



dbFramework

The Premier Rapid Web-Development Platform

User Guide

Written By John Wood

RiaForm Technology

Date: 20th May 2011

Revision: 0.12

Status: DRAFT

Contents

User Guide	1
1. Introduction.....	13
1.1. Background	13
1.2. What can I create using dbFramework?	14
2. Application Design.....	17
2.1. Concepts	17
2.1.1. Workflow	18
2.1.2. The Data Model	18
2.1.3. Automation.....	20
2.1.4. Business Logic	20
2.1.5. One-Click Features.....	22
2.1.6. Reports.....	23
2.1.7. Advanced Programming	24
2.1.8. Deployment	25
3. Installation.....	26
3.1. Licensing	26
3.2. Installation Procedures.....	26
3.2.1. dbFramework Express and WorkflowFirst.....	26
3.2.2. dbFramework Professional	27
3.2.3. PDF Reports	27
4. Accessing dbFramework.....	28
4.1. A Brief Tour of the OfficeSuite Launcher	28
5. The Common Web Interface	30
5.1. Page URL	30
5.2. Logo and User Options	31
5.3. Application Tabs	31
5.4. Navigation Bar	32
5.5. Record List	32
5.6. Record Detail	36
5.7. Auto-Expands.....	38
5.8. Edit Mode	38
5.9. The Actions List.....	42
5.10. Reports List	43
5.11. Parent Records.....	43
5.12. Cell Phone Access	44
6. Creating An Application Data Model.....	45

6.1.	Application Wizard.....	45
6.2.	Publishing the Application.....	46
6.3.	Application Configuration.....	46
6.4.	Understanding Data Modeling	49
6.5.	Creating and Editing Type Definitions	53
6.6.	Creating and Editing Field Definitions	54
6.6.1.	Adding a Field	54
6.6.2.	Naming Your Field.....	55
6.6.3.	Specifying a Data Type.....	56
6.7.	Creating a Hierarchy	57
6.8.	Example: An Invoice Manager	59
6.9.	Anatomy of a Field.....	63
6.9.1.	Field Options.....	63
6.9.2.	Field Flags.....	72
6.10.	Creating Dropdown Lists.....	73
6.11.	Conditional Coloring	75
6.11.1.	Overview	75
6.11.2.	Configuring Renditions	75
6.11.3.	Ordering.....	77
6.12.	Specifying Constraints and Indexes	78
6.13.	Inheriting Types	79
6.14.	Links	79
6.15.	Advanced Links	80
6.16.	Adding Features.....	82
7.	Workflow.....	83
7.1.	Introduction	83
7.2.	Workflow Basics.....	84
7.2.1.	Creating a New Application	84
7.2.2.	Adding the Workflow.....	85
7.2.3.	Workflow Stages.....	89
7.2.4.	Concepts	89
7.2.5.	Adding a Stage	95
7.2.6.	Adding Fields to Collect	97
7.2.7.	Making Sub-Lists	98
7.2.8.	Publishing the Application	100
7.2.9.	Adding Script.....	100
7.2.10.	Deleting or Moving a Stage.....	101
7.3.	Workflow Features	101
7.3.1.	Combining Workflows	101
7.3.2.	Notifications and Alerts	102
7.3.3.	Workflow Reports.....	103

7.3.4.	Workflow Report Subscriptions.....	104
7.3.5.	Custom Reports	105
7.3.6.	Converging Workflow Paths	105
7.3.7.	Integrating Approval Workflow	106
7.3.8.	External Initiators.....	106
7.3.9.	Mobile Access	107
7.3.10.	Customizing Forms.....	107
7.4.	Workflow Examples	107
7.4.1.	Purchase Request	107
7.4.2.	Help Desk	108
7.4.3.	Product Release	109
7.4.4.	Credit Application (Advanced).....	111
8.	Business Logic.....	113
8.1.	Workflow	113
8.2.	Actions	113
8.2.1.	Action Parameters	114
8.2.2.	Creating An Action	115
8.2.3.	Example Scripts.....	118
8.3.	Calculated Fields	119
8.3.1.	Creating Calculations	119
8.3.2.	Recalculating.....	122
8.4.	Events	123
8.4.1.	Overview	123
8.4.2.	Adding Events	124
8.4.3.	Event Throttling	128
8.4.4.	Writing Event Handler Scripts.....	128
8.4.5.	Example Event Handlers	129
8.4.6.	Writing Layout Events.....	130
8.5.	Scheduled Jobs.....	131
8.5.1.	Overview	131
8.5.2.	Adding a Scheduled Job	132
8.6.	Notifications.....	133
8.6.1.	Overview	133
8.6.2.	Adding a Notification	133
8.6.3.	Using Notifications with DbfScript.....	135
9.	Creating Reports.....	137
9.1.	Overview	137
9.2.	Quick Reports.....	138
9.2.1.	Overview	138
9.2.2.	Quick Report Concepts	139
9.2.3.	Getting Started	141
9.2.4.	Report Summary	155

9.3.	Automatic Schedule (Tickler) Reports	156
9.3.1.	Overview	156
9.3.2.	Creating a Tickler Report	158
9.3.3.	Using the Tickler Report	160
9.3.4.	Anatomy of the Tickler Report	161
9.4.	Custom Reports	162
9.4.1.	Overview	162
9.4.2.	Creating a Custom Report: The Basics	164
9.4.3.	Word-Specific Functions	166
9.4.4.	Creating Charts	167
9.4.5.	Adding the Report to the System	169
9.4.6.	Configuring the Report's Data Query	172
9.4.7.	Example	173
9.5.	Report Subscriptions	175
9.5.1.	Overview	175
9.5.2.	Setting up a Report Subscription	175
9.5.3.	Managing Report Subscriptions	176
9.5.4.	Email Format	177
10.	Creating Custom Views	178
10.1.	Overview	178
10.2.	Custom Record View HTML	179
10.2.1.	Designing the HTML	179
10.2.2.	Enabling the Custom View	181
10.3.	Using Quick Reports to Create Dashboards	183
10.4.	Custom Field Viewers and Editors	183
10.5.	Skins	186
11.	Enterprise Features	187
11.1.	Overview	187
11.2.	Approval Workflow	188
11.2.1.	Overview	188
11.2.2.	Scenario	188
11.2.3.	Concepts	189
11.2.4.	Configuring Approval Workflow	190
11.2.5.	Reviewing Workflow Messages	193
11.2.6.	Approval Workflow Report	194
11.2.7.	E-Mail Integration	195
11.2.8.	Reviewing the Request	197
11.2.9.	Advanced Workflow Concepts	197
11.2.10.	Example: Expense Approval	198
11.2.11.	Example: Vacation Request	200
11.2.12.	Example: Content Management Workflow	201
11.3.	Tasks	202

11.3.1.	Overview	202
11.3.2.	Configuring Tasks	203
11.3.3.	Using the Tasks	203
11.4.	Live Updates	204
11.5.	Show Calendar	205
11.5.1.	Overview	205
11.5.2.	Configuring the Calendar	206
11.6.	Show Map	207
11.6.1.	Overview	207
11.6.2.	Concepts	208
11.6.3.	Configuring the Map	209
11.7.	Audit Trail.....	210
11.7.1.	Overview	210
11.7.2.	Configuring Audit Trail	210
11.8.	Last-Modified Dates.....	211
11.9.	Data Entry Reminders	212
11.9.1.	Overview	212
11.9.2.	Configuring Data Entry Reminders	212
11.9.3.	Email Structure	215
11.9.4.	Escalation	215
11.10.	Version Control	216
11.10.1.	Overview	216
11.10.2.	Concepts	216
11.10.3.	Configuring Version Control	217
11.10.4.	Using Version Control	217
11.11.	Wiki: Community Content	219
11.11.1.	Overview	219
11.11.2.	Configuring Wiki Support.....	220
11.11.3.	Using the Wiki.....	220
11.11.4.	Comments.....	222
11.12.	Ratable	222
11.12.1.	Overview	222
11.12.2.	Configuring Ratings.....	223
11.12.3.	Using Ratings.....	223
11.13.	Document Review and Routing	223
11.13.1.	Overview	223
11.13.2.	Configuring Document Routing	225
11.13.3.	Using Document Routing	225
11.13.4.	Example: Product Ideas Database	227
11.14.	Comments.....	228
11.14.1.	Overview	228
11.14.2.	Configuring Comments	229
11.14.3.	Using Comments.....	229

11.15. Forum (BBS)	229
11.15.1. Overview	229
11.15.2. Configuring Forum Support	230
11.15.3. Using the Forums	230
11.16. Supports RSS	232
11.16.1. Overview	232
11.16.2. Configuring RSS	232
11.16.3. Using RSS	232
11.17. Tagging	233
11.17.1. Overview	233
11.17.2. Configuring the Tagging Engine	234
11.17.3. Using the Tagging Engine	235
11.18. Quick Charts	236
11.18.1. Overview	236
11.19. Gantt Charts	240
11.19.1. Overview and Use	240
11.19.2. Configuring the Gantt Chart	240
11.20. User Positioning	242
11.20.1. Overview	242
11.20.2. Configuring User Positioning	242
11.20.3. Using the User-Positioning	242
11.21. Tickler Reports	243
11.22. Composite Lists	243
11.22.1. Overview	243
11.22.2. Concepts	244
11.22.3. Configuring the Composite List	245
11.23. Holiday Schedules	248
11.23.1. Overview	248
11.23.2. Configuring the Holiday Schedule	248
11.23.3. Holiday Schedules in Scripting	249
11.24. What-If Server	249
12. Help System	251
12.1. Overview	251
12.2. Adding Help	252
12.2.1. Entering Tooltips	252
12.2.2. Entering Help Pages	252
12.2.3. Linking Help Pages	254
12.3. Using the Help System	254
13. Security	255
13.1. Overview	255
13.2. Concepts	255

13.3.	The Administrator User	256
13.4.	Creating Users.....	256
13.5.	Creating User Groups.....	258
13.6.	Creating a Role.....	258
13.7.	Restricting Individual Record Access	261
13.8.	Assigning Roles to Users	262
13.9.	Disabling All User Editing	262
14.	Deployment	264
14.1.	Creating Multiple Servers	264
14.2.	Publishing.....	266
14.3.	Change Log and Notifications	266
14.4.	Feature Modules.....	267
14.4.1.	Overview	267
14.4.2.	Creating Feature Modules	268
14.4.3.	Allocating Features	268
14.4.4.	Selecting Feature Modules	269
14.5.	Default Database	269
14.5.1.	Overview	269
14.5.2.	Adding the Default Database.....	270
14.5.3.	Keeping Data Private	271
14.5.4.	Removing Sample Data.....	271
14.5.5.	Reset Sample Data	271
14.6.	Synchronizing Servers.....	272
14.6.1.	Overview	272
14.6.2.	Configuration	272
14.6.3.	Locking Records	273
14.6.4.	Technical Details	274
14.7.	Upgrading	274
14.8.	Beta Testing	276
14.9.	Using DbfServer Independently.....	277
14.10.	Using dbFramework with Microsoft IIS	277
14.10.1.	Installing IIS.....	277
14.10.2.	Configuring dbFramework	278
14.10.3.	Troubleshooting.....	279
14.11.	Internet Hosting.....	279
14.11.1.	Dedicated hosting.....	280
14.11.2.	Shared hosting	280
14.11.3.	Cloud and Virtualized Servers.....	280
14.11.4.	Configuration	281
14.11.5.	Licensing	282
15.	Searching	283

15.1.	Overview	283
15.2.	Background	283
15.2.1.	Categorizing Documents	283
15.2.2.	Discovering Documents	285
15.3.	Configuring Search in dbFramework	289
15.4.	Using Search	289
15.5.	Search Ranking.....	291
15.6.	Security	291
15.7.	Installing IFilter Plugins	292
15.8.	Query Syntax.....	292
15.8.1.	Terms	292
15.8.2.	Wildcard Searches	293
15.8.3.	Fuzzy Searches	293
15.8.4.	Proximity Searches	293
15.8.5.	Range Searches.....	294
15.8.6.	Boosting a Term	294
15.8.7.	Boolean Operators.....	294
15.8.8.	Grouping	296
15.8.9.	Special Characters.....	296
16.	Skinning.....	297
16.1.	Overview	297
16.2.	Platforms.....	297
16.3.	Understanding the Process.....	299
16.4.	Adding a New Skin	300
16.5.	Creating a Template.....	301
16.6.	The Data Structures	302
16.6.1.	The Master Template.....	302
16.6.2.	The Dropdown Template	305
16.6.3.	The Login Page Template.....	306
16.6.4.	The Error Page Template	306
16.7.	Replacing Skins in dbFramework Itself	307
17.	Localization	308
17.1.	Overview	308
17.2.	Adding a New Language	309
18.	Collaborative Design	312
18.1.	Overview	312
18.2.	User Stories.....	312
18.3.	Glossary	314
19.	Automated Testing	315

19.1.	Overview	315
19.2.	Concepts	315
19.2.1.	Basics	315
19.2.2.	Things to Keep In Mind	316
19.3.	Configuring the Testing System	318
19.4.	Testing Features.....	320
19.5.	Recording Tests.....	321
19.6.	Playing Back Tests.....	323
19.7.	When Tests Fail.....	324
19.8.	Testing Reports	325
19.8.1.	Overview	325
19.8.2.	Running a Report Test	325
19.9.	Having Tests Run Automatically	326
19.10.	Testing Performance.....	327
20.	DbfScript Reference	329
20.1.	Introduction	329
20.2.	Terminology.....	329
20.3.	Variables	330
20.4.	Program Control	330
20.5.	Errors and Notification	331
20.6.	Reports.....	332
20.7.	Database Access	333
20.8.	General Functions.....	336
20.9.	Path Functions	341
20.10.	Date/Time Functions	342
20.11.	File Functions.....	344
20.12.	Tokenizer Namespace.....	345
21.	Connectivity	347
21.1.	Overview	347
21.2.	Emailing Page Links.....	347
21.3.	Spreadsheet Import/Export.....	348
21.3.1.	Importing Spreadsheet Structures.....	348
21.3.2.	Exporting.....	348
21.3.3.	Importing	349
21.3.4.	Importing Hierarchical Data.....	349
21.4.	Excel Add-In	350
21.4.1.	Overview	350
21.4.2.	Installing DbfRtdServer	351
21.4.3.	Using the RTD Server	351

21.4.4.	Advanced Usage.....	353
21.5.	Using the .Net SDK.....	354
21.5.1.	Overview.....	354
21.5.2.	Prerequisites.....	354
21.5.3.	Setting up Your Project.....	354
21.5.4.	Concepts.....	356
21.5.5.	Creating a Service.....	358
21.5.6.	Paths.....	360
21.5.7.	Dropdown Lists.....	363
21.5.8.	Nodes.....	364
21.5.9.	Sequences.....	366
21.5.10.	Retrieving Data.....	368
21.5.11.	Updating Data.....	370
21.5.12.	Inserting Data.....	371
21.5.13.	Deleting Data.....	372
21.5.14.	Services.....	372
21.5.15.	Adding Member Functions.....	373
21.5.16.	Using Visual Basic.Net and other .Net Languages.....	373
21.5.17.	Debugging.....	374
21.6.	Using the XML API.....	374
21.6.1.	Overview.....	374
21.6.2.	Logging In.....	375
21.6.3.	Querying Data.....	376
21.6.4.	Inserting Records.....	377
21.6.5.	Updating Records.....	378
21.6.6.	Deleting a Record.....	379
21.6.7.	File Data.....	379
21.7.	Querying iCal Data.....	380
21.8.	Using the JSON API.....	380
21.8.1.	Logging In.....	381
21.8.2.	Querying Data.....	381
21.8.3.	Inserting Records.....	382
21.8.4.	Updating Records.....	383
21.8.5.	Deleting a Record.....	383
21.8.6.	File Data.....	384
22.	Maintenance.....	385
22.1.	Backing up Your Application.....	385
22.1.1.	Overview.....	385
22.1.2.	Backing up SQL Server.....	386
23.	Advanced Topics.....	387
23.1.	Admin Tools.....	387
23.1.1.	Exporting to XML.....	387
23.1.2.	Importing XML.....	387

23.1.3.	DbfScript Command Window	388
23.2.	System Log	388
23.3.	About DbfServer.exe.....	388
23.4.	In-Memory XML Database.....	389
23.5.	Database Concurrency and Deadlocks	390
23.6.	Database Schema Upgrading.....	391
23.7.	SQL Table Structures.....	392
23.7.1.	Overview	392
23.7.2.	Table Structures.....	393
23.7.3.	Special Fields.....	395
23.7.4.	Path Index Tables.....	396
23.7.5.	Schema Table	396
23.7.6.	Session Table	397
23.7.7.	File Data	397
23.8.	Overriding the Server Port Number.....	397
24.	References	398
24.1.	Culture ID Reference.....	398
24.2.	Icon Reference	402
24.3.	Contacting RiaForm Technology	403

1. Introduction

dbFramework is a revolutionary new way to create web-based software applications.

What makes dbFramework different to other databases is firstly its emphasis on workflow. You can define your business processes using the graphical workflow editor, and dbFramework will create the database model and the form design and even some reports for you – without any designing or programming at all.

You don't have to worry about designing the way the application looks: all of the application layout is constructed for you. Of course you can override and customize any part of it, but by having dbFramework generate the user interface for you, you save a significant amount of time letting you concentrate on what's really important in the application.

The second major difference is its use of a hierarchy to define the data that goes into the application. Most database systems require the design of tables and complex joins and programming, but dbFramework requires none of this: applications can be created quickly using a simple point-and-click interface that reflects how most people think about their data. Because of this, dbFramework better understands your data and can then add sophisticated functionality that would otherwise take months to develop, with just a simple click.

If you want to skip the introduction and jump right in, then go to section 3, on installing dbFramework. If you want more information on how precisely it's different to other systems, then enjoy the next few sections explaining the background to dbFramework, its concepts and capabilities.

1.1. Background

Getting computers to do what we want is a hard process. Most people rely on off-the-shelf-software that often does what *others* want, but only superficially what *we're* after. The net result is that we waste a lot of time trying to manipulate this software to work for us, and often end up performing the hard bits manually, even on paper, and then additional time trying to integrate that process with the software product.

Applications such as Microsoft Excel help tremendously, because they allow us to create simple software applications without the need for complex programming.

But spreadsheets are not really applications, and the model adopted by spreadsheets doesn't scale too well, and so as the spreadsheet grows, so does its complexity. As it becomes more complex, it becomes dramatically less manageable until, more often than not, we just give up and go back to working on paper.

Larger companies have the luxury of employing computer programmers to create custom software that is tailored precisely to their requirements. Yet even this is not a panacea: the development of the software proves to be expensive, the maintenance of the software even more expensive, and the quality never quite matches that of commercial software. Features that we come to expect in software, such as the clipboard, the undo button, help pages, security and so on – these are all very difficult, and very expensive, to implement in custom software. And so, more often than not, they're missing. The result is sub-par quality, a sub-par experience and frustrated users.

What's more, with no single system that is capable of satisfying all of these requirements, we instead end up with a multitude of software systems that each work differently, each require different training and each require significant (and constant) manual intervention to integrate.

This may paint quite a dismal picture of the software industry today, but it goes to show the great opportunity that remains: software has the potential to touch and transform every aspect of our business if we could just remove these hurdles by providing better tools.

This is exactly the position that dbFramework seeks to take. By providing an end-to-end tool that lets ordinary business users have a role in creating the software that they use, we effectively remove the obstacles that normally make software such a frustrating and inefficient solution. dbFramework provides all of the functionality that is needed to create easy to use, robust, flexible, secure and scalable business applications, with very little effort in next to no time.

1.2. What can I create using dbFramework?

There are many different development systems available on the market today, and each is focused on creating different kinds of applications.

dbFramework is a general-purpose development system. It can create a wide variety of web-applications. From simple web-sites to loan-administration systems, customer relationship management to document management systems.

dbFramework excels at creating workflow or other data-oriented web applications, and one might consider all web applications to be data-oriented.

dbFramework provides a lot of functionality out of the box. Features such as:

- User-management (login etc.)
- User-interface (data entry, form design, views, navigation, clipboard etc.)
- Data linking (relationships between records translate into links on the screen automatically)
- Database management (interfaces with the database transparently: you never have to perform any database administration)
- Concurrency and data-integrity (provides full transaction support)
- Deployment/upgrades (Takes care of installing and applying database modifications transparently whenever an upgrade is provided)
- Scalability (Built to support a multitude of concurrent users)
- And capabilities to quickly add a help system, multiple language support, Visual Studio integration and much, much more...

And these features are automatically enabled just by creating your application and clicking 'Publish'. Where an application created using dbFramework needs additional functionality, custom HTML views can be generated to allow more freedom over user-interface innovations.

Here are a few of the sample applications that will be shipped with dbFramework:

Customer Relationship Manager (CRM)	Add customers, maintain correspondence logs and schedule, map where customers and prospects reside, track calls, automatic issue elevation, search across entire database, get daily task lists.
Document Manager	Add any kind of document file, create unlimited folders, maintains version archive, roll-back to specific versions, lock a file while it is being edited, subscribe to get email notifications when documents are submitted, get a schedule report for expiring documents.
Project and Task Manager	Using the Scrum Methodology, lets you create multiple projects, each with their own backlog of tasks. Then lets you assign tasks to a specific team. Automatically assigns and schedules tasks based on individual availability and workload.
Invoice Manager	Create clients and invoices, broken down by invoice item. Automatic

	invoice numbering. Search across all invoices in an instant.
Money Tracker	Create multiple accounts, enter multiple transactions to keep track of bank balances. Create unlimited, hierarchical account groups to tally up totals across types of accounts (eg. personal / business / investment accounts). Tag transactions with categories.
Knowledge Base	Create rich-text knowledge-base entries on any subject you wish, and put these into a free-form hierarchical structure. Create a comprehensive database of tags and assign multiple tags to each knowledge base entry to quickly organize and locate information when you need it.
Bug List Manager	Maintains a database of product defects. Lets you quickly prioritize bugs and schedule them to be fixed. Create a daily report of pending and scheduled tasks.
LenderEx Loan Manager	Example line-of-business application demonstrating sophisticated loan-administration features for the loan-factoring business. Maintain multiple customers, multiple hierarchical loans and collateral, with support for factoring of accounts receivables. Centralizes all events into one tickler file showing up-coming and past-due payments emailed out each day.

You can get going in dbFramework very quickly by starting with one of these sample applications and modifying it to suit your needs.

2. Application Design

2.1. Concepts

Before delving into dbFramework it's a good idea to be familiar with the concepts and processes involved in creating applications using the system. While there are some similarities with programs such as Microsoft Excel, Microsoft Access or FileMaker, what makes dbFramework superior in its simplicity is that the approach it takes to creating applications is refreshingly, and fundamentally, different.

Unlike Microsoft Excel, dbFramework employs two different modes of use: a design mode and a run mode. The design process occurs using dbFramework, editing the data model, adding reports and so on. When the design is complete, the developer publishes the application. This tells dbFramework to put together all of the bits and create the application. When you access the application you will be in 'run mode', where you actually use the application and cannot make any changes to its design. This is important because it protects the application from unintended modification, a mistake that is easy to make with systems like Microsoft Excel.

Having two distinct modes of operations helps the user from being confused with features that are specific to designing or creating the application. Likewise, the features pertaining to using the application don't get in the way of the design process.

There are three main aspects to the design of any application in dbFramework: The data model, the business logic and the reports. These refer to the structure of information entered and displayed, the rules that determine how that information is processed, and analysis and presentation of that information in the form of printed reports, respectively.

Because dbFramework is an end-to-end, web-based development platform, each of these aspects are managed through a simple and intuitive web-interface that can be accessed from any Internet browser. It features a graphical workflow editor and point-and-click report generator that make application development efficient and enjoyable.

By adopting a consistent user interface across all the major features in dbFramework, once the user is familiar with a few of the basic concepts entailed in using the system they will be able to re-use that experience and apply it to new areas without the need for extensive training. And the consistency extends to

dbFramework itself because even dbFramework was developed using dbFramework!

Being familiar with these concepts is essential to getting the most out of dbFramework. This section will highlight these concepts to give you a head-start in developing your dbFramework application.

2.1.1. Workflow

Workflow, also called Business Process Modeling, refers to the ability to take business processes and automate or facilitate them using software.

dbFramework, incorporating the features of WorkflowFirst, provides second-to-none workflow features. With a graphical flowchart designer and the ability to build a data model as you are designing the workflow, you can create complete sophisticated applications simply using the workflow designer.

The workflow in dbFramework is defined as a hierarchy of stages. A workflow can only be in one stage at any one time, but multiple workflows can be active at the same time.

Any number of workflows can be added to any type in the data model. Buttons will appear automatically when a record of that type is visible, allowing you to start or progress the workflow associated with it.

As the workflow is designed, the system automatically creates the data model to match the information that will be collected. This is where the knowledge of the data model comes in useful, because after it has been created you can easily go in and tweak the fields, adding help text, sort keys and other features that are only available in the data model.

2.1.2. The Data Model

The data model refers to the *terminology* and *structures* involved in the particular problem you're solving or feature you're implementing.

While the workflow features of dbFramework will create much of the data model for you, it's worth understanding how data models are created in dbFramework so you can easily go back and add fields to forms and other structures, or modify existing fields.

Let's take a simple, familiar example to demonstrate the data model: a to-do list. In this example the terminology involves tasks, due dates, priorities and state. If we

were to put these into a structure we would need to define how these different categories relate to one another. In this case we would say that we have a list of tasks, and each task consist of a title, a due date, a priority and a current state. The current state would be that it's either a pending task, a task in-progress, a deferred task or a finished task.

Creating the data model is perhaps the hardest part of creating any application, which is why the workflow features are built to save you from having to design the data model yourself.

Luckily, dbFramework makes it extremely easy to enter the data model quickly and efficiently. If mistakes are made, or if the data model expands or needs to be reworked, it is a very easy process to move sections around, delete or add new sections to the data model.

What also makes dbFramework different to other systems where data models are designed is that data models in dbFramework are *hierarchical*. We should all be familiar with hierarchical relationships: families, corporate structures, or the more mundane folder structures we create on our computers all reflect how easily we can comprehend hierarchical relationships.

On a more practical level, we tend to naturally think of data structures in hierarchical terms: a team has a list of projects, a project has a list of tasks, a task contains fields. When we say 'this has a that', 'has a' relationship defines the *primary* (most important) relationship between the two items. There may be other ways of relating the data that is not strictly hierarchical (for example, a project may involve two or more teams), but focusing on the *primary* relationship, which is the hierarchical relationship, allows us to greatly simplify the design process.

There are various terms adopted in dbFramework when working with the data model, and it's important to become familiar with them before starting your application design:

- **Field:** A field refers to a category of simple data. 'Age' would be a field, 'Task Priority' would also be a field. These are categories that contain simple data, such as numbers or words.
- **Type:** A type refers to a collection of fields that make up a particular entity. It is like the 'template' or 'design' for all specific instances of some data. An example of a type would be a 'Project'. This is not a simple category, it consists of multiple attributes. A project would have a number of fields: a title, a due date, a project manager, and perhaps a budget, and so on.

