemplate uses a QFIND to retrieve the database field, "RATED". We then drag that same field, from the palette, to the value attribute of the input. This is very fast, but does require a bit more design. What would happen if the QFIND returned more than one record? You would get two or more inputs with the same name and probably not look too good either. So, to complete this approach, we need to use a queryicon that limits the QFIND to one, and only one, record. This is easily done if the user chooses a record identifier on a previous screen. This value can then be used by the queryicon.

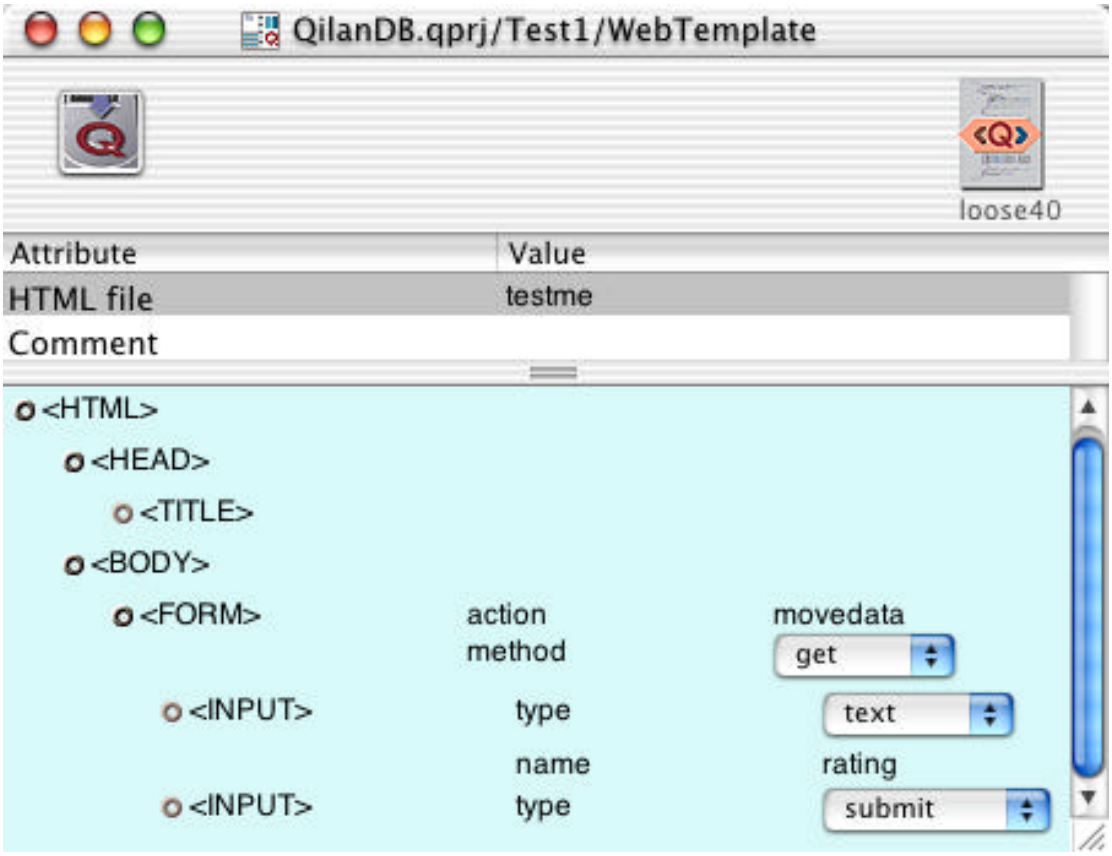
```
( _rowid = ( acquire <Test1/row_id> ) )
```

Observe how the queryicon uses the field value, "row_id", from the Framework, "Test1". If "row_id" is used in the queryicon, why doesn't it appear on the WebTemplate? A field value does not have to be on a WebTemplate to be valued. The previous form action, identifying this WebTemplate, submitted an input named, "row_id".



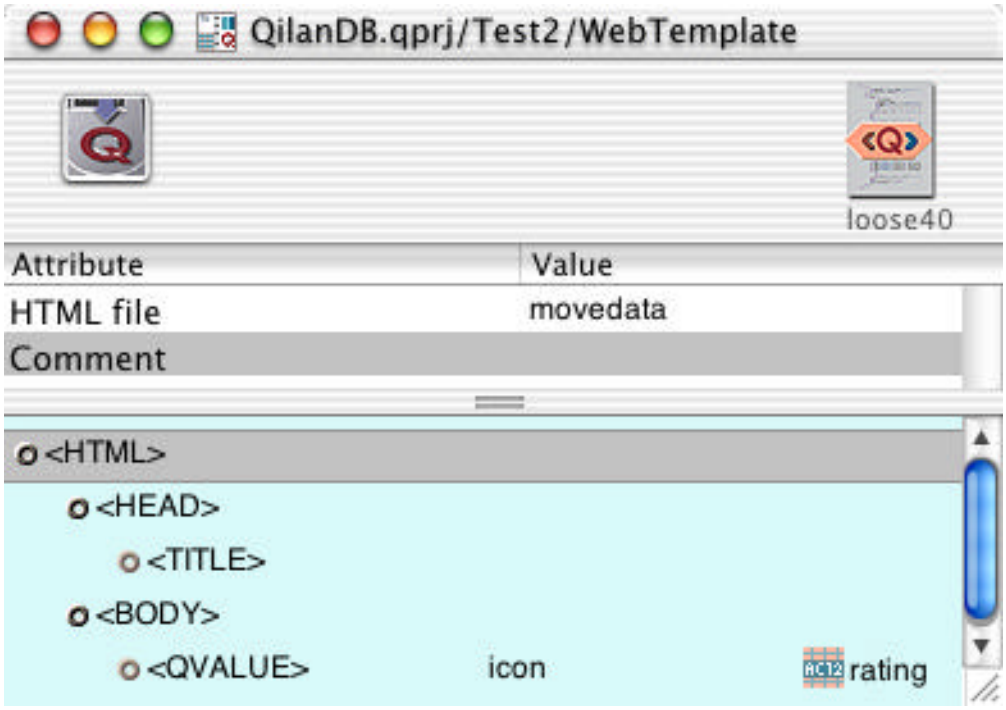
Before we leave this, a new concept has been introduced: valuing fields by name. Let's take a moment and see how this works.

This WebTemplate, located in the Framework, "Test1", submits the input named, "rating". The form action points the another WebTemplate named, "movedata".



What we hope will happen is that the field, "rating" will be valued. So far, we have been working with a single Framework, valuing fields by name. Qilan extends this functionality between Frameworks, and even between Projects.

Here's what the WebTemplate looks like in the Framework, Test2". All that is necessary is a field of the same name and the value submitted by the previous WebTemplate will appear.



 Take a deep breath.

Dynamic Sorting

Qilan is designed so that the sort order for lists (QFIND, QTABLE, QSELECT, etc.) is defined by the selection of a specific Access > Table field. There does not appear be any way of dynamically controlling this selection other than re-creating the entire list tag with a different Access > Table field selected for the sort order.

To allow a user or data specification to control a list's sort order, create an abacus in the Framework containing the webtemplate list tag. Here is an example:

In the Framework containing the webtemplate, create an abacus (sort_by) that 'acquires' the Access > Table field you want to sort by. For example:

```
acquire (company/people/age)
```

Where the database is 'company', the table, 'people', and the field, 'age'. The selection is made from the palette and dragged into the "acquire __" operator. Place this abacus into the list tag sort attribute. Note that the list is retrieved from the same database and table as specified by the sort.

```
QFIND       fields  company/people/name
                    company/people/address
                    company/people/salary
        sort    sort_by
```

This will retrieve a list of people sorted ascending by age.

Now create a form INPUT on the form that allows the user to choose between sorting by name or age.

```
INPUT       type    radio
            name    sort_order
            value   Y

INPUT       type    radio
            name    sort_order
            value   N
```

The construction will create two radio buttons, which the choices are mutually exclusive. The value of 'Y' means to sort by name; 'N' means to sort by age. Now, back to controlling the sort order. Let's modify the abacus, 'sort_by'.

```
IF (acquire (sort_order))
                    THEN (acquire (company/people/name))
                    ELSE (acquire (company/people/age))
```

Why do we have to 'acquire' sort_order (a Framework icon) when it is valued upon a form submit? The reason is a bit obscure, but recall that the list sort attribute is defined from Access > Table, therefore any value used in the definition of the sort attribute must behave as though it originated from Access > Table. The operator, 'acquire __', was designed to look out from Access > Table. All we are doing is being consistent with this paradigm.

One more point. When the list attribute 'sort' is selected, it must be defined; otherwise Qilan will report an error. So to insure this is the case, another modification needs to be made to the abacus, sort_by:

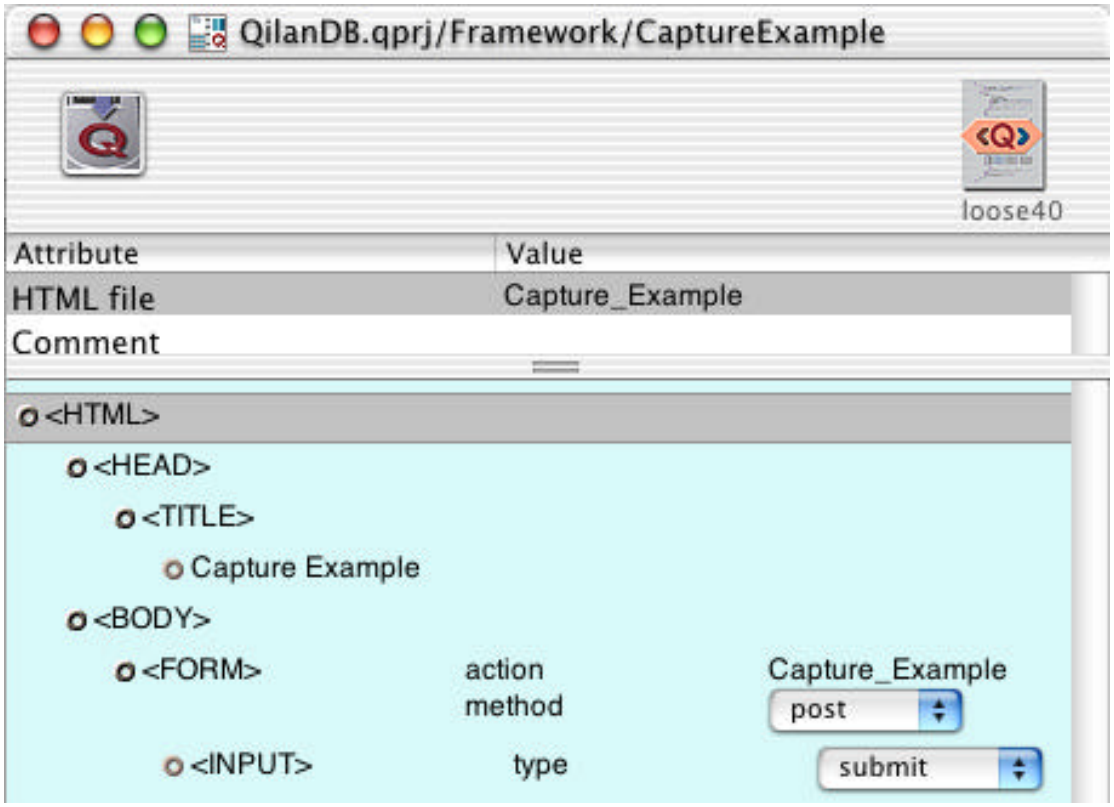
```
IF ((acquire (sort_order)) defined/or (Y))
                    THEN (acquire (company/people/name))
                    ELSE (acquire (company/people/age))
```

When the page is first accessed, the value of sort_order will be undefined, this construction ensures the list will be sorted by 'name' and avoid any errors.

Capturing Values

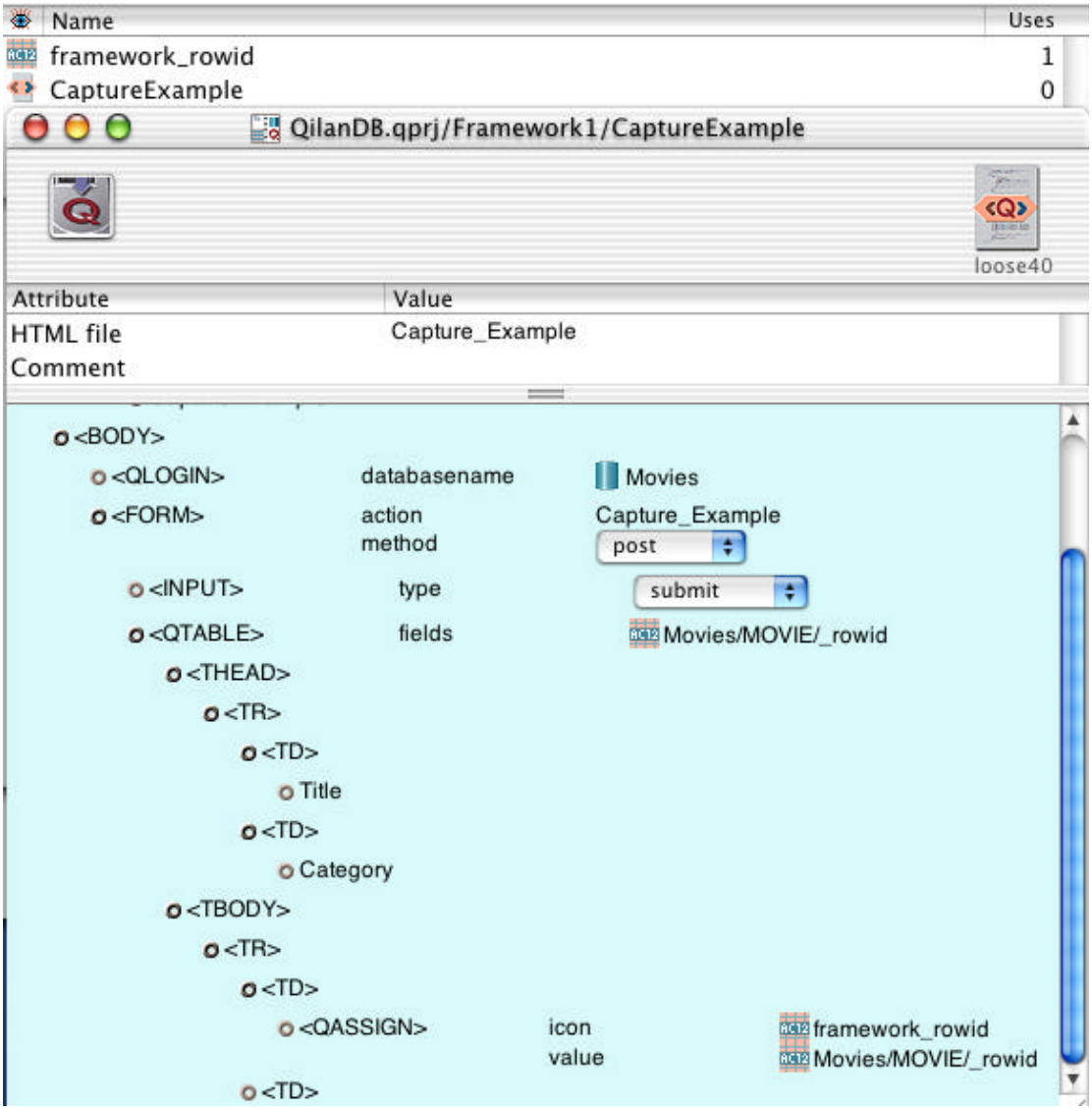
Qilan makes it possible to build entry forms from lists. Imagine a grid like arrangement on a web page, where rows correspond to records and columns to fields. Other software products can produce lists, but what you usually encounter are submit buttons linked to each row. Qilan introduces a technique to submit multiple records at one time (without iteration or loops). This speeds data entry and improves the interaction between the user and the database.

