Encodo Systems AG – Garnmarkt 1 – 8400 Winterthur
Telephone +41 52 511 80 80 – www.encodo.com

The Quino Metadata Framework

Fact Sheet

Abstract

This document offers an analysis of the advantages of working with metadata in general and with Quino in particular. It includes a high-level introduction to the metadata core of Quino as well as the constellation of components that work with that metadata.

Authors

Marco von Ballmoos
Remo von Ballmoos

Version 1.0.1 – 20.10.2009

Table of Contents

Version History

Version	Datum	Author	Comments
1.0	30.09.2009	MvB	Initial Version
1.0.1	19.10.2009	MvB	Updated "2.3 – Expressions" with usage examples; added date and version to title page

Referenced Documents

Nr./Ref.	Document	Version	Date
[1]			

Open Issues

Nr./Ref.	Document	Version	Date

Terms and Abbreviations

Term / Abbreviation	Definition / Explanation

1 Overview

What is Quino? Quino is a metadata framework written in C# 3.5. How does a *metadata* framework differ from an *application* framework? Application frameworks generally dictate much of the infrastructure of an application. An application can extend the framework only if it offers some way of extending it; if not, the application developer is often left without help at all. If a developer wants to extend or improve the user interface, for example, they have to work within the bounds defined by the user interface support provided by the application framework.

Quino, being a metadata framework, is different. The philosophy behind Quino is that metadata is great. Metadata enables generic programming and allows a developer to write or generate entire swathes of your application with very little effort—and great results—and therefore free up previous time to fine-tune the parts of the application that make it really stand out. It's about spending time on the stuff that really matters instead of down in the trenches, connecting to databases, marshalling objects or painstakingly placing controls on forms or web pages.

1.1 Reinventing the Wheel?

That sounds great, but doesn't .NET already offer support for metadata? With a lot of attributes and the reflection library, it's possible to use the application's .NET code as a model, isn't it? In principle, yes; in practice, the .NET metadata isn't such a good fit for defining application models. Not only that, but the reflection API is quite low-level and nowhere near as easy to work with as the Quino model API. The Quino model is available as a graph of completely standard .NET objects. Applications can navigate and browse this metadata intuitively because the application model is composed declaratively instead of gleaned from model intended for another purpose (i.e. generating an executable).

Having established the greatness of metadata—and Quino metadata in particular—it should be obvious that the first thing one does when developing with Quino is to define the metadata for your application in terms of classes, properties and relationships. This model and its components provide the information on which a whole constellation of supporting components are built, wonderful things like automatically-generated user interfaces, automatic schema migration, seamless integration with reporting tools, generation of business objects, pain-free object-relational mapping (ORM), remote query and method execution for thin clients and so much more.

1.2 Defining a Model & Workflow

Building a model in Quino doesn't involve esoteric mapping from a graphical modeling tool to code and back; instead, developers get to work in their comfort zone: .NET code. This has several advantages:

- Developers don't have to learn a new language in order to define a model. (Did we forget to mention that you should know a .NET language in order to use Quino?)
- The construction of the model is statically checked by the compiler.
- You have the full power of code-completion—and any other Visual Studio add-ons—at your disposal.
- The model can contain *any* .NET-code, which means that components that use the Quino model can use not only the more abstract Quino metadata, but also highly-specific application code.

Building a model also doesn't require a developer to know everything about an application before beginning. With Quino, designing and coding can be a simultaneous process, ironing out kinks in the model that only become apparent when something is actually on-screen instead of specified in a document.

Those paying attention will have noticed that we slipped in the feature "automatic schema migration" in with many others. We're not kidding about that (see below for more information).

2 What Are the Kinds of Metadata?

The main part of a Quino model is a list of metadata classes (meta-classes) and the relationships between them. Meta-classes can be grouped into modules to keep larger models organized. Meta-classes have properties, methods, relations and indexes. Meta-classes and relationships can set default filtering and sorting options. All textual metadata can be specified in multiple languages.

Methods can bind application code to the model, which can, for example, be made available by user interface components on instances of that meta-class. Properties can be persistent (stored in the database), transient (not stored in the database) or calculated, retrieving a value from either application code or from a Quino expression (more on that later).

The main types of metadata are:

- Models
- Classes
- Relations
- Properties
- Methods
- Indexes
- Aspects (see below)
- Constants/Resources/Assets (see below)

The supported data-types for properties are:

- Text
- Byte
- Int16
- Int32
- Int64
- Boolean
- DateTime
- Time
- Date
- TimeSpan
- Stream
- Binary
- Key
- Currency
- Double
- Password

The "Key" type above is used for primary and foreign keys and can be automatically generated for new objects (e.g. "identity" or "serial" on the database).

2.1 Aspects

Aspects are type-safe but customizable metadata that can be attached to other metadata in order to extend it in component- or application-dependent ways. For example, the Quino user-interface components render a meta-class without any aspects very nicely. However, almost any application will want to customize the appearance of that meta-class when rendered, perhaps by putting properties on different tabs or into labeled field-sets.