- **Nested Type:** This is a mixture of a field and a type: it's a field that is itself a type. You can think of a nested type as a subfolder. If, following on from the previous example, one had a type called a Project, there may be a field within a Project called a 'Task List'. This is a component of a project, but it is not a simple category: it is not a number or a series of words. It is a type, and because it is defined *within* a type we call it a nested-type.

The 'Task List' nested-type would hold fields such as a title, priority, due date, assignment, status and so on. A 'Task List' may itself have additional nested-types (a task log, for example), and that goes to show how a hierarchical data model simplifies defining the relationships used in an application.

- **Data Type:** When defining a field, one must specify the *kind* of data that the field will hold. Will it hold a number? If so, would it allow fractions? Can it hold a yes/no value, known as a Boolean? Will it hold a name? A sentence? A paragraph of text? A file attachment? These are all different data types that can be chosen when defining fields.

2.1.3. Automation

Once the data model is complete, dbFramework will create the application automatically, creating default layouts, entry forms, navigation, actions – an entire functioning application. Of course any aspect of the application can be overridden and customized, but dbFramework gives you something that will let you start using your application immediately.

dbFramework also automatically upgrades the database that is used behind the scenes. Normally utilizing Microsoft SQL Server can involve a lot of complexity in keeping the database schema up to date, but dbFramework does all of this for you so you never have to write a line of SQL.

2.1.4. Business Logic

Creating applications in dbFramework is simple, and what makes it so simple is that entering the workflow or even just the data model alone is sufficient to create a fully-functional, professional application. No programming is required at all.

However, to add more complex intelligence to the application, you may need to define more advanced business logic. Business logic includes rules that control how

information flows from one aspect of the data model to another, what calculations occur on that data, and any validation of that data that needs to be performed.

Advanced business logic appears in a few different areas in dbFramework:

- Formulae, in calculated fields.
- Events, which are actions that run when changes are made to the database, or when data needs to be validated.
- Transactions, which usually take input from the user and perform specific processing on information in the database.
- Report logic, which is used to perform analysis or other processing on data before it is presented to the user in reports, where it can be printed.

Each of these aspects of advanced business logic are defined using commands that are altogether simpler, yet more powerful, than the formula language used in Microsoft Excel, for example. The programming language that is adopted, referred to as DbfScript, allows easy access to any part of the database for your application, using the same terminology that you choose when designing your data model. Very little other complexities are involved.

The same language is used across all features of dbFramework, so once you have learned the handful of commands you can combine them to create powerful and sophisticated rules.

Let's look at an example. In the previous section we discussed a Task management application. In this example, let's say we had a couple of fields in the task called 'Estimated Days' and 'Actual Days'. These would refer to the estimated number of days considered necessary to complete the task, and the actual number of days it has taken to complete that task.

Now let's say we wanted to add a third field called 'Days Late'. This would refer to the number of days over the estimate it has actually taken. The formula for this is simple: **Actual Days minus Estimated Days**. This would be an example of business logic that you would implement in your application. In DbfScript syntax, the formula would be:

```
#input/ActualDays - #input/EstimatedDays.
```

At first glance it may look a little more cryptic, but it should be easy enough to see

how that relates to the formula we specified, and it will become clearer as we delve further into the DbfScript language later on.

2.1.5. One-Click Features

dbFramework provides a lot of built-in functionality that reduces or eliminates the need for a lot of business logic to be written. These 'one-click features' can enable an enormous amount of functionality with a single click. And they're integrated right into the data model that is created by the user.

Some example of features that are provided by dbFramework:

- **User Signup:** Allows users to create their own accounts in the system.
- **Task Management:** Adds the fields related to working on the record as if it were a task, such as the due date, estimates and assignment.
- **Live Updates:** Changes that occur in the system will be reflected on the screen without the need to manually refresh the page.
- **Workflow:** Any modifications to the system can enter a workflow system and be sent to others for approval.
- **Tagging:** Records in the database can be 'tagged', and those tags navigated to see all related records in one place.
- **Reminders:** For records that need to be entered periodically, the ability can be enabled for users to configure how often that data needs to be entered, and by whom.
- **Report Subscription:** Specific reports can be enabled for users to be able to manually subscribe to a report, whereby that report is automatically generated and emailed to them on a periodic basis.
- **RSS:** Lets users with RSS (Really Simple Syndication) readers subscribe to a part of the database so they can be notified of updates.
- **Calendar Support:** Can show any list of records that contains date fields as a calendar view.
- **Mapping:** Can show any list of records that contains geographic data (an address, for example) on a map.
- **Charting:** Showing any data series as a line chart.

- **Localization:** Ability to download an automatically-generated language dictionary, that can then be translated by the user and then uploaded in the system, to quickly add another language to the application.
- **Audit Trail:** Ability to record all changes made to a record in a history list.
- **Modification Dates:** Adds dates to a record that keeps a record of when the data was created, when it was modified and by whom.
- **Version Control:** Enables the ability to control changes to a record. The record has to be 'checked-out' before it can be edited, whereby it is locked not allowing others to change the record until it is 'checked-in' by the editor. Once checked-in, a copy of the record is added to the version history, which can then be accessed at a later date.
- **Ratings:** Allows users to rate records in the database, voting anonymously to contribute towards the overall rating of the record.
- **Comments:** Provides an area to add unlimited comments to a record.
- **Forum:** Provides an area to create topics and manage discussions related to a specific record.
- **Schedule Reports:** Automatically creates reports that put any records with date information into a schedule of upcoming events.

More information on these features, how to enable them and what exactly they provide, is provided in later sections.

2.1.6. Reports

dbFramework makes it very easy to create structures to store your data, and it won't take long to create a comprehensive information store for your business or personal use. But what makes this information store even more useful is to be able to analyze that data, to extract new information, statistics or other facts, from that data – what is known as business intelligence. This can be presented in a format that can be printed, shown to others or archived away. Because these reports become hard-copies they are often generated in Adobe's PDF format, which is easy to print and also provides additional assurance that the report cannot be easily modified once it has been generated.

dbFramework provides three different ways to create these kind of reports from your data: An ad-hoc report, a custom report or a Microsoft Excel report.

An ad-hoc report can be generated quickly just by selecting the fields that you wish to incorporate into the report. With just a few clicks you can select fields, create multiple levels of detail, group those lists, add sub- and grand- totals, add calculations and pie, bar or line charts to create a sophisticated, professional looking report without any graphical design or programming.

For more unusual reports, a somewhat more advanced technique can be adopted whereby a report template is designed that will allow for absolute flexibility in how the report will present data. In conjunction with this, custom business logic can be implemented to retrieve and transform data, and templates can be designed either using Microsoft Word or an HTML editor. The result is then generated as a PDF file that can easily be printed and archived.

Finally, there are two quick ways to extract data from dbFramework on an ad-hoc basis: On every screen there is an 'Export to Excel' link that will immediately transfer data from the currently selected page to a Microsoft Excel spreadsheet. Additional analysis can then be performed from this spreadsheet, such as adding charts or performing pivot transformations. Secondly, using the administration tools from any page one can export all the data (from the selected page downwards in the database hierarchy) to XML format. This XML format can be imported into other, third-party applications for further analysis.

Once reports are created, whether ad-hoc reports or custom reports, one can easily enable user-subscription capabilities. This allows users to see that the report is available and subscribe to that report, so that it will be scheduled to be generated on a periodic basis and emailed to them automatically.

Reports can also, optionally, have input forms. These input forms will request additional parameters from the user that will be used to filter or otherwise change the selection or processing of the data that is used in the report, giving the user more control over the type of report they want generated.

2.1.7. Advanced Programming

While dbFramework is an end-to-end system, these ends are not closed. Rather dbFramework is a self-sufficient platform which can also serve as the foundation for the application. Industry standard interfaces are provided that allow access from programming environments such as .Net, and other Internet-enabled environments.

Through a secure session, these interfaces allow the ability to query any aspect of the database, to send transactions and even to run reports. They allow business

logic to be written in other languages, such as C#, Visual Basic, or JavaScript and then easily incorporated into formula, events or report data processing.

Special support is provided for easily integrating .Net modules into dbFramework applications. An SDK (Software Development Kit) can be automatically generated from a single-click and used within a C# project, for example, to access all the aspects of your dbFramework application. Once built, this module can then easily be imported into the dbFramework application and seamlessly integrated at various points in the development process.

Beyond this, any screen in your dbFramework application can be replaced with a custom HTML page, that can itself host a Flash page, or a Silverlight application. This enables huge potential for integrating new technologies to provide innovative visualizations of data or user interaction designs.

The ability to create custom views also applies to individual fields: custom views or custom editors can be specified for any field in the system. For example, dbFramework itself employs a custom editor for the color selector of the application. This uses some simple JavaScript to display a color-chart so the user can easily see the color they want to utilize in the application. Third-party components can also be integrated this way: both the CodeMirror code editor and charting components were integrated into dbFramework using the same mechanism.

By providing these powerful integration points, the possibilities with dbFramework are literally endless.

2.1.8. Deployment

dbFramework applications can be hosted either within the dbFramework Server, or in Microsoft IIS (Internet Information Server), and can use either SQL Compact, a simple in-memory XML file for data storage, or Microsoft SQL Server (from Express to Professional editions). This allows the application to scale to any number of users, and even to provide the functionality behind Internet sites serving hundreds of users.

3. Installation

3.1. Licensing

There is a base license for dbFramework that covers the designer and any applications running on that server machine. With that license any application can be accessed with one login or guest access.

Additional logins (a login refers to a username and password for accessing an application) must be purchased as a separate license, but this license can be shared between all applications running on that machine.

Purchasing additional licenses is a simple process and dbFramework will tell you if you need to, and will provide the necessary links to purchase and activate the license without ever leaving the web browser or restarting the server.

Please contact RiaForm for more information about licensing.

3.2. Installation Procedures

You can download and install dbFramework for free. You can download this from the following link:

<http://dbframework.com/>

The download file is relatively small, and installation should be a quick and easy process.

There are three editions of dbFramework: WorkflowFirst, dbFramework Express and dbFramework Professional. The WorkflowFirst and Express edition uses a database called SQL Compact which is suitable for running on desktops and takes up very little resources. The Professional edition uses Microsoft SQL Server, which usually runs on its own server machine and can support a large number of users.

3.2.1. dbFramework Express and WorkflowFirst

dbFramework Express and WorkflowFirst use the Microsoft SQL Compact Edition database. This is automatically installed during the installation procedure, so no user intervention is required.

For more information on maintaining the SQL Compact database used by each dbFramework application, please see the section on maintenance.

3.2.2. dbFramework Professional

dbFramework Professional requires a Microsoft SQL Server or SQL Server Express installation to store its data, but this is downloaded installed automatically during the setup process. Under normal circumstances, the dbFramework Professional Installation program will automatically detect whether SQL Server is installed, and if not it will download and install SQL Server itself.

Once dbFramework is installed and first accessed, it will automatically create the connection to the Microsoft SQL Server, create the database and all the necessary configuration required to use the system.

Note: If you are installing SQL Server manually, dbFramework requires that the default settings are adopted when SQL Server is installed: the instance name must be called SQLEXPRESS, and Windows authentication must be enabled. Unless you deliberately change the setup settings, or have a pre-existing installation of SQL Server, you shouldn't have to worry about this.

Note: Many computers, especially if they have been purchased after 2005, will have Microsoft SQL Server Express installed already. It's recommended that you try to install dbFramework first, and let it advise you to install SQL Server if it finds it to be missing.

If you choose to install Microsoft SQL Server Express manually, it can be downloaded for free from the following link:

<http://www.microsoft.com/express/database/>

3.2.3. PDF Reports

Also, to generate reports in PDF format, Microsoft Word 2007 must be installed on the same machine that the OfficeSuite Launcher or DbfServer is running. Also the Microsoft Save-As-PDF plugin must be installed. dbFramework uses Microsoft Word to convert the HTML or Word template into a PDF file programmatically.

If you do not wish to utilize this feature, you can switch off the PDF generation in the Server record, within the Application record. Once it's switched off and republished, all reports will be generated in HTML and viewed in the browser. Some functionality, such as pagination control, may not be available.

4. Accessing dbFramework

4.1. A Brief Tour of the OfficeSuite Launcher

Once RiaForm OfficeSuite is installed, you will need to run the OfficeSuite Launcher application. Once that is running you will be presented with a screen similar to this:

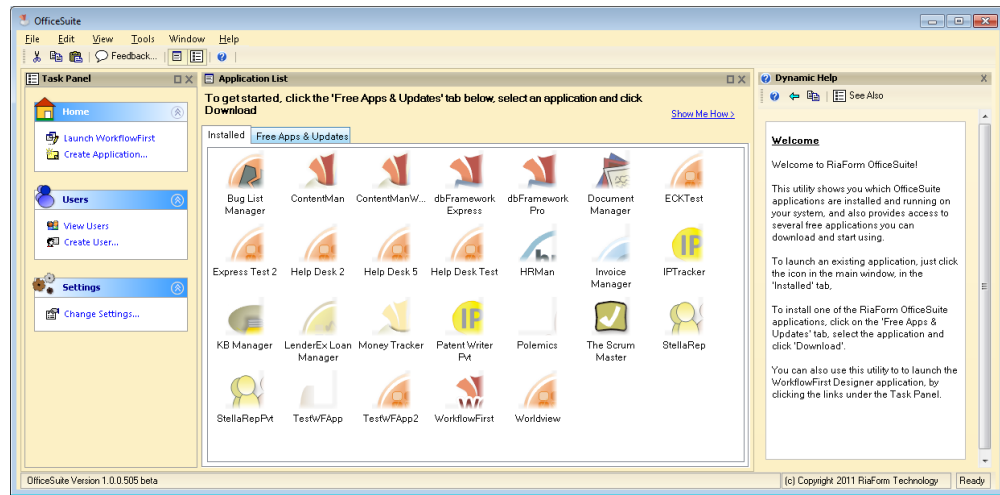


Figure 1: The application

This launcher application is effectively the server, and oversees the dbFramework and WorkflowFirst applications that are running on your machine, including the designer.

Web-browsers usually access web-servers that run on the Internet, but because dbFramework is for creating web applications on your own machine or network, OfficeSuite Launcher runs the web-server on your local machine, letting you access it from the browser as if you're accessing an Internet web-site.

Note: dbFramework and WorkflowFirst, along with all the applications it creates, can be accessed from all major Internet browsers.

This OfficeSuite Launcher application manages all of the applications that you have created, providing a portal for quickly accessing them.

After you have created your application, you will need to make sure that OfficeSuite Launcher is running in order to use the application. That's only the case if you're not using Microsoft IIS to host your application, but we'll discuss that in a later section.

You probably won't use the OfficeSuite Launcher application directly, and may end up just minimizing it once you're comfortable bringing up the browser and finding your way around dbFramework. But to get started you can click the 'Launch dbFramework' link on the left side (the Task Panel). This will bring up a browser pointing to the following URL:

```
http://localhost:86/
```

Assuming everything is installed correctly, that will take you to the main screen in dbFramework. The web server 'localhost' always refers to the machine on which you're working, and by default dbFramework runs on port 85.

Advanced: For advanced users, you can also right click an application icon in the monitor window and select 'Debug...'. This will launch a debugger and attach to the specified application's DbfServer instance. This may be useful if you have a plugin installed in your application as you can set breakpoints and trace through your code.

5. The Common Web Interface

Now that you're accessing dbFramework, you'll be presented with the main screen:

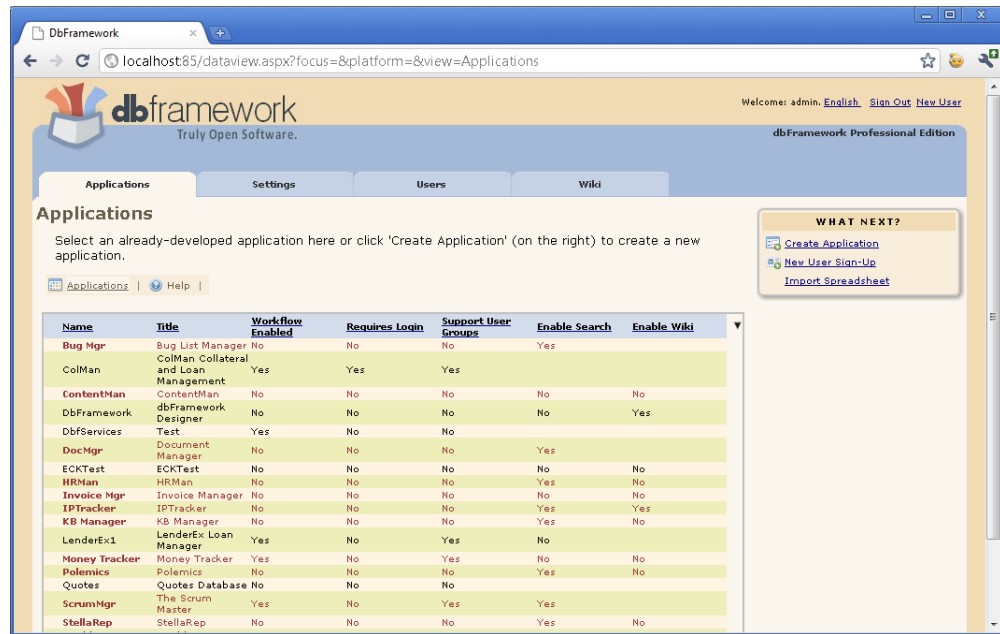


Figure 2: Main screen in dbFramework

The main screen shows you a list of all of the applications you have created on your system. You will also see the sample applications and if you're using dbFramework on a network you may also see applications created by other users.

Before we get started with creating a new application, it's worth taking a minute to get familiar with the different aspects of the web interface.

This same interface is adopted across all dbFramework applications, so getting familiar with the concepts, where they're located and how they're used, will be critical in visualizing your application, and will help tremendously during the design process.

There are several different sections we'll discuss.

5.1. Page URL



Each page in dbFramework has its own address, known as a URL. This is useful to know because it means you can create bookmarks ('favorites') of specific pages in any dbFramework application, create shortcuts on your desktop, and even email those links to people, so that you can return to that exact screen at any time with a single click.

5.2. Logo and User Options

The first section, in the top-left corner of the screen, is the application logo. The logo overlaps the navigation bar a little, for aesthetic reasons and to make it fit in better with the rest of the page.

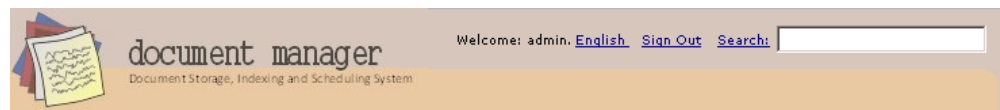


Figure 3: Logo and User Options

The next section, in the top-right corner of the screen as shown in Figure 3, is the user options. This where it tells you the user name that you are logged-in with, provides a link where you can log-out, in order to switch users. It also shows you the language options that are available, if additional languages are installed. If search is enabled you will also see the search box here.

Underneath that area is the byline or description for the application.

5.3. Application Tabs

The next section is, below the application logo, are the application tabs, as shown in Figure 4. These tabs let you access different, top-level aspects of the application. In the case of dbFramework these tabs let you access the Applications, the Users and the Settings. Clicking a tab will take you to that area. In your own applications the tabs will depend on the top-level, visible fields in the model.

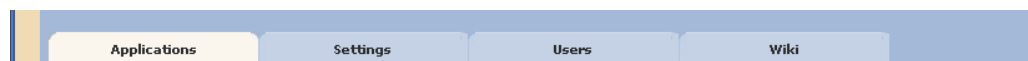


Figure 4: Application Tabs

Tip: You can always click the tab from anywhere in the application to go back to the top-level of that area of the database. For example, clicking the Applications tab from anywhere in dbFramework will take you to the list of applications in the system.

5.4. Navigation Bar



Figure 5: Navigation Bar

Underneath the selected tab are a series of links which show you your position within the database, known as the Navigation Bar. If you are at the top level of the tab, the navigation bar will not be shown.

As you navigate through the database, this navigation bar changes to reflect your new position. The bar should be read from left to right, with the list name at the top and the record title at the bottom. Each vertical entry is followed by the drill-down record to the right.

The items displayed in the navigation bar are themselves links that will take you to that part of the database, which makes for a convenient means to go back to where you were with a single click.

5.5. Record List

The next section, below the tabs, is the screen for the currently selected tab. This shows either a list of records, or the selected record. A screen showing a list of records is shown in Figure 6.

Nested Fields [Create New](#)

Quick Entry (Alt+Q)

Name: Text (Few Words)

Name	Title	Group Title	Data Type	Existing Type	Dropdown List	Link
Name			Text (Few Words)			
FacilityType	Facility Type		Text (Few Words)		FacilityTypes	
Borrower			Text (Few Words)			
Lender			Text (Few Words)			
CashLoan	Cash Loan		Boolean			
Indicates if this loan is an actual loan involving cash transactions, or is simply a loan from another borrower that is being tracked.						
Country			Text (Few Words)			
StartDate	Start Date		Date/Time (General)			
MaturityDate	Maturity Date		Date/Time (General)			
PaidOffDate	Paid-Off Date		Date/Time (General)			
InsuredAmount	Insured Amount		Number (Money)			
The total insurance coverage for this facility.						
CurrentLimit	Current Limit		Number (Money)			

Figure 6: Record list

The first line is the name of the list, in this case the 'Nested Fields' list. To the far right of that title is a link 'Create New' that will take you to an empty form that, once filled out, will be used to create a new record in the list.

Note: Create New may not always be available. You can disable that link for lists that you don't want the user to change, for example.

Underneath that is the 'Quick Entry' area. This may or may not be available depending on the list you're viewing. It provides a quick way to enter a new record by just typing the required fields and clicking 'OK'.

The next line is the list of column titles. Any column title can be clicked to resort the list by that column. The first click will sort by that column in ascending order. Clicking the field again will switch to descending order. Clicking the column a third time will group by that field and show you only the unique values, clicking a group will show only the records that have that value – providing a quick way to filter the list. Click again to remove the sort or grouping and revert to the default sort order for that list.

Note: When you sort columns that are dropdown lists, the order will be based on the order of the values in the dropdown.

Underneath that is the list of records themselves, each showing values under the corresponding column header. Each record is shown with alternating, banded colors. Hovering the mouse over a record will highlight it. Click a record will show all of the record details. Also, some field values are links themselves that will take you to the corresponding record in the system.

As will be seen later, when there are too many items to show on one screen, links will be provided at the bottom for jumping to the next page. Each page shows 20 items.

/2010	7/16/2010	3	20,000.40
/2010	6/27/2010	22	23,100
/2010	5/19/2010	61	2,224.92
/2010	5/22/2010	58	1,725.3
/2010	5/30/2010	50	6,565.49
2010	6/2/2010	47	3,424.26
2010	6/2/2010	47	222.02
2010	6/3/2010	46	7,571.57

[First Page](#)
[Previous Page](#)
[Next Page](#)

Figure 7: Showing the links to jump between pages

Tip: Each of the links displayed on the screen take you to a different URL. This means that every aspect of both dbFramework and your own applications has its own web address. This can be very useful when sending links to others, or creating bookmarks/favorites to specific screens.

For some lists you will see a '+' expansion icon for each row on the far-left of the list. Clicking this will be the same as opening the record and clicking one of the 'GoTo' items. Normally the 'GoTo' item that is of the same type as the list will be the one used by the '+' icon. An example of a list with expansion icons is shown below:

<u>Name</u>	<u>Title</u>	<u>Group Title</u>	<u>Data Type</u> ^	<u>Existing Type</u>
PrincipalSchedule	Principal Schedule		Nested Record	
InterestPayment	Interest Payment Schedule		Nested Record	
PrincipalParticipa	Principal Participation		Nested Record	
SubFacility			Nested Record	Facility
Fees			Nested Record	
Ledger			Nested Record	
TermChanges	Term Changes		Nested Record	
Tasks			Nested Record	
InterestParticipa	Interest Participations		Nested Record	

In the top, right-hand corner of the list, there is a triangle pointing downwards. Clicking this will display a context menu offering some clipboard options, as shown in Figure 8.

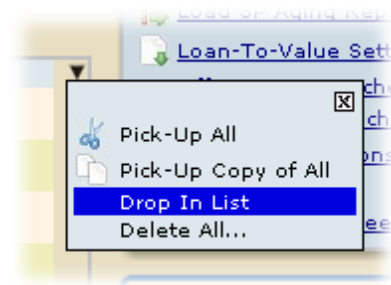
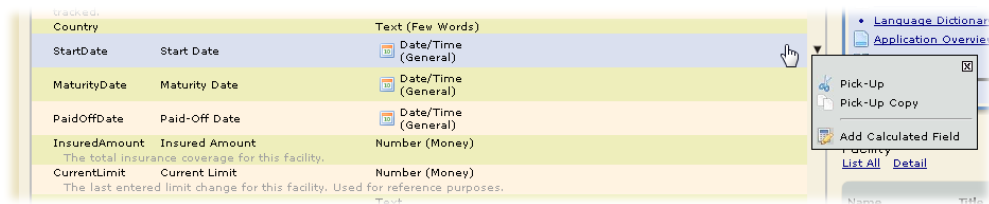


Figure 8: Context menu

This same menu is also available through a dropdown button at the far right of each row:



This menu offers the same options but without first having to view the detail of the record.

These are options are explained below:

- **Pick-Up:** Similar to the 'Cut' option you find in desktop applications, this prepares the record to be moved from one place to another.
- **Pick-Up Copy:** Similar to the 'Copy' option you find in desktop applications, this prepares the record to be copied to somewhere else in the database.
- **Drop On This:** Similar to 'drag and drop' functions in desktop applications, this allows one record to be dropped onto another record. For example you may wish your database to perform a specific operation when you drop a product onto a customer (create a proforma invoice, for example).
- **Drop Above This:** Similar to 'drag and drop' functions in desktop applications, this pastes the record into the list at the location specified.
- **Drop Below This:** Similar to 'drag and drop' functions in desktop applications, this pastes the record into the list at the location specified.

When you're viewing a list of records, different options are shown that act on all of the records on the screen:

- **Pick-Up All.** This option puts all items in the list (across all pages) into the clipboard. This is similar to selecting all items, and selecting 'Cut' in traditional desktop applications.
- **Pick-Up Copy of All.** This option puts *a copy of* all items in the list (across all pages) into the clipboard. This is similar to selecting all items, and selecting 'Copy' in traditional desktop applications.
- **Drop In List.** This option takes the records in the clipboard and inserts them into the list.
- **Delete All....** This option deletes all of the records in the current list. Please use this option with extreme caution!

5.6. Record Detail

Clicking on any item in the record list will display a record detail screen. An example of a record detail screen is shown in Figure 9.