We begin with a FORM tag, as we will be entering data. As a bit of departure, a submit button will be placed after the FORM tag. Actually, submit buttons can be placed anywhere inside the FORM tag.



The name of the exported WebTemplate, “Capture_Example” is the same as the FORM action. This will cause the page to be reloaded each time the submit button is clicked the the web user.

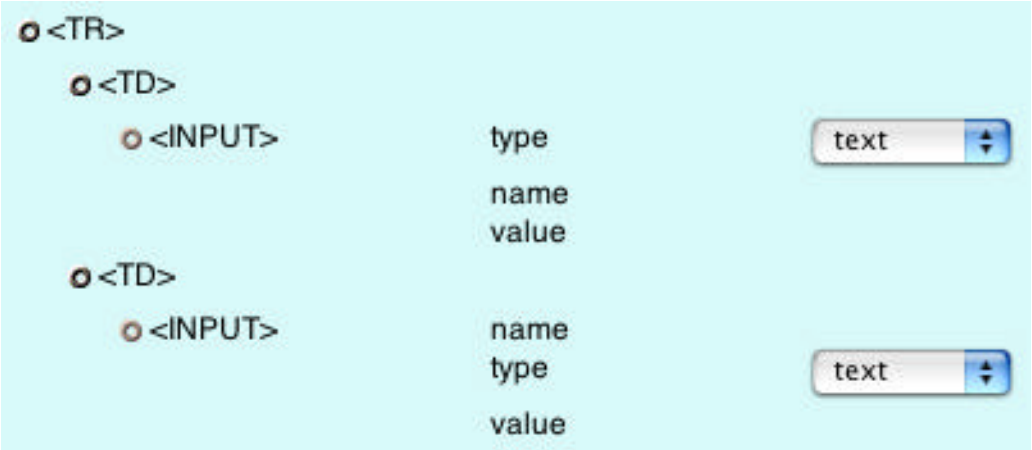
Within the FORM tag, we retrieve our data using a QTABLE. This tag will produce a formatted list. These will be the records that we will be modifying. We could use a query to restrict which records are retrieved, but for this example, let's show them all.



After dragging out the QTABLE, the field, "_rowid" is selected. This field is the primary key for the table, "MOVIE", thus guaranteeing a unique value for each record. In a few moments, we will need to use "_rowid" in the Framework, so the field, "framework_rowid" is created. QASSIGN is used to assign "_rowid" to "framework_rowid". The assignment will be performed for each record retrieved. Note that QASSIGN is inside the TBODY. This is because only a table's TBODY iterates.

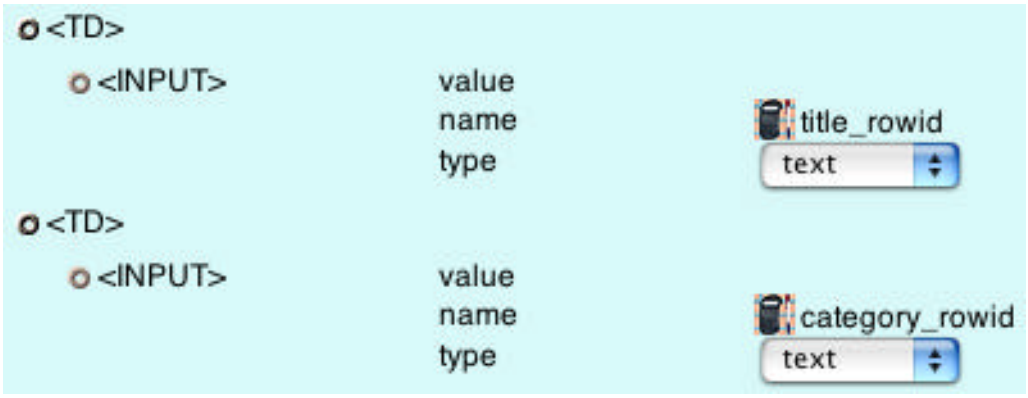
Recall that a field will retain its value until re-assigned. Re-assignment occurs each time that a new record is retrieved.

A little formatting is added to help readability. The THEAD will create headings for the data we are about to modify.
The next step is to add the form INPUTs.



This looks pretty standard, but creates an immediate dilemma. What should the input attribute names be? Because the INPUTs are contained within a list, each record will contain the same two names. Submission would result in all the records being updated with the same data. This will not work! But wait, we have a method to uniquely name inputs – "framework_rowid". Let's see how this is done.

Two abacus expressions are created that link a name with the framework rowid. The name can be anything you want, just as long as it is unique for the record. The result of this concatenation will be that all inputs will have a unique name, regardless of how many records are retrieved. Next, drag the abacus expressions to the respective input name attributes.

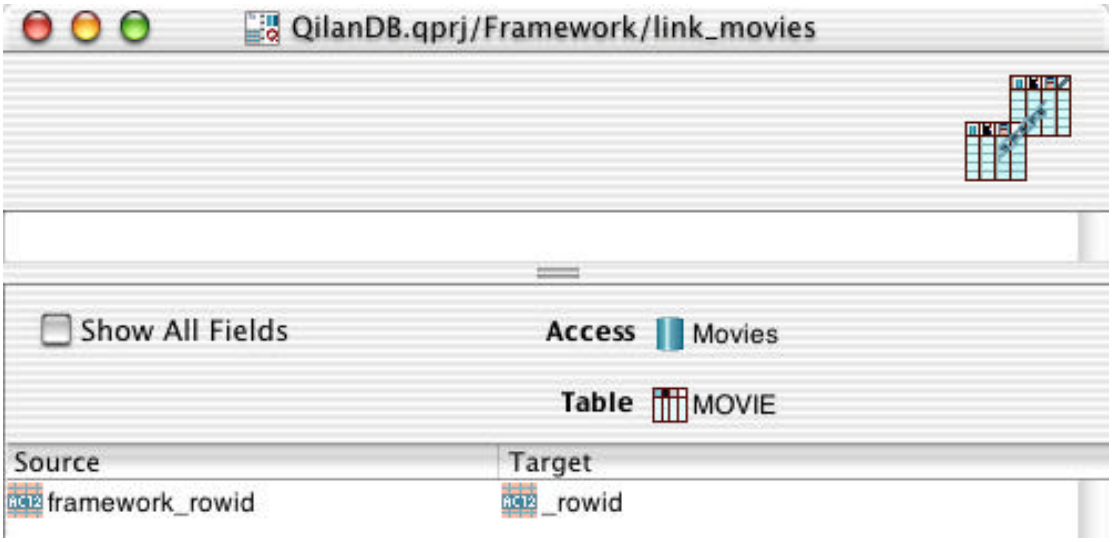


In traditional systems (including Qilan), a field would be valued by naming it the same as the input name attribute. If we did that here, we would have to create two fields for every possible record retrieved. Qilan eliminates that need with the Capture abacus expression. Providing Capture with an INPUT name will return the input's value.

```
( capture category_rowid )
```

```
( capture title_rowid )
```

These two abacus expressions will return the value of each input for each record retrieved by the QTABLE. While it might seem the next step is to place these abacus expressions into the INPUT value attribute, we have not actually retrieved any data from the table, just "_rowid". Placing the capture abacus expressions into the value attribute would just echo the data back. Another approach needs to be considered.

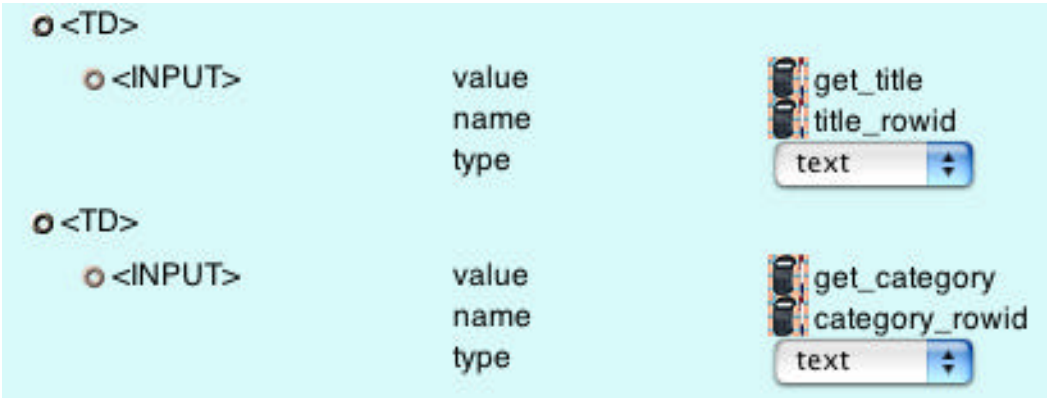


A relationship link is established between “framework_rowid” and “_rowid” in MOVIES. This will allow us to obtain database data as the QTABLE is retrieving data. These ‘get’ abacus expressions are then used as input values.

link_movies.TITLE link_movies.CATEGORY

If you don’t understand why this method is being used, be patient.

The WebTemplate now appears as…



So far, we have created a method to display values from the database and discern unique input names and values. What’s missing is the QUPDATE tag that sends updated data back to the database. A quandary though, where should we put the QUPDATE? We cannot place it outside the QTABLE because each row (being a unique record) needs to be processed individually. Should we update before the first row or after the last row? Before we can answer this question, we need to review how this form will be processed.

Assume the form is shown on the browser. As designed, it will be a list with two inputs on each line. When submit is clicked, here is the sequence of events:

Qilan will re-load the form (recall the form action is the same as the page name).

QTABLE will retrieve the first record.

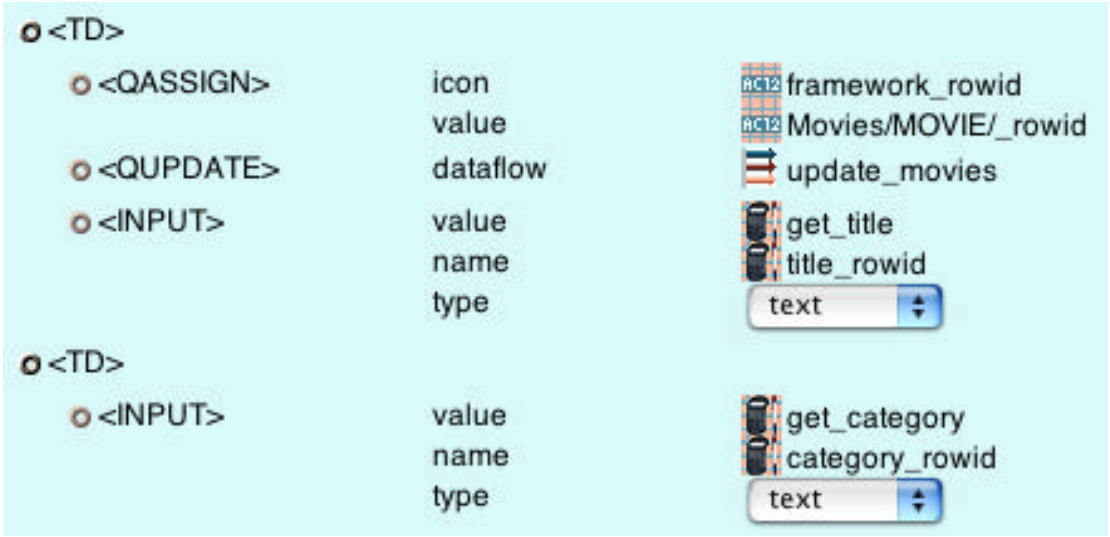
QASSIGN assigns "_rowid" to "framework_rowid"

INPUT value is retrieved based on the relationship link.

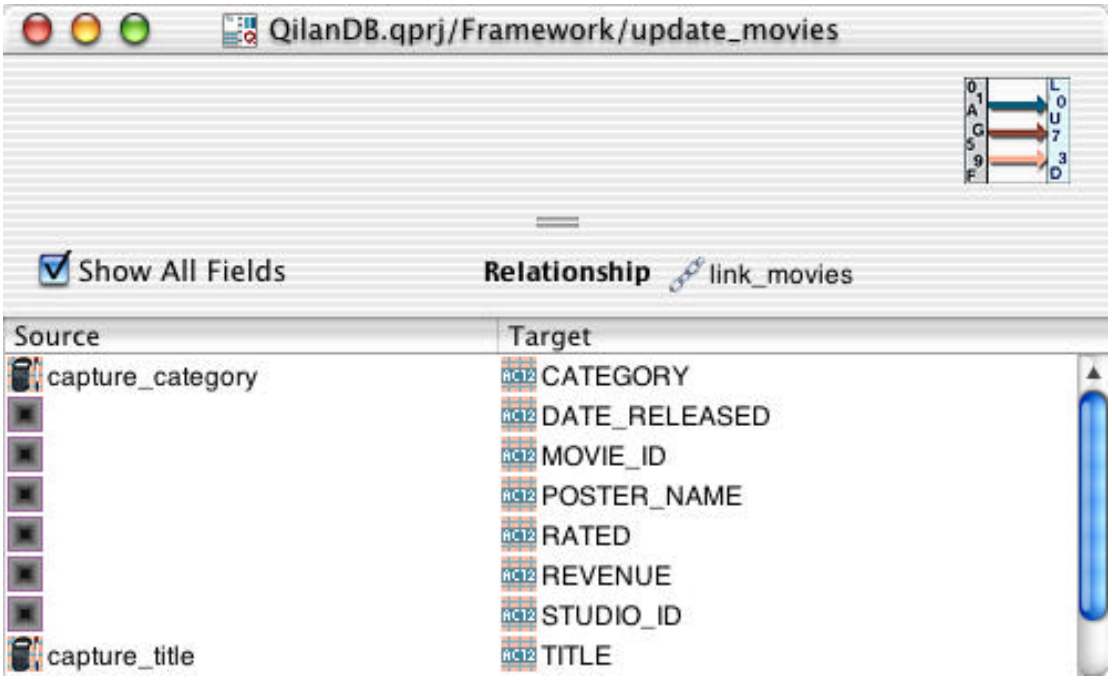
If any data changes are made, we will want them to occur before the data is re-displayed. This will give the user immediate feedback. Therefore, the QUPDATE needs to be placed after the QASSIGN, but before any of the INPUTs.

Previously, the inputs were defaulted using a relationship. Here's the reason… Data retrieved from the QTABLE would not show the results of the QUPDATE. This is because the retrieval would occur before the QUPDATE is processed. On the other hand, if we use a relationship, the retrieval would be current. That is, after the QUPDATE.

The WebTemplate now appears as…



The DataFlow used by the QUPDATE uses the ‘captured’ values to update the database.