That's where aspects come in: the user interface components will use certain types of aspects as rendering hints, if they are present on the properties of the meta-class. Applications use aspects to customize the short description or columns in a list for a meta-class, as well. Similarly, the Quino ORM and Reporting components check for certain types of aspects in order to let the metadata customize data-retrieval.

This standardized approach makes it easy for developers to attach their own aspect types and to develop components (user interface or otherwise) that use those aspects. And, aspects that are general enough can be used by multiple components in different ways (e.g. the "color" aspect can be used by both the ASP.NET and Winform user interfaces).

2.2 Constants/Resources/Assets

Metadata elements need to reference external assets, such as images, icons or strings. .NET offers resource files as a solution to collecting these into an assembly. Quino provides support for integrating such resources, as well as standard constants like strings, integers and other base types into collections of constants in the metadata. Instead of making direct references to external resources, all other metadata refer instead to these constants. Text constants support multiple languages.

2.3 Expressions

Quino models include support for an expression language based on the C# language syntax. Various values in the model—like text captions, Booleans like "enabled" or "visible" or colors—can refer either to a fixed value or an expression. The language includes the following features:

- Standard arithmetic
- Standard Boolean comparisons
- String comparison (e.g. "begins with", "contains", etc.)
- String formatting templates (e.g. "{Name}, {FirstName}")
- Path expressions (e.g. "Company.Contact.MainPhoneNumber")

The evaluation engine is extensible and can include application-specific functions. As detailed in the ORM section, many of the metadata-only expressions can be mapped directly to the database.

2.4 Methods & Remoting

Quino metadata includes two kinds of method: meta-class–based methods and remotable methods. Methods defined in a meta-class are treated as properties and can appear automatically in dynamically generated user interfaces. Remotable methods are static business logic that works with either generic or application-specific (generated) objects. The client application remains unaware of whether a method is executed locally or remotely.

A Quino application server to handle remote method calls and queries need not be based on IIS; instead, use a standard .NET HTTP server and keep things simple, at least during development.

2.5 Runtime Extensibility

Having your application metadata defined in one place is one thing, but what about defining metadata on the fly in your application? The Quino user interface components do this quite a lot, generating metadata based on your application metadata—on the fly—in order to render a user interface without key fields or to let the user customize the displayed columns and so on. Much of this is accomplished by *wrapping* metadata, which allows one piece of metadata to be based on another.

Wrappers are very powerful and, together with automated schema-migration, help your applications solve the problem of deployment-specific models. With Quino, adding a few extra fields or hiding some classes for a single client is no longer a problem.

3 Data-access, ORMs & More

3.1 Generic API & Generalized data access

It almost goes without saying that the Quino API provides highly generalized access to your metadata and to the data loaded by the ORM. Unlike many other solutions, you're not stuck working with *only* your business objects: Quino lets you easily write generalized components for your applications that will work with *any* Quino model instead of being application-specific. The generic API means that components that use Quino metadata can much more easily provide a generalized interface to external components (e.g. reporting tools).

When developing multiple products, this saves you time, effort and, above all, money.

3.2 The Quino ORM

Quino's ORM uses a code-based querying API that works directly with the metadata, which means that an application is not *required* to generate code in order to query the database. The generic API allows a development team to quickly build the model, testing it with unit tests or to see how it looks in a user interface before committing to a particular representation by generating code and writing business logic against it. This approach allows for super-fast prototyping.

Whether or not an application uses generated code or not, the following features are available in the Quino ORM:

- Restrictions of all kinds (e.g. expression-based, direct, even on :n relations)
- Inner and left outer joins
- Sorting on properties and some expressions
- Lazy loading or both objects and delay-loaded properties
- Support for retrieving object graphs (beta)
- Freely mix custom SQL in to the query for those parts of the query that cannot be expressed with Quino's API
- Caching (extensible by the application)

- Support for multiple data languages (i.e. values in different languages for the same property)

Queries that include expressions that cannot be mapped to the database (e.g. those that include .NET code) cannot be executed.

3.3 Schema Migration

With the full metadata of an application available, Quino can offer a full-fledged schema migration solution that maximizes safety, consistency and developer comfort and efficiency.

The Quino schema migrator supports many of the most common model changes right out-of-the-box, allowing developers to fine-tune the application model as they go along, changing types, names and adding or removing relations and properties as requirements change or solidify. No more manual updates to a shared schema versioning script; no more worrying that downloading from source control will break the local build: just download the changes and Quino will update the schema of the local database to match the model.

The schema migrator includes the following features:

- Schema import (quality of imported model highly dependent on quality of database schema)
- Comparison of two models to produce a migration plan
- Schema migration, with support for:
- Tables
- Fields
- Static default values
- Indexes (standard "btree")
- Foreign key constraints with cascading actions
- Seamless support for .NET enumerated types

3.4 Supported Databases

Quino supports data-retrieval and full schema-migration on the following databases:

- PostgreSql 8.1.x–8.3.x (8.4 support is still in beta)
- Sql Server 2005; includes specialized support for automatically establishing triggers for unsupported cascades (2008 support is still in beta)

3.5 Remote Query Execution

But, wait, there's more! The ORM also has a driver that provides support for remote query execution. That is, clients can request data from a metadata-aware application server instead of directly from the database. This functionality is completely transparent to the rest of the application. Developers can use a direct database connection while the application is in development, and then switch to the remoting driver for testing and deployment.