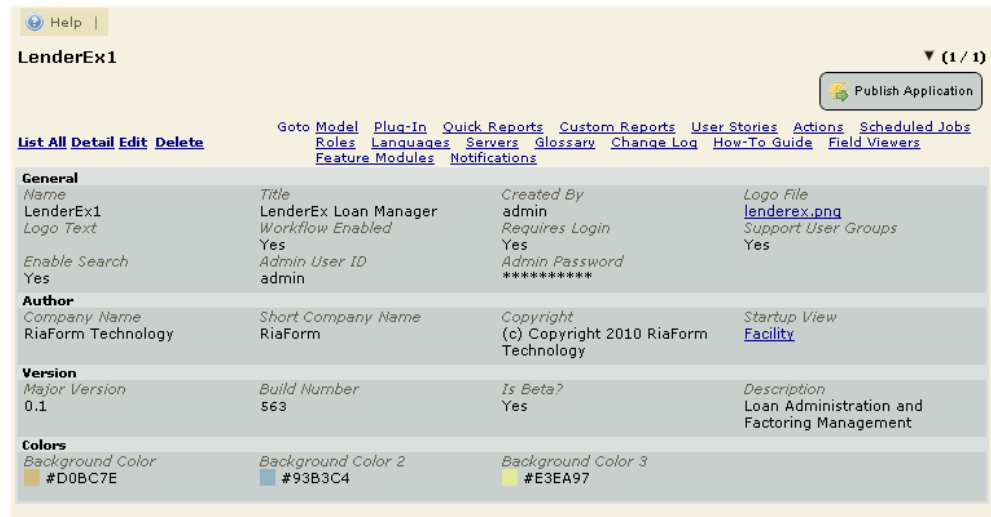


Figure 9: Record detail view

There are several sections in the record detail view that deserve some explanation.

The very top of the section is a list of tabs that let you switch between different screens or views of the record. In this example, the 'Help' link is provided that will display the help page for that record.

Underneath the screen tabs there is the *title* of the record. For every record, or rather every type, you can designate a field to be the title field. The title of the record is a short bit of text that best describes the record. For example, the title field for a User record would probably be the user name. The title field for a payment record might be the date it was entered. The title of the record is also used to identify the record in dropdown lists.

At the far right of the title is a dropdown menu button that shows clipboard options for that record. Many of these options were discussed in the previous section. Beside this is a record index that shows you the ordinal index of the record in the list, and also the total number of records in the list.

The next line down shows any optional action buttons that can act on the current record. These are known as 'primary actions'. Actions will be discussed in more detail in a later section.

Underneath that, on the far left, are four important links:

- **List All:** This option goes back to the list holding this record.
- **Detail:** Shows additional details for this record, if available.
- **Edit:** Goes to the Edit mode for this record, allowing you to change the values that you see in this view.
- **Delete:** After confirmation, deletes the record.

You will use these options to modify records anywhere in the system.

Note: These options may not be displayed if the logged-in user does not have permission to perform those actions.

To the right of those links, there is a section of links which is prefixed with the label 'Goto'. These 'Goto' links allow the user to drill-down to either more detail or further-related records and record lists.

In the main part of the screen are the fields names and values, underneath. These are separated into groups, and each group section has a title that is highlighted with a shaded background.

To learn more about a particular field, the user can hover their mouse cursor over a field title and often a more detailed description, known as a Tool-Tip, will appear. Moving the cursor away will dismiss this window. An example tool-tip is shown below.



Figure 10: Tool-Tip Window

Some field values will appear in blue and underlined. These are often links to other parts of the database. In dbFramework these are known as 'foreign links'. You will learn more about foreign links later in this user guide. The value may also be a link to a file that is stored in the system, or it may be a URL link.

Some field values may have custom views. That means that the value isn't displayed as plain text, but might have a picture or some other kind of rendition. These can appear in any way as determined by the custom view configuration. An example of a custom field value is displayed below.



Figure 11: Custom field views, showing the selected color

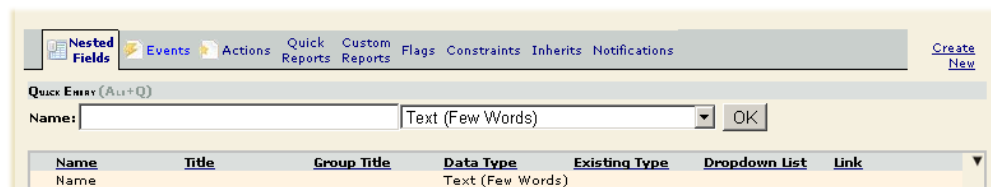
Tip: Because this is a web page, any fields can easily be copied to the clipboard so they can be pasted into documents or spreadsheets.

5.7. Auto-Expands

Beneath a record detail view, you will see a number of tabs that show the content of the 'GoTo' items for a record on the same screen as the record. You can switch tabs to display each 'GoTo' item. These are referred to as 'Auto-Expands', because they are automatically expanded drilldown options from the record.

For each of these auto-expands you can also create a new record for that drilldown by clicking the smaller 'Ceate New' link to the far right, or there may also be a Quick Entry box below the tab. Once you click these, or try to sort the columns, it'll show the record list full-screen.

An example of the auto-expands is shown below:



5.8. Edit Mode

By clicking the 'Edit' link on a record view, you will be taken to a form with editable fields allowing you to change their values. In a similar fashion, clicking the 'Create New' link will show an empty form (except for default values) where initial values can be entered and the record submitted for insertion into the list.

An example of a record in edit mode is shown below.

Figure 12: A record in Edit Mode

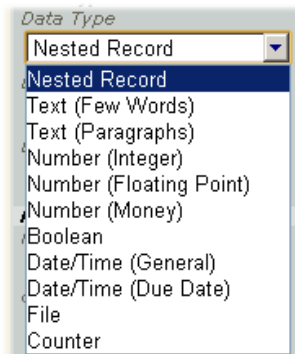
Tip: Should you decide to abort editing, you can easily click any other link on the page to abandon the changes you made and go back to navigating the database.

There are a few different types of fields you'll encounter when editing a record that should be explained.

- **Simple Edit Box:** This can be a box that will let you enter either text or a numeric value.

- **Record Selector:** The gray area of the box is not editable. By clicking the ellipses (...) button, a dropdown list of records will be displayed. The user can then select one of those records to set the value of the field. Once selected, the field will be displayed as a link that, when clicked, will take you to that area of the database. You can also drilldown through types by clicking the '+' icon next to the name. To make the value empty, the user can select '(reset)'. To cancel the selection they can just click the X close button in the top, right corner.

- **Dropdown List:** The dropdown list is not directly editable in that you cannot type new text into the white area. But you can click the dropdown arrow to select a new value from a predefined list of available options.



- **File Attachments:** The file attachment field lets you upload a file to that record. A record may have multiple file attachments configured.

Clicking the 'Browse' button will let you select a file to upload. When you click OK on the record editor the file(s) will be uploaded to the system.

- **Free-Form Text:** The free-form text box allows multiple lines (paragraphs) of text to be entered. When more lines of text are entered than can be displayed, the scroll bar can be used to navigate through the entire text.

When you are done filling in the field values you can click the 'Update' or 'Create' button at the bottom of the form.

If the edit was successful, you will be taken back to the record view. If there was a problem committing the changes, an error will be displayed at the top of the record view. You will then be given a chance to correct the error and submit the form once more.

Sometimes these errors messages are actually warnings which can be accepted or rejected. These will appear the same way as error messages but will be prefixed with a checkbox (a box that can be checked). Checking the box and clicking Update/Create again will accept the warning and submit the form regardless. Alternatively the Back button can be clicked to abandon the changes.

Types

The field 'Name' must be entered

Types

General

Name

Title

Figure 13: An error message.

Sometimes, when creating new records, you will see links within the form with some text such as 'Click to add X', where 'X' is the name of a Go To item. An example of this is shown below:

Custom Reports

Title Icon

Report File No file chosen

[Click to add Default Filters](#)

Input Type File Extension (Override)

[Click to add a String Table record](#)

Editable Input Data Transform (Plugin)

Clicking this link will expand the link to show a nested, empty form relating to the Go To item that was mentioned in the link. An example of an expanded nested form is shown below:

Custom Reports

Title Icon

Report File No file chosen

Default Filters

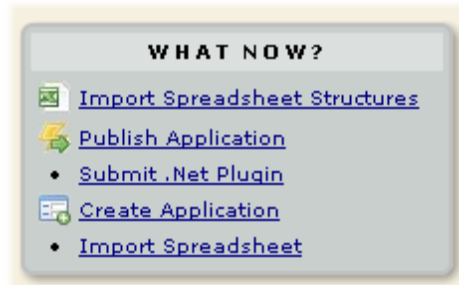
Field Operator Value Special Value

[Click to add Default Filters](#)

Input Type File Extension (Override)

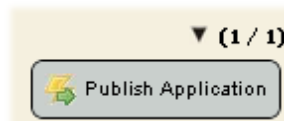
You can delete that nested form by clicking the 'X' in the corner of the form.

5.9. The Actions List



To the right of the screen, near the top, is a list of entries under the heading 'What Now?'. These are actions that pertain to the selected record, and also to any parent records (i.e. Records you went through to get to this record).

The primary actions will also be displayed as buttons at the top of the record view.



Clicking an action link will open a form, or a confirmation window. The form may ask for additional parameters required to run the action. Clicking OK will then invoke the action, and either display an error message or return to the previous screen. An example of an entry form for an action is shown in Figure 14.

A screenshot of the 'Publish Application' form. The title 'Publish Application' is at the top. Below it is a 'Help' link. The main section is titled 'Publishing Options'. It contains two input fields: 'Release Date' with the value '11/15/2010 11:08:36 AM' and 'Target Server'. Below these is a 'Release Notes' section with a rich text editor toolbar and a text area. At the bottom right is a 'Create' button.

Figure 14: Action entry form.

5.10. Reports List



Beneath the actions list is the report list under the heading 'See More...'. The list of reports also pertains to the selected record and its parent records. The report list is also split up into two main sections: Relevant Reports, which refers to reports pertaining to the selection, and Other Reports, which are global reports that run across all data in the system without concern for the current selection.

5.11. Parent Records



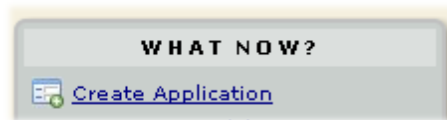
Below the report list is a summary of parent records, showing the first couple of columns for each line. Above each parent record is a link to show the detail or a link for that record. This information is useful for referencing parent information without having to navigate backwards.

6. Creating An Application Data Model

6.1. Application Wizard

dbFramework allows for an unlimited number of applications to be created. Each application that is created is displayed as an entry on the main page of dbFramework.

Creating a new application is a simple process. To begin with, click the 'Create Application' action link, which will be available from anywhere in dbFramework.



This should show the following entry form:

A screenshot of a web form titled "New Application". The form has two input fields: "Name" and "Description". The "Name" field is a single-line text box, and the "Description" field is a larger, multi-line text box. To the right of the "Description" field is a "Create" button.

The name of the application should be short, like a person's name. The longer description can be entered into the 'Description' field, but even this should be limited to just a few words so it can be properly displayed on the screen.

When you are done entering these fields, click 'Create' to let dbFramework set up the application.

You should then be returned to the main screen. Your application has been created. Right now it's pretty empty, but you can still take a peek to see what it will look like.

Tip: It's highly recommended that you back up your application configuration as frequently as possible. See section 22.1 for information on how to do this.

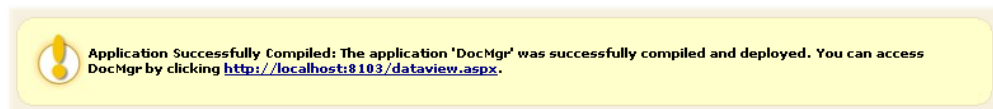
6.2. Publishing the Application

Before the resultant application can be used, it must be published. This effectively 'creates' the application output that can be run by the users. You can do this at any point.

So click the 'Publish Now' action. This is displayed as a button on the application record, or in the action list from anywhere in dbFramework.



Once you click that button you can click confirm. When it has finished publishing the application, it will display an information bar similar to this:

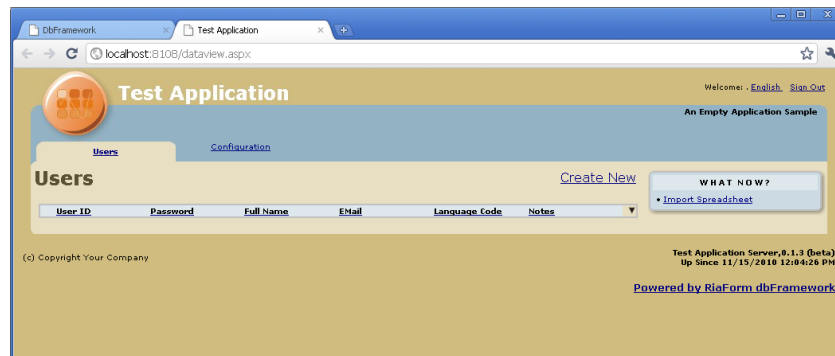


Depending on the complexity of the application, this should only take a few seconds to complete.

Once published, to see the application click the given link.

6.3. Application Configuration

The application screen should look pretty familiar: it will use the common web-interface that was described in a previous section. It will even show a default-logo and color theme for the application. Don't worry: all of these can be changed at any time, the system just selects defaults for you to get you started.



Only two tabs are displayed: The User list and the Configuration tab. You'll always see these two tabs in your application, but there's plenty of room to add more tabs for your more important application data.

Switch back to the dbFramework window so that we can review the application configuration.

Click the Applications tab to go to the list of applications in the system. You should see your newly created application. Click that to show the record view. Then click the 'Edit' link to edit your application configuration.

Note: Any changes made here or anywhere in the application's data model won't be reflected in the run mode of the application until it is published again. This is useful because you don't want your users to see half-finished features!

There are a few options that can be edited here. We'll review some of them here:

- **Name and Title.** These are the name and title that were entered in the 'Create Application' form. They can be changed at any time.
- **Logo File:** A simple logo, consisting of the application name, was created for you automatically when the application was created. This can easily be replaced with a new logo by uploading a new file here. This file should be a PNG file, approximately 300 pixels wide by 200 pixels high and preferably should have an transparent (alpha-channel) background. It will be positioned so that half of the logo overlaps with the top of the tab bar, allowing for some appealing aesthetic blending effects.
- **Logo Text:** If the logo file is not specified, the logo text defines the text that will be displayed in the automatically generated logo. If you specify a logo file, then the logo text field should be set to empty.
- **Requires Login.** If this is set to Yes then the user will be prompted for login information when they first access the application. The user will only be able to access the application if they enter a valid username and password.

- **Enable Search:** Enables searching, just like adding a 'Google' search to your entire web-application. We'll discuss this in a later section.
- **Company Name, Short Company Name and Copyright.** These are self-explanatory, except that the short company name is typically a one-word name used to refer to the company. For example the short name for International Business Machines would be IBM.
- **StartUp View.** This lets you indicate which of the tabs should be selected when the application is first launched.
- **Major Version:** Indicates the major version of the software. This can be increased when major features are added to the application.
- **Is Beta:** Indicates if the software version is still in the development or testing phase.
- **Colors section.** There are three base-colors used in any dbFramework application, and these can be selected here using the ellipses buttons. The first color is primarily used for the background, the second color for the navigation bar, and the last color for the actions list. But derivations of all three colors are used in various places throughout the application.
- **Admin User ID and Password.** This login information is used to indicate the administration access for the application. Before any users are created in the system, this administration user ID should be used to access the system. This user login will always be available, even if all of the users are deleted.
- **Workflow Enabled.** When this is set to 'Yes' various structures will be provided to enable the workflow capabilities of dbFramework. Workflow is discussed in more detail in a later section.
- **Support User Groups.** When set to 'Yes' will create an area to define user groups. If this is not set then only individual users can be configured in the application.

There are also several Go To items under an application that can be further explored. The most important of all of these is the link to the Model. Under this we store the entire data model for the application.

By clicking Model, and then clicking Types, you can start to edit the data model for your application. This will be discussed in more detail in the next section.

6.4. Understanding Data Modeling

The process of ‘modeling’, or coming up with the data model, is central to the developing of a dbFramework application. Therefore it’s worth spending some time getting familiar with data models as they are used and defined in dbFramework.

For those unfamiliar with data modeling, a data model defines the structures in which the information in the database will be held. In a spreadsheet, for example, the data model consists of the column, sheet and range names. Those are quite distinct from the data itself, which is usually held in rows underneath the column header. Broadly speaking the data model consists of the categories into which each record of the data is separated. The structure or class of a record is known as a ‘type’. An example of a type is a customer, or an order. All the types related to a specific domain are collectively called the data model.

Creating an application in dbFramework involves designing a type of data model called a ‘hierarchical data model’. A hierarchical data model is unique in that the types within the model follow a parent-child relationship.

To understand this better it's a good idea to look at a concrete example. A spreadsheet designed to hold company information might have one sheet with columns at the top such as 'Company Name', 'Address'. In another table, or another sheet, there may be another list of contacts at that company, with columns such as ‘Name’ and ‘Position’. Each contact may have a column called 'Company Name' also, so that we know which contact belongs to which customer. That is a classic example of a relational model.

An example of this relational structure can be seen below:

Sheet 1 : Companies

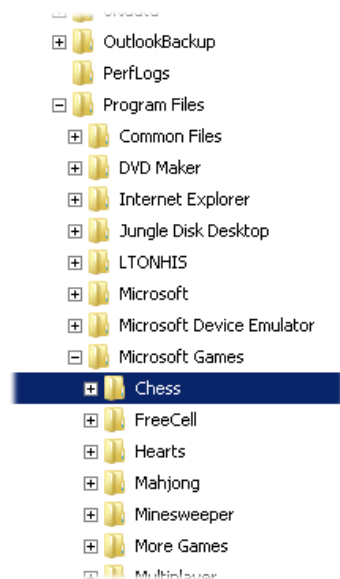
Company Name	Address
ACME Corp	51 Middle Street, Some Town, NY
Microsoft Corporation	1026 Helen Court, Some Town, NY
Google	11 Straw Road, Some Town, NY
Sun Microsystems	89 Gardens Road, Some Town, NY

Sheet 2 : Contacts

Contact Name	Position	Company Name
Tom Crinsley	Product Manager	ACME Corp
Mike Smith	Vice President Marketing	ACMR Corp
Bob Crown	11 Straw Road, Some Town, NY	ACME Corp
Sarah Kent	89 Gardens Road, Some Town, NY	Google

In a hierarchical data model there are not two, separate tables that can then be related. Instead there is one top-level table, the Customers, and that has a *child* table called Contact. Each Customer record can then contain one or more Contact records. The relationship is explicit and not implicit. It is that container/ownership structure that allows the model to be hierarchical.

You may be familiar with hierarchical structures in your experience of using folders and files, using Microsoft Windows' explorer for example.



The benefit of a hierarchical data model is that the relationship between different tables (or, as we call them, 'types') is very clear. It is not a matter of seeing multiple

tables of data and then wondering how to relate that information. Instead all of the information and its relationships are immediately obvious and accessible.

An example of a hierarchical data mode is shown below:

Company Name		Address
ACME Corp		51 Middle Street, Some Town, NY
	Contact Name	Position
	Tom Crinsley	Product Manager
	Mike Smith	Vice President Marketing
	Bob Crown	11 Straw Road, Some Town, NY
Microsoft Corporation		1026 Helen Court, Some Town, NY
Google		11 Straw Road, Some Town, NY
	Contact Name	Position
	Sarah Kent	89 Gardens Road, Some Town, NY
Sun Microsystems		89 Gardens Road, Some Town, NY

In this example you can see that underneath the company ACME Corp we have the list of Contacts related to that company. We no longer need to include the Company Name in that list because the company is implied from its parent. In a hierarchical database, if we were to retrieve the ACME Corp record we would automatically have access to the contacts belonging to that company.

Designing a hierarchical data model involves making decisions upfront about the type of relationships between types. The key to deciding whether a type should be a child of another type is a matter of ownership: If a record in the parent type is deleted, would you expect the records of the child type to be deleted also? If the answer is yes then you are probably looking at a parent-child relationship.

There may be a time when the relationship between two types isn't that of containment or ownership. If the relationship is rather a matter of reference, and you would *not* expect the parent record to delete the record it is referring to, then you should consider it a link.

The difference between ownership or reference (link) can sometimes be difficult to decide upon. To look at a more concrete example consider our Customer Information File example that stores information about companies. What if we wanted to record a list of countries. In each country we want to record information on their GDP, population, import/export restrictions and things of that nature. Now each customer belongs to a specific country (for simplicity we'll assume that a customer only has one country). If we put the industry definition underneath the

customer type as a child type, then when we delete the customer we will also delete the country. That might be fine if we were not sharing the information, but each country may refer to several customers. So we wouldn't want to delete the country because it would also delete information that is used by other customers. Because this relationship is not of ownership, but rather a shared reference, we would define this as a **link**. Links are described in more detail in a later section.

The process is not always clear cut. There may be times when the choice is more ambiguous. In the above example, should the Country be the top level type, and the Customer type be underneath that? Another example, let's say we add 'letters' to our customer information database. In there we record any correspondence from or to the customer. On the one hand we might say that each letter is owned by the customer. But then what if we send a letter to multiple customers? Should we have a copy of it under each customer? Or would it have been better to have the 'Letters' type as a separate, independent table that itself can link to one or more customers?

Fairly often the need will arise to choose a *primary* relationship that finds a compromise based on the most common scenario. This may make exceptions to the norm harder to deal with, but overall the focus should be on the most common scenarios and making them the easiest to handle. This is why a hierarchical data model demands an understanding of typical business operations, it demands that developers talk to the users to understand what their most common scenarios actually are, so they can be prepared when coming to make these design decisions.

The final point to make about data modeling is naming. Coming up with sensible names for types and field names is critical in creating data models. These names should describe exactly what is intended to be expressed, without being too generic or too specific. It may often be a good idea to come up with a glossary of terms that all the users can generally agree upon, and use this glossary as your reference point. Names that are too ambiguous lead to confusion.

Tip: dbFramework provides a section to maintain a glossary of terms used throughout an application. In your application record, drilldown to the Glossary section and there you'll easily be able to add terms and definitions that everyone can see and share.

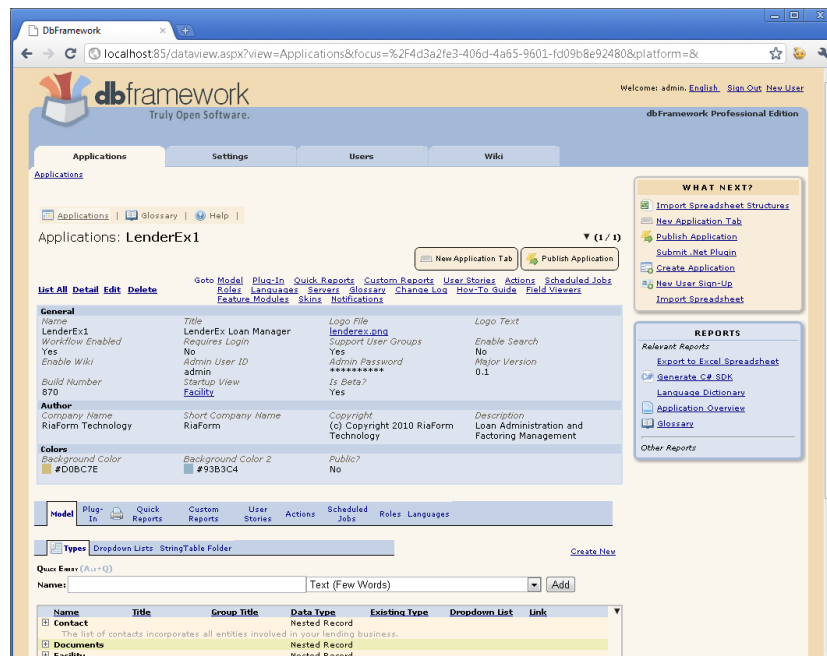
So it should be clear now that understanding how to structure a data model is the key to understanding business requirements and being able to translate those into a format that a computer system can utilize to create a software application.

6.5. Creating and Editing Type Definitions

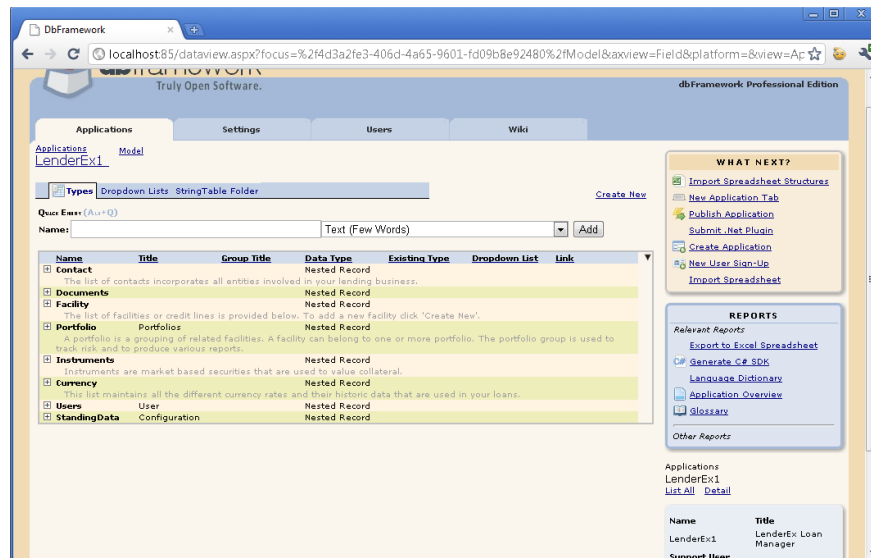
Because it's so easy in dbFramework to add types and field definitions in those types, to move them around or delete them, you can easily use dbFramework as a kind of scratch pad to figure out your design. You can even create a whole application design and delete it again without leaving any trace, in seconds.

The first step is to get to the model screen of your application. If you're not there already, to get to your application click the Applications tab and then click your application.

From there you can see the data model underneath the application record in the 'auto-expands' section of the application, where you'll see a 'Types' tab. For an example look at the screenshot below.



You just scroll down a bit to see the data model. Alternatively just click 'Model' GoTo item in the application record to bring the data model into focus.



This screen provides links to either the list of type definitions or the list of dropdowns. Dropdown lists will be described in a later section. For now we'll focus on the 'Types' drilldown.

Whether you're editing types or fields, the editor is exactly the same. But what makes types more important than field definitions is that this list of types is used to determine the tabs at the top of your application.

Tip: If you want a type in this list but do not want it to appear as a tab, then you need to set the Visible flag of the field to 'No'.

Another Tip: You can easily change which tab is selected when you first access the application. By default the User tab will be selected. You can change this in the Application configuration, under 'Startup View'.

Because editing types and field definitions is pretty much the same, I'll refer you to the next section on entering field definitions for more information.

6.6. Creating and Editing Field Definitions

6.6.1. Adding a Field

The top line of the list of types or field definitions is known as a Quick Entry line, and looks something like this:

This is a convenient alternative to clicking 'Create New' and filling in the form. If you click the field next to Name, and type in a type name and press Enter, it will create a field definition (or type) but with default values for all of the fields. You can also select the type of the field. Once you create the field, you can still edit the entry some more by clicking it and clicking 'Edit'.

Tip: Another way to get to the Quick-Entry line is to press either Ctrl+Q or Alt+Q, depending on your web browser.

Another Tip: You can create your own 'Quick Entry' lines simply by setting the 'Quick Entry' field to Yes on the field. We'll discuss this more later on.

There are a few tips worth keeping in mind when entering fields using the Quick Entry, and we'll go over them now.

6.6.2. Naming Your Field

It's important to choose an appropriate name for the field you're going to add. It should reflect what it will store. For example the date that a payment is effective could be named 'Effective Date'.

You can always change the name later, but if you do that after it's been used in your application you may lose the values that have already been entered into that field.