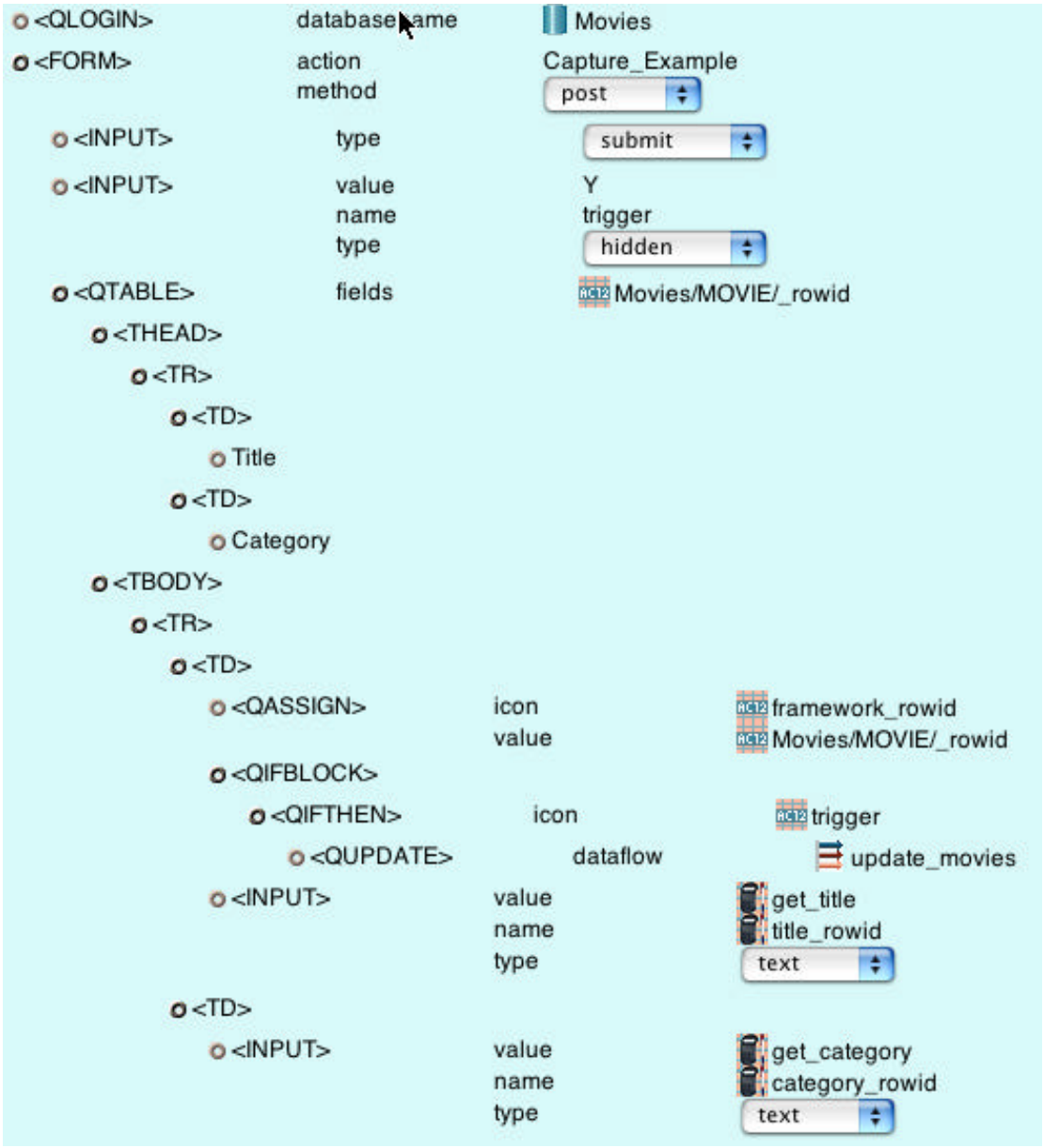


Are we done yet? Almost. Consider what happens when this form is first opened. QTABLE will retrieve the records, then QUPDATE will attempt to update the records using the Capture values. Without first submitting this list, the Capture values will all be undefined. If we did not consider this, the user would be very surprised to discover the inputs would all be empty each time the form is accessed. An easy solution is to create a field that is only valued after the submit button is clicked. When the form is accessed, the field will be undefined. We then can use this field to trigger the QUPDATE.

The approach to use, although there are several variants, is to create an INPUT of the type, ‘hidden’. The value of ‘Y’, will be submitted when submit is clicked. The QUPDATE will be processed when the field, ‘trigger’ is set to ‘Y’.

The completed WebTemplate is shown on the next page.

A completed example of the use of capture.



Appendix

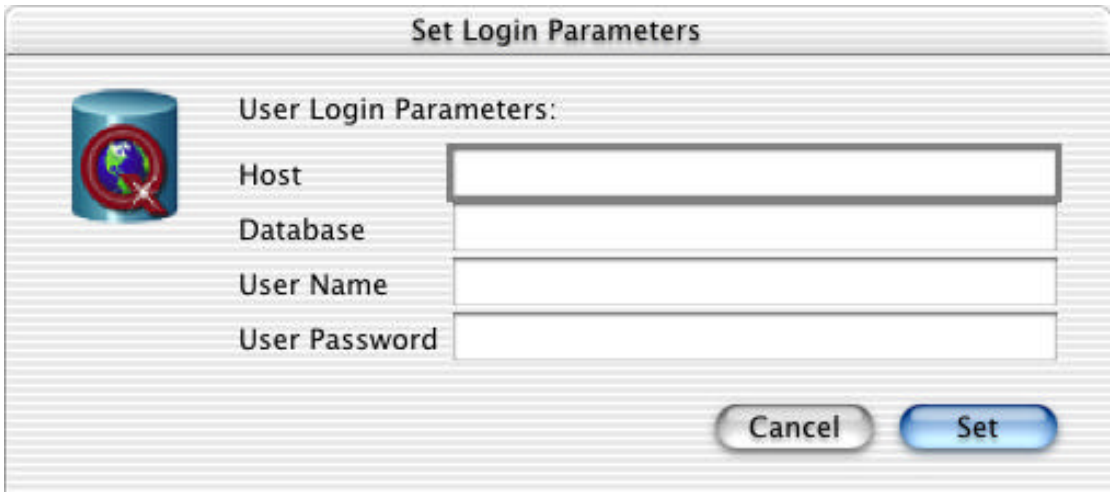
Database Adapters

Microsoft SQL Server® Supplement

Qilan includes a specially developed adapter for Microsoft's, SQL Server. The adapter is a thinweb TDS derivative.

FreeTDS is a project to document and implement the TDS (Tabular DataStream) protocol. TDS is used by Sybase and Microsoft for client to database server communications. The FreeTDS project also includes a JDBC driver for Microsoft's SQL Server and Sybase, which are used by Qilan. More information can be found at http://www.freetds.org/; the source code can be obtained by visiting the Qilan web site.

The Qilan implementation has been tested with SQL Server versions 7 and 2000. Other versions of SQL Server could function erratically and should be tested before deployment.



Limitations and other Considerations

- Fields indexed by MS SQL are not represented in the Access > Table > Field window.
- Field designated as primary keys by MS SQL are not represented in the Access > Table > Field window.
- Import schema is supported, but may take several minutes. Be patient.

MS Access® Supplement

Qilan includes a specially developed adapter for Microsoft's, Access database. The adapter is based on an RmiJdbc bridge. Additional information can be obtained by visiting:

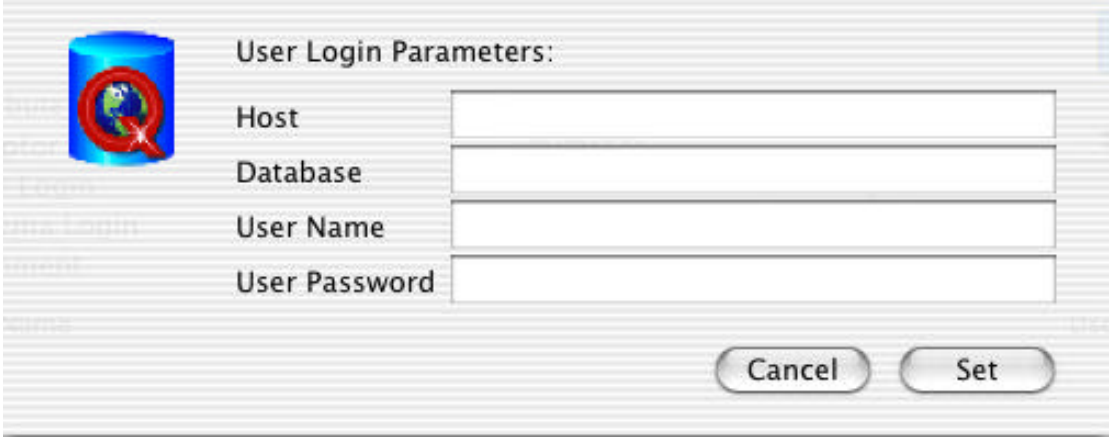
http://www.objectweb.org/rmijdbc/

MS Access does not, in its native form, support JDBC connectivity. Therefore a 'bridge' is necessary to convert JDBC calls to ODBC commands that MS Access does support. The bridge software is known as the 'server' and must be installed on a MS Windows machine accessible the database. Qilan communicates with the server, not directly with the database.

The software server component can be downloaded, along with complete documentation and troubleshooting, from rmijdbc. (See 'Installation on the PC' which follows)

http://www.objectweb.org/rmijdbc/current/RmiJdbc/doc/index.html

You do not need to download the Rmi 'client' software as this is already built into the Qilan adapter.



The Qilan Login panel is shown above. When identifying the 'Host', insure you enter the IP or host name of the machine running the Rmi server software.

Installation on the PC

Installation on the PC (the machine running the MS Access database) requires as series of simple steps and minimal configuration. Here are the basic steps to follow:

Go to: http://www.objectweb.org/rmijdbc/download.html

Download either:
Zip format: RmiJdbc.zip
Gzipped tar format: RmiJdbc.tar.gz

Uncompress the downloaded file.

Copy the file "RmiJdbc/dist/lib/RmiJdbc.jar" to an appropriate location on your system. The specific location is not critical.

Start the MS-DOS Prompt program. At the dos prompt, cd (change directory) to the directory containing the RmiJdbc.jar file.

Type: set CLASSPATH=<path_to_rmijdbc.jar_file>;.

(don't forget the semi-colon followed by a period on the end)

```
set CLASSPATH=C: \RmiJdbc\dist\lib\RmiJdbc.jar;.
```

Press return.

Type: java org.objectweb.rmijdbc.RJJdbcServer
Press return.

This will start the Rmi server. To stop the server, hold down the control key and press 'c'.

To set up your ODBC data souce (database), refer to:

http://www.objectweb.org/rmijdbc/Access/access.html

Limitations and Other Considerations

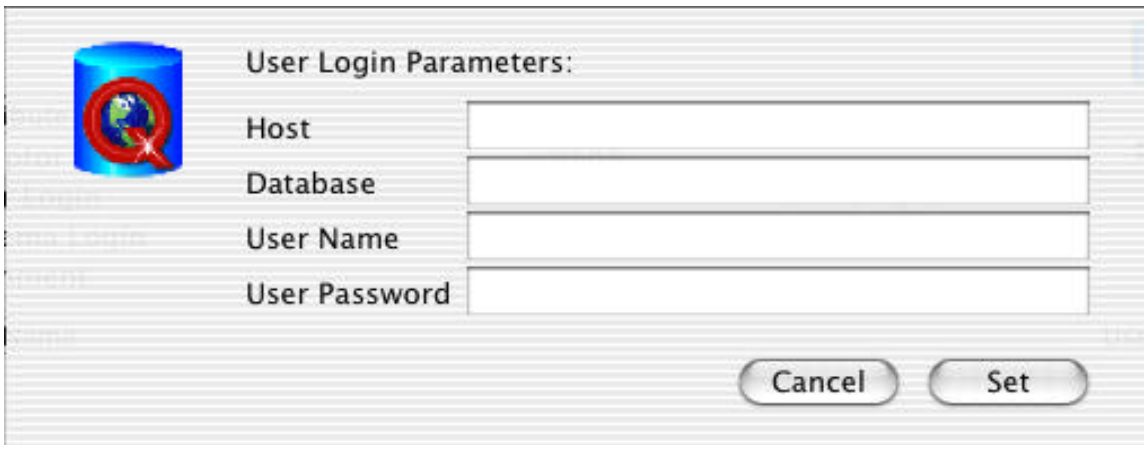
When using queries that contain dates, a special convention must be used so that MS Access does not interpret a date as a character string or mathematical formula. Dates must be proceeded and followed by the pound sign as follows: (datetime_datatype > #11/01/2002#).

The wildcard character for ‘like’ searches is the percent mark (%). This is a JDBC convention. When using MS Access via its traditional interface, the asterisk (*) is used.

Import schema is supported; Export schema is not supported.

MySQL™ Supplement

Qilan includes an adapter for the MySQL database. MySQL is an open source, high performance, relational database. It is included as part of the Mac OS X 10.2 Server installation. Additional information, current downloads and full documentation can be obtained by visiting http://www.mysql.com.



Qilan's login panel is shown above. These values must be set using the command line interface for MySQL or other tool before attempting to login with Qilan. Additionally, Qilan cannot create, start or stop a database, or set start-up parameters for MySQL. Before attempting to login, make sure a database is started and user permissions are set correctly.

Once successfully logged in, Qilan can create, alter or delete tables and fields, or if a database exists, Qilan can import the schema.

Limitations

Data types ENUM and SET cannot be exported.

> ENUM is an *enumerated* data type. It is defined as a set of permissible values, of which only one can be stored.
>
> SET is similar to the enumerated data type except that multiple enumerated types may be stored within a single field.

If you wish to use either of these data types, they must be created via the MySQL command line or other tools. Either type can be imported into Qilan.

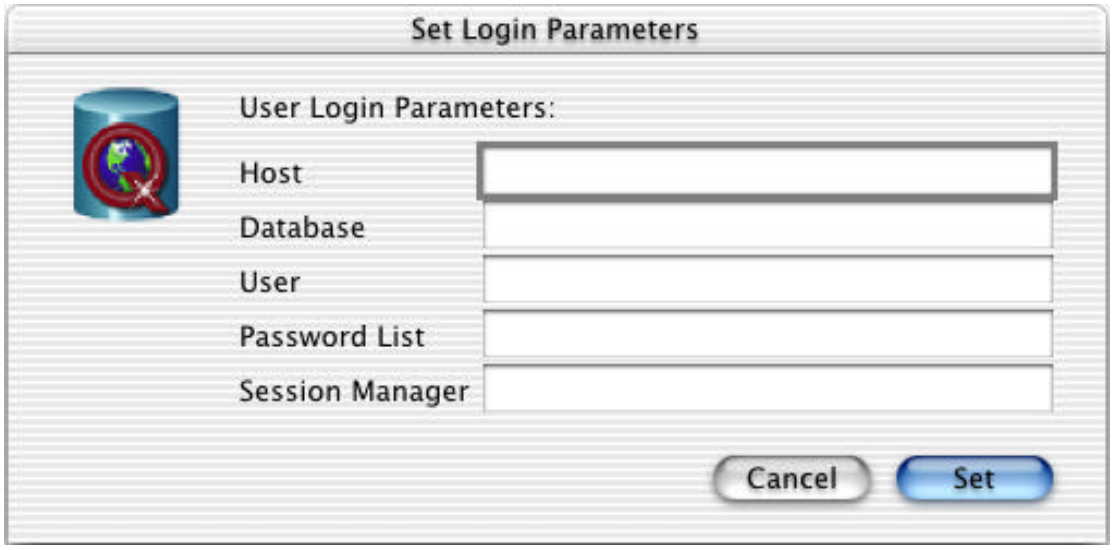
Use of wildcards with the LIKE abacus operator must follow the following syntax:

The % (percent sign) means to accept any character. This is different from standard MySQL documentation and is due to nature of JDBC connectivity.

The _ (underscrore) means to accept one character.

Paradox® Supplement

Qilan includes a specially developed adapter for Corel's Paradox. Insure you properly install the JDBC database module before you attempt to log into a Paradox database. This is usually supplied with Paradox.