3.6 Creating Models & Model Persistence

Models are easiest to create directly in .NET code, but that's not the only way to get the job done. Once a model exists, it can be persisted and restored from elsewhere. Together with dynamic model generation (as described above), an application could provide highly extensible and customizable solutions for individual customers.

A hypothetical application could load its main model and then load and apply customer changes—in whatever form—from a separate file or perhaps even a database. Once the application has merged all changes into its main model, extending it at run-time, Quino can take over again, migrating the database (if necessary) and running the application.

Quino can also import models from supported databases. That means Quino can be useful on projects that don't even use Quino for anything else (see how non-intrusive a metadata framework can be?) Import the model from an existing database and browse around the data in a completely generic way. Or, store the model as C# .NET code and work from there, using a Quino model instead of a schema maintenance script to keep your database up-to-date.

4 User Interfaces, Reporting & More

4.1 User Interface

4.1.1 Winform

Note: Many of the Winform components are based on components from the DevExpress libraries.

Another huge advantage in using Quino access to automatically generated user interfaces. Quino offers several components, from individual metadata-aware controls to multi-control user controls (e.g. tree with list/detail) to entire forms that make integrating Quino data into applications a breeze. Using these components, developers can build their own forms, using Quino only where needed or desired. Quino can also handle all user interface needs with a main form for an SDI or MDI application.

Quino's Winform support includes:

- Lists
- Trees
- Grids
- Various edit controls (e.g. edit, lookup, checkbox, image etc.)
- Edit panel (single-object editor)
- Explorer component (tree with list/detail)
- Navigator component (web-style with breadcrumbs and back/forth UI)
- Winform-compatible dynamic data-binding support
- Toolbars (object manipulation, reporting, timeline, etc.)

Some Quino controls (e.g. lists, trees, etc.) work together using a dynamic context concept, which allows them to be chained together and automatically update on context-changes (e.g. change in selection or object state).

4.1.2 ASP.NET (beta)

Note: Many of the ASP.NET components are based on components from the DevExpress libraries.

Quino's metadata support also extends to web applications, with the following features:

- ASP.NET-compatible data-source
- Grids
- Automatic per-request connection handling

4.1.3 ASP.NET MVC (beta)

With the release of ASP.NET MVC, Microsoft has delivered what we think is a very compatible companion for Quino on the web. So far, we have support for the following:

- Various HTML-formatter extensions for rendering properties, classes, read-only lists
- Controller that delivers information out of the metadata (captions, images, etc.)
- Automatic per-request connection handling

4.2 Reporting Integration

Note: The primary implementation of Quino's reporting library integrates with reporting components from DevExpress.

An essential component for almost any application is reporting, for which the Quino framework provides the following components:

- An end-user designer for Winform applications with access to the metadata (classes, properties, relations)
- Context-sensitive reporting integrated into the standard user interface
- Extensible context-detection (e.g. show reports for sub-objects of the selected object, as well)
- Export to Excel, Word, PDF, HTML, Text, CSV
- Platform-independent reports (i.e. use the same reports for both Winform and ASP.NET)
- A graphical previewer for both Winform and ASP.NET applications
- Templates (beta)
- Sub-reports (beta)
- Persistence for reports in the database or the file system
- Support for parameterized reports with a Winform user interface for choosing values
- Pre-processing for report data (based on metadata aspects)

4.3 Other Modules

Quino comes with metadata-based solutions for some common application domains:

- Timeline – a persistent aspect that adds data to a timeline, easily allowing different objects to be valid in a particular relationship at different times
- Change-tracking & notification – a persistent aspect that automatically tracks changes to objects as well as a publication system for collecting and distributing notifications
- Security (beta) – a module that provides highly extensible support for ACL-based metadata- and content-dependent security as well as code-based security (virtual ACLs).
- Membership and Role provider implementation for .NET

4.4 Tools Support

Quino comes with a few tools to aid in development:

- Schema Migrator (Winform) – a wizard-based schema migrator appropriate for end-user IT
- Schema Migrator (Console) – a command-line schema migrator for developers
- Database Browser – imports a model from a supported database and displays it in a data explorer (supports CrUD and is useful for non-Quino databases, as mentioned above).
- Code Generator (Console) – a command-line generator for Quino business objects

4.5 Application Support

Quino isn't an application framework, but it does come with some extra goodies to make building applications—especially Quino, metadata-based applications—easier:

- Support for configuration of all facets of a Quino application: database, model
- Command-line handling for console and Winform applications
- Integrated logging support
- Support for automatic application updates
- Robust initialization support for web, console and Winform applications using the configuration above to load the model and database, verify and migrate the database schema and generally ensure that the application is ready to run