There are two kinds of names that a field can have. The 'Name' of the field is the identifier that is used in scripting, reports and other programming interfaces. It's never really displayed to the user, but just used internally.

The internal name has a few limitations that you need to be aware of:

- It must start with a letter, and not a number or symbol.
- It cannot contain spaces. Typically you can get around that by either using underscores ('_') or using what's called pascal casing, where two words are separated by a capital letter, for example UserName.
- It needs to be fairly short, no more than 4 words.

The other kind of name is called the 'Title'. This is what is displayed to the user. The field name and field title can be completely different, but you should try to keep

them similar to avoid confusion. Unlike field names, titles can contain spaces and be any length you like – although you should still try to keep them a reasonable length so they will fit into column headers and forms.

When you create a field entry using the quick entry, you can enter the title of the field. The system then automatically creates a name out of that title.

For example, if we try to create a top-level type definition to store a history of maintenance on your house, you might type in 'House Maintenance' in the Quick Entry. The system will create a field with the name 'HouseMaintenance', and with a title of 'House Maintenance'. By default all fields are created to hold text, but there are quite a few different data types that you can choose.

What's That? A 'Data Type' refers to the kind of information a field will hold. The different kinds include numbers, text, dates and files.

You can then enter another top-level type by pressing Alt+Q and entering another name, you can enter some fields into that type, or you can click the field to make some changes to the field definition.

6.6.3. Specifying a Data Type

Keep in mind that, at the top level of the data model you can only create nested types which appear as application tabs at the top of the screen. So after you enter something at the top level you can click the '+' expansion icon to add fields to that type.

When entering the type in the Quick Entry you can select the data type from the dropdown list. There is also an even quicker way to choose a different data type when using the Quick-Entry. By using some special codes at the end of the field title, you can designate the field as you're typing. An explanation of the different data types, and the codes you would use to specify those types, is given below:

- **Text (Few Words)** – Specifies a short text field. This is the default if no data type is specified.
- **Text (Paragraphs) (...)** - Specifies a long text field, used for entering lots of text (paragraphs).
- **Nested Record** – Specifies that this will be a 'GoTo' item instead of a field. There are some variations on the nested record type you can specify by putting a special character at the end of the name. Alternatively you can select these options from the field definition's 'List Type' dropdown when editing the field.

- > - Specifies that the field is a single-item nested type, which means that it's a type definition that can hold more field definitions.
- >> - Specifies that the field is a nested type that can hold a list of records.
- >>> - Specifies that the field is a nested type that can hold a LONG list of records, which is shown in a list in which paging is adopted to only show a part of the list at a time.
- **Number (Integer) (%)** - Specifies an integer number, that does not allow fractions.
- **Number (Money) (\$)** - Specifies a decimal number type, often used to represent money.
- **Number (Floating Point) (!)** – Specifies a floating point number, that does allow fractions.
- **Date/Time (General) (@)** - Specifies a date or date and time type.
- **Boolean (?)** – Specifies a boolean type, which is used to hold either a 'Yes' or 'No' value, or a 'True' or 'False' value.
- **File (^)** - Specifies a file type, that lets you select a file that is uploaded into the system.
- **Counter (+)** - Specifies a counter. This field automatically increments each time a new record is added, creating a sequence number. It also provides a special screen for quickly locating records by their sequence number.
- ***** - The asterisk is used to indicate that the field links to another part of the database. After the asterisk you would put a path to the location in the database where the user can choose the record to which it will point. This is discussed in more detail in a later section.

So if I wanted to enter a field that records the date and time of a sale, I could enter into the Quick-Entry the following:

Date Of Sale@

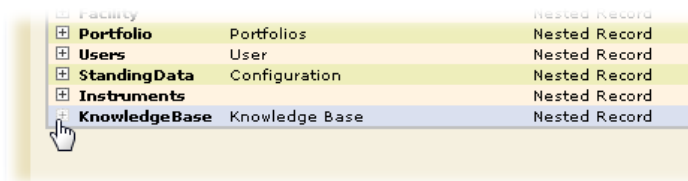
Or, if you would prefer not to use the codes, you can just enter 'Date Of Sale' and select the data type to 'Date/Time' in the next text box.

6.7. Creating a Hierarchy

Often the field you're adding will become a 'drilldown', which appears in the GoTo list of the record, where you can store additional structures in a hierarchy.

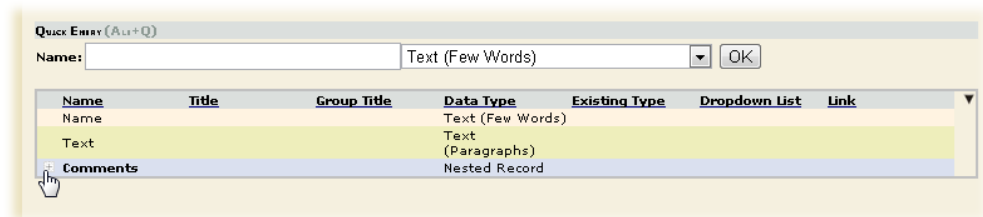
To turn a field into a 'GoTo' item, just click the gray '+' icon next to the field after it is created.

For example, enter a field in the Quick Entry called 'Knowledge Base' and select 'Nested Record':

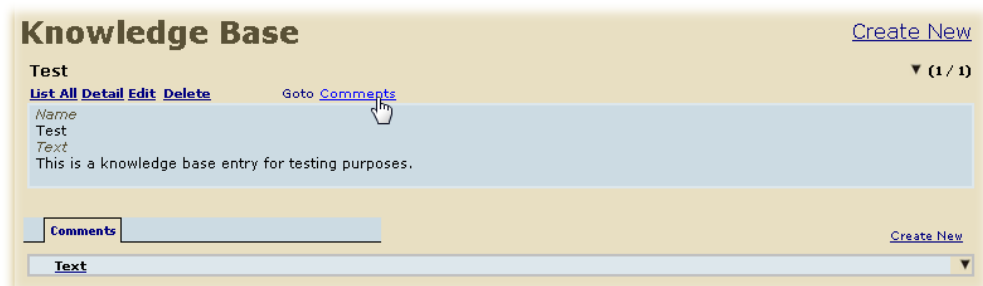


Then click the gray icon next to 'KnowledgeBase' to start entering that field's child fields. This is how we create a hierarchy.

If in that list we enter a few fields, such as Name and 'Text...', we can then add another field called 'Comments>>' and click the plus next to that to create a further level for comments. Note that we put '>>' at the end of the name as a quick way to say it will be a nested type containing a list. We can also just select 'Nested Record' in the type entry box next to the name.



Once you publish, when you go to a Knowledge Base record, you'll see a drilldown to 'Comments' to reflect what you defined.



You'll also see 'Comments' appear in the 'auto expanded' area below the record, and when you create a Knowledge Base entry you'll see a place to drilldown in the new form to add new comments:

Knowledge Base

Name

Text

[Click to add Comments](#)

Create

Figure 16: Showing the area to insert comments

Knowledge Base

Name

Text

Comments

Text

I can enter a comment here.

[Click to add Comments](#)

Create

Figure 17: After clicking, showing the area to enter the comments

6.8. Example: An Invoice Manager

Let's go through an example of setting up a type. For this example we'll set up something to record invoices. We'll do this just using the Quick Entry, for simplicity.

Note: Make sure you enter the field names exactly as shown, to avoid any problems!

At the top level of the model, enter the following, with an enter at the end of each line. You can press Ctrl+Q to get back to the Quick-Entry after you enter each line.

- Customers>>>>
- Invoices>>>>

This will create two types, Invoices and Customers, both of which will hold a (potentially) long list of records. That's why we put the '>>>>' at the end of the

name: it's a shortcut code to tell the system to make it a potentially long list. It just saves us the extra clicks to edit the field and set that up afterwards.

Because these are nested types, an expansion icon will appear next to each type for you to drilldown to the field list.

Next, click the expansion ('+') icon next to Customers, and enter the following into the Quick-Entry. These will be the fields that we'll store in the Customer record.

- Company Name
- Address Line 1
- Address Line 2
- Town
- City
- Country
- Zip Code
- Office Telephone Number

All of these will be entered with the default data type of 'Text (Short String)'. When you're done with these, go back to the top level of the model and this time expand 'Invoices' to enter some fields for the Invoice record. Enter the fields below, entering the specified name and selecting the appropriate data type.

Field Name	Data Type
Client	Text (Short String)
Invoice Number	Counter
Invoice Details	Text (Paragraphs)
Date Sent	Date/Time (General)
Due Date	Date/Time (Due Date)
Sales Tax	Number (Money)
SubTotal	Number (Money)
Amount Paid	Number (Money)
Original Invoice File	File

Once you're done entering those, click on the 'Client' field. This is currently set to be a 'Text' field, meaning you can just enter any text you like into it. However this isn't really ideal: it would be better if we could select a client from the client record we just defined.

Click 'edit' to edit the 'Client' field. Then look for the entry box labeled 'Link'.

Client ▼ (1 / 1)

General
 Name: Client Title: Group Title: Icon: ...

Field Type
 Data Type: Text (Few Words) Existing Type: Dropdown List Link: << Up
 Default Value: Expression (for calculated fields): Format Code: (reset)
 List Type: Single Item Display Level: Summary Visible Condition: Customers
 Attributes: Mandatory? No Calculated? No Hidden? No Unique? No
 Quick Entry? Is Title Field? Custom Field Viewer: Model

This entry box lets you select another place in the data model that the field will point to. In this case we want to select 'Customer' so that we can link the Client to our list of Customers.

Now you're done with a basic data model, we can go ahead and create the application. To do this click the 'Publish Now' action in the 'What Now?' list, and click create. It should compile and deploy in just a few seconds. Then click the application link to have a look at what you've created.

Note: When you access the application after publishing there may be a slight delay before the screen appears, with the message 'Please Wait'. This generally only happens once, and it's because the database is being upgraded with the changes you made.

You should see four tabs at the top of the screen: Customers, Invoices, Users and Settings. The latter two tabs are created automatically by the system.

The screen will look something like this once some data has been entered:

Invoice Mgr Welcome: admin. [English](#) [Sign Out](#)

[Customers](#) [Invoices](#) [Users](#) [Configuration](#)

Customers [Create New](#)

Quick Entry (A, L, + Q)
 Name: [] OK

Name	Address Lines	City	State	Country	Zip	Bad Standing
BLK Distributions, LLC	14 Enterprise Road	New City	New York	United States	10950	No
Figero Catering Services LLC	25 La Mont Road	New York City	New York	United States	10003	

(c) Copyright Your Company

Invoice Mgr Server: 0.1.14 (beta)
 Up Since 11/15/2010 1:00:54 PM

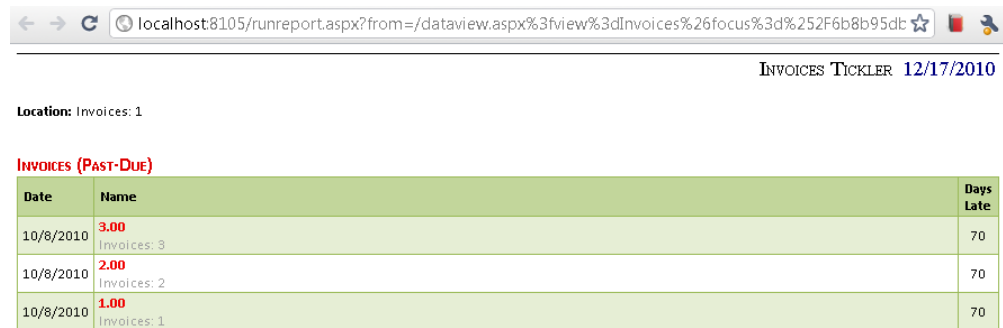
Powered by RiaForm dbFramework

Click the Customers tab and click Create New to add a new customer. You'll notice that dbFramework creates the entry form for you, with all of the layout, navigation and controls for editing.

Fill in some details and click 'Create'.

Then click the 'Invoices' tab, and click 'Create New' to add a new invoice. The Client field should be a dropdown field that lets you select the customer. When you click the dropdown you should see the customer you just entered appear in the list.

Because we made the Due Date field a 'Date/Time (Due Date)' type, the system automatically created a tickler file report that shows all invoices by their due date – very useful to keep track of outstanding invoices. You can access this report by clicking the 'Invoices Tickler' report on the right hand side.



Date	Name	Days Late
10/8/2010	3.00 Invoices: 3	70
10/8/2010	2.00 Invoices: 2	70
10/8/2010	1.00 Invoices: 1	70

There's a lot more we can add, but that just goes to demonstrate just how quickly you can create useful applications using dbFramework.

6.9. Anatomy of a Field

6.9.1. Field Options

Now let's look in more detail at a field definition. Click on a field and click Edit to see a list of the editable properties. It will look something like Figure 18:

Figure 18: Field Editing Form

Below is an explanation of each editable entry in the field definition:

Name

The name of the field. As discussed in a previous section, the 'Name' of the field is the identifier that is used in scripting, reports and other programming interfaces. It's never really displayed to the user, but just used internally. It must not contain spaces, and must start with a letter. It must be less than 65 characters (including spaces) in length.

Note: It's best to keep the names of a field constant and not to update them too often. For example, changing the name of a field after it has been used to store information in the application will result in that data being lost.

Title

The title of the field is what is displayed to the user in the application. It will appear at the top of columns in tables, and will appear as labels next to field values in forms.

The title doesn't have to be the same as the name, it is usually a little more descriptive. However it shouldn't completely different to the name. You shouldn't, for example, have a field named 'UserName' and a title of 'Address'. That would just lead to lots of confusion.

Group Title

When your type has a lot of different fields, you'll likely want to group them. You can do this by indicating that a field is starting a new group, and to do that you simply type in the title of the group here.

The group is indicated by a shaded bar above the field names that follow. For example, setting the group to 'Dates' on the 'Last Modified' field will lead to something that looks like this:

Dates	
<i>Last Modified</i>	<i>Last Modified By</i>
10/5/2010 5:53:35 PM	admin

Icon

In various places in dbFramework the field can be displayed with an icon, or a small picture that graphically represents the information being held. You can select an icon for your nested record, field or drilldown member in the 'Icon' entry box. Clicking the dropdown shows a gallery of icons provided by dbFramework. You can also add your own icons, and this will be discussed in a later section.

Data Type

The data type refers to the kind of information that you will allow this field to store. You could get away with leaving all fields as a text type, because text allows any value to be entered. But for proper validation, correct sorting and optimal storage in the database, it's worth carefully selecting the appropriate data type.

dbFramework provides a number of different data types. We'll run through them below:

- **Nested Type:** This indicates that the field is a type that contains records itself. You will be able to drilldown to its sub-type to add additional fields.
- **Text (Few Words):** The field will contain a short amount of text.

- **Text (Paragraphs):** The field will contain more text, in a box with a scroll-bar. There will be no limit to the amount of text that can be entered.
- **Number (Integer):** The field will contain a number without a fraction.
- **Number (Floating Point):** The field will contain a number with a fraction.
- **Number (Money):** The field will contain a decimal type with a fraction, suitable for storing money values.
- **Boolean:** The field will contain a Boolean type, of 'Yes' or 'No'.
- **Date/Time:** The field will contain a date and optionally a time value. There are two types of Date/Time fields, one for general dates and another for 'Due Dates'. These are dates that represent an event. dbFramework automatically creates a tickler report for each type that contains a field with a 'Due Date'. For more information on this, see section 9.3.
- **File:** The field will contain a file. A special box will be provided for uploading a file to the system. Once committed the value will show the name of the file as a hyperlink that, when clicked, will download the file.
- **Counter:** The field will contain a sequence number that is automatically generated. Each new record is assigned a number in sequence. The same number cannot be re-used.

Tip: You can specify the data type with a special code in the field name when using the Quick Entry. More information on this is in section 6.6.3.

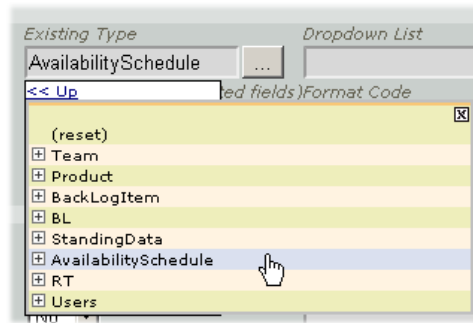
Existing Type

Specifying that the field has a 'Nested Type' data type is just one way of creating a field that can be expanded into a record. Another way is to refer to an existing type that has been defined, allowing you to re-use types elsewhere in the data model.

To do this you can select the type using the 'Existing Type' dropdown. You can drilldown to any place in the data model.

Once selected, any changes that occur to that type will automatically be reflected in this field because now they're linked.

A particularly powerful feature is the ability to select the *same* type in the Existing Type in order to create a self-referencing hierarchy. This kind of recursive data model allows for an unlimited number of levels deep, such as a folder hierarchy.



Dropdown List

You'll use this field if the data type is a text field and you want the user to select a value that comes from a dropdown list of hard-coded values. By hard-coded we mean a list that won't be editable by the user, but instead will be pre-defined by the application. It'll be a list that doesn't change very often.

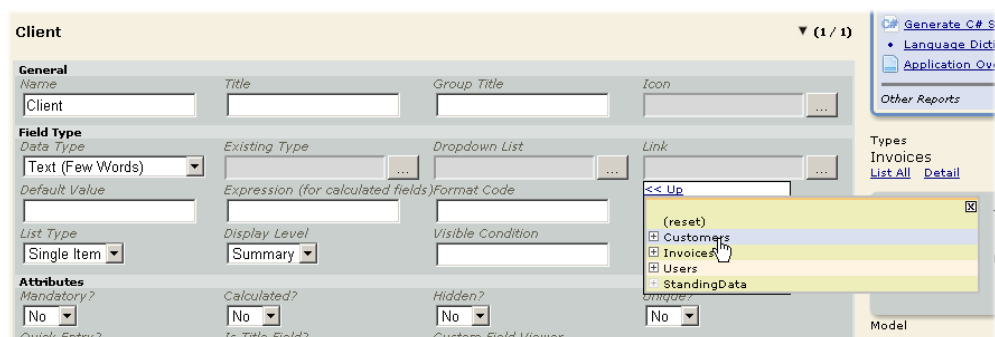
An example of a dropdown list would be a task state. It'll have one of a number of pre-set states such as 'Pending', 'Complete', 'Not Started' – and these options won't really change. Another example may be the gender, which would be either 'Male' or 'Female'. These options generally don't change.

There are two ways to create dropdown lists, and we'll discuss this in more detail in the next section.

Link

A link or 'foreign link' refers to a link to another record in the database somewhere else in the data model. In this field you will select the *type* of the record that you will let the user select by selecting an area of the data model.

To do this, just click the dropdown next to the entry box labeled 'Link'.



You can drilldown to anywhere in the data model, but make sure it's accessible from that particular place in the data model. You can select top level types, or any type that is a parent of the current location in the data model.

We'll discuss more advanced options that can be entered into this field in section 6.14.

Default Value

The default value for a field is the value that the field will have when the edit form is first displayed to the user. It'll also be the value that is used if the form is bypassed, like for example when using the Quick-Entry.

While you can simply enter a text value, or a number, in this field, some special values can also be entered:

- **True or False**
In the case of a boolean field, putting 'True' will make a default value of 'Yes', and putting 'False' will make a default value of 'No'.
- **=DateTime.Now**
This is used to set the value of a date/time field to the current date and time. The date and time the edit form is displayed to the user will be used.
- **=DateTime.Now.Date**
This is used to set the value of a date/time field to the current date only, omitting the time. The date the edit form is displayed to the user will be used.
- **=DateTime.Empty**
A null, empty date value will be selected.
- **=Environment.UserName**
For a text field, this is used to select the name of the currently logged-in user.
- **=CreatePosition()**
A unique number is selected for the field.
- **=Rnd()**
A random number is selected for the field.

Expression (for Calculated Fields)

This field provides space for a formula that can be used to calculate a value for this field based on the values of other fields, or other records. The language that is used is a simplified version of DbfScript, that is described in more detail in a later section.

A basic formula to calculate a display name from a first and last name may look like this:

```
=#input/LastName + ", " + #input/FirstName
```

All formula can be prefixed with an equals sign.

Whenever we're referring to a field of the current record, we prefix that with **"#input/"**.

But we can also reference other records in our calculations. The following formula calculates a count of all of the sub-tasks in a task:

```
=Count( &path/SubTask )
```

In this case we're asking the system to count all of the items in the SubTask record relative to the current path. Whenever we're referring to the current path, we use the identifier '&path'.

Tip: The term 'path' is used a lot in dbFramework. It means the location that the user has navigated to within the database hierarchy.

These formula are also 'live': Whenever the fields or records referenced in our formula are changed, this field will automatically get updated.

More information on calculated fields can be found in section 8.3.

Format Code

The format code controls how numbers and dates will be formatted when displayed to the user. If this field is left blank, then the default formatting will be used.

For example you may enter 'N' in the format code, and this will format a number that would otherwise display as 1123456.78 as "1,123,456.78".

A full list of valid format codes can be found in the appendix.

Tip: dbFramework uses the .Net Custom Format String specification, affecting both numeric values and date/time values. You can refer to Microsoft documentation for a full discussion on how to use these format codes.

Sort Key?

If the field should be used to control the sort order of the records when displayed in a list, then select either Ascending or Descending from this list.

Multiple fields can be combined to create sort compound keys, and they will be combined in the order that the fields appear in the type.

The way in which the data is sorted depends on the data type that is selected.

Note: The sort order can affect the order of rows in reports, if no override is specified, and any calculated fields that reference the #previous variable. Be cautious before changing the sort order to make sure it won't have any unintended consequences.

List Type

The list type controls how many records for this type can be created by the user. The following values can be selected:

- **Single Item:** Only one record can be created. Once a record exists, the Create New link will disappear.
- **Short List:** Multiple records can be created, but they will all be displayed on the screen at the same time, in full detail.
- **Long List:** Multiple records can be created, and they will be displayed in a paged list. That means that if the list gets larger than, say, 20 items, additional items will be accessed only by switching to the next page. This is suitable for large lists because the paging allows any number of records to exist and be navigated without impacting the speed of the system.

Display Level

The display level refers to when the field will be displayed to the user.

If Summary is selected here, the field will be displayed both in lists as a column, and in the full detail view.

If Detail is selected here, the field will NOT be displayed in the list view, it will only be displayed when entering the detail view.

The Summary fields should reflect the most important fields that you wish the user to see when navigating a large list.

Visible Condition

Specifies a formula that is evaluated to determine if this field should be visible to the user. For example, for a task's 'Complete Date' field, we may enter a visible condition of 'State=Completed'.

The visible condition also applies to nested type fields. These will only appear in the Go To list if the condition is met, otherwise they won't be displayed.

Note: In the current version the visible condition only works for nested types, not individual fields.

Mandatory

Fields set as 'Mandatory' must be entered by the user. If they are not entered, then an error is automatically displayed to the user asking them to provide a value.

 **The field 'Name' must be entered**

Warning: Use this field with caution. Setting a field to be mandatory *after* data has been created in the database may result in loss of data if records exist where data for this field is not present. To avoid loss of data ensure that data is entered in this field in existing data before setting this field to mandatory. You can automate this process with the use of a script in a scheduled job that will run at startup.

Calculated

Fields set as 'Calculated' will not be editable by the user. In the edit form the field will simply not appear. In view mode the field will appear and the value would usually be determined based on scripting or a formula.

Hidden

A hidden field will not be displayed to the user in either edit mode or view mode. Hidden fields are usually fields that you wish to temporarily take out of the view, or fields you wish to use only in scripting or business logic.

Tip: Hiding fields can also be used to create top-level types that you do not wish to become application tabs. If a top-level type is marked as 'Hidden' it will just be a type you can reference elsewhere in the data model, it won't become an application tab.

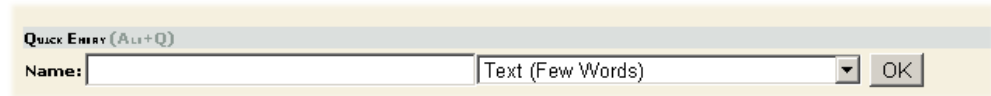
Unique

A unique field indicates that values entered into this field must be unique, and cannot exist elsewhere in that list at that particular location in the database hierarchy.

Quick Entry

Marking a text field as 'Quick Entry' will automatically create a Quick-Entry box on top of the list letting the user quickly enter new records by just entering a value for this field.

The Quick-Entry looks like this:



Any number of fields can be marked as 'Quick Entry' fields.

It's important to note that there cannot be any mandatory fields (other than the selected 'Quick Entry' fields) in the record if you have a Quick-Entry field.

Is Title Field

In various places, such as dropdowns or other lists, and at the top of the detail view, the record is represented by a short title. Selecting a field as the title field means that the value of this field is used as the title of the record.

If you don't select a title field, dbFramework will automatically select one for you. To be safe it's best to set the title field explicitly.

Custom Field Viewer

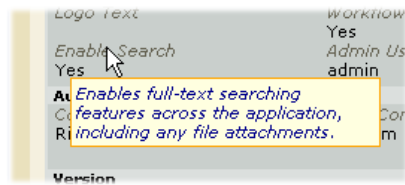
dbFramework allows the developer to create new ways to display and edit fields. By adding a number of 'field viewer' entries under the Application record, with appropriate HTML code, you can select which field viewer you want associated with this particular field using this entry box. This will be discussed in more detail in a later section.

Description

This is a text area that lets you enter a description of the field.

The description of the field is displayed in a tool-tip when the user hovers over the field title in forms. This is a yellow popup window that disappears when the user moves their mouse away from the field title.

An example of a tooltip is shown below:



6.9.2. Field Flags

From the field definition form there is a drilldown called 'Flags' that lets you select some additional behavior of the field.

The flags are selected by adding a flag record and selecting the specific flag from the list. The available options are detailed below.

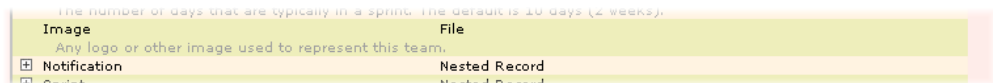
Refers to Owner

This indicates that this field holds the name (or foreign link reference) to the user who owns the record. By default when a record is created it is owned by that user, however ownership can be transferred by updating the value of this field.

The ownership of the record is important when using security permissions that only allow the owner access to the record.

In-Line Preview

When a field is set to 'In-Line Preview' it will appear in the list underneath each record, indented slightly and often in a lighter color and smaller font. An example of a preview is shown below.



Hide Content

The 'Hide content' flag lets you specify that the value of this field should never be shown to the user after it is entered. Instead a series of asterisks will be shown. When the field is edited it will be empty unless a new value is entered. This is particularly useful for password fields, social security numbers or other, sensitive information. The information can be shown on a report, however, if the field is explicitly selected.

Modified Date

This flag indicates that the value of this field is used to store when the record was last modified. This functionality is probably easier to configure using the Features, as discussed in a later section.

Created Date

This flag indicates that the value of this field is used to store when the record was created. This functionality is probably easier to configure using the Features, as discussed in a later section.

Rich Text Editor

This flag indicates that the free-form text (with a data type of 'Text (Paragraphs)' can contain rich-text, with rich formatting such as bold, different font sizes and so on.

Note: This feature is not supported in the current version.