Limitations and other Considerations

- Import and Export schema are not supported. (This is due to a limitation of the Paradox JDBC driver).
- Insert and update is not supported for data type autoincrement.
- Field length specification is not supported.
- Data types supported:

Alpha	Number
Money	Short
Long Integer	Logical
Autoincrement	Date
Time	Timestamp

- Data types not supported:

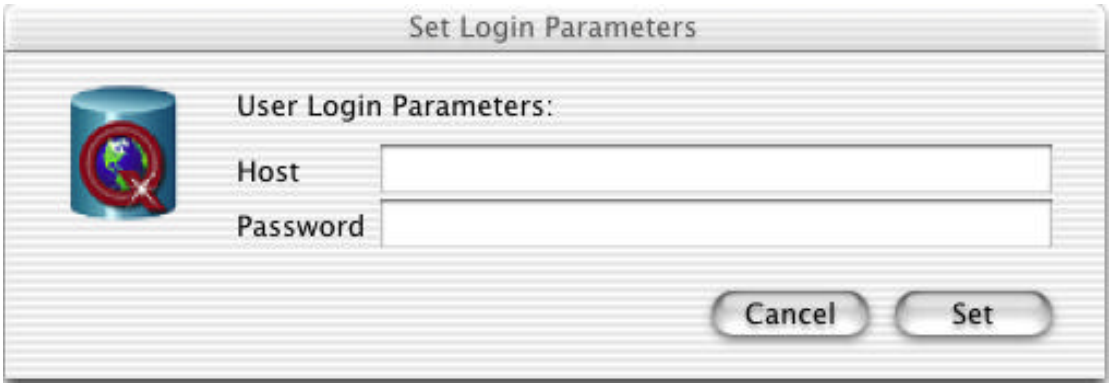
BCD	Memo
Formatted Memo	Graphic
OLE	Binary
Bytes	

- The NOT SQL keyword works inconsistently.
- Supported Operators:

Defined	UpperCase
LowerCase	Undefined
All math operators	Like
Contains	Endswith
Startswith	

FileMaker® Supplement

Qilan supports JDBC access to a FileMaker (FM) database (JDBC v1.3 or higher). A FM database is not, strictly speaking, a relational database. A FM database consists of a single table, not multiple related tables as with standard SQL databases. When several databases are opened at the same time however, Qilan will import a table for each database.



The FileMaker JDBC Driver connects to FileMaker Pro through a standard HTTP connection. Therefore, when configuring FileMaker databases, insure the "web companion" plug-in is enabled.

By default, FileMaker uses port 80. If you are running the Apache webserver on the same machine as FileMaker, it is suggested you change the web companion port to another value (e.g., 81). The host name would then be entered as: [IP]:[port_number].

Qilan supports the importation of FM database schemas; export is not supported. Table relationships are not supported.

‘Q’ Tags supported by FM

All ‘Q’ tags supported except QGROUP

‘Q’ Tag special attributes supported by FM:

QFIND ‘sort’ attribute
QFIND ‘descending’ attribute

Flag type abacus operators supported by FM:

__ and __
__ or __
__ = __
__ not equal to __
__ > __
__ ≥ __
__ < __
__ ≤ __
__ like __ escape__ [escape is not supported]
not __
undefined __

Qilan does not support calling FM scripts.

Pseudofields Supported by Qilan

When a record is created, FM automatically numbers records uniquely as well as timestamps each modification. As this data is considered by FM as 'internally defined', they are not considered 'real' fields and therefore not imported by Qilan. To access them, you must manually create them in the Access > Table. You can retrieve (although not change) this data by creating table fields as follows:

External Name	FM Data Type	Qilan Data Type
RecordID	Integer	Integer
ModID	Date	Date

The RecordID can (and should) be used as the table's primary key. FM external names are not case sensitive.

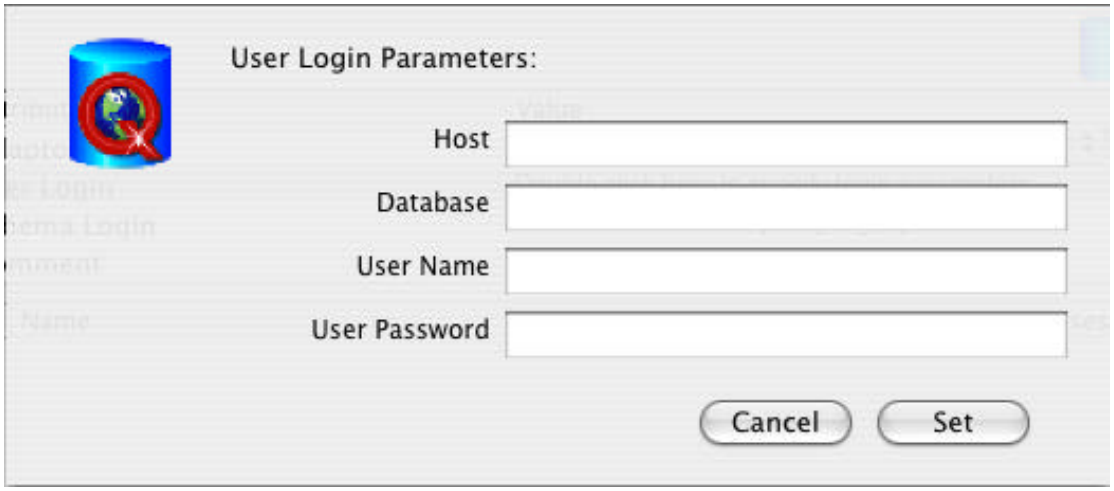
Qilan users, who access FileMaker databases via JDBC, are required to adhere to license restrictions required by FileMaker, Inc. Specifically, section 1.d of the FileMaker Software License:

> *…use of an JDBC Driver to connect to a FileMaker Pro database shall count toward the number of guests for which that copy of FileMaker Pro is licensed to host.*

Sybase® Supplement

Qilan v2.8 is designed for OS X, version 10.3. To access a Sybase database, you must use the '`jconn2.jar`' file. This file is automatically installed when Qilan is installed or can be found with the Sybase JDBC distribution archive designed for Mac OS X.

Installation of the JDBC file is very easy. Log in as the root user, then copy the file to `/Library/Java/Extensions`. Restart the Qilan developer, and if you are using FastCGI, stop then restart the webserver.



The login parameters are shown above. Access to a Sybase database typically requires the identification of a port address. To enter this information, when necessary, append the host name with the port number separated by a colon. For example:

```
19.168.1.1:1599
```

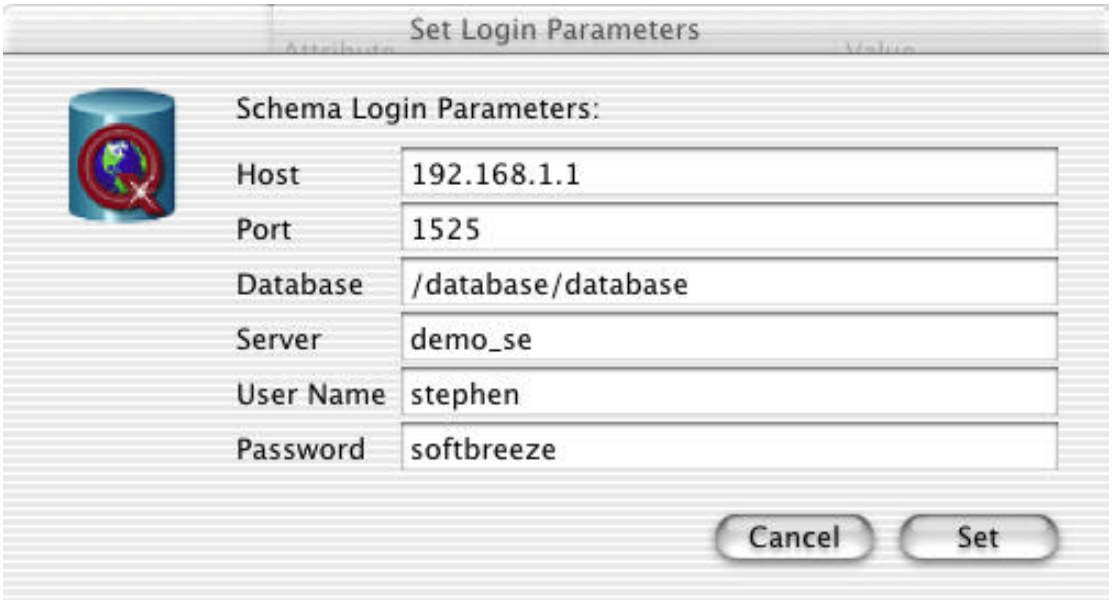
Where '1599' is the port value.

Qilan supports both schema import and export. SQL operators are supported, including joins.

Some Sybase data types such as, char, the width attribute is not imported. The width attribute will appear to be undefined in Qilan's Table > Field window. If import schema is followed by export schema, the field width may be set to zero in Sybase. This may result in data loss. We suggest you carefully review field attributes after using schema import or export.

Informix® Supplement

Qilan enables JDBC access to Informix databases via the JDBC jar file (`ifxjdbc.jar`) as provided by Informix to their registered users.



For some versions of Informix, the name of the database must be prepended by its path, starting from the root volume. Please consult your JDBC documentation if you experience difficulty accessing the database.

Limitations and Other Considerations

- When importing or exporting the database schema, indexed fields and primary keys are not identified or created. Use iSQL.
  - Only the relationship semantic, "Inner Join", is supported.

Date queries use a special format, namely: yy/mm/dd. A two-digit year (without the century) and forward slashes are required. This format may be changed with JDBC environmental variables. Please refer to Informix documentation for details.

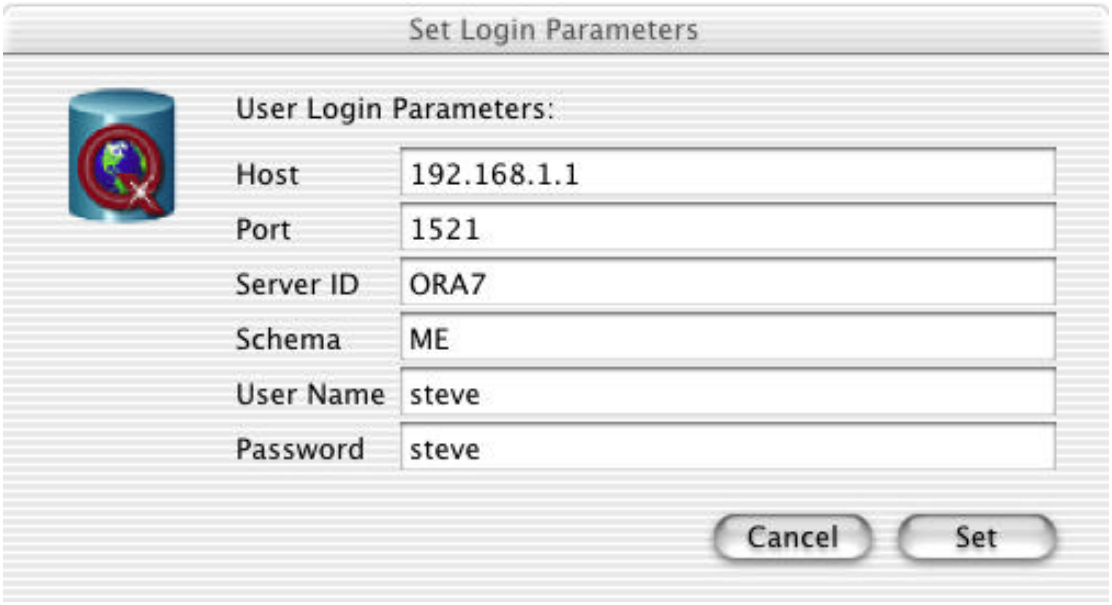
Oracle® Supplement

Qilan enables “thin client” access to Oracle 8i (v8.1.6 and higher) and 9i databases via JDBC. JDBC drivers are provided by Oracle to their registered users and are available for download at the Oracle website:

`http://otn.oracle.com/software/tech/java/sqlj_jdbc/content.html`

Qilan v2.8 is designed for OS X, version 10.3. To access an Oracle database, you must use the ‘`classes12.jar`’ file. This file is automatically installed when Qilan is installed or can be found in the Oracle JDBC distribution archive designed for Mac OS X.

Installation of the JDBC classes file is very easy. Log in as the root user, then copy the file to `/Library/Java/Extensions`. Restart the Qilan developer, and if you are using FastCGI, stop then restart the webserver.



The login parameters are shown above. Note that the 'Schema' parameter is required. The Schema refers to the name given to a set of Oracle tables that comprise a database. Qilan treats each database as a separate Access.

Limitations and Other Considerations

- When importing the database schema, indexed fields are not identified.
- The relationship semantic, "Full Outer", is not supported.
- Export Schema has been disabled.

Helix® Supplement

Qilan communication with Helix databases (collections) is accomplished using the Osmosis Gateway®. Unlike most other databases, which use SQL, Helix uses AppleEvents as the communication method to remotely access data.

Selection of a Helix DTD (Project Settings) adds special 'QOG' tags to WebTemplates. These tags enable communication with the Osmosis Gateway via TCP/IP, which in turn communicates with Helix via AppleEvents.

The Helix AppleEvent suite is limited. There are no explicit 'query' or aggregate functions. However, creating process methods (similar to the concept of stored procedures) are easily and quickly accomplished. These methods can be activated, or called, when data is entered or retrieved.

Qilan's Import and Export data schema options are not available for Helix.

Data retrieval can include both formatted (text, HTML, etc.), as well as raw data. For users who wish to take advantage of Helix's unique development environment, Qilan offers the QOGOUTPUT. This tag will return data as output by Helix. The data itself cannot be manipulated by Qilan, other than displayed on the client browser. The QOGFIND, on the other hand, enables Helix data fields to be mapped directly to Qilan Framework fields.

The role of the Osmosis Gateway is more than a passive communications conduit. The Osmosis Gateway controls access between Qilan and Helix and concurrency control for requested processes. We strongly suggest you refer to the documentation provided with the Osmosis Gateway for complete information. Qilan allows the user to define concurrency control within the WebTemplate using the QOGPROCESS tag.

Designing Helix for AppleEvent access is beyond the scope of this document. If you require assistance, CommonGround Softworks, Inc. maintains an active list of experienced developers familiar with Helix and AppleEvents. We strongly urge you to download the Qilan <-> Helix tutorial, available on our website, for more extensive information.

Communicating with the Osmosis Gateway

Qilan communicates with Helix databases via the Osmosis Gateway. The Osmosis Gateway is a Macintosh application that 'listens' for TCP/IP data on a user specified IP address/port. A special telnet protocol is employed to control communication. The Osmosis Gateway will respond to IP inquiries, including pinging, however messages will not be passed to Helix unless proper keywords and user controls are used.

Creating a Access to a Helix Database

An Access to Helix requires a login specification. While in the Project window, drag an Access icon from the palette then double click on the Access icon.

To open the Login Panel, double click on User Login line. The Schema User Login is not used and may be left empty.

Login parameters:

IP/Domain Name: Qilan communicates with Osmosis Gateway via TCP/IP. Enter the IP address (or domain name) of the machine hosting the Osmosis Gateway. Qilan automatically sets the port number to 3034, but you may enter another if necessary.

Examples: mydomain.com

192.165.1.1
192.165.1.1:3030

Collection Name: The name of the Helix database as appearing in the Macintosh Finder. The name is not case sensitive, but its spelling must be exact.

User Name: The name of the access user. This is the user menu containing all the Views specified for this Access. The name is not case sensitive, but its spelling must be exact.

Helix sub-menus cannot be specified as users.
Custom User icon names will override default names.