Triggers In-Line Change Event

This flag is used in conjunction with the 'InLine Change' event, and is used to manipulate the form values as they are being entered by the user. We'll discuss this in more detail in a later section.

Disallow Create/Edit/Delete

These flags control whether the 'Create New' link, along with 'Edit' and 'Delete' links at the top of records, will be displayed to the user. While permissioning can control whether these appear for different users, this flag disables the options completely. This is often used for instances where you wish this type of interaction with the data to be controlled by the system, perhaps through actions or scheduled jobs.

6.10. Creating Dropdown Lists

Dropdown lists provide a means of forcing a field value to be selected from a pre-defined list, rather than being edited or selected from a part of the database.

An example of a dropdown list might be a task status. The possible values might be 'Not Started', 'Pending', 'Complete', or 'Deferred'. These are predefined values and won't change after the application is published. You wouldn't want the user to type in their own task status as this will make it more difficult to determine the actual status of the task.

You'll notice dropdown lists in the dbFramework application, such as the data type dropdown. When you click them there is no way to just freely edit the content, a simple white-background list appears from which you must select a value.

Creating dropdown lists in dbFramework is a simple task.

The easiest way to create a dropdown list is to use the 'Dropdown' 'GoTo' item under a field definition. From there you can just enter in the dropdown options, and dbFramework will automatically create a dropdown list to reflect those options.

Sometimes you may want to have a dropdown list that is shared between multiple fields. For example a week day selector may appear in several places in your application. To do this you will want to create a dropdown in a different place, and then select that dropdown in the 'Dropdown List' entry box in the field.

First, go to the application and select the Model drilldown. From there, next to where you normally drilldown to the Types you can instead drilldown to the 'Dropdown Lists'.

Then click 'Create New'.

Tip: Dropdown lists can be hierarchical. That means you can group related dropdown lists together. To do this, just create a dropdown list with the name of the group, and click 'SubDropDownList'. You can put your grouped dropdown lists in there.

In the Dropdown List entry form you can specify the name of the dropdown list and the title. The limitations of a dropdown list name are similar to those of a field definition, as discussed in section 6.6.2. The dropdown list name should not contain spaces, and must start with a letter. The title of the dropdown list has no such limitations, and is optional.

Just like entering the name of the field, if your dropdown list name does contain spaces or other characters, dbFramework will automatically take them out to create a valid name, and leave the original name to be the title.

After entering the name, you can add the members. The dropdown list members are the possible values that can be selected by the user.

To add a member, click where it says 'Click to add Members'. This will show a nested edit form for a member. The values for this are the same as the dropdown list: you enter a name, a title and also an optional description.

In our example of a task state, we would create:

1. A dropdown list called 'Task State'
2. A dropdown list member called 'In Progress'
3. Another member called 'Deferred'
4. Another member called 'Complete'
5. Another member called 'Not Started'

The order of the members on the screen is the order of how they will be shown to the user in the list. You can change the order of the members by using the clipboard operations, 'Pick Up' a member, and then 'Drop Above' to position it above another record.

Tip: When you sort a record list by a field that is a dropdown, it will use the order of the members to determine the order of the list.

You can then use this dropdown list anywhere in the data model.

To use a dropdown list in a field, go to the field definition, edit it, and in the field called 'Dropdown List' you can select the name of the dropdown list you just created.

You can now publish the application and see how this looks in the form.

6.11. Conditional Coloring

6.11.1. Overview

Sometimes it's useful to have rows or individual cells in a list appear differently to highlight a specific state. To do this in dbFramework you can use something called 'Renditions'.

Renditions can be applied to both fields or types, and each can have multiple renditions applied. A rendition has a name, an optional condition that must evaluate to true for the rendition to be applied, and then a list of formatting flags that control how the text will appear.

6.11.2. Configuring Renditions

Renditions can be created on either fields or types. Go to the field definition for which you wish apply the rendition, drilldown to 'Renditions' and click Create New.

The following fields can be entered:

Title

Specify a brief description of what your rendition does.

Condition

This is the condition that must evaluate to true for the rendition to be applied. This is a DbfScript expression. The #input variable provides access to all of the fields in the current record.

An example of a condition might be:

```
#input/State == "Started"
```

Apply Font Bold

Indicates whether the 'Font Bold' setting should be applied.

Set Font Bold

Indicates whether, if applied, the font should be bold or regular weight.

Apply Font Italics

Indicates whether the 'Font Italics' setting should be applied.

Set Font Italics

Indicates whether, if applied, the font should be italicized or straight.

Apply Font Color

Indicates whether the font color should be changed.

Set Font Color

Indicates the font color to set, if applied.

Apply Background Color

Indicates whether the cell background color should be changed.

Set Background Color

Indicates the background color to set, if applied.

Note: In the current version background colors cannot be modified in conditional renditions.

Apply Strikethrough

Indicates whether the font strikethrough setting should be changed.

Set Strikethrough

Indicates whether to set or clear the strikethrough of the font, if applied.

6.11.3. Ordering

When multiple renditions are defined, each with different conditions, the renditions will be combined in the order in which they were created. They will be combined based on the application of each.

For example if one rendition applies the font bold change and makes it bold, but the next one applies font bold change but clears the font bold, then the resultant rendition will have regular weight.

But if one rendition applies and sets the font bold, and the doesn't apply font-bold change but does apply strikethrough, then the final rendition of the cell will show the data bold and stricken.

and goes to the first frequency.

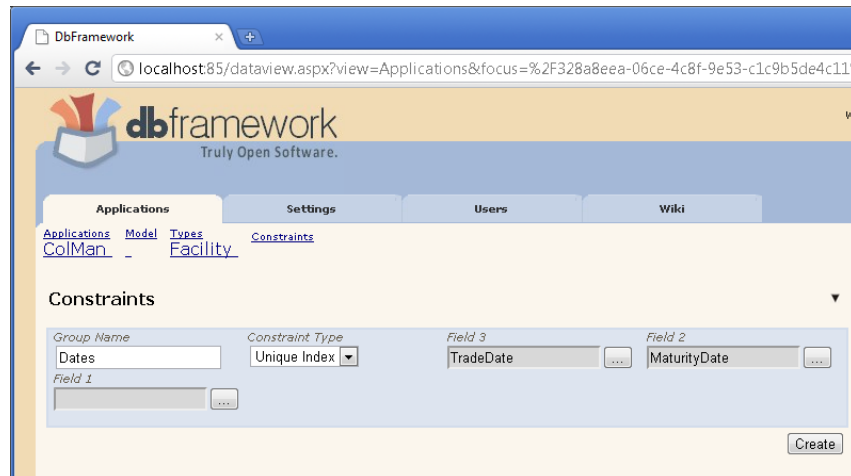
Review: Order Lunch for Wednesday	☐ Not Started	10	Daisy Castillo	0	11/20/2010	11/20/2010
The task 'Order Lunch for Wednesday' completed by 'John Wood' needs to be reviewed.						
Set up scenario server	☐ Not Started	10	Adam Rice	1	12/1/2010	12/2/2010
Write ColMan administrator guide	☐ Not Started	10	John Wood	5	1/13/2011	1/21/2011
Need to write an administration guide that includes details of setting up the scenario server.						
Review: Install security cameras	☒ Complete	4	Daisy Castillo	0	12/9/2009 12:00:00-PM	12/9/2009 12:00:00-PM
The task 'Install security cameras' completed by 'John Wood' needs to be reviewed.						
Prepare Holiday Cards	☒ Complete	40	John Wood	0	2/23/2010	2/24/2010
Order lunch for Wednesday	☒ Complete	40	John Wood	0	2/10/2010 12:00:00-PM	2/10/2010 12:00:00-PM

Figure 19: Shows a conditional rendition that shows complete tasks

6.12. Specifying Constraints and Indexes

The 'Unique' flag on a field is an example of a constraint: it stops the user from entering two records that have the same value for that field. That's a basic example of a constraint, but dbFramework supports other, more advanced constraints that can be specified in the 'Constraints' drilldown for a field.

You can add multiple constraints. Click 'Create New' to add a new constraint.



The first thing you need to do is to enter a name for your constraint. It doesn't matter too much what this name is, so long as it roughly describes what the constraint means.

Next you can specify the type of constraint. Currently there are two types of constraint: Either an index, or a unique index.

An index constraint simply optimizes how records are found in the database. A bit like an index in the back of a book, it allows dbFramework to quickly locate a record for a given value, rather than having to search through the whole database one record at a time.

A unique index constraint is similar, it also optimizes how records are found but in addition to that the combination of field values must be unique in that list. If, for example, the Last Name and First Name must be unique as a combination, rather than individually, then you would create a Unique Index and select the First Name and Last Name fields in the field selections that are provided.

Unique index constraints are also known as a database key.

Note: Only up to three field selections for an index are currently supported.

6.13. Inheriting Types

Sometimes when we're creating a type, it's very similar to an existing type – except perhaps it has some additional entries. To save the developer from having to re-enter all of those fields, dbFramework allows a type to *inherit* from another type. When a type inherits another type, the *parent* type, it automatically takes on all of the parent's fields. Any additional fields that are defined in the type will appear before the parent's fields.

dbFramework also allows multiple inheritance, allowing the developer to select multiple parents for a particular type. All of the parents' fields are added to the type.

To specify that a nested type inherits from another type, use the 'Inherits' drilldown on the field definition. In that list, click 'Create New' and select the type that this nested type inherits from.

The inherited fields won't be shown until the application is published.

Note: An advanced feature, special support for inheritance is provided in the SDK generation, allowing these types to be used polymorphically. This is discussed in more detail in the section on developing business logic.

6.14. Links

As discussed briefly in a previous section, a foreign link refers to a link to another record in the database. When editing the foreign link we indicate the *type* of the record that you will let the user select.

Foreign links are extremely useful when designing a data model, because they allow relationships between completely different parts of the database to be established.

The simplest way to create a foreign link is to just select the area of the data model using the 'Link' entry box dropdown. You must either select a type at the top level, or a field that is a parent of the current field in your data model.

The screenshot shows the 'Client' configuration window. It has a 'General' tab with fields for Name, Title, Group Title, and Icon. Below this is the 'Field Type' section with fields for Data Type (set to 'Text (Few Words)'), Existing Type, Dropdown List, Link (set to '<< Up'), Default Value, Expression (for calculated fields), Format Code, List Type (set to 'Single Item'), Display Level (set to 'Summary'), and Visible Condition. The 'Attributes' section includes Mandatory? (No), Calculated? (No), Hidden? (No), Quick Entry? (No), and Is Title Field? (No). A dropdown menu is open next to the Link field, showing options: (reset), Customers, Invoices, Users, and StandingData. On the right, there are links for 'Generate C# S...', 'Language Dict...', 'Application Ov...', and 'Other Reports'. At the bottom right, there are links for 'Types', 'Invoices', 'List All', and 'Detail'.

6.15. Advanced Links

You can also utilize more advanced expressions to create these links by clicking the 'Advanced' GoTo item from a field and entering the 'Advanced Link' entry box. You'll do this using a special notation known as an XPath. You'll need to be a little familiar with the data model you entered in order to use this field.

An XPath is like an address to a part of the data model. XPaths are similar to how we might refer to paths in Microsoft Windows or filing systems in other platforms. Basically, it's a list of type names. Each type name refers to the name of the field definition that you have created, and each is separated by a forward slash '/' character. For example a basic XPath might look like this:

```
/Products/News
```

Often, the first item in the path is a forward slash, which indicates it starts at the top-level of the data model. However this first item in the path list can also indicate that we're starting from somewhere else in the data model.

If the path starts with a double dot, it indicates that the path starts from the parent type of the current type. For example, in a more complex data model for a task manager, you might want Task Assignment to be selected from the list of Team Members of the Team. The Team is the parent type of the task, so for the 'Task Assignment' we might have a foreign link such as this:

```
../Members
```

You can also specify another foreign link field in the current record to act as the root of your path. To do this you start the path with a row function. For example:

```
row(Project)/Task
```

This indicates that the Project field in the record will be used to start the path, and the Task drilldown from the selected project will be used as the selection list.

The start of the path can also indicate a type name to find. The type name to find uses its fully-qualified name, which means that it is prefixed with all of the types that come before it. For example if I had a type called Task under a type called Project, then the fully qualified name of the Task would be the ProjectTask.

In the previous example, rather than starting with “../Members”, we could be more explicit and indicate the actual type we want to find based on the current location. To do this we start with an asterisk, followed by the fully qualified type name, and then the fields we want to traverse from that location. For example:

```
*/Team/Members
```

Finally, you can also specify filters to apply to the list. You do this by specifying the filter (also known as a *predicate*) in square-brackets. The filter must refer to a field in the type that is specified by that part of the path. Each filter predicate is specified as a field name, a comparison operator, and a value. Multiple filter predicates can be specified, but each must be separated by a comma. If two predicates have the same field name, then the predicates are ORd together, otherwise they are ANDed together.

For example, if we wanted to only show the team members that were available for assignment, we might have a foreign link such as this:

```
*/Team/Members[Available=True]
```

This will then show a list of members for the team that have their available field set to ‘Yes’ (which is indicated as True in predicates).

You can use equals (=), not equals (!=), greater (>), less than (<), greater than or equals to (>=) or less than or equals to (<=).

Note: Complex filter values, such as formula, are not currently supported. Instead you can create a separate formula field and reference that in the filter.

6.16. Adding Features

One of the most powerful drilldowns from a field definition is the 'Features' drilldown.

The Features drilldown consists of a number of flags, each of which (when enabled) automatically implements enhanced functionality into your application. These features can be switched on or off at ease, without any additional administration. For example, switching on the 'Tagging' feature will automatically add a tag navigator tab, and the ability to select tags for that record. And you can just as easily switch a feature off, and all of the structures added for that feature will be deleted.

These features only appear once the application is published.

Warning! Once you have published your application with features enabled, and information was entered into the feature-related structures, when you switch that feature off and publish the application any data you entered relating to that feature will be deleted, and cannot be recovered unless the database is first backed up. Use the feature list with caution!

Each of the available features is discussed in detail in section 11.

7. Workflow

7.1. Introduction

Workflow commonly refers to the automated routing of tasks and documents, and it's a powerful tool that can quickly help a well-managed business run more efficiently, streamlining and speeding up fundamental business processes.

There are several ways to implement workflow in dbFramework:

- **Process workflow, using the workflow designer** (this section)
Enables you to graphically design the routing and stages of tasks, known as process workflow or business process modeling. An example of this type of workflow is a support ticket. This starts with a form where the customer enters the details of the problem. Once submitted it is screened by the general support technician where it is either answered immediately, or escalated to an expert. The expert will then optionally request more information, to which the customer will respond, and finally the support ticket will be resolved.
- **Approval workflow** (see section 11.2)
Lets you specify areas of the database that cannot be entered or modified without approval by a specific user group, optionally with conditions. An example of this type of workflow would be a purchase request. The request cannot be entered without approval by the supervisor. If the purchase is for more than \$5k, for example, it would also have to be approved by the department manager.
- **Custom actions and event handlers** (see section 8)
A highly extensible framework for adding custom actions and event handlers using programming languages. For workflow that involves complex decisions involving advanced calculations and data aggregation, this type of scripting can be employed.
- **Integration with external systems** (see section 21)
The ability to interact and integrate with external systems using the XML API. For example, workflow may be started by an event in a separate system. Or some stage in a workflow may involve triggering an action in an external system.

This section will detail the workflow designer features of dbFramework.

Note: The screenshots in this section are based on the WorkflowFirst product. All options are the same as those in dbFramework. WorkflowFirst provides a *subset* of the functionality available in dbFramework.

7.2. Workflow Basics

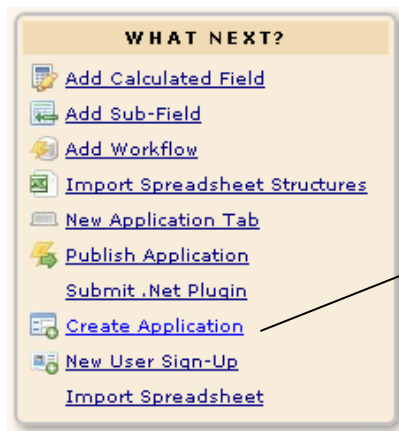
Workflow can be added to any area of the database, and any area of the hierarchical data model. This means that you can have workflow available on an Invoice itself, on the Invoice Items or on the Customer record.

Just like anything created in dbFramework or WorkflowFirst, you first design the workflow and then, when you want to *use it*, you must click 'Publish Application' and run the application in another window.

Creating workflow using dbFramework or WorkflowFirst is far simpler than other business process modeling systems because you don't have to design any forms or databases, instead it creates these for you as you go. It lets you focus on the workflow itself instead of having to design forms.

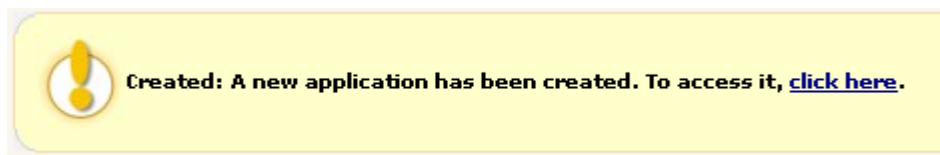
7.2.1. Creating a New Application

If you're starting a new workflow application, click the 'Create Application' link in the tasks list:



Click the 'Create Application' link

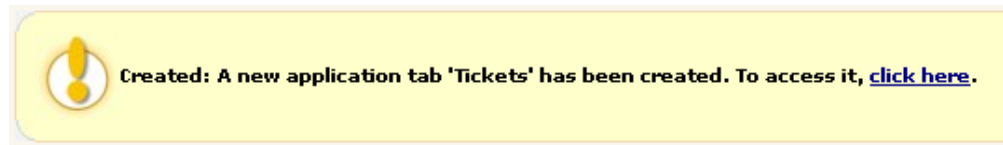
Once created, it will display a link like this:



Click that link to take you to the application record. Next you need to add an application tab. This also adds a record type. To do this click the 'New Application Tab' button:



Once you create the application tab you'll see a message like this:



Click the link to take you to the type. For an existing type that you created previously, just go to the type record where you wish to add the workflow.

7.2.2. Adding the Workflow

An example of an existing type definition is shown in Figure 20. You can then either go to the Workflows drilldown (go-to item) and click 'Create New', or you can click the 'Add Workflow' button.

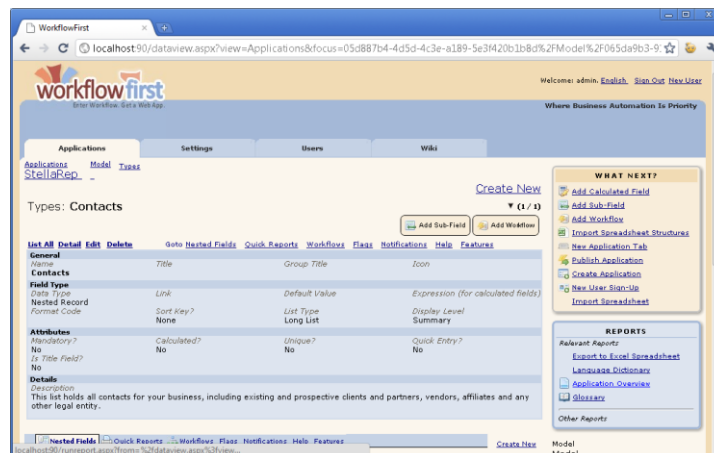


Figure 20: Type definition screen, showing workflow button



Figure 21: The 'Add Workflow' button

Clicking the 'Add Workflow' button (or clicking Create New on the workflow list) will display a form where you can enter the name of the workflow.

Add Workflow: **Workflows**

Title

Workflow Type **Updates Existing Item**

Condition

Show State **Yes**

Create Report **No**

Email Notification **No**

Assignable **No**

Audit Trail **Yes**

Locked? **No**

Notes

Create

Figure 22: Adding a new workflow

The following fields can be entered:

Title

The name of the workflow, which is just used for internal reference purposes and isn't displayed to the user. For example, "Help Desk Ticket System".

Workflow Type

Either 'Updates Existing Item' or 'Creates New Item'. This refers to how the workflow starts.

- **Updates Existing Item:** The workflow can be started once a record has already been created. It will modify (update) the existing record as it goes.
- **Creates New Item:** The workflow can be started *before* a record has been created. The workflow creates a new record and then updates it as the stages progress.

Condition

This is an optional, simple condition that is evaluated to determine if the workflow start button should be visible. For example you could put 'Complete=False' here to ensure that the first workflow stage will only be visible if the Complete field is set to False.

Show State

Indicates if the current workflow stage of the record should be visible as a field.

Create Report

Indicates if a workflow report should be provided. This will provide a list of all records in the workflow, grouped by their current stage. This report can be subscribed to so that it is automatically emailed to your inbox each day.

Email Notification

Indicates if a Notifications area will be added, letting users subscribe for email alerts when records reach specified workflow stages. The email will include a link to the workflow record so that further actions can be taken.

Assignable

If this is set, then the record will be turned into a task. Various fields will be added related to tasks, such as progress, estimated completion dates and a user assignment field. These fields can be selected as input fields when creating the workflow stage. When the 'Assigned To' field is set, the system will automatically send an email to that user to tell them that a task has been assigned to them. They can then click the included link and participate in the workflow.

Audit Trail

Indicates if a history will be recorded of all changes made to the record, whether through workflow or by manual editing. An 'Audit Trail' drilldown will be added and the system will automatically create an entry in this list whenever a change is made to the record.

Locked

Indicates if you wish to lock the workflow so that it won't be updated by any upgrade process. This is only applicable if you are customizing a sample application.

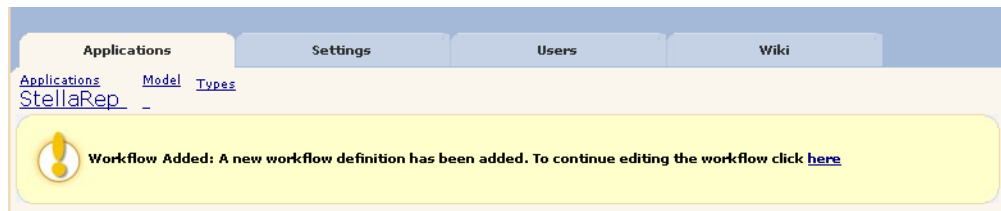
Notes

This is an area for entering in any notes you want to make on the workflow design or purpose. These are not shown to the user and are used for internal reference purposes.

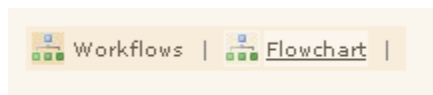
Once the workflow is created, it will display a message with a link that will take you to the workflow designer.

Note: Any number of workflows can be added to a field, and any number of workflows can be in progress at any time. Because they share access to the same record,

workflows can also interact with one another, allowing you to model some very sophisticated business processes.



You can also click on the 'Workflows' drilldown, select the workflow you just created, and once that's displayed click the 'Flowchart' view:



...to switch to the designer. The 'workflows' view lets you edit the data of the workflow without the graphical designer to assist you. This may be useful for certain operations like rearranging stages.

Once the flowchart is displayed, the initial screen will just show the main workflow starting point:

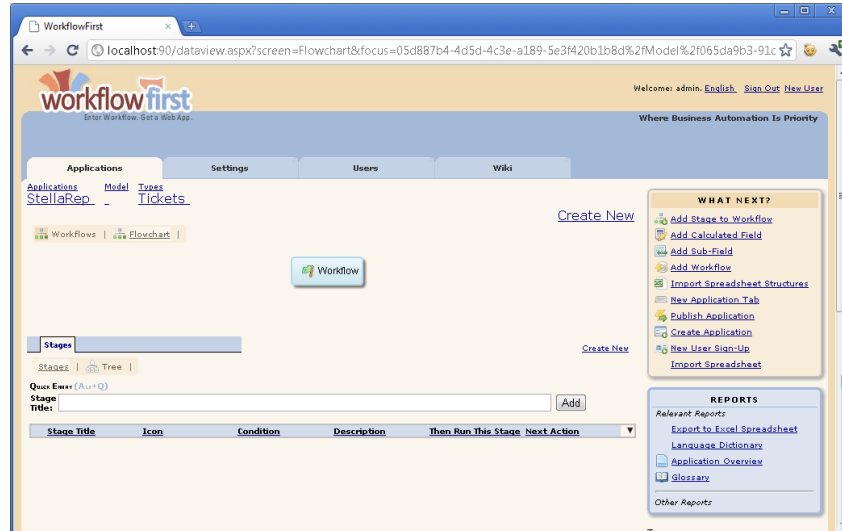


Figure 23: Initial view of the workflow designer

If you put your mouse cursor inside the blue box, a number of toolbar options will appear:

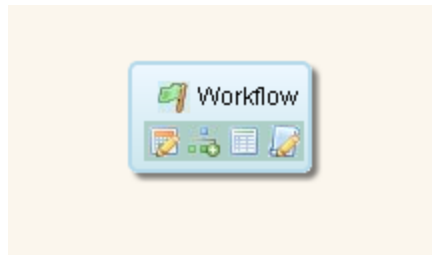


Figure 24: Action buttons available from within a workflow item

From left to right, the icon buttons are:

Edit

This command takes you to the edit screen for this workflow stage (or the main workflow record). Here you can edit the workflow stage title, icon and other setting.

Add Sub-Stage

This command takes you to the 'New Sub-Stage' form where you can enter the details of a new stage that will go under this stage. We'll be covering this next.

Select Input Fields

For a workflow stage that has user-input, this lets you select the fields that will appear in the form generated for that step in the workflow. This can consist of existing fields, or new fields you can specify as you create the stage. This will be covered in this section.

Edit Script

In addition to letting the user enter fields and allowing the user to make decisions, you can also add some ad-hoc scripting. The scripting will have complete control over what can be set in the stage and also provides the ability to easily start a secondary workflow. Details of the scripting language, DbfScript, are provided in section 20.

7.2.3. Workflow Stages

7.2.4. Concepts

The next step is to add some workflow stages. These are displayed in the workflow designer, and the process flows from top to bottom. An example of a workflow in the designer is shown below:

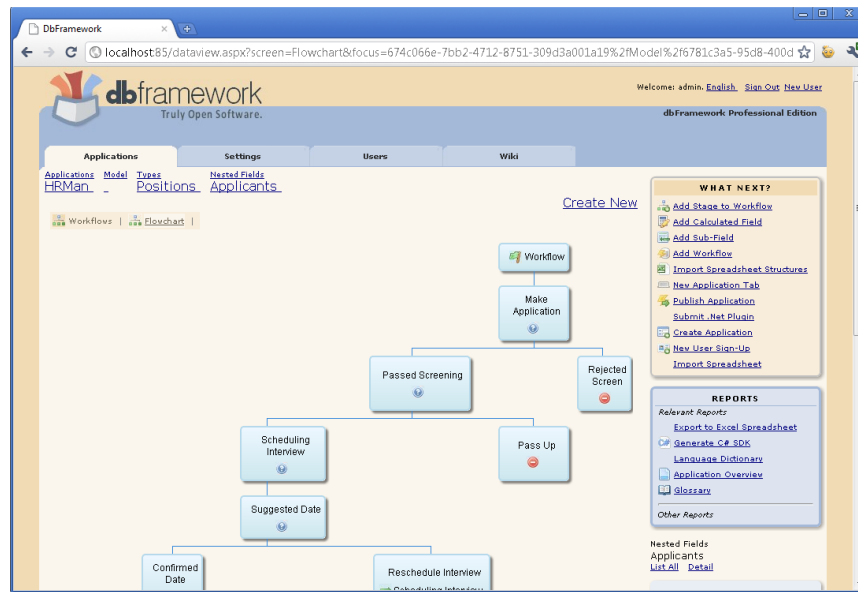
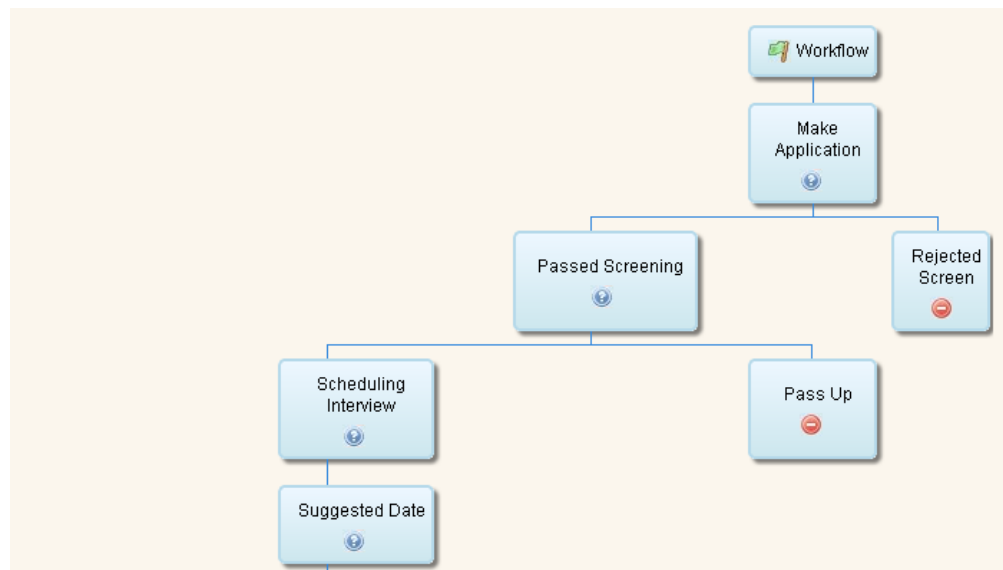


Figure 25: Example workflow in the designer

We'll look more closely at this particular workflow to better understand how the structures relate to the business processes. In this case we're modeling the process of applying for a vacant position at a company.



The top most box in the flowchart is the Workflow Record, and every workflow has one of these.

Underneath that are all the stages. The 'Make Application' box is the first stage. This collects information from the user about the position application. Once published in the application, this translates into an action and a form.

Positions: Accounting Clerk (1 / 1)

Change Filled Status Make Application

List All Detail Edit Delete Goto Skills Applicants Notifications Audit Trail Report Subscriptions

Title	Filled	Notes
Accounting Clerk	No	

Skills Applicants Notifications Audit Trail Report Subscriptions Create New

Quick Entry (Alt+Q)

Skill: Add

Skill Critical

Figure 26: The 'Make Application' button appears automatically on a Position record

hrmanager Access Your Company

Welcome: admin English Sign Out New User Search

Human Resources Manager

Department Positions Skills Interview Guides Org Chart Users Configuration Tag Navigator

Make Application

Make Application

Full Name: Terry Greenfield Email: tgreen@gmail.com

Address: 5 Rossmore Road, Stony Point, NY 10983

Telephone: 845 512 2156

Resume Attachment: (Choose File) my_resume.docx

Create

(c) Copyright RiaForm Technology 2011

HRMan Server: 0.1.125 (beta)
Up Since 5/27/2011 4:2:00 PM

Powered by RiaForm dbFramework

Figure 27: Once clicked, an empty form will appear with the fields you need to enter

After the user fills out the 'Make Application' form and submits that, it automatically creates a new record in the 'Applicants' drilldown from the Position record.

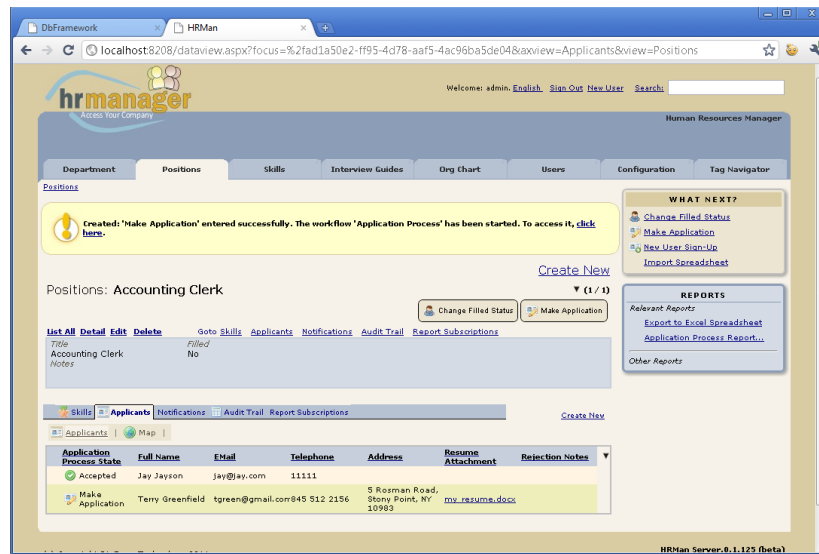


Figure 28: Once submitted, the new record appears at the bottom, and a message at the top.

Anyone subscribing to notifications (we'll touch on this in a later section) will receive an email telling them that a new Applicant record has been added, and when they click on that link they'll be taken to the record.

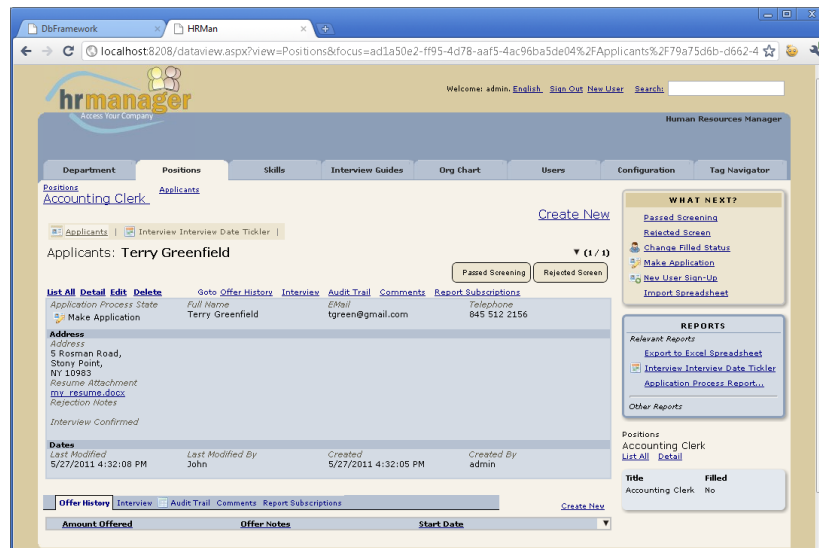


Figure 29: The newly created Applicant record, with action buttons at the top

Now, getting back to the workflow designer in Figure 25. Underneath the 'Make Application' stage, processing splits between two further stages: "Passed Screening" and "Rejected Screen". It's those two stages that correspond to the buttons you'll now see above the new Applicant record. By adding those sub-stages, we've added a choice for the user to make.

Once they click one of those buttons, workflow processing will continue underneath that stage.

So in this case we may click 'Passed Screening'. Refer back to the flowchart and you should see what options this will now provide on the record:

Applicants: Terry Greenfield

[Scheduling Interview](#) [Pass Up](#)

[List All](#) [Detail](#) [Edit](#) [Delete](#) [Goto](#) [Offer History](#) [Interview](#) [Audit Trail](#) [Comments](#) [Report Subscriptions](#)

Application Process State	Full Name	Email	Telephone
Passed Screening	Terry Greenfield	tgreen@gmail.com	845 512 2156

We now see the options of 'Scheduling Interview' and 'Pass Up'. If we click 'Scheduling Interview', this can then be sent to whomever schedules interviews – the secretary for example. Again refer to the flowchart and you'll see the option 'Suggested Date' under that. Once we click that button we get a form like this:

Suggested Date

Interview Date: Interviewer:

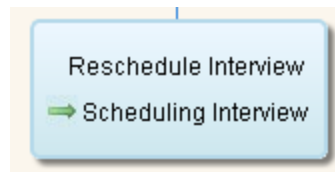
Calendar: May 2011. The 27th is selected.

[Create](#)

As we enter this data in, it automatically populates the fields in the Applicant record. At the end of the workflow we'll have a complete record. Because WorkflowFirst and dbFramework are hierarchical, this record can contain some pretty complex structures, enabling some very sophisticated workflow to be modeled.

Lets continue to look at the rest of the workflow for the Position Applicant, continuing on from 'Suggested Date' for the interview, in Figure 30.

The 'Suggested Date' stage forks into two stages, 'Confirmed Date' and 'Reschedule Interview'. We'll look at the 'Reschedule Interview' stage because, if you look closely, you'll notice a green arrow at the bottom of the box:



This means that if the 'Reschedule Interview' option is selected, then the workflow will jump back to the 'Scheduling Interview' stage, going back to showing the 'Suggested Date' option where you can enter in a new date. This is controlled by selecting the stage in the 'Then Run This Stage' option for that stage.

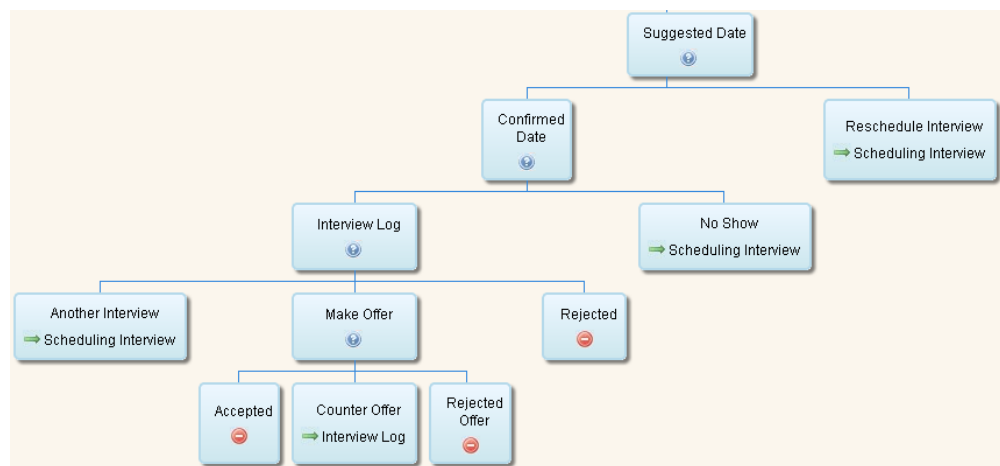


Figure 30: The remaining of the Position Applicant workflow

We use the same technique in the 'Another Interview' stage, so that we can schedule a new interview and *re-use* the processes involved in that.

Now this brings up an interesting issue: we want to store a history of each interview that takes place, we don't want it to overwrite the interview details for each one that is scheduled. How can we do this?

When you add a field to a stage, which goes to constructing the form the user will fill out, you have an option to specify a 'Separate List Tag'. This lets you put the entered value into a separate sub-list of the record. It appends values to that list, so that every time that stage is reached a new entry is made in that sub-list. This lets you construct highly structured data in one simple workflow.

The final concept worth mentioning in this section are the terminators.

For each stage, you have to specify what happens next. This is put into the 'Next Action' dropdown at the end of the stage. This is what controls the icon that appears in the workflow stage box. This can be:

User Decision

This means that the next stages will appear as buttons on the record when this stage is reached.

Auto Decision

This means that the next stages will be evaluated automatically. The Condition on the stage will hold an express that the system will automatically evaluate to determine which path to take. No user action is required.

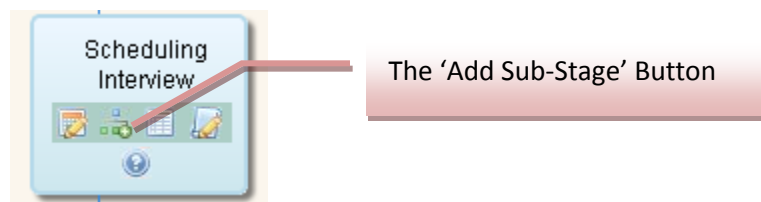
Finish

This means that the workflow is complete and no further actions can be taken.

If the 'Then Run This Stage' is specified, then the above options are ignored.

7.2.5. Adding a Stage

To add a new stage, you hover your cursor over a box in the flowchart diagram, and select the second button 'Add Sub-Stage':



This will bring up a new stage form for you to fill out:

Sub-Stages

Stage Title

Icon

Condition

Description

Add a 'Fields To Collect' entry

Then Run This Stage

Next Action

Create

The following fields can be entered:

Stage Title

The title of the stage is used to title the action button, and is also used for internal reference purposes.

Icon

An optional icon for this stage, that will be displayed in the task name and action buttons.

Condition

For automatic decisions, a DbfScript expression that will be used to determine whether this stage will be selected. If multiple stages evaluate to true, then the last stage will be selected.

For 'User Decision' stages, the Condition field holds a simple expression that defines whether the option will be visible to the user. This works the same way as the Visible Condition, described in section 6.9.1. It can be used to effectively suspend a workflow until a certain condition is met, often triggered by a secondary workflow or process.

Description

The description of the stage. This is used for reference purposes and isn't displayed to the user.

'Fields to Collect'

Click this to add a field reference or a new field to use in generating the input form for this stage. This is discussed below.

Then Run This Stage

Optional. If you wish this stage to jump to another stage on completion, then select the stage to jump to here.

Next Action

Defines how the next set of stages (the sub-stages) should be processed. If 'Auto Decision' is selected, then the conditions of each sub-stage is used to determine if that stage should be followed. If 'User Decision' is selected then the sub-stages are displayed as buttons presented to the user.

Once you are done entering the details of the stage, click 'Create'. This should take you back to the flowchart where you can continue adding additional stages.

7.2.6. Adding Fields to Collect

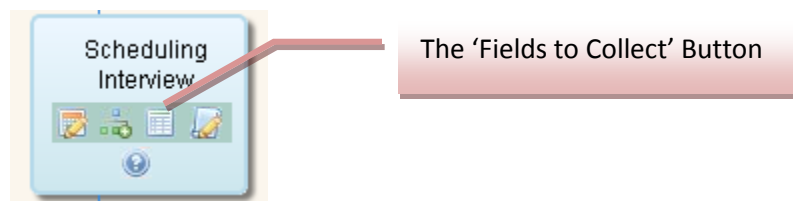
Any stage in the workflow can collect information from the user. In many ways the workflow features in dbFramework and WorkflowFirst focus on assisting in a process of gradually filling out a form.

There are several ways to specify the fields that should be collected at a particular stage.

When you are entering a new stage, you will see a 'Add 'Fields to Collect' Item' link which you can click for each field you want to add.

[Add a 'Fields To Collect' entry](#)

Alternatively, from the flowchart you can access the fields associated with that stage by clicking the third toolbar button:



From that list, you can then click 'Create New' to add a new field.

Lastly, from the stage record view, you can select the 'Fields to Collect' drilldown to get to the same list.

Once you are on the form for adding a new field to collect, you will see something like this:

<i>Field Reference</i> 	<i>New Field Title</i> 	<i>New Field Type</i> Text (Few Words) ▼	<i>Is Mandatory</i> No ▼
<i>Ask User</i> Yes ▼	<i>Value To Use</i> 	<i>New Group Title</i> 	<i>Separate List Tag</i>

The following fields can be entered:

Field Reference

Here can you select an existing, previously created field in this record, or in a sub-record.

New Field Title

Alternatively, just enter in a new field title here. If the field doesn't exist, it will be created for you automatically when the stage is created, and it'll then be available to select as a 'Field Reference'. Also, the next time you view this entry, you'll notice that the field will be selected in the 'Field Reference' automatically.

After creating the stage, you can also click on the Field Reference link to go to the actual field definition, where you will find a multitude of additional options to modify the behavior of the field. See section 6.6 for more information.

Tip: The ability to create the data model as you are entering the workflow saves a lot of time and makes creating workflow that much more efficient.

New Field Type

If the previous entry was used to create a new field, then you can specify the data type (that is, the type of data it will hold) using this dropdown. You can review the different data types in section 6.6.3.

Is Mandatory

Specifies whether this field will be required to be entered by the user. If this is set and the user does not enter this field, an error will be displayed.

Ask User

Specifies whether this field should be included in a form so that the user can enter the value. If not, then the next field will be used to seed the value for the field, instead of it being asked of the user.

Value To Use

The value to assign to this field if 'Ask User' is set to No.

New Group Title

Allows you to specify a field group title for this field, starting a new field group.

Separate List Tag

Allows you to record this field value in a sub-record. This is a powerful feature and is discussed in more detail below.

7.2.7. Making Sub-Lists

Normally, when the workflow type is set to 'Creates New Item', the workflow system will create just one record. Then each stage can fill-in different fields in that record as the workflow progresses.

However there are times when you want to record the same field more than once, and want to keep a record of each value that was entered. To do this, we introduced the 'Separate List Tag' field.

By entering a name in the 'Separate List Tag' field, you are telling the system to store the value for this field in a sub-record of that name. You can store multiple fields in that sub-record. Each time the stage is reached (including when it is jumped back-to) a new record will be added to that sub-record list, and the values transferred into that entry.

Example: Adding Notes

Let's look at the previous workflow where we were dealing with a job applicant. At some point in the workflow we are arranging and performing interviews. Let's say we wanted to record some notes for that interview.

Now that interview can repeat 2 or 3 times, for when we want a second or third interview. Normally, if we just added a field called 'Notes' and set that, each time we interview someone that field will be overwritten. Rather, we want to add those notes to a list.

So, to do this, we add a field called Result, but also specify a separate list tag of 'Interview', like this:

The screenshot shows a configuration form for a new field. The fields are arranged in a grid:

- Field Reference:** A text box with a dropdown arrow.
- New Field Title:** A text box containing the word "Result".
- New Field Type:** A dropdown menu showing "Text (Few Words)".
- Is Mandatory:** A dropdown menu showing "No".
- Ask User:** A dropdown menu showing "Yes".
- Value To Use:** A text box.
- New Group Title:** A text box.
- Separate List Tag:** A text box containing the word "Interview".

A "Create" button is located at the bottom right of the form.

Figure 31: Specifying the separate list tag

Now, when we publish the application and come to enter in the details of the interview, in our workflow form we'll see a simple field called 'Result':

The screenshot shows a form titled "Interview Log" with a dropdown arrow. Below the title is a text area labeled "Result" containing the following text:

The interview went very well. He had great communication skills and answered all of the interview questions correctly and was very enthusiastic about our products. I definitely recommend him.

A "Create" button is located at the bottom right of the form.

But once the record is created, it puts this value into a sub-list of the record:

The screenshot displays an Applicant record interface. At the top, a status bar indicates 'Interview Confirmed' and 'Yes'. Below this, a 'Dates' section shows 'Last Modified' as '5/27/2011 11:39:48 PM' and 'Last Modified By' as 'admin'. The 'Created' date is '5/27/2011 4:32:05 PM' and 'Created By' is 'admin'. A navigation bar includes links for 'Offer History', 'Interview' (which is active), 'Audit Trail', 'Comments', and 'Report Subscriptions'. To the right of the navigation bar is a 'Create New' link and an 'iCal' button. Below the navigation bar, there are tabs for 'Interview' and 'Calendar'. The main content area shows a table with the following data:

Interview Date	Interviewer	Result
5/26/2011	John	The interview went very well. He had great communi...

Figure 32: Showing Applicant record, with the Interview sub-list at the bottom

There is no limit to the number of sub-lists that can be created under a record. And even those sub-lists can themselves have sub-lists, and so on.

You can also select a record field, instead of a simple field, in the 'Field Reference' list and this will add a link to a sub-form in your workflow form. This will let the user fill out the entire record, and that entire record will be added to the sub-list.

7.2.8. Publishing the Application

To see your workflow application, you'll need to click Publish. The 'Publish Application' link is available from anywhere in dbFramework or WorkflowFirst in the 'What Next?' list. Once you click that (and click OK) it will create your application for you, and give you a link you can click to see the application.

You'll usually find the task that starts the workflow in the 'What Next' list.

7.2.9. Adding Script

If the user decision making and conditional decision making is not enough, you can also add script to any stage. This script will be executed once the stage is complete.

Scripts can be used to make certain calculations or decisions that are hard to describe in the flowchart design. They can also be used to start secondary, parallel workflows by creating records in other parts of the database that have workflow associated with them.

In the script, the following variables are preset and available for use:

#input

The #input variable holds the current record associated with this stage. It can be modified and then updated in the database using the save command.

&path

The &path variable holds the path to the current location in the database.

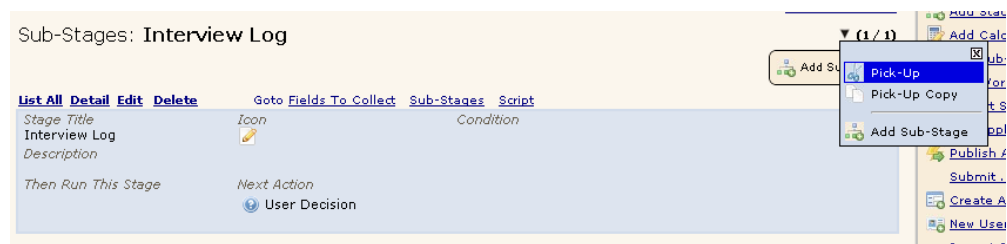
For more information on DbfScript, see section 20.

7.2.10. Deleting or Moving a Stage

At some point you may want to delete a stage. Keep in mind that when you delete a workflow stage you are also deleting all the sub-stages below it, along with all the field references. If you want to salvage some of the sub-stages, you should first **move** those stages to a different branch that won't be deleted.

To delete a stage, click the title of the stage in the flowchart diagram. This will take you to the stage record. Then click the 'Delete' link on the far left.

To move a stage, use the dropdown clipboard menu:



Select 'Pick-Up' to move the item, or 'Pick-Up Copy' to copy the stage to another area. Once it is picked up, you should go to another area of the workflow and drop it there. To do this, follow the procedure to add a sub-stage, but click the dropdown arrow in the top right corner and select 'Drop In List'. Then click the back arrow to see the changes.

7.3. Workflow Features

7.3.1. Combining Workflows

There's no limit to how many workflows you can define on a particular record, or how many types of records you can create in your data model. This opens the possibility for modeling vastly complex systems using dbFramework and WorkflowFirst.

It's therefore possible that you will need to combine two or more workflows. By combining workflows I am referring to the ability for one workflow to spawn another, secondary workflow. It would then either wait for that workflow to complete before resuming, or will continue in parallel.

There are a few ways you can do this.

The first involves spawning a separate workflow. The original workflow will continue running in parallel with the secondary workflow. By using the 'Separate List Tag' feature to create a sub-list, you can then go to the sub-list field and add a new workflow there. The workflow type will be set to 'Updates Existing Record'. This means that as soon as the record in the sub-list is created, that workflow will be started.

The second method is to suspend the workflow at a certain point until a specific condition is met, such as a field value being set. An external process or workflow would then set that value when it completes, using a simple script, which would then cause the workflow to resume. You can utilize the 'Condition' field in the workflow stage to accomplish this.

7.3.2. Notifications and Alerts

To make designing the workflow easier, the email notifications in WorkflowFirst are handled in the resultant application, rather than in the designer. When the workflow is created it provides an area in the resultant application where users can sign-up to receive notifications for particular stages of a workflow. This works much better because it means the application design doesn't have to be changed when the users participating in the workflow changes.

The 'Notifications' is added automatically as a drill-down from the parent record of the record where the workflow is operating. If the record is at the top level of an application tab, the notifications area will be added in the Configuration tab.

List All	Detail	Edit	Delete	Goto	Skills	Applicants	Notifications	Audit Trail	Report Subscriptions
<i>Title</i>				<i>Filled</i>					
Accounting Clerk				No					
<i>Notes</i>									

Figure 33: A record showing the 'Notifications' goto item (drilldown).

Once you add the notification, you'll first get to select the user that will be subscribing to the notification, and in the 'Event' list you'll see a list of workflow stages, as shown below.

Each stage in the workflow is shown as an event to subscribe to. This means, for example, if you were the CFO you may want to be notified of a 'Make Offer' stage so you can monitor the amount being offered to make sure it fits into the budget.

More likely you would be the head of the department who would be responsible for making the offer itself.

For more information no Notifications, please refer to section 8.6.

The screenshot shows a 'Notifications' window with a 'User' field containing 'jwood' and an 'Event' dropdown menu. The dropdown menu is open, displaying a list of workflow stages as events. The 'Make Offer' option is highlighted. A 'Create' button is visible to the right of the dropdown. The background of the window is light beige with a blue header bar.

User	Event
jwood	Make Offer

Event options in dropdown:

- Another Interview
- Accepted
- Counter Offer
- Rejected Offer
- Make Offer**
- Rejected
- Interview Log
- No Show
- Confirmed Date
- Reschedule Interview
- Suggested Date
- Scheduling Interview
- Pass Up
- Passed Screening
- Rejected Screen
- Make Application

Figure 34: Notification entry, showing the workflow stages as events

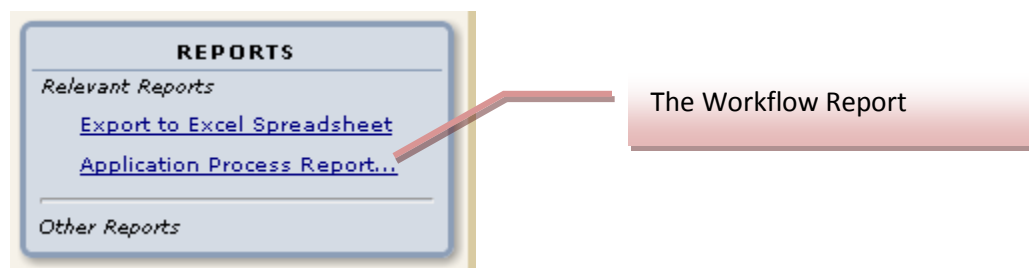
Note: You will need to define a user for each person subscribing to the workflow stages notifications. Additional licenses may be required to define those users.