Password: The User Name password. Qilan will display the password as bullet characters. Passwords are case sensitive. We urge you to use passwords, although they are not required.

Timeout: The length of time, in seconds, Qilan will wait for the Osmosis Gateway to respond. If undefined, Qilan will wait indefinitely. We suggest a timeout of 60 seconds. A longer timeout may be necessary if extensive processing or large amounts of data are to be transferred, whereas a shorter time out is suggested if a fast reply is anticipated.

The lower portion of the Access window, containing Table icons, is not used for Helix. This area should be left empty.

Helix Specific ‘QOG’ Tags

Helix specific ‘QOG’ tags become available in the WebTemplate palette after a Helix DTD is selected (See Project Settings).

The prefix, “QOG” (Qilan Osmosis Gateway) is used exclusively for all Helix specific tags.

QOGLOGIN: Specifies an Access. Overrides the Access login settings by modifying one or more of the tag attributes: IP location of the Osmosis Gateway, Collection Name, User Name, or Password.

> If you want to accept the default settings for the Access, the QOGLOGIN is optional, as an Access specification is required for all QOG tags.

QOGENTRY: Specifies the view in which the data is to be processed. The data is passed to Helix in alphabetical order, by field/abacus name. The field/abacus order will be shown on the web template after selections are made. Required attributes are Access, Relation, View and Fields. Note that Helix does not process data entry by field name, but rather by tabbing order on a View.

To choose Framework fields or abacii, double click on the attribute line, “Fields” on the WebTemplate. You may use as many fields or abacii as you require.

QOGOUTPUT: Returns the output of a View. Required attributes are Access, Relation and View. OuterAbacii is an optional attribute. If the OuterAbacii attribute is selected, OuterAbacii will be output first. The concept of ‘OuterAbacii’ refers to data displayed on a Helix View, but outside of a list. The output will be sent directly to the client browser.

QOGFIND: Returns the output of a View and places those values into Framework fields. Required attributes are Access, Relation and View. Fields and OuterAbacii are optional attributes.

Selecting one or more Fields, without an OuterAbacii specification, will iterate through each record, ignoring OuterAbacii.

Selecting one or more OuterAbacii, without a Field specification, will iterate once with the values of the OuterAbacii.

Selecting one or more Fields and OuterAbacii will iterate through each record. The values of both Fields and the OuterAbacii are available.

Selecting neither Fields nor OuterAbacii will result in the QOGFIND tag being ignored.

Firstrecord: The first record (starting at 1) to be extracted from the database. Accepts any numeric value.

Recordlimit: The maximum number of records to be extracted from the database, starting as the 'Firstrecord'. Accepts any numeric value.

After the QOGFIND tag is added to the WebTemplate, Fields or OuterAbacii attributes can be added by selecting them from the Icon > Attributes menu. To choose Framework fields, double click on the attribute line on the WebTemplate. You may use as many Fields or OuterAbacii as you require.

Helix outputs data to Qilan via the Helix "View". As Views are based on the icon structure, "Template", the arrangement of data rectangles defines order in which fields/abacus expressions will be output. Generally speaking, the order will be left to right, top to bottom.

Qilan will receive data using the order of fields specified by the QOGFIND. By default, an alphabetical listing will be used. The correspondence between Helix fields and Qilan fields is by placement, not Name.

When naming Qilan fields for Helix, consider using a two digit numeric prefix, e.g., 00, 01, 03, 04 and so forth. This will ensure they will be arranged in the order you require. Fields cannot be re-arranged within the QOGENTRY or QOGFIND tags beyond their natural alphabetical order. If, at some point in the future you need to add a new field after '03', but before '04', you can use the prefix, '031'.

Framework fields can then be used in other calculations, displays, formats or Qilan functions. Data placed into Framework fields does not have to be used or displayed, however it is discarded when cgi processing is completed.

Placing HTML tags within a QOGFIND block will repeat with each record. As may result in improper formatting, we suggest you use the QOGFIND HTML derivative tags:

QOGSELECT
QOGTABLE
QOGUL
QOGOL
QOGDL
QOGMENU

These special tags integrate HTML lists and QOGFIND. They should be used whenever the contents of an HTML list reflect database information. HTML and Qilan attributes (except OuterAbacii) can be selected from the Icon > Attribute menu while the tag is highlighted.

To build a table using the results of a QOGFIND, you would instead use a QOGTABLE as follows:

```
QOGTABLE      db          [Access]
              Relation    [MyData]
              View        [MyView]
              Fields      [FirstName]
                          [LastName]
    <TableBody>
        <TR>
            <TD>
                QVALUE    [FirstName]
            <TD>
                QVALUE    [LastName]
```

QOGTABLE creates a single <TABLE> tag, then iterates the results, one record per <TR> tag.

QOGPROCESS: Sets a process control for tags within the outline structure. Required attributes are Access and Processname. When a Processname is used, the Osmosis Gateway uses concurrency control for processes of the same name. A Processname can be any text string that uniquely identifies the specific process.

QOGTRANSFER: Transfers data to/from views within the same collection or between collections. Required attributes are FromDataBase, FromRelation, FromView ToDataBase, ToRelation, and ToView.

Helix does not use Relationship or DataFlow icons. These objects are used exclusively for SQL databases.

SQL Addendum

SQL Reserved Words

The following words should be avoided when naming *external* field names. Some SQL databases interpret these words as commands or functions and if used as external field names, unexpected errors may result.

ABSOLUTE	ACTION
ACTOR	ADD
AFTER	ALIAS
ALLOCATE	ALTER
ARE	ASSERTION
ASYNC	AT
ATTRIBUTES	BEFORE
BETWEEN	BIT
BIT_LENGTH	BOOLEAN
BOTH	BREADTH
CALL	CASCADE
CASCADED	CASE
CAST	CATALOG
CHARACTER_LENGTH	CHAR_LENGTH
COALESCE	COLLATE
COLLATION	COLUMN
COMPLETION	CONNECT
CONNECTION	CONSTRAINT
CONSTRAINTS	CONVERT
CORRESPONDING	CROSS
CURRENT_DATE	CURRENT_PATH
CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	CYCLE
DATA	DATE
DAY	DEALLOCATE
DEFERRABLE	DEFERRED
DEPTH	DESCRIBE
DESCRIPTOR	DESTROY
DIAGNOSTICS	DICTIONARY
DISCONNECT	DO
DOMAIN	DROP
EACH	ELEMENT
ELSE	ELSEIF
END-EXEC	EQUALS

EXCEPT	EXCEPTION
EXECUTE	EXTERNAL
EXTRACT	FACTOR
FALSE	FIRST
FULL	GENERAL
GET	GLOBAL
HOLD	HOUR
IDENTITY	IF
IGNORE	IMMEDIATE
INITIALLY	INNER
INPUT	INSENSITIVE
INSTEAD	INTERSECT
INTERVAL	ISOLATION
JOIN	LAST
LEADING	LEAVE
LEFT	LESS
LEVEL	LIMIT
LIST	LOCAL
LOOP	LOWER
MATCH	MINUTE
MODIFY	MONTH
NAMES	NATIONAL
NATURAL	NCHAR
NEW	NEW_TABLE
NEXT	NO
NONE	NULLIF
OBJECT	OCTET_LENGTH
OFF	OID
OLD	OLD_TABLE
ONLY	OPERATION
OPERATOR	OPERATORS
OTHERS	OUTER
OUTPUT	OVERLAPS
PAD	PARAMETERS
PARTIAL	PATH
PENDANT	POSITION
POSTFIX	PREFIX
PREORDER	PREPARE
PRESERVE	PRIOR
PRIVATE	PROTECTED

READ
REF
RELATIVE
RESIGNAL
RETURN
REVOKE
ROLE
ROW
SAVEPOINT
SEARCH
SENSITIVE
SESSION
SIGNAL
SIZE
SQLEXCEPTION
SQLWARNING
STATE
SUBSTRING
SYSTEM_USER
TERM
THEN
TIME
TIMEZONE_HOUR
TRAILING
TRANSLATE
TRIGGER
TRUE
TYPE
UNKNOWN
USAGE
VALUE
VARIABLE
VIRTUAL
WAIT
WHILE
WRITE
ZONE
RECURSIVE
REFERENCING
REPLACE
RESTRICT
RETURNS
RIGHT
ROUTINE
ROWS
SCROLL
SECOND
SEQUENCE
SESSION_USER
SIMILAR
SPACE
SQLSTATE
START
STRUCTURE
SYMBOL
TEMPORARY
TEST
THERE
TIMESTAMP
TIMEZONE_MINUTE
TRANSACTION
TRANSLATION
TRIM
TUPLE
UNDER
UPPER
USING
VARCHAR
VARYING
VISIBLE
WHEN
WITHOUT
YEAR

Field Formatting Reference

Databases can define formats in a variety of ways. There is general agreement on the generic types, such as CHAR, but considerable variation with numeric types. Also, some databases extend the specificity of formats, especially date and time. Also be aware that the operating system, on which the database is installed, effects storage formats.

The following reference section is provided as a primer to the data formats. It is not intended to be exhaustive or cover all databases.

Database Types

A data type that which the database defines. INTEGER, CHAR, DATE, and DECIMAL are examples of data types.

Choosing a Data Type

When you choose a data type, you format (constrain) the field so that it contains only values that can be represented by that type.

Every field in a table must have a data type that the database supports. The choice of data type is important for the following reasons:

> It establishes the basic properties of the field; that is, the set of valid data items that the field can store.
>
> It determines the kinds of operations that you can perform on the data. For example, you cannot apply aggregate functions, such as SUM, to fields with a character data type.
>
> It determines how much space each data item occupies on disk. The space required to accommodate data items is not as important for small tables as is for large tables.

The Field Formatting Reference Section was adapted, in part, from the: Informix Guide to SQL: Tutorial, version 9.1

Using Data Types in Relationships

Almost all data type combinations must match (or be able to be coerced) when you are trying to match source fields/abaci and target fields.

Refer the diagram in the Appendix, which shows the decision tree that summarizes the choices among data types. The choices are explained in the following sections.

Numeric Data Types

Most database applications support eight numeric data types. Some are best suited for counters and codes, some for engineering quantities, and some for money.

Counters and Codes: INTEGER, SMALLINT, and INT8

The INTEGER and SMALLINT data types hold small whole numbers. They are suited for fields that contain counts, sequence numbers, numeric identity codes, or any range of whole numbers when you know in advance the maximum and minimum values to be stored.

Both types are stored as signed binary integers. INTEGER values have 32 bits and can represent whole numbers from -(231 -1) through 231-1, that is, from -2,147,483,647 through 2,147,483,647. (The maximum negative number, -2,147,483,248 is reserved and cannot be used.)

SMALLINT values have only 16 bits. They can represent whole numbers from -32,767 through 32,767. (The maximum negative number, -32,768, is reserved and cannot be used.)

The INTEGER and SMALLINT data types have the following advantages:

> They take up little space (2 bytes per value for SMALLINT and 4 bytes per value for INTEGER).
>
> Arithmetic expressions such as SUM and MAX as well as sort comparisons can be done very efficiently on them.

The disadvantage to using INTEGER and SMALLINT is the limited range of values that they can store. The database does not store a value that exceeds the capacity of an integer. Of course, such excess is not a problem when you know the maximum and minimum values to be stored.

The INT8 (LONGLONG) data type is stored as a signed binary integer, which uses 8 bytes per value. Although INT8 takes up twice the space as the INTEGER data type, INT8 has the advantage of a significantly larger range of data representation. INT8 can represent integers ranging from -9,223,372,036,854,775,807 through 9,223,372,036,854,775,807. (The maximum negative number, - 9,223,372,036,854,775,808, is reserved and cannot be used.)

Approximate Numbers: FLOAT and SMALLFLOAT

In scientific, engineering, and statistical applications, numbers are often known to only a few digits of accuracy, and the magnitude of a number is as important as its exact digits.

The floating-point data types are designed for these applications. They can represent any numerical quantity, fractional or whole, over a wide range of magnitudes from the cosmic to the microscopic. Their only restriction is their limited precision. Floating-point numbers retain only the most significant digits of their value. If a value has no more digits than a floating-point number can store, the value is stored exactly. If it has more digits, it is stored in approximate form, with its least-significant digits treated as zeros.

This lack of exactitude is fine for many uses, but you should never use a floating-point data type to record money or any other quantity whose least significant digits should not be changed to zero.

Two sizes of floating-point data types exist. The FLOAT type is a double-precision, binary floating-point number as implemented in the C language. A FLOAT data type value usually takes up 8 bytes. The SMALLFLOAT (also known as REAL) data type is a single-precision, binary floating-point number that usually takes up 4 bytes. The main difference between the two data types is their precision. A FLOAT field retains about 16 digits of its values; a SMALLFLOAT field retains only about 8 digits.

Floating-point numbers have the following advantages:

They store very large and very small numbers, including fractional ones.

They represent numbers compactly in 4 or 8 bytes.

Arithmetic functions such as AVG, MIN, and sort comparisons are efficient on these data types.

The main disadvantage of floating-point numbers is that digits outside their range of precision are treated as zeros.

Adjustable-Precision Floating Point: DECIMAL(p)

The DECIMAL(*p*) data type is a floating-point data type similar to FLOAT and SMALLFLOAT. The important difference is that you specify how many significant digits it retains. The precision you write as *p* can range from 1 to 32, from fewer than SMALLFLOAT up to twice the precision of FLOAT.

The magnitude of a DECIMAL(*p*) number ranges from 10^{-130} to 10^{124}.

It is easy to be confused about decimal data types. The one under discussion is DECIMAL(*p*); that is, DECIMAL with only a precision specified. The size of DECIMAL(*p*) numbers depends on their precision; they occupy 1+p/2 bytes (rounded up to a whole number, if necessary).

DECIMAL(*p*) has the following advantages over FLOAT:

Precision can be set to suit the application, from highly approximate to highly precise.

Numbers with as many as 32 digits can be represented exactly.

Storage is used in proportion to the precision of the number.

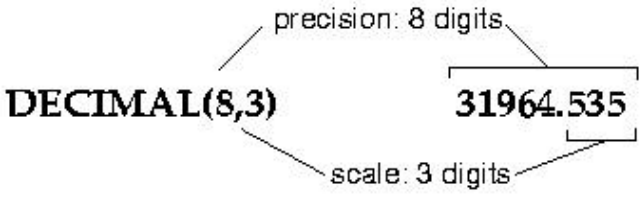
The DECIMAL(*p*) data type has the following disadvantages compared to FLOAT:

> Performing arithmetic and sorts on DECIMAL(*p*) values is somewhat slower than on FLOAT values.
>
> Many programming languages do not support the DECIMAL(*p*) data format the way that they support FLOAT and INTEGER. When a program extracts a DECIMAL(*p*) value from the database, it might have to convert the value to another format for processing.

Fixed-Point Numbers: DECIMAL and MONEY

Most commercial applications need to store numbers that have fixed numbers of digits on the right and left of the decimal point. Amounts of money are the most common examples. Amounts in U.S. and other currencies are written with two digits to the right of the decimal point. Normally, you also know the number of digits needed on the left, depending on the kind of transactions that are recorded: perhaps 5 digits for a personal budget, 7 digits for a small business, and 12 or 13 digits for a national budget.

These numbers are *fixed-point* numbers because the decimal point is fixed at a specific place, regardless of the value of the number. The DECIMAL(*p*,*s*) data type is designed to hold them. When you specify a field of this type, you write its *precision* (*p*) as the total number of digits that it can store, from 1 to 32. You write its *scale* (s) as the number of those digits that fall to the right of the decimal point. The graphic below shows the relationship between precision and scale. Scale can be zero, meaning it stores only whole numbers. When only whole numbers are stored, DECIMAL(*p*,*s*) provides a way of storing integers of up to 32 digits.