7.3.3. Workflow Reports

7.3.3.1. Workflow Status Report

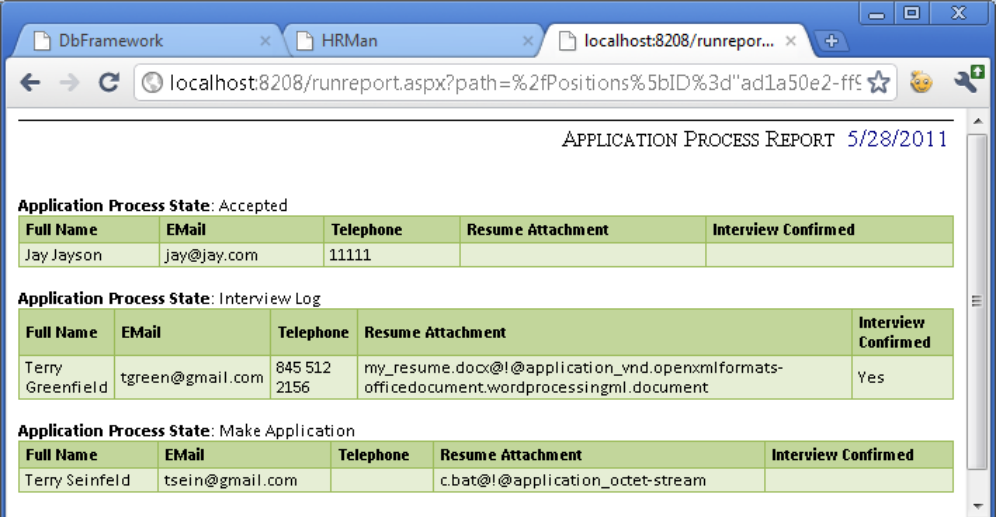
If you selected 'Create Report' when creating the workflow record, the system will automatically create a report for you that lists all of the outstanding tasks grouped by their workflow stage.

In the resultant application, the report will be available in the 'Reports' section on the right hand side of the screen. It will only be available when you are on the parent record of where the workflow is operating. If the workflow is operating on a top-level type, it will be available from anywhere.



The report will be named based on the name you gave to the workflow, followed by a 'Report' suffix.

When you go to run the report you will be given a filter form which you can optionally complete. The form lets you filter only the selected workflow stages in your report. If you just click create without selecting a stage, it will include all of them.



APPLICATION PROCESS REPORT 5/28/2011

Application Process State: Accepted

Full Name	EEmail	Telephone	Resume Attachment	Interview Confirmed
Jay Jayson	jay@jay.com	11111		

Application Process State: Interview Log

Full Name	EEmail	Telephone	Resume Attachment	Interview Confirmed
Terry Greenfield	tgreen@gmail.com	845 512 2156	my_resume.docx@!@application_vnd.openxmlformats-officedocument.wordprocessingml.document	Yes

Application Process State: Make Application

Full Name	EEmail	Telephone	Resume Attachment	Interview Confirmed
Terry Seinfeld	tsein@gmail.com		c.bat@!@application_octet-stream	

The first seven (summary level) fields in the record will be included in the report, and each record will be grouped by their current stage.

This report can also be subscribed to, so that it is sent to your inbox each morning, for example. We'll discuss that in the next section.

7.3.3.2. Tickler Reports

If any field you create in your workflow stage has a data type of 'Date (Due Date)' the system will automatically create something called a 'tickler report'. This report will include all a list of all the records that are upcoming or just passed. You can subscribe to this report as an easy way to get a list of upcoming tasks, for example, or to track any tasks that have passed. For more information on tickler reports, see section 9.3.

7.3.4. Workflow Report Subscriptions

The workflow report is automatically set up for subscriptions. This allows any user to subscribe to have this report generated and emailed to them automatically at a specified frequency, and at a specified time.

Note: To subscribe to the report you will need a user record for your login. Defining user records may require additional licenses.

To enable the user or administrator to set up these subscriptions, a special area is added automatically in the parent of the record on which the workflow is operating. If the workflow is operating on a top-level record, the report subscriptions list will be maintained in the Configuration record, on the Configuration tab.

For more information about report subscriptions, please see section 9.5.

7.3.5. Custom Reports

Because the workflow operates on a record in the database defined with a regular data model, all of the features available to dbFramework and WorkflowFirst applications are also available to these records. One of the more powerful features is the reporting system.

You can create quick, ad-hoc reports using a simple point-and-click interface. These can include hierarchical sections, graphs and a variety of sophisticated layout structures. Alternatively you can utilize the custom reporting feature to design reports using either HTML or Microsoft Word, providing an unlimited degree of customization.

For more information on creating reports, see section 9.

7.3.6. Converging Workflow Paths

Sometimes you want the user to select all sub-stages before continuing, instead of just one. This is called converging paths.

To do this you will need to have each sub-stage set a field value (in the 'Fields to Collect' you specify a field, but place a value to set rather than asking the user). You will also have one sub-stage called a convergence point.

The convergent point will need to have a 'Condition' set that checks that all of those fields have been set before it will be displayed. This stage will remain invisible until each of those stages have been visited.

The other stages will have to be set to return to the parent stage every time they are executed. Also, to have them disappear when they are run, you should have a condition on that stage that checks the field hasn't been set.

7.3.7. Integrating Approval Workflow

A very common workflow situation is where a transaction must be approved by a certain group or individual. While you could model this using the workflow designer, WorkflowFirst and dbFramework make this even easier by supporting approval workflow as part of its security system.

This means that, as well as disallowing someone the rights to modify a certain area of the database, you can also require approval before that person can modify a certain part of the database. A specific user group signs up to govern changes in that area, and they will be emailed for approval before the change will be committed. All of this is handled automatically by the system, all you have to do is add the permissions.

Because workflow involves collecting information from the user and setting fields, these same permissions can be applied to fields in the record on which the workflow is operating. This allows you to easily combine approval workflow with your process workflow.

For example, in the above example of a process workflow, in the 'Make Offer' stage, you could set up a permission so that the offer amount field cannot be set to a value larger than \$100,000 without approval from the CFO. The amount and the user group requiring approval would both be configured in the application after it is published.

For more information on setting up approval workflow, see section 11.2.

7.3.8. External Initiators

Workflow is often a company-wide operation and therefore often involves more than one system. The ability to integrate data and triggers from external systems is critical where business process automation and efficiency is priority.

WorkflowFirst and dbFramework provide many ways to integrate third party systems.

The workflow stage is controlled by a field that is added when the application is published. The field is named based on the workflow name (with spaces removed) followed by 'State'. So, for example, if I named my workflow 'Ticket System', then the workflow state will be held in a field called 'TicketSystemState'.

The value of this field will be the name of the stage, with spaces removed.

This makes it very easy to initiate a workflow or to control the stage of a workflow from an external system. The system can simply utilize the XML interface described in section 21.6, for example, and update a record to set the state to a particular value. This would then send out any email notifications and start the workflow from that stage.

There are many other ways to interact with workflow, and these are covered in depth in section 21.

7.3.9. Mobile Access

As with any application created using dbFramework and WorkflowFirst, when the application is viewed on a mobile phone a special rendition of the application is provided that is simpler and suited to the size constraints of typical cellular phones.

7.3.10. Customizing Forms

One of the major benefits of using WorkflowFirst or dbFramework is that it creates forms for you, removing the necessity to spend time and resources with designing.

However sometimes you may want more control over how the forms are designed. WorkflowFirst and dbFramework provide several ways to customize the look and feel of your application, including providing HTML to replace the default form layout, providing HTML components for a specific field, or even re-skinning the entire application.

For more information on creating custom designs, refer to section 10.

7.4. Workflow Examples

7.4.1. Purchase Request

7.4.1.1. Requirements

The purchase request begins with an employee filling out a form specifying what they want to purchase, the approximate amount it will cost, the suggested vendor, and the project it is for. They will also provide details of how the product will be used, and how long the product will last before it will need to be upgraded or replaced.

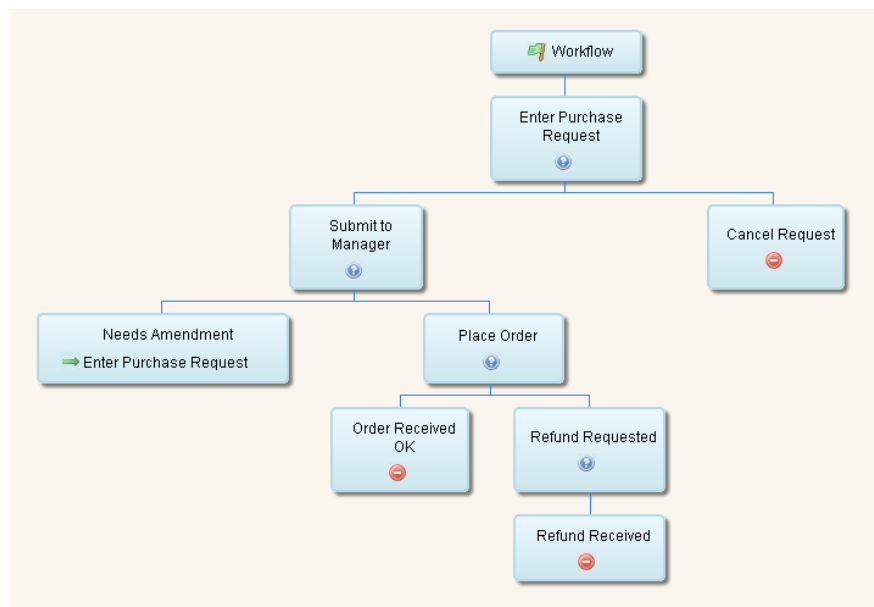
The request then goes to their manager who will approve the necessity of the product for the given project. They may accept it as-is, or they may reject it either

as being unnecessary, or that the chosen vendor is too expensive or otherwise unsuitable. At this point the originator will have to amend the request or give-up.

Once accepted, if the amount is greater than \$5,000 then the CFO needs to approve the purchase. It will then go to the accounting department who will place the order.

Once the order is received, the request will be closed and marked as fulfilled. Alternatively, if the order is never received then it will be marked as refunded.

7.4.1.2. Flowchart



7.4.1.3. Special Features

For the most part this is a simple flowchart. However the special case for the \$5,000 or over CFO approval will be implemented using the approval workflow feature, and so is not included in this flowchart. This works better because it means that the user(s) granting the approval, and the threshold amount, can both be defined at runtime without having to change the application itself.

7.4.2. Help Desk

7.4.2.1. Requirements

The Help Desk workflow starts with a customer having an issue they need to resolve. They provide information on the problem, such as the product they're

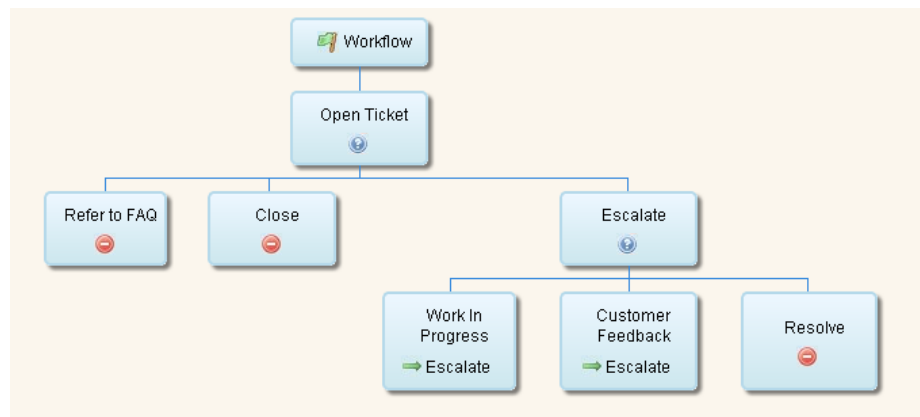
using, the version number, the steps to reproduce the problem, and also any identification of a support contract they have established. This is called a ticket.

The ticket is then moved to a primary support responder. They will screen the issue. If it is trivial, or can easily be found in a help file or FAQ, they will respond immediately and close the ticket. Otherwise they will escalate the ticket.

Once escalated, a technical expert will respond. They will have the option of asking the customer for more information, and the customer can respond. This can happen any number of times.

Finally the technical expert will mark the issue as resolved and it will be closed.

7.4.2.2. Flowchart



7.4.2.3. Special Features

Once escalated, the technical expert can effectively correspond with the customer using workflow. This consists of two workflow stages, 'Request Info' and 'Customer Feedback', both of which take the stage back to 'Escalate' at the end. This is effectively a loop. To break the loop the technical expert will select 'Resolved'.

7.4.3. Product Release

7.4.3.1. Requirements

At the end of product development, a number of stages are executed before the product can be released to the customers.

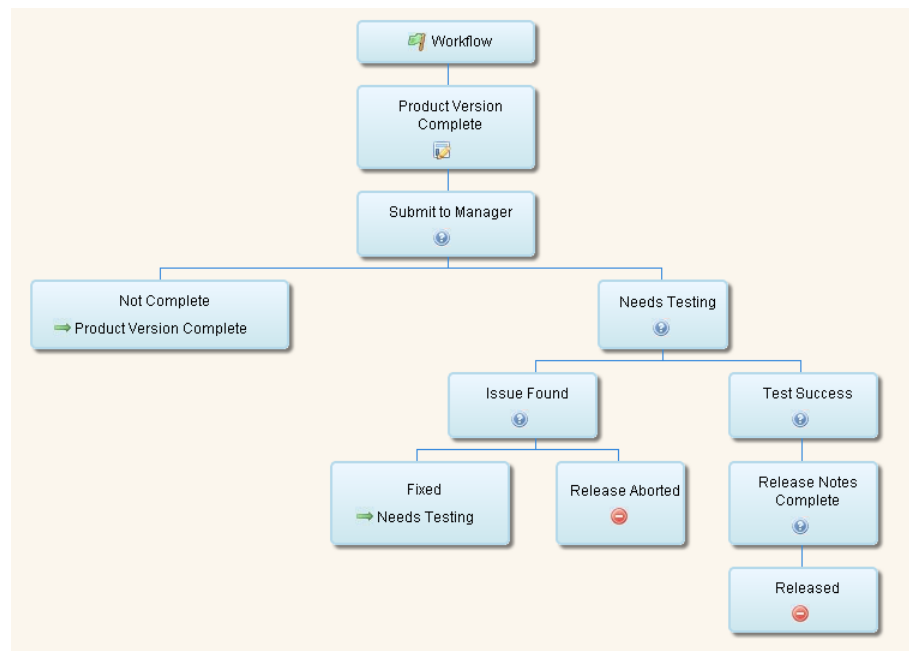
The first stage is for the manager to confirm that the necessary task list related to features in this release has been completed. The manager will enter those tasks into the workflow item so it can be passed on.

Once confirmed, a notification will go to Quality Assurance (QA) to start testing. They will be provided this with a deadline. They must test that all of the new features work as expected, and also that any existing functionality hasn't been broken. They will also have to confirm that the documentation is complete and meets the required standards.

If any problem is found, the work will return to development for completion, with a note explaining what was wrong.

Once QA are satisfied that the features are complete and the product is of an acceptable quality, a product release document will be generated. This will be sent to the marketing, sales, support and the deployment departments so they can confirm that they fully understand the new features and how to convey them to prospective, new or existing clients.

7.4.3.2. Flowchart



7.4.3.3. Special Features

If QA find a problem with the product, it is sent back to development. This effectively sends the stage back to the first stage, where it can be submitted to the manager and QA all over again. This can repeat any number of times. The loop is broken only when QA accept the product.

In the final stage, where the product release notes are submitted, we do not explicitly stage which departments must receive this. Rather this is set up through notification subscriptions. A representative from each department will subscribe to receive notification of when the product release is made.

7.4.4. Credit Application (Advanced)

7.4.4.1. Requirements

The credit application process starts with an entry by a potential borrower, the applicant, who wants to borrow some money.

The sales rep then receives that application and checks to make sure all of the required information is provided and that it calculates correctly. If not, it will be sent back to the applicant.

Once acceptable, the application is passed on to the credit rep. They perform their own validation and work with the applicant to ensure it is complete. They will then order a credit report. Once received they will determine the security requirement. They will then pass that requirement back to the applicant, who must respond with the necessary documentation.

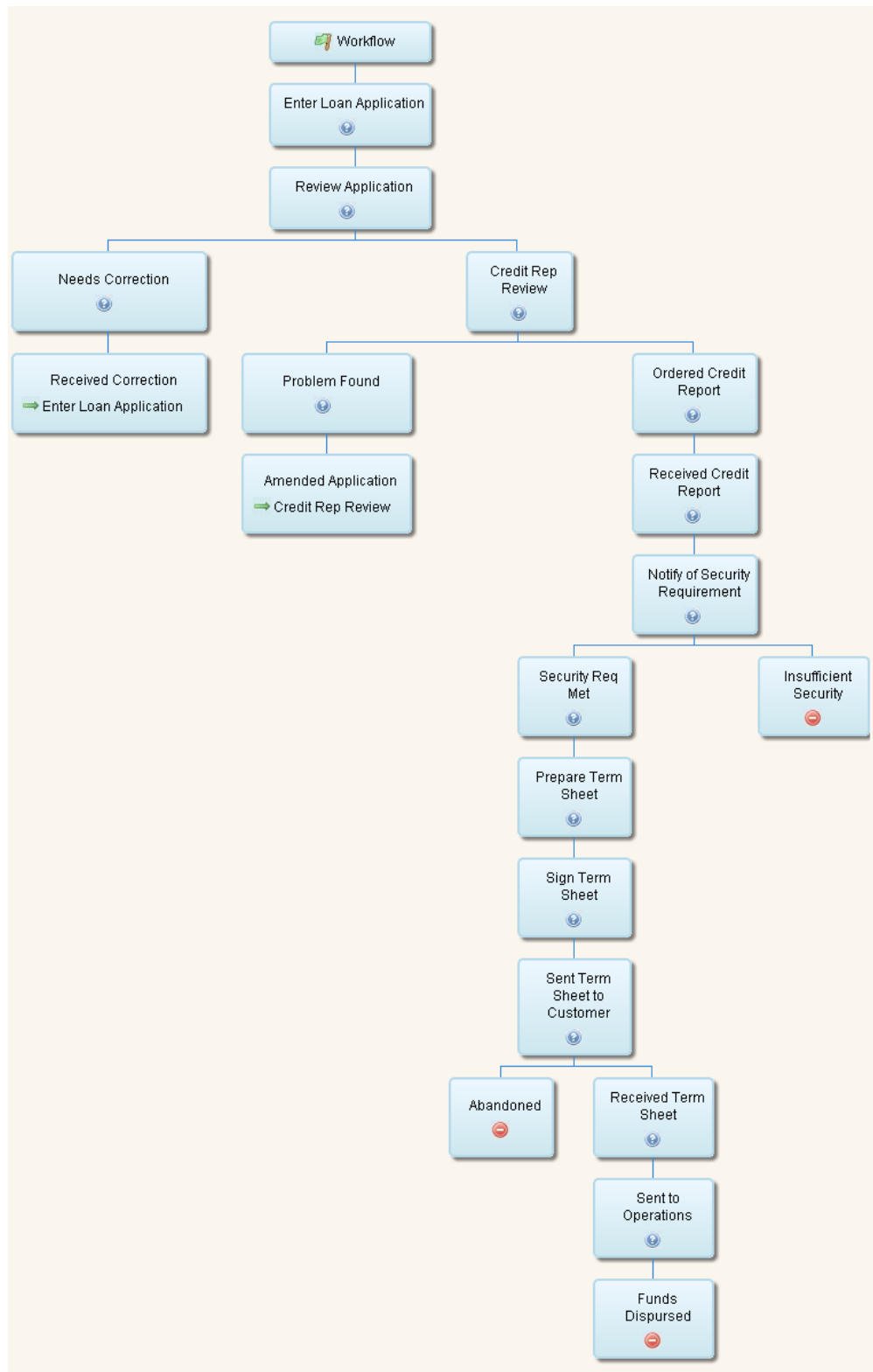
The documentation will then be reviewed, and requests for more information or documentation will loop back to the applicant.

At this point the credit rep determines the credit approval. If declined, the sales rep and the applicant will be notified and the application closed.

If accepted, a formal notice will be sent to the applicant (through the word processing clerk). This will be signed by the credit rep before being sent to the applicant and to the sales rep. This must be signed by the applicant.

Once received back from the applicant, a copy will be sent to the loan operations department so that the necessary funds can be dispatched.

7.4.4.2. Flowchart



8. Business Logic

8.1. Workflow

Probably the most common type of business logic you'll define will be in the form of 'Workflow'. Workflow refers to business procedures that are mapped out into a flowchart. Each element of the workflow is referred to as a 'stage' and can perform actions, collect information from the user and allow the user to make decisions.

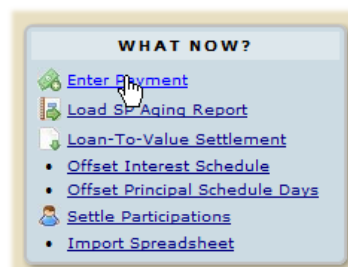
There are several ways to implement workflow in dbFramework, but the easiest is probably using the workflow designer that dbFramework inherits from the WorkflowFirst product. With this designer you can graphically create the flow chart from within your browser. You simply have to enter the workflow stages, and dbFramework will create the data model and the actions for you.

For a detailed guide to creating workflow, please refer to section 7.

8.2. Actions

For more advanced functionality that involved making complex decisions based on data, or running calculation-processes on data, you may want to create an action.

These actions usually appear as tasks in your dbFramework application:



But they can also be used to handle events, run as periodic scheduled jobs, be used to generate the data to display in a report or to control the layout of a screen.

There are two kinds of actions that can be created in dbFramework: Scripts and plug-ins.

Scripts can be created from within the web browser, and are written using the DbfScript language. Plug-ins allow you to create actions using a third-party .Net

compatible language such as C#. Ultimately both of them are used in the same way, however.

Whereas DbfScript allows you to create relatively short procedure very quickly, the plug-in code allows you to use more sophisticated development tools such as Microsoft Visual Studio to create your actions. These benefit from having integrated debuggers, which make it easier to track down problems, and also allow you to use third-party tools and libraries to enhance your actions further. Using a language such as C# allows you to structure your actions using object-orientation, letting you re-use code and create abstractions that are difficult to create using a scripting language. Visual Studio also provides features to re-factor code that gets too complex, and provides many compile-time errors that will help create more reliable code. More information on using the .Net SDK to create plug-in DLLs can be found in section 21.

Saying that, there is nothing as convenient as typing a few lines into the web-browser interface of dbFramework to create a new action. The turnaround can be in seconds. As well as being a powerful language, DbfScript employs a special syntax that makes working with a hierarchical database more intuitive. This is the primary reason that a new language was created, rather than using an existing language. DbfScript provides a wealth of functionality allowing you to query data from anywhere in the database. A comprehensive list of functions and keywords can be found in section 19.

8.2.1. Action Parameters

Actions can either run with or without any parameters. The parameters to the action input are defined the same way you create a data model type. And, in the same way, dbFramework will automatically create the input form layout, or use any custom-views you have designed. The automatically-generated form will have the same features allowing you to have complex, nested records, links to other parts of the database and so on.

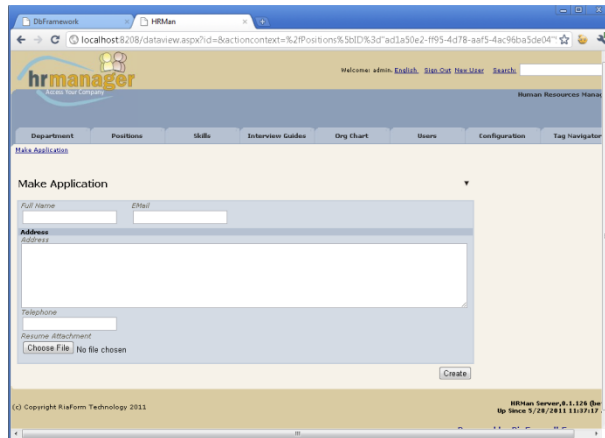


Figure 35: Example of an action parameter screen

8.2.2. Creating An Action

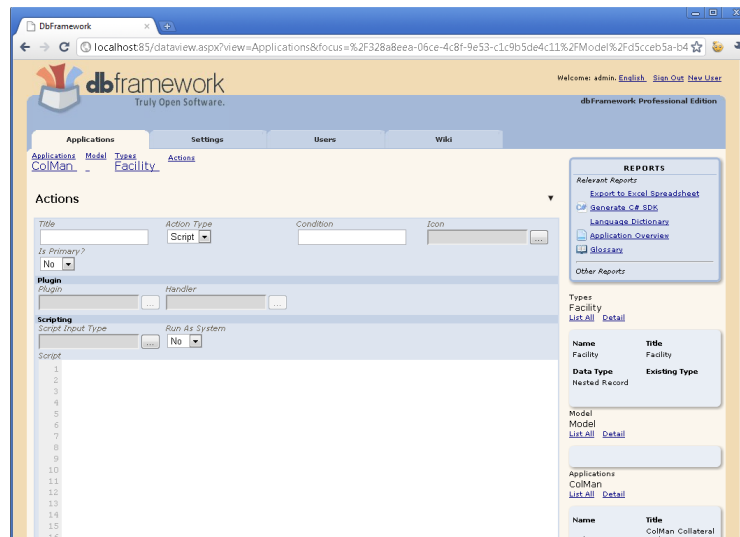
There are two places that actions can be added: Against a specific type, or globally. Global actions are available no matter what tab you are on. Type-based actions are available only when you are viewing a record that is of that type.

You will notice this in dbFramework: The action “Add Calculated Field” is available when you’re viewing a list of fields, but if you are on the Application record then this action is not displayed. Type-based actions are referred to as *contextual* actions.

To create a global action you need to go to the Application record, and then click on ‘Actions’.

To create a contextual (type-based) action, first find the field for which you wish to have the action visible, and then click ‘Actions’.

Then click ‘Create New’ to creation the action. The screen will look something like this:



You'll have to enter the following fields:

Title

This is the title of the action that will be displayed in the “What Now?” list.

Action Type

The action type is where you specify whether this will be a script or a plug-in. The default is script.

Plugin

If the action type is set to ‘Plugin’, then you select the plugin to use here. If there is nothing in the list then you will probably need to submit the plugin first (using the ‘Submit .Net Plugin’ action).

Handler

If the action type is set to ‘Plugin’ and you have selected a plugin, this is where you select the handler defined in the plugin. This will relate to a function you have defined in the plugin DLL.

Script Input Type

Lets you select an existing type within the data model to use for the action parameters.

If you want to create the action input private to this action, then create the action and then use the “Script Nested Type” drilldown to create the parameters for the action.

Run As System

If this is set, the action will run without any permission restrictions. If it is not set, then the permission restrictions of the user invoking the action will be applied.

Icon

Optionally select an icon to display next to the action title wherever it is displayed.

Is Primary?

Indicates if this action will be displayed more prominently on the screen. When this is set to 'Yes', the action will also display as a button above the record, making it easier to access. This is especially useful for actions that are invoked frequently.

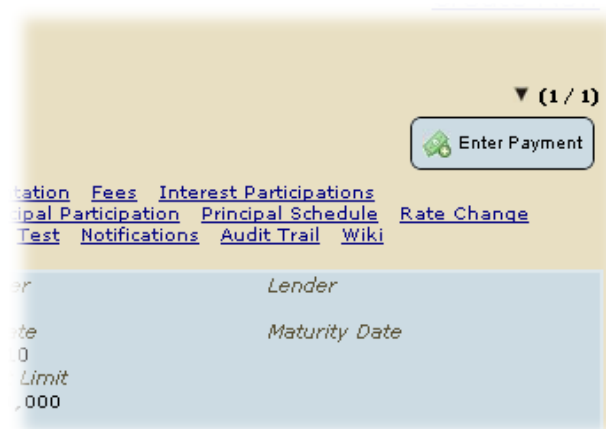


Figure 36: Showing the 'Enter Payment' action as a button by setting 'Is Primary'

Script

This is the code editor where you enter the actual script. This shouldn't be used if a plug-in was selected.

For more information on creating a DbfScript, please refer to section 19.

When the script is invoked, two special variables will be passed in to the script:

- **#input**
This holds the action input (parameters) if any was provided.
- **&path**
This holds the path to the current record where this action was invoked.
- **&root**
The root variable is always available and lets you access any path in the database from the root.

For example, if the parameters to the action included a “PaymentDate” field, you could access that in a formula such as:

```
set #record/Date = #input/PaymentDate
```

And, as another example, you can use the path variable to load that part of the database, or a related part.

```
var #payment = load(&path)
```

8.2.3. Example Scripts

8.2.3.1. Hello World

It’s quite common when new programming languages or environments are introduced to demonstrate how one uses the language to output ‘Hello World’ on the screen. The DbfScript ‘Hello World’ application is rather simple. Create an action called ‘Hello’ with a script and in the script put the following statement:

```
message "Hello", "Hello World"
```

After creating the action, publish the application and you’ll see the new action in the ‘What Now?’ list:



If you click that ‘Hello’ action you should see the following appear:



The ‘message’ keyword is useful for outputting information to the user when the action is finished.

8.2.3.2. *Modifying a Record*

DbfScript has special support to make it easy to access parts of the database seamlessly. The following script loads the record at the path, updates a field, and saves the record again.

```
var #record = load (&path)
set #record/State = "Complete"
save #record
```

8.2.3.3. *Looping Through Records*

Often you may wish to load a drilldown from the current record which contains a list of records. You would use the loop keyword to achieve this:

```
var #list = load (&path/Invoices)
loop through #list as #record
  if (#record/Overdue) then
    var #overdue = #overdue + 1
  end if
end loop
message "Overdue", "You have " + #overdue +
  " invoices overdue."
```

In this example we first load all of the invoices underneath the current record. We then loop through them. We then check each record if it is set as overdue. If it is we increment a counter. At the end of the function we output to the user the number of overdue invoices.

This example also demonstrates using conditional statements: The IF statement is used to make a decision and the statements between the IF and the END IF are only executed if that condition is true.

8.3. Calculated Fields

8.3.1. *Creating Calculations*

When defining a field in a type that field can be designated as being entered by the user or calculated. If the field is calculated it means it is not shown to the user when the record is being edited. In this case the value is either set through actions (through tasks that are invoked or events being raised), or it is set by a formula on the field itself.

To set a formula on a field you enter a simple DbfScript into the 'Expression' entry box on the field definition.

Simple DbfScript gives you the ability to use everything in DbfScript except program control statements such as IF, LOOP, WHILE and EXIT SCRIPT. The expression can optionally start with an equals sign. It also looks like a formula, much like formulae in Excel, and so the formula should always be in a format that can be evaluated to a value that can be assigned to something.

The DbfScript is executed with the variables #input and &path provided, and these are used to access the other fields in the record.

For example, if you had two fields in the record called SubTotal and SalesTax, then you could add a field called Total with a formula such as:

```
=#input/SubTotal + #input/SalesTax
```

You can also reference drilldown ('Go To items') in aggregate functions, such as putting:

```
=Count (&path/Orders)
```

And this will automatically recalculate if the number of Orders in the drilldown changes.

In addition, you can access sub-fields of fields that hold *links* to other records. dbFramework will automatically retrieve the linked record and allow access to fields within that record.

For example:

```
#input/Price * #input/Currency/Rate
```

This formula multiplies the 'Price' field by the rate of the selected currency. Because the currency is selected in a dropdown field called 'Currency', by referring to '/Currency/Rate', the system knows that this field must be accessed via another record, and so it loads the selected currency record, takes the 'Rate' value from the record, and uses that value in the calculation.

And what's more, this creates a 'live link' between the selected value and the result: If the rate changes, then this formula will automatically be recalculated to reflect the new value. This kind of dynamic updating saves a lot of time that would otherwise have to be spent implementing complex workflow programming to achieve the same result. For the most part dbFramework hides this from you and seamlessly connects these implicit links in formula much the same way as Excel does.

Although you cannot use an IF statement in an expression, you can achieve a conditional statement by using the IIf function (The letter 'I' followed by 'If'). This takes three parameters. The first is the condition, which must be true or false. The second is the value to use if the condition is true, and the third is the parameter to use if the condition is false. For example:

```
=IIf(#input/State == "Complete", 100, #input/Progress)
```

Note: Unlike a regular IF statement, the unused value is still evaluated even though the result is not used.

You can also access previous records in the formula, which you may do to create a running total, for example. To do this you access the variable #previous, which provides access to all of the fields of the previous record.

For example, to keep a running total of PaymentAmount, you might use a formula such as:

```
=#input/PaymentAmount + #previous/Total
```

This also means that rows will be recalculated if records are deleted or inserted, as the #previous value would change.

Keep in mind, however, that #previous always refers to the previous row based on the default sort order for that type. If the sort order changes, then the behavior of this formula will no longer work as expected.

Any fields that reference themselves in a previous row will invoke special processing when field values are changed. Because a change to one record can affect all of the subsequent records, known as a 'waterfall effect', updating a single record can have a significant performance impact. Care should be taken to take this into consideration when writing business logic.

Because of this overhead it may be preferable to move the waterfall effect of the field's calculation into DbfScript, in an event handler that runs on idle, to more efficiently calculate changes to whole lists that occur when one record changes. Because these can be processed on idle, and throttled so they only run once for multiple updates, your application will end up performing significantly less work and will reduce the overall impact of updating the table.

On another note, it's important that the formula used in the expression cannot produce an error. If the formula produces an error then the original action or update will not complete and an error will be displayed to the user. Because the error may not make much sense to the user, you should be careful to ensure the formula will not fail.

8.3.2. Recalculating

The calculated field will automatically recalculate when the row is created or if any dependent fields change. As mentioned in the previous section, even dependents that are referenced through *linked* records will also have their changes automatically reflected in the calculated field.

However, sometimes you may add a calculated field *after* data has been added to the list. If you do this then the value of the calculated field for the *existing* rows will be empty.

You can remedy this quite easily by forcing dbFramework to recalculate the field for all of the existing rows.

To do this you need to write a little bit of DbfScript.

Firstly, go to the list that contains the existing records with the new calculation. Keep in mind the name of one of the fields that is used in the calculation.

Then go to the address bar, and change the 'dataview' part of the URL to be 'admintools'. This will bring up a page with a large area for entering in ad-hoc scripts.

Note: You will need to be logged-in using an administrator account to access the admin tools.

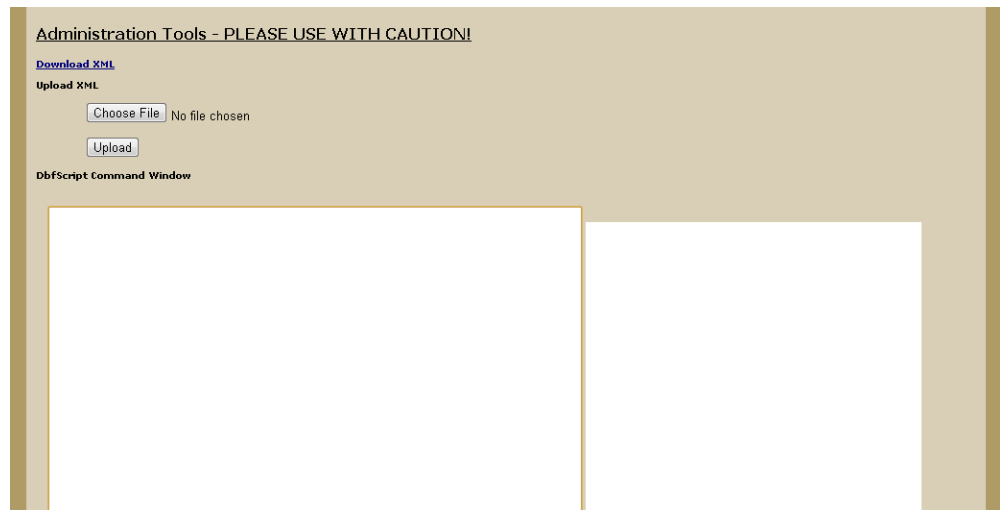


Figure 37: The administrator tools section lets you enter ad-hoc scripts

In the left-side script window, enter a script such as this:

```
loop through load(&path) as #record
  set #record/DependentField = #record/DependentField
  save #record
end loop
```

...but replace 'DependentField' with the name of the field that the calculation depends on.

This script runs through all of the rows in the list, and saves a pretend change to a field that will trigger the calculation to run.

Note: If you have a significant number of rows, this script may need more time to run or otherwise the system will say it's taking too long. Just insert a 'more time' statement at the start of the script to have the system give the script more time to run. It may be a pain to have to do this, but it's useful because it stops your scripts getting stuck in an infinite loop.

8.4. Events

8.4.1. Overview

Events in dbFramework applications refer to specific consequences or special circumstances that occur on types of data in your data model. Event *handlers* are actions that are invoked when these circumstances are met.

The event-driven architecture of dbFramework is what allows it to be more than just a simple database: the ability to easily tie actions to certain events on the data makes the data intelligent. An event can be used to set the progress percentage to 100% when a task is set to complete, for example. Or an event can be used to check that the invoice being updated hasn't already been sent and so shouldn't be modified.

It's through events that you can create *business rules*.

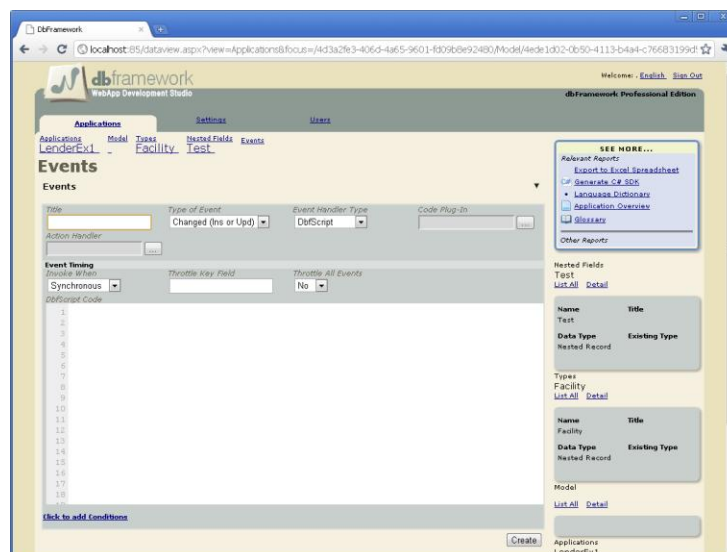
Event handlers can be added to any field in the data model, and are guaranteed to run whenever data of that particular type is changed.

If an event handler itself makes changes to the data model, those changes will also raise events, and if those make changes those will trigger events and so on. This event-oriented architecture allows for encapsulation of business logic and a simple but powerful means of functionality re-use.

As such, events provide a means to *centralize* business logic to turn your database into a true application.

8.4.2. Adding Events

To add an event, first go to the type in your data model where you wish to add the event handler. Then click on the Events drilldown. You will see a number of fields that can be entered:



Title

This is an internal name you can optionally give to the event for reference purposes.

Type Of Event

This dropdown specifies the particular event or circumstance that the handler will be invoked. The following options are available:

- **Changed (Ins or Upd)**
The event is raised when any record in this list is changed, either by being inserted or updated. A delete does not constitute a change.
- **Updated**
The event is raised when any record in the list is updated. An insert or a delete does not constitute an update.
- **Inserted**
The event is raised when a new record is inserted into the list.
- **Before Delete**
The event is raised before a record is deleted, while the record still exists in the database.
- **After Delete**
The event is raised after a record is deleted, when the record no longer exists in the database.
- **Prepare**
The event is raised as a request to set default values in a blank form used to insert a new record. This allows the default values to be calculated using business logic.
- **Field Layout Request**
The event is raised as a request to provide guidance on how the form should be laid out, in terms of field order and grouping. This allows the form layout to be calculated using business logic. See the section on controlling layout.
- **Drag and Drop**
The event is raised when one record is dropped onto another record. Note: This is currently not used in the web interface for dbFramework.
- **Validate**
The event is raised prior to changing a record in the database, allowing business logic to validate and potentially stop the change.
- **Inline Change**
The event is raised when specific entry boxes are changed in an edit form, allowing the business logic to set the values of other entry-boxes in

reaction to that change. This occurs dynamically as the form is being edited.

Event Handler Type

Like actions, the event handler can be either a script (DbfScript) or a .Net module (External Code). The default is DbfScript, and the script can be entered in the script code editor at the bottom of the form. If 'external code' is selected, the next fields are used to select the action in the .Net module to use.

Code Plug-In

If the event handler is set to external code, select the plugin to use in this dropdown.

Action Handler

If the event handler is set to external code, select the action in the plugin to use.

Invoke When

Specifies when the event handler will be invoked after the event occurs:

- **Synchronous**
The event handler will be invoked as soon as the event is raised. Any errors that occur in the event handler will stop the original action from committing.
- **Asynchronous**
The event handler will be invoked shortly after the event is raised. Any errors that occur in the event handler will NOT stop the original action from committing. The errors will only be viewable in the system log.
- **Idle**
The event handler will be invoked once system activity has settled down. Any errors that occur in the event handler will NOT stop the original action from committing. The errors will only be viewable in the system log.
- **Idle (Throttled)**
The event handler will be invoked once system activity has settled down, but the number of events raised will be limited (throttled) based on the name of a field in the path. Only one event will be raised for that path's field. See section below for more information

Throttle Key Field

As explained below, when Idle (Throttled) is selected, this field specifies the field in the path that defines the key to throttle the number of events.

Throttle All Events

If this is set, throttles all event handlers by the throttle key. Only one of the event handlers will be raised in a given time.

Note: Setting 'Throttle All Events' means you can no longer guarantee that any specific event will run. Rather only one of the events will run in a given time period. Use with caution.

Add Conditions

Event conditions allow you to specify which fields must be present in an update for the event to trigger. It's important to set conditions on events to limit what specific changes will cause your event handler to run. For example an event handler that changes the state of a task when the progress is updated should have a condition set for the field 'Progress'. That way the event will only be invoked if the Progress field is changed, and not any other field.

The reason this is so important is that, without it, you may end up with an event that invokes itself, which invokes itself, and so on – what is referred to as an infinite loop. dbFramework detects infinite loops and stops them so they don't consume all system resources and bring down the server, displaying an error explaining that the operation was stopped for security reasons. But the situation should still be avoided if at all possible.

When adding the condition you can specify the field from a list, and also whether that field needs to be present or not. If the field is set to NOT be present, then the event will only raise when that field is NOT updated.

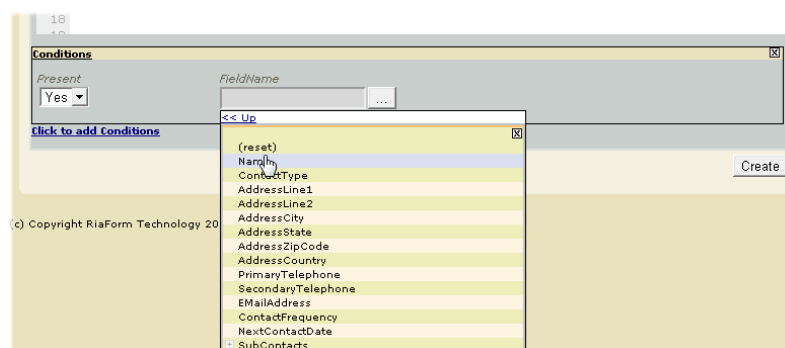


Figure 38: Screenshot of a condition being edited.

8.4.3. Event Throttling

Event throttling allows the developer to control how often an event is raised for a particular area of the database. This is particularly useful if you want to perform a complex operation that takes time to run when something updates, but the updates occur too frequently. If the event was run synchronously then it would significantly slow down the update. By throttling the updates it leaves the updates to run freely without performance impediment.

The throttling is controlled by specifying a field name that must occur in the path of the database. Up to the occurrence of that field, only one event will be raised. If the throttled field is the parent field name in the data model, then only one event will ever be raised for a number of updates in a list. Once the event is raised, another can be raised, but only 1 ever 2 seconds will ever be raised.

8.4.4. Writing Event Handler Scripts

The event handler script is entered in the code editor at the bottom of the event form. The script is written much the same as the action script (see section 8.1).

The same functions can be used, including 'message' to convey information to the user when the action raises the events completes. The 'warning' keyword can also be used to stop the original action from running until one or more warnings are accepted by the user, and 'error' will stop the original action from running and display an error to the user.

Some keywords and functions may behave differently if the event is set to run asynchronously or on idle. For asynchronous or idle event handlers, errors, warnings and messages will not be displayed to the user but will rather place an entry in the system log.

The script in the event handler will have access to two important variables:

&path

This contains a reference to the path where the event occurred.

Note: For Idle Throttled events, the path of the event that is raised is not significant. The path should be cropped to the throttle key field and any predicates or fields beyond the throttle key should not be relied upon.

#input

This contains the data related to the specific event. For a Changed or Updated event, #input will contain the fields and values of the data that was changed. For an Insert event, #input will contain all of the data of the newly inserted record. For Delete events #input will be empty.

8.4.5. Example Event Handlers

Setting a Task to Complete

The following script could be put on the Changed event of a fictitious Task record. A condition would be set on the 'State' field being present.

```
if (#input/State=="Complete") then
    set #input/Progress = 100
    save #input
end if
```

Validating an Invoice State

The following script could be put on the Changed event of a fictitious Invoice record. A condition would be set on the State field NOT being present.

```
// load all of the information in the invoice
var #invoice = load(&path)

if (#invoice/State=="Sent") then
    error "An invoice cannot be modified after it has been sent to the
customer"
end if
```

Checking for Account Limit Breach

The following script could be put on the Changed event of a fictitious Payments record in a Bank Account record. The Payment record will have a running Total (calculated field). A change in that Total will be the condition for this event. A warning will be raised if that limit is exceeded, but the user can accept the warning to allow the operation to commit.

```
// retrieve the bank account
var #account = load(Parent(&path))
// check the total against the account limit
if (#input/Total > #account/Limit) then
    warning "The total " + #input/Total + " exceeds the bank account
limit of " + #account/Limit
end if
```

8.4.6. Writing Layout Events

8.4.6.1. Overview

The Layout event allows business logic to control the layout and captions used in generating forms. While normally settings such as the order of the fields, the group names and Display Priority will allow dbFramework to determine how to layout the form, this can be overridden in script so that the layout is based on decisions that can utilize data and other conditions it determines necessary.

This may be useful, for example, if some fields are not relevant to a particular record. For example if you have a contact list, and underneath that you have addresses for different people at that company, you may want to have a different format for the addresses based on the country in which the company is located. You could use a layout script to accomplish this.

8.4.6.2. Creating a Layout Event

There are two ways you can create a layout event. The first is through writing some DbfScript, and the second is through a .Net plugin.

In both cases the event needs to return a list of SelectedColumn instances, each of which must have its column name, title and optionally have its group name set.

In DbfScript you don't have to create a SelectedColumn directly. Instead you use a special function called AddLayout.

AddLayout takes four parameters:

- The value returned from a previous call to AddLayout. Internally this is the list of 'SelectedColumn' instances. If this is the first time it has been called, pass in Nothing().
- The name of the field in the data model.
- The new title for the field.
- Optionally the group title for the field, if it's starting a new group. If this is not starting a new group, pass in Nothing().

At the end of the script, return the value returned from AddLayout.

The DbfScript is initialized with the path of where the layout is being requested. Other data from the database can be requested using that path to make business decisions about what should be in the layout.

Note: Because the layout event is invoked every time the user displays a record of this type, care should be taken to have the event execute and return quickly. Any delays in this event will slow down the user interface experience. Care should also be taken not to have any code that can raise errors. Any errors thrown during the code will result in the inaccessibility of that record.

8.4.6.3. Example

The following event returns a custom layout using DbfScript:

```
var #layout = AddLayout(Nothing(), "Name", "Facility Name", Nothing())
call AddLayout(#layout, "Amount", "Facility Amount", Nothing())
return #layout
```

8.5. Scheduled Jobs

8.5.1. Overview

Scheduled jobs are actions that run in the background in your application at a specific frequency. For example you can set up an action to run every 15 minutes. Any number of scheduled jobs can be configured. The frequency is specified in minutes.

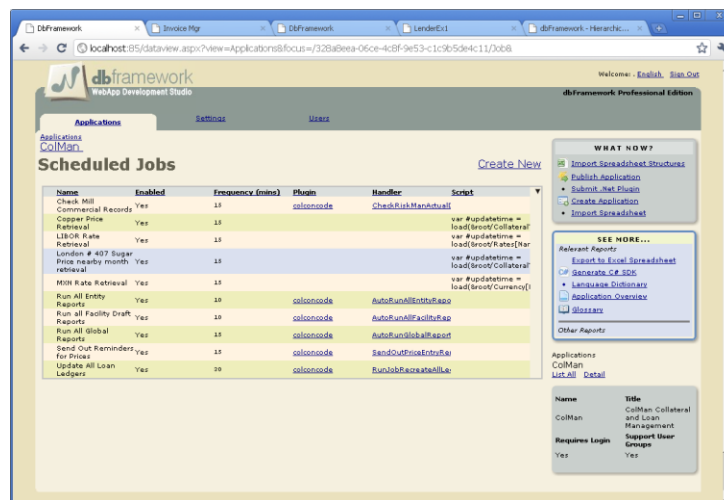


Figure 39: List of scheduled jobs in an application

Like actions and events, scheduled jobs can be either scripts or external modules such as .Net code.

Note: All scheduled jobs run once as soon as the server starts.

8.5.2. Adding a Scheduled Job

There is only one area to add scheduled jobs, and this is under the Application record, in the 'Scheduled Jobs' drilldown. To add a scheduled job click 'Create New'.

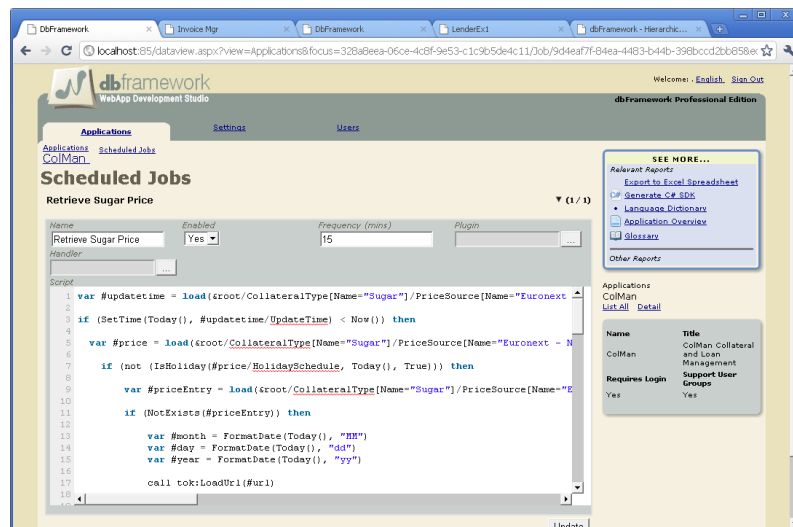


Figure 40: Example of a scheduled job script being edited.

The following fields can be entered:

Name

The name given to the scheduled job. This name is not shown to the user, but appears in the system log when the job is run or an error occurs.

Enabled

Provides a quick way to disable a job from running without having to delete the job.

Frequency (Mins)

Specifies how frequently the job will run. This refers to the number of minutes between job runs. The counter doesn't start until the job action completes, so the same job cannot run more than once at a time.

Plugin and Handler

Specifies the plugin and handler for an action in an external module. If a script is specified, the plugin will be ignored.

Script

Specifies the DbfScript for the scheduled job. If a script is specified, the plugin will be ignored.

Unlike actions and events, scheduled jobs have no variables passed in except &root that provides access to the root of the database.

Because scheduled jobs are run asynchronously, any errors that occur while the scheduled job is running will be reported in the system log. You can also find out when the scheduled job runs, and how long it took to run, by looking in the system log. See section 23.2 for more information.

8.6. Notifications

8.6.1. Overview

‘Notifications’ is a feature that provides a convenient mechanism for users to subscribe to receive notification of specific events occurring in the application. By creating a Notification, dbFramework automatically creates an area under a specific part of the database that people can use to subscribe themselves to receive a notification. The notification will be received by email that is automatically generated by dbFramework.

8.6.2. Adding a Notification

Several kinds of notifications can be created. To create a notification go to the area of the data model where the notification will be raised, and where you expect the user to subscribe. Then click on the ‘Notifications’ drilldown, and click ‘Create New’.

Notifications

Notifications ▼

Name: Title: Event: Condition:

Description:

Create

The following fields can be entered:

Name

Specifies the name of the notification. This name can be used in scripts and so should not contain spaces or start with a number or symbol.

Title

Specifies the name of the notification that will be displayed to users.

Event

Specifies the event that causes this notification.

- **Inserted**
The notification is raised when a new record is inserted.
- **Updated**
The notification is raised when an existing record is changed.
- **Changed**
The notification is raised when a new record is inserted OR if an existing record is updated.
- **Deleted**
The notification is raised when a record is deleted.
- **Script-Invoked Only**
The notification is only raised when a script invokes the notification.

Description

The description is any text that you wish to be incorporated into the email that is sent to each user that subscribes to the notification.

Once the notification has been set up, you will need to publish to see the notification area in the application.

The notification subscription area will appear as a drilldown called 'Notifications' under the area in the data model where you created the notification. In this the user can add themselves and select the event they wish to be notified of.

When an event occurs, an email will be generated and sent to the user via the email address they have set up in their user area in that application.

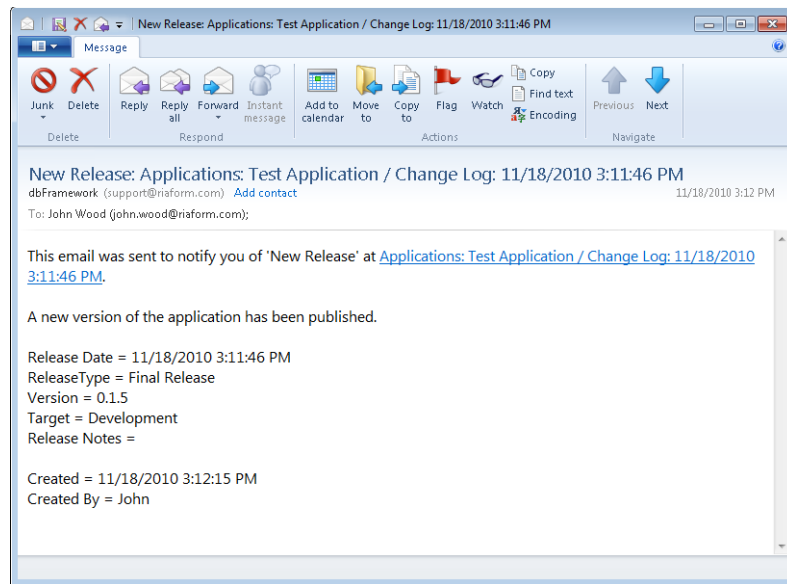


Figure 41: An example email sent from a notification

Note: The email server must be configured properly in the Configuration tab of the application for the notifications to work correctly. Please see