The Relation Between Precision and Scale in a Fixed-Point Number

Like the DECIMAL(*p*) data type, DECIMAL(*p*,*s*) takes up space in proportion to its precision. One value occupies 1+*p*/2 bytes, rounded up to a whole number of bytes.

The MONEY type is identical to DECIMAL(*p*,*s*), but with one extra feature. Whenever the database server converts a MONEY value to characters for display, it automatically includes a currency symbol.

The advantages of DECIMAL(*p*,*s*) over INTEGER and FLOAT are that much greater precision is available (up to 32 digits as compared with 10 digits for INTEGER and 16 digits for FLOAT), and both the precision and the amount of storage required can be adjusted to suit the application.

The disadvantages are that arithmetic operations are less efficient and that many programming languages do not support numbers in this form. Therefore, when a program extracts a number, it usually must convert the number to another numeric form for processing.

Choosing a currency format

Each nation has its own way of displaying money values. When a database displays a MONEY value, it refers to a currency format that the user specifies. The default locale specifies a U.S. English currency format of the following form:

$7,822.45

For non-English locales, you can change the current format by modifying Qilan's Project Setting defaults, explicitly changing the symbol using the 'formatted by' abacus operator and/or modifying the International defaults using the System Preferences.

Chronological Data Types

Most databases support three data types for recording time. The DATE data type stores a calendar date. DATETIME records a point in time to any degree of precision from a year to a fraction of a second. The INTERVAL data type stores a span of time; that is, a duration.

Calendar Dates: DATE

The DATE data type stores a calendar date. A DATE value is actually a signed integer whose contents are interpreted as a count of full days since midnight on January 1, 2001.

The DATE format has ample precision to carry dates into the far future (58,000 centuries). Negative DATE values are interpreted as counts of days prior to the epoch date; that is, a DATE value of -1 represents the day December 30, 2000.

Because DATE values are integers, database servers permit them to be used in arithmetic expressions. For example, you can take the average of a DATE field, or you can add 7 or 365 to a DATE field.

The DATE data type is compact, at 4 bytes per item. Arithmetic functions and comparisons execute quickly on a DATE field.

Choosing a date format

You can punctuate and order the components of a date in many ways. When a database server displays a DATE value, it refers to a date format that the user specifies. The default locale specifies an U.S. English date format of the form:

MM/DD/YY

Some systems, notably Informix, may use the text format: YY/MM/DD to represent the date in queries and other functions.

Timezones

A timezone is the offset in hours and minutes a locale is from GMT (Greenwich Mean Time). Databases that support timezone append it to the DateTime format as follows:

2001-03-25 18:30:22 –0500 [yyyy-mm-dd hh:mm:ss ±hhmm]

where the first two digits are the number of hours and the last two digits the number of minutes. The example above indicates the timezone for the locale is a minus five hours and no minutes from GMT. We could also interpret this as Eastern Standard Time.

Exact Points in Time: DATETIME

The DATETIME data type stores any moment in time in the era that begins 1 A.D. In fact, DATETIME is really a family of 28 data types, each with a different precision. When you define a DATETIME field, you specify its precision. The field can contain any sequence from the list *year*, *month*, *day*, *hour*, *minute*, *second*, and *fraction*. Thus, you can define a DATETIME field that stores only a year, only a month and day, or a date and time that is exact to the hour or even to the millisecond. The size of a DATETIME value ranges from 2 to 11 bytes depending on its precision.

The advantage of DATETIME is that it can store dates more precisely than to the nearest day, and it can store time values. Its sole disadvantage is an inflexible display format, but you can circumvent this disadvantage.

Durations: INTERVAL

The INTERVAL data type stores duration, that is, a length of time. The difference between two DATETIME values is an INTERVAL, which represents the span of time that separates them. The following examples might help to clarify the differences:

> An employee began working on January 21, 1994 (either a DATE or a DATETIME).
>
> She has worked for 254 days (an INTERVAL value, the difference between the TODAY function and the starting DATE or DATETIME value).
>
> She begins work each day at 0900 hours (a DATETIME value).
>
> She works 8 hours (an INTERVAL value) with 45 minutes for lunch (another INTERVAL value).
>
> Her quitting time is 1745 hours (the sum of the DATETIME when she begins work and the two INTERVALs).

Like DATETIME, INTERVAL is a family of types with different precisions. An INTERVAL value can represent a count of years and months; or it can represent a count of days, hours, minutes, seconds, or fractions of seconds; 18 precisions are possible. The size of an INTERVAL value ranges from 2 to 12 bytes.

INTERVAL values can be negative as well as positive. You can add or subtract them, and you can scale them by multiplying or dividing by a number. This is not true of either DATE or DATETIME. You can reasonably ask, "What is one-half the number of days until April 23?" but not, "What is one-half of April 23?"

Forcing the format of a DATETIME or INTERVAL Value

The database always displays the components of an INTERVAL or DATETIME value in the order *year-month-day hour*:*minute*:*second*.*fraction*. It does not refer to the date format that is defined to the operating system, as it does when it formats a DATE value.

Choosing a DATETIME Format

When a database displays a DATETIME value, it refers to a DATETIME format that the user specifies. The default locale specifies an U.S. English DATETIME format of the following form:

1995-10-25 18:02:13

Boolean Data Type

The BOOLEAN data type is a one-byte data type. The values are case insensitive.

You can compare a BOOLEAN field against another BOOLEAN field, or against Boolean values.

Character Data Types

Most databases support the CHAR data type and VARCHAR.

Character Data: CHAR(n)

The CHAR(*n*) data type contains a width of *n* bytes. These characters can be a mixture of English and non-English characters and can be either single byte or multibyte (Asian). The width of *n* (for the database) can range from 1 to 32,767.

> JDBC does not impose a character length restriction, however use of fixed width or a specific database parameter may limit field length.

When a CHAR(*n*) value is retrieved or stored, exactly *n* bytes are transferred. If an inserted value is shorter than *n*, the database server extends the value by using single byte ASCII space characters to make up *n* bytes.

Data in CHAR fields is sorted in code-set order. For example, in the ASCII code set, the character *a* has a code-set value of 97, *b* has 98, and so forth. Most databases sort CHAR(*n*) data in this order.

The advantage of the CHAR(*n*) data type is its availability on all database servers. The only disadvantage of CHAR(*n*) is its fixed length. When the length of data values varies widely from row to row, space is wasted.

Varying-Length Strings: VARCHAR(m,r)

The VARCHAR(*n*) data type contains a maximum width of *n* bytes

VARCHAR (m,r) is a data type for storing character data of varying length.

The advantages of the VARCHAR(*n*) data type over the CHAR(*n*) data type are as follows:

It conserves disk space when the number of bytes that data items require vary widely or when only a few items require more bytes than average.

Queries on the more compact tables can be faster.

The following list describes the disadvantage of using the VARCHAR(*n)*, data types:

Table updates can be slower in some circumstances.

Varying-Length Execution Time

When you use the VARCHAR(*n*) data types, the records of a table have a varying number of bytes instead of a fixed number of bytes. The speed of database operations is affected when the rows of a table have a varying number of bytes.

Because more rows fit in a disk page, the database can search the table with fewer disk operations than if the rows were of a fixed number of bytes. As a result, queries can execute more quickly. Insert and delete operations can be a little quicker for the same reason.

When you update a row, the amount of work the database must do depends on the number of bytes in the new row as compared with the number of bytes in the old row. If the new row uses the same number of bytes or fewer, the execution time is not significantly different than it is with fixed-length rows. However, if the new row requires a greater number of bytes than the old one, the database server might have to perform several times as many disk operations. Thus, updates of a table that use VARCHAR(*n*), data can sometimes be slower than updates of a fixed-length field.

OBJECT Data Types

Object data types include those denoted as BLOB and CLOB. The primary difference being BLOB types store data of any type, while CLOB data types are used for textual data.

Object data types are generally undeclared as a 'type' and can contain text, graphics, sounds or movies. Object data types can be of any size. Qilan treats all Object data types as "LONGVARCHAR", internally converting the data to the STRING type.

Caution should be exercised if the known Object data type is NOT textual. The default HTML content/type declared by Qilan is "text/html". This means that the browser will display Object data as text. If the data is an image for example, Qilan will display raw 'code', not the image. In order to display an image, sound, movie, etc., a separate window must be opened that explicitly sets the html content/type to the desired type.

Objects can be logically manipulated similar to any other STRING type. For instance, most databases support 'like' searches on Objects. Depending upon the Object's size, retrieval may take more time than standard CHARACTER types.

Changing the Data Type

After the table is built, you can change the data type that is assigned to a field. Although such alterations are sometimes necessary, they should be avoided due to the potential for data loss.

Understanding Dates and Times with Qilan

With the advent of Qilan 2.0, processing of dates and times in Qilan became a little bit more complicated. This section describes those complications, explain why they occur, and help the Qilan developer work with them.

Why is it so complicated?

The complications stem from the attempt to combine at least three different systems, each of which has it's own idea about what a date and/or time is: SQL, JDBC and Qilan.

The SQL Standard.

The SQL Standard defines five different data types for dealing with dates and times:

DATE
TIME
TIMESTAMP
TIME WITH TIME ZONE
TIMESTAMP WITH TIME ZONE

These types hold the kind of data one would expect. A DATE specifies a date, including a month, day, and year. A TIME indicates a time, including the hour, minute, second, and fractions of a second down to a microsecond or smaller. A TIMESTAMP is a combination of both DATE and TIME. By adding WITH TIME ZONE, the time includes the number of seconds away from GMT this time is.

The standard allows some conversion between these types. TIMESTAMPS may be converted to and from DATES and TIMES, but DATES and TIMES may not be converted to each other. TIMES and TIMESTAMPS without WITH TIME ZONE represent local times.

JDBC

JDBC defines only three distinct data types for dealing with dates and times:

DATE
TIME
TIMESTAMP

These correspond, roughly, to their SQL counterparts. But where the SQL DATE, TIME, and TIMESTAMP represent an abstract, anywhere in the world, date or time or both; in JDBC, these represent the date, time, or both specifically where the JDBC Client is located.

JDBC has a very different idea TIMEZONE. To JDBC, a TIMEZONE is an area of the globe that is keeps its clocks at some fixed offset from GMT, and changes to daylight savings time for some (possibly empty) period of the year.

Each database system has its own JDBC driver. Each driver may interpret the dates and times it gets from the database system anyway it deems necessary. It should, however, always present dates/times to Qilan in the local time zone. This means, for example, that a timestamp read from the database for a summer date will typically be one hour later than the a time stamp read from the database for the same time on a winter date.

JDBC also has no mechanism for dealing with the WITH TIME ZONE construct in SQL. When retrieving data from a WITH TIME ZONE column, the JDBC specification requires the driver to translate that to local time, and otherwise discard the time zone information. When storing data into a WITH TIME ZONE column, the JDBC specification does allow the program (qilan.cgi in this case) to specify what time zone to store, even though it refuses to reveal that information when retrieving that same column.

The generous and cooperative folks at FrontBase have extended the JDBC specification for us so that we can completely utilize the WITH TIME ZONE construct in FrontBase databases.

Qilan

Qilan.cgi defines only one time/date data type. It is effectively the same as the SQL data type TIMESTAMP WITH TIME ZONE. To be consistent with JDBC, times and timestamps created in qilan.cgi without time zones are assigned to the local time zone. A date (without a time) is kept internally as midnight (00:00:00) on that date in the local time zone. A time (without a date) is kept as that time on January 1, 1970 in the local time zone. These all follow from the JDBC specifications.

Various Databases

Of course, various databases may implement all, some, or none of the SQL data types, and the various JDBC drivers that go with the databases may interpret the specification differently.

What do I need to know to develop with Qilan?

Qilan takes care of most of the idiosyncrasies of each JDBC driver automatically. No matter how the database stores the data, Qilan endeavors to present this information in the same manner as each database's data access tool does.

In the following descriptions, the "local time zone today" is the time zone of the system time on the machine where qilan.cgi is running. That is subtly distinct from the "local times zone of the data", which is the time zone of the system time on the machine where qilan.cgi is running on the date represented by the data. These two could be different by an hour because of differences in Daylight Savings Time. Unfortunately, the different databases use different interpretations of the JDBC requirement to use the local time zone. The phrase "local time zone" denotes the interpretation used by the database data access tool, whichever it may be.

DATE

When retrieving a date from a database, the system delivers a value that reflects midnight on that date in GMT.

When storing into a date field, the system translates the value to GMT, and stores the date part in the database.

TIME

When retrieving a time from a database, the system delivers a value that reflects that time on January 1, 1970 in the local time zone.

When storing into a time field, the system translates the value to the local time zone, and stores the time part in the database.

TIMESTAMP

When retrieving a timestamp from a database, the system delivers a value that reflects that time in the local time zone.

When storing into a time field, the system translates the value to the local time zone, and stores the time part in the database.

TIME WITH TIME ZONE

When retrieving a time with time zone from a database, the system delivers a value that reflects that time on January 1, 1970 in the specified time zone.

When storing into a time with time zone field, the system discards the date, and stores the time and time zone part in the database.

TIMESTAMP WITH TIME ZONE

When retrieving a timestamp with time zone from a database, the system delivers a value that reflects that time on that date in the specified time zone.

When storing into a timestamp with time zone field, the system stores the date, time and time zone in the database.

Database Specifics

OpenBase

OpenBase implements only date, time, and timestamp (which it calls datetime). It uses the local time zone today for the local time zone.

FrontBase

FrontBase implements the complete SQL specification. It uses the local times zone of the data for the local time zone.

Informix

Informix uses server (or user) specific environmental variables to control many of its settings and parameters. Before you start using Qilan to access Informix databases, we strongly urge you to carefully review the environmental variables. Specifically, when DATE, DATETIME and INTERVAL data types are used to format and query data, environmental variables, such as 'DBDATE', can alter how text data is interpreted as dates or times. Depending upon the server configuration, Qilan conversions may not work correctly. Timezone, when configured, is supported by Informix.

Modifying Program Defaults

Modifying a DTD (Document Type Definition)

The document is provided so Qilan designers can modify standard DTDs.

> *Significant damage to the application or project, or functional anomalies may result when DTDs are changed, therefore CommonGround Softworks, Inc. assumes no responsibility for modifications or consequential damages resulting from such changes*.

DTDs define the general structure of an HTML document. Tags, attributes and their 'allowed' placement are specified. Qilan ships with six DTDs, corresponding to the version and use of Helix tags:

HTML 3.2
HTML 4.0 (strict)
HTML 4.0 (loose)
HTML 3.2 Helix
HTML 4.0 (strict) Helix
HTML 4.0 (loose) Helix

Accessing a DTD

The standard DTDs are located within the Qilan application; the Helix DTDs are located in the Helix.Extension.Bundle, inside the Qilan application. Each file is a text file and may be opened and edited with the Text Edit application. If you routinely use a single DTD, then you only need to edit that DTD.

Navigate to the Applications folder and locate Qilan. Press and hold the control key and click on the Qilan icon. A small submenu will appear. Select, "Show Package Contents". The Qilan package will open displaying the Contents folder. Double click to open the folder, then double click to open the Resources folder. The full path is as follows:

/Applications/Qilan/Contents/Resources/

The folder, "Resources", contains the DTDs for HTML 3.2, HTML, 4.0 (strict) and HTML 4.0 (loose). They can be identified by their suffix, ".dtd".

Modifying the DTD

Once you select the DTD you wish to modify, double click to open it. It should open with Text Edit. The top of the file appears as follows:

```
<!--
This is the HTML 4.0 Transitional DTD, which includes
presentation attributes and elements that W3C expects to phase out as
support for style sheets matures. Authors should use the Strict DTD
when possible, but may use the Transitional DTD when support for
presentation attribute and elements is required.

HTML 4.0 includes mechanisms for style sheets, scripting, embedding
objects, improved support for right to left and mixed direction text,
and enhancements to forms for improved accessibility for people with
disabilities.

Draft: $Date: 1998/04/02 00:17:00 $

Authors:
Dave Raggett <dsr@w3.org>
Arnaud Le Hors <lehors@w3.org>
Ian Jacobs <ij@w3.org>

Further information about HTML 4.0 is available at:

http://www.w3.org/TR/REC-html40
-->
```

The DTD is organized into sections defining each element. Do not make any changes, beyond the two suggested modifications unless you seek professional assistance. Doing so may make the DTD unusable or cause Qilan to behave erratically.

Requiring a Tag Attribute

Most tags have attributes. Attributes allow for the addition of specific characteristics, such as names, actions and methods. When a tag is dragged onto the WebTemplate, an attribute may be implied or required. When an attribute is implied, it is considered optional and appears on the Icon > Attribute submenu. Required attributes appear on the WebTemplate as soon as tag is dragged from the palette.

When an attribute is required by the DTD, we suggest that this not be changed. However, you may make any implied attribute required.

Here is an example of how to make a TABLE border attribute required. First, locate the 'TABLES' section in the DTD. Here is the start of this section:

```
<!--====================== Tables =======================-->

<!-- IETF HTML table standard, see [RFC1942] -->

<!--
The BORDER attribute sets the thickness of the frame around the table.
The default units are screen pixels.

The FRAME attribute specifies which parts of the frame around the table
should be rendered. The values are not the same as CALS to avoid a name
clash with the VALIGN attribute.

The value "border" is included for backwards compatibility with <TABLE
BORDER> which yields frame=border and border=implied.  For <TABLE
BORDER=1> you get border=1 and frame=implied. In this case, it is
appropriate to treat this as frame=border for backwards compatibility
with deployed browsers.
-->
```

Attribute descriptions follow. Let's move right to the "ATTLIST TABLE", as this describes the attribute list for the TABLE tag itself.

```
<!ATTLIST TABLE                       -- table element –

%attrs;                               -- %coreattrs, %i18n, %events
summary         %Text;     #IMPLIED   -- purpose/structure for speech
                                         output
width           %Length;   #IMPLIED   -- table width --
border          %Pixels;   #IMPLIED   -- controls frame width around
                                         table
frame           %TFrame;   #IMPLIED   -- which parts of frame to render
rules           %TRules;   #IMPLIED   -- rulings between rows and
colscellspacing %Length;   #IMPLIED   -- spacing between cells
%Length;                   #IMPLIED   -- spacing within cells --
align           %TAlign;   #IMPLIED   -- table position relative to
                                         window
bgcolor         %Color;    #IMPLIED   -- background color for cells
%reserved;                            -- reserved for possible future
                                         use
datapagesize    CDATA      #IMPLIED   -- reserved for possible future
                                         use
>
```

Locate the attribute, "border", in the first column. Note how the default is, "#IMPLIED". To make this attribute required, change, "#IMPLIED" to "#REQUIRED" as follows:

```
border          %Pixels;   #REQUIRED  -- controls frame width around
```

Don't forget the pound sign (#) and use all capital letters. Save your changes.

Now, when you drag out a TABLE tag, the attribute, 'border' will appear automatically.

Adding an Attribute

Under some circumstances, it may be necessary to add an attribute. JavaScript, for instance, requires a ‘name’ attribute for the FORM tag. None of the DTDs specify this attribute for the FORM tag. Here is the relevant DTD section:

```
<!--================ Forms ===================================-->

<!ELEMENT FORM - - (%flow;)* -(FORM)   -- interactive form -->

<!ATTLIST FORM
%attrs;                                 -- %coreattrs, %i18n, %events
action         %URI;          #REQUIRED -- server-side form handler --
method         (GETlPOST)     GET       -- HTTP method used to submit
                                           the form--
enctype        %ContentType; "application/x-www-form-urlencoded"
onsubmit       %Script;       #IMPLIED  -- the form was submitted --
onreset        %Script;       #IMPLIED  -- the form was reset --
target         %FrameTarget;  #IMPLIED  -- render in this frame --
accept-charset %Charsets;     #IMPLIED  -- list of supported charsets -
name           CDATA          #IMPLIED  -- NOT in the original dtd, but
                                           required by Javascript --
>
```

The last line shows the ‘name’ attribute added to the FORM attribute list. The first column is the attribute label, the second column the data source and the third column whether it is implied or required. We have chosen to use ‘CDATA’ as the source (user defined).

Adding an attribute requires knowledge of HTML, what attribute is supported by different browsers and platforms, and what types of attribute values can be used. Caution is advised.

Modifying Defaults.plist

The document is provided so Qilan designers can modify some of the Qilan Developer standard defaults.

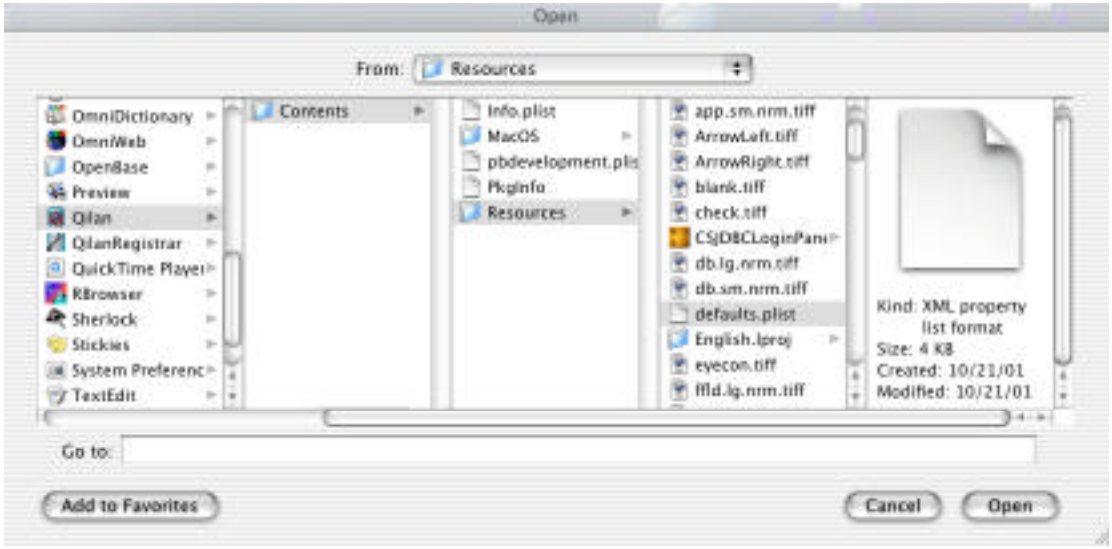
> *Significant damage to the application or project, or functional anomalies may result when the default.plist is changed, therefore CommonGround Softworks, Inc. assumes no responsibility for modifications or consequential damages resulting from such changes.*

The defaults.plist contains standard default settings for abacus fonts, sizes and colors, the order of abacus expressions on the palette and various WebTemplate settings such as indent measurements.

First log in as the root user, then navigate to the Applications folder and locate Qilan. Press and hold the control key and click on the Qilan icon. A small submenu will appear. Select, “Show Package Contents”. The Qilan package will open displaying the Contents folder. Double click to open the folder, then double click to open the Resources folder. The full path is as follows:

```
/Applications/Qilan/Contents/Resources/
```

Locate the file, ‘defaults.plist’. Double click on the file to open it.



Default.plist is in a very specific format, which must not be changed. Although values can be modified, be careful not to change names or remove brackets, equal signs, commas or semicolons.

Although you can edit default.plist directly, it is a lot easier (and safer) to use the developer application, "Property List Editor". The application is available from Apple Computer and is installed automatically with the OS X Developer Tools.

To change abacus fonts, sizes or colors, first identify the abacus object you wish to change (i.e., operators). The font family is displayed, by name, followed by the point size.

Abacus colors are within the range of 0 to 1, where 0 is black and 1 is white. The first and second line, refer to the unselected/selected colors.

The list of abacus tiles is used to order how abacus operators are shown on the palette. Blank lines are ignored.

At the bottom of the document, you will find settings for the WebTemplate. The values are in picas (12 points or about 1/6 of an inch).

Property List	Class	Value
▼Root	Dictionary	7 key/value pairs
▶abacusfonts	Dictionary	4 key/value pairs
▶abacusSelectedBackgroundColor	Array	3 ordered objects
▶abacustiles	Array	67 ordered objects
▶backgroundColor	Array	3 ordered objects
▶buttonbar	Dictionary	3 key/value pairs
▼templateAttributes	Dictionary	5 key/value pairs
nameTab	String	3.0
nameWidth	String	120.0
tabWidth	String	10.0
tagTab	String	3.0
tagWidth	String	130.0
▶templateHighlightColor	Array	3 ordered objects

Qilan must be restarted in order for changes to take effect.

QRUN Shell Scripts

Sample QRUN Shell Scripts

When using QRUN to execute a Unix shell script, you need to carefully consider several factors: where is the script located, what permissions are needed to execute the script, necessary parameters and the output, if any. From a technical point of view, you are automatically running an executable file. Computers are very fast, but not terribly flexible. Therefore, precision is a valuable commodity.

Calling a system script, such as 'cat', requires a complete path specification:

```
/bin/cat
```

The leading slash (/) is essential. 'cat', for example, can be used to import a file's contents directly into Qilan for display on a web page. The way this is performed is by specifing the file, then entering its full path after the executable specification.

```
/bin/cat /MyDocuments/ShowFile
```

The space after 'cat' separates the file to be executed from the file to be accessed. In this case, 'ShowFile' is located in the folder, 'MyDocuments'.

The attribute, outputicon is designed to accept standard out (stdout), meaning whatever the executable will be returning. A Qilan field is used to hold stdout for use in other constructions or 'Q' tags.

You are not limited to running system scripts. You can write your own scripts of any complexity, using Unix or any other scriptable language. Here are several examples. Each script is a unformatted text file with execute permissions given to the user www. Note that lines starting with the pound sign (#) are comments and not executed. The last line must end with a carriage return.

Example #1: AutoRun

```
#!/bin/tcsh

#Sets an environmental variable
#Sets the path to the Qilan Script to be executed
setenv PATH_TRANSLATED "/Library/WebServer/Documents/MyWebTemplate"

#Sets the path to the Qilan support files
cd /Library/WebServer/CGI-Executables

#Redirects qilan.cgi output to a null device (no output)
/Library/WebServer/CGI-Executables/qilan.cgi > /dev/null
```

This script, when called using a QRUN, executes a Qilan WebTemplate named, “MyWebTemplate”, located in /Library/WebServer/Documents. It passes no parameters and returns no output. The last line tells the script to send all output to /dev/null. The Unix way of saying ‘don’t bother’.

The first line, although starting with a pound character, is actually a command. It means to open a shell (tcsh). QRUN is not actually executing this script it is being run in the shell tcsh.

Qilan templates can be written to perform automated activities requiring no user input, such as moving data between databases or periodically updating files. QRUN can trigger AutoRun within a Qilan WebTemplate using Cron, on a time basis, or any other mechanism capable of running an executable file.

Example #2: AutoRunOutput

AutoRunOutput will execute a Qilan WebTemplate, then return its output to Qilan as standard out (stdout). Use QRUNs outputicon attribute for stdout.

```
#!/bin/tcsh

setenv PATH_TRANSLATED "/Library/WebServer/Documents/[HTML_Name]"

/Library/WebServer/CGI-Executables/qilan.cgi
```

Example #3: AutoRunParameters

This example will also run a Qilan WebTemplate, but will pass parameters, such as a date or record number. In this example, we will be using the QRUN attribute, inputicon or standard in (stdin).

```
#!/bin/tcsh
#Sets environmental variables

#Sets the path to the Qilan Script to be executed
setenv PATH_TRANSLATED "/Library/WebServer/Documents/[HTML_Name]"

setenv QUERY_STRING "[fld1name]=[fld2value]&[fld2name]=[fld2value]"
#Names/Values inside brackets should be form encoded; omit the brackets

setenv CONTENT_LENGTH "0"
#Redirects qilan.cgi output to a null device (no output)

/Library/WebServer/CGI-Executables/qilan.cgi > /dev/null
```

This example will ‘accept’ two fields and their corresponding values as stdin. Formatting the fields is easy, so long as you conform to the syntax:

```
[field1name]=[field2value]&[field2name]=[field2value]
```

Field names or values containing spaces, equal signs or ampersands, should be form encoded using the abacus, URL Encode. Otherwise the script will fail.

This script will pass field names and corresponding field values. The Framework containing the target WebTemplate should have fields of the same name as the source fields. When the target WebTemplate is run by the script, those fields will automatically be valued.

Example #4: AutoRunDynamic

This example will also run a Qilan WebTemplate, but will pass dynamic parameters, those that can vary. In this example, we will be using the QRUN attribute, inputicon or standard in (stdin). The syntax is '$#'. A number corresponding to the order in which the variable appears replaces the pound sign.

```
#!/bin/tcsh

#Sets environmental variables

#Sets the path to the Qilan Script to be executed
setenv PATH_TRANSLATED "/Library/WebServer/Documents/$1"
#Where '$1' is the first parameter

setenv QUERY_STRING "$2=$3&$4=$5"
#Where Names/Values are parameters

setenv CONTENT_LENGTH "0"

#Redirects qilan.cgi output to a null device (no output)
/Library/WebServer/CGI-Executables/qilan.cgi > /dev/null
```

```
( "/Public/AutoRunUpdate" followed by" " followed by"mrn_unit_number" followed by" " followed by
mrn_unit_number followed by" " followed by"session_id" followed by" " followed by session_id )
```

In the above example:
The WebTemplate name, 'AutoRunUpdate is $1.
The field name, 'mrn_unit_number' is $2.
The field value, 'mrn_unit_number is $3.
The field name, 'session_id' is $4.
The field value, 'session_id' is $5.

Note the spaces between the field names and field values.

Additional Appendices

Installing FastCGI

Before beginning the installation of FastCGI, insure you have a valid internet connection and have installed Apple's Developer Tools for your operating system. These can be found on your system installation CD. If you already have FastCGI installed, there is no need to re-install and this section can be omitted.

Do not attempt to use a pre-compiled FastCGI installer unless specifically designed for your Apache application and configuration parameters. Doing so otherwise may result may result in damage to your httpd.conf file and/or Apache application.

If you are updating the Qilan engine from a version prior to 2.8, you *must* re-intall FastCGI by running the installation scripts. The reinstallation process will replace earlier versions.

The Qilan installation includes two FastCGI installation scripts, one for Apache version 1.3.x and another for 2.0.x. Before you begin, insure you have one or both of the Apache applications completely installed. By default, Mac OS X will install Apache 1.3.x. Mac OS X Server provides the option of installing Apache version 2.0.x. Apache 2.0.x is also available as a separate install from versiontracker.com.

Installing FastCGI and Apache should follow a specific scenario. Our recommendation is as follows:

- Determine your OS – OS X Client or OS X Server.
- Determine which version of Apache will be used – 1.x or 2.x.
- Determine which Apache package will be used – Apple or Faby.

We do not recommend running Apache versions 1.3.x and 2.0.x on the same machine at the same time. You can safely install both versions, as long as one or the other is disabled. If you need to enable both versions on the same machine, please contact support@commongrnd.com for assistance.

Installation for Mac OS X Client

Locate the script, “installfastcgi_apache13.command” or “installfastcgi_apache20.command”, found in `/Library/Qilan/fastcgi`. Double click to begin the download of FastCGI and installation process. Your administrative password will be required.

The following processes will occur:

- The Terminal application will launch and execute the script
- Your system will be checked to insure required files are present (Apache source files, System files necessary for compilation, etc.)
- FastCGI source will be downloaded from http://www.fastcgi/com
- Mod-fastcgi compiled and installed
- Removal of existing fastcgi.conf Include statements from httpd.conf; installation of new fastcgi.conf Include statements
- Apache will be restarted
- Removal of all temporary files, including the FastCGI source.

Quit the Terminal application.

Installation for Mac OS X Server

Locate the script, “installfastcgi_apache13.command” or “installfastcgi_apache20.command”, found in `/Library/Qilan/fastcgi`. Double click to begin the download of FastCGI and installation process. Your administrative password will be required.

The following processes will occur:

- The Terminal application will launch and execute the script
- Your system will be checked to insure required files are present (Apache source files, System files necessary for compilation, etc.)
- FastCGI source will be downloaded from http://www.fastcgi.com
- Mod-fastcgi compiled and installed
- Removal of all temporary files, including the FastCGI source.

Additional and required changes to the httpd.conf file must be performed manually. You will be prompted to do this at the end of the script's execution. Apache’s configuration file, “httpd.conf”, will be opened for

you. If you do not have write permission, log out and back in as the root user, then navigate to `/etc/httpd/` and open the file manually.

Scroll to the bottom of the file. It should appear as follows:

```
#LoadModule jserv_module         /usr/libexec/httpd/mod_jserv.so
LoadModule  fastcgi_module       libexec/httpd/mod_fastcgi.so
#AddModule  mod_jserv.c
AddModule   mod_fastcgi.c
#Include                         /private/etc/httpd/tomcat.conf
```

Note, lines proceeded by '#' are not executed (commented out).

Now, change the above lines so that they appear as follows:

```
#LoadModule jserv_module     /usr/libexec/httpd/mod_jserv.so
LoadModule  fastcgi_module   /usr/libexec/httpd/mod_fastcgi.so
#AddModule  mod_jserv.c
AddModule   mod_fastcgi.c
#Include                     /private/etc/httpd/tomcat.conf
Include                      /Library/Qilan/fastcgi/fastcgi.conf
```

The LoadModule fastcgi_module path is changed along with the addition of a new 'include' line at the end.

Confirm your modifications, then save and close the file. Using the web server administration application, restart the web server.

FastCGI Configuration

Locate the file, “fastcgi.conf” in `/Library/Qilan/fastcgi/`. This is the configuration file for FastCGI. We have configured the file for basic usage, but you may alter it according to your situation and needs. Please refer to the FastCGI website http://www.fastcgi.com for more information and parameter settings.

FastCGI Error Checking (OS X and OS X Server)

To confirm that FastCGI has been successfully installed, access a Qilan exported WebTemplate, from the browser, using the path:

```
[hostname]/fcgi-bin/qilan.fcgi/[WebTemplate_name]
```

If your WebTemplate is returned, then FastCGI has been properly installed.

Installing Apache 2.0.x

You can obtain Apache 2.0.x from a website or install it from your OS X Server installation disks.

Installing the Faby Distribution

The Faby distribution, available from www.versiontracker.com, is the commercial version of Apache 2.0.x. It is a complete installation package, including a system preference pane to control the startup and basic setup. Following the instructions, Apache will be installed in a directory (/Library/Apache2) that differs from the default Apple location (/Library/WebServer). As defined by the Apache 1.3.x configuration file (httpd.conf), references to cgi and web documents will refer to /Library/WebServer as the root location. Rather than changing all enclosed items to the new 'root' directory for Apache 2.0.x, we suggest modifying the configuration file. Changes are made in httpd.conf Apache 2.0.x as follows:

Change *Document Root*
from: /Library/Apache2/htdocs
to: /Library/WebServer/Documents

Change *Directory*
from: /Library/Apache2/htdocs
to: /Library/WebServer/Documents

Change *ScriptAlias /cgi-bin/*
from: /Library/Apache2/cgi-bin/
to: /Library/WebServer/CGI-Executables/

Change *Directory*
from: /Library/Apache2/cgi-bin
to: /Library/WebServer/CGI-Executables

> The Faby distribution will assign port 8080 to Apache 2.0.x. When you reference web pages, you must specify the port as port 80 will be used by default. Using the URL syntax can do this: http://localhost:8080/. Alternately, you can change the default port in your System Preferences. We do not recommend running versions Apache 1.3.x and Apache 2.0.x at the same time.

Modifying the FastCGI Installation Script

After installing and configuring Apache 2.0.x, FastCGI can be installed. Locate the file, “installfastcgi_apache20.command”, located in /Library/Qilan/fastcgi/.

> IMPORTANT: Please insure you are logged on as a system administrator, have access to the internet and installed Apple’s Developer Tools.

The Apache 2.0.x installation script shipped with Qilan is designed to be used with the Apple distribution. In other words, it expects to find Apache 2 in /opt/apache2. To use the installation script with the Faby distribution (or any other location), you must manually edit the installation script. To do this, use a text editor and open, “installfastcgi_apache20.command”. Go to first section (line 14) and change the location:

```
set apache_dir = /Library/Apache2      {Faby default}
set apache_dir = /opt/apache2          {Apple default}
```

When you are ready, double click the installer to proceed. The installer will download the correct version of FastCGI, modify configuration settings and install the necessary files for Apache2.

Qilan Field Coercion Reference

OpenBase Field Type	Qilan Coercion
Char	String
Date	Date
Datetime	Date
Float	Float
Int (Integer)	Int (Integer)
Long	Int
Longlong	Int
Money	Float
Object	String
Time	Date
Varchar	String

FrontBase Field Type	Qilan Coercion
Boolean	Int
SmallInt	Int
Integer	Int
Interval year to month	Int
Numeric	Float
Decimal	Float
Float	Float
Real	Float
Double precision	Float
Interval day with second	Int
Time	Date
Time with zone	Date
Date	Date
Timestamp	Date
Timestamp with time zone	Date
Character	String
Varchar	String
Character Varying	String
Bit	**Not Supported**
Bit varying	**Not Supported**
Blob	String
Clob	String

Command Key Equivalents

Function/Action	Key/Key Combination		
QILAN MENU			
Preferences…	⌘		**,**
FILE MENU			
New Project	⌘		**N**
Open	⌘		**O**
Save	⌘		**S**
Save As…	⌘	Shift	**S**
Revert to Saved	⌘		**U**
Export HTML	⌘		**H**
Export All HTML	⌘	Shift	**H**
Page Set-up	⌘	Shift	**P**
Print	⌘		**P**
Quit	⌘		**Q**
EDIT MENU			
Undo	⌘		**Z**
Redo	⌘	Shift	**Z**
Cut	⌘		**X**
Copy	⌘		**C**
Paste	⌘		**V**
Paste Into	⌘	Shift	**V**
Clear	⌘	Shift	**X**
Clear		Delete	

Duplicate	⌘		**D**
Select All	⌘		**A**
Spelling…	⌘		**:**
Check Spelling	⌘		**;**

ICON MENU

New Access Icon (Project Window)	⌘	Option	**A**
New Framework Icon (Project Window)	⌘	Option	**F**
New Abacus Icon (Child Windows)	⌘	Option	**A**
New Field Icon (Child Windows)	⌘	Option	**F**
New Table Icon (Access Window)	⌘	Option	**T**
New Relationship Icon (Child Windows)	⌘	Option	**R**
New DataFlow Icon (Child Windows)	⌘	Option	**X**
New WebTemplate Icon (Child Windows)	⌘	Option	**W**
Open Icon	⌘	Option	**O**
Open Uses	⌘	Option	**U**
Open Parent	⌘	Option	**P**
Expand All	⌘	Shift	**{**
Expland Selection	⌘		**[**
Contract All	⌘	Shift	**}**
Contract Selection	⌘		**]**

WINDOW MENU

Close	⌘		**W**
Minimize	⌘		**M**
Show Palette	⌘		**/**

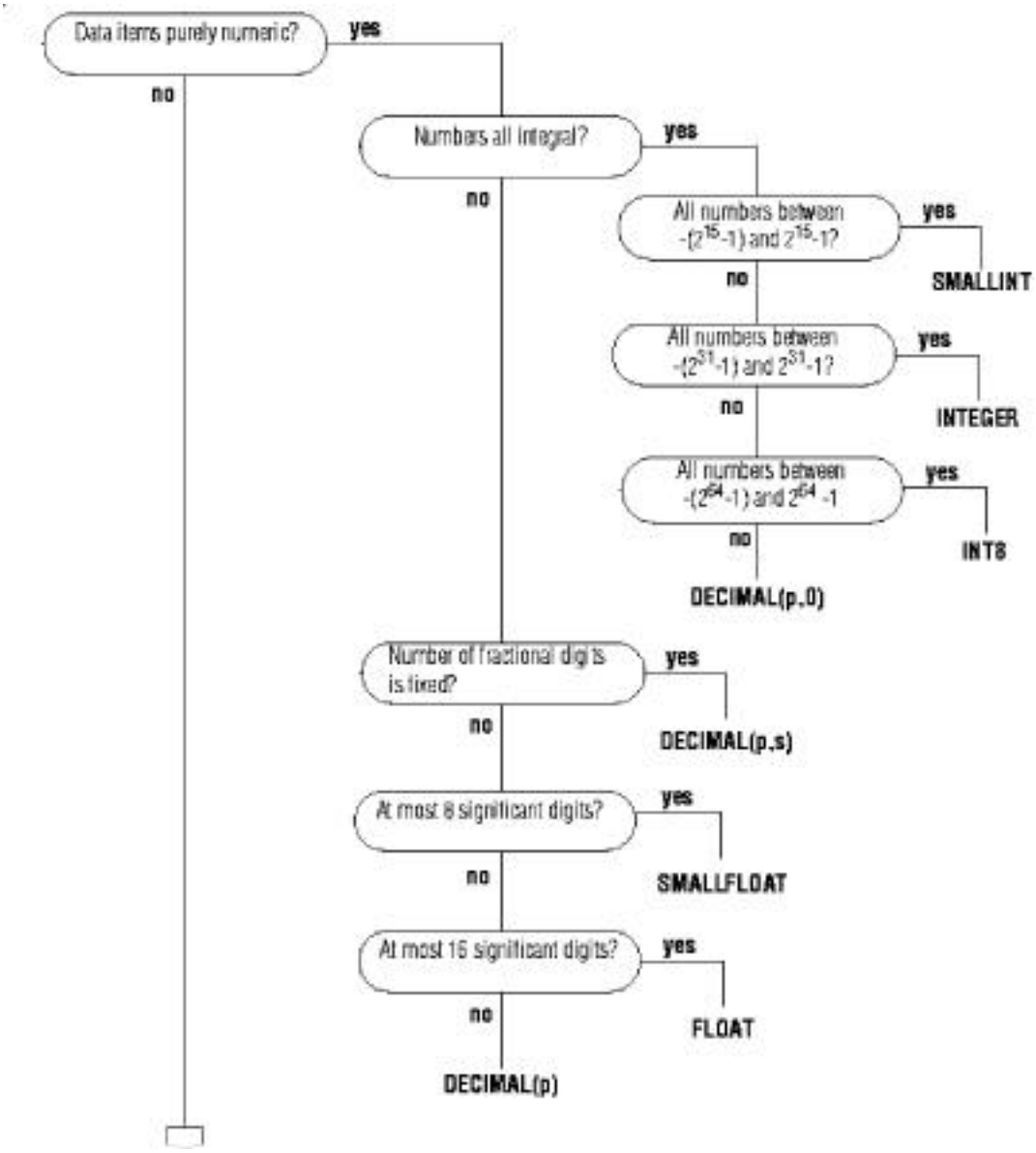
HELP MENU

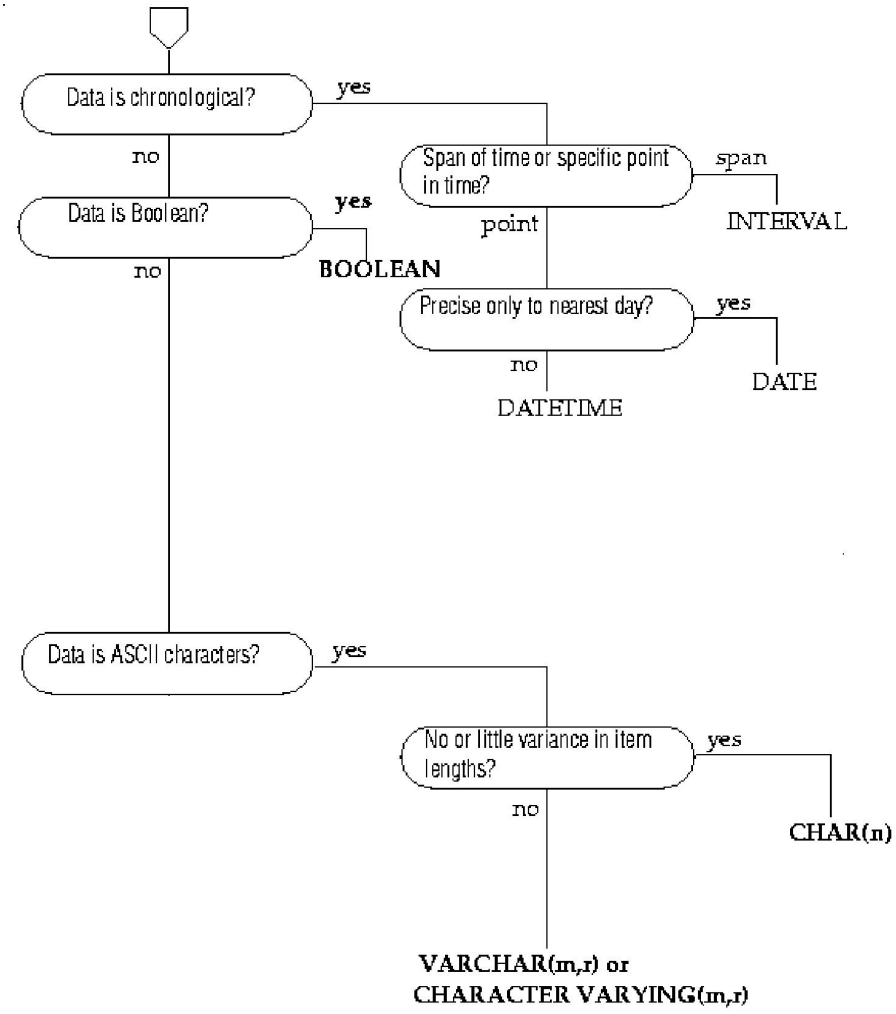
Qilan Help	⌘		**?**

Using Data Types in Relationships

Almost all data type combinations must match (or be able to be coerced) when you are trying to match source fields/abaci and target fields.

The following diagrams show the decision tree that summarizes the choices among data types.




Data is chronological?
yes
no
Span of time or specific point in time?
span
INTERVAL
point
Data is Boolean?
yes
BOOLEAN
no
Precise only to nearest day?
yes
DATE
no
DATETIME
Data is ASCII characters?
yes
No or little variance in item lengths?
yes
CHAR(n)
no
VARCHAR(m,r) or
CHARACTER VARYING(m,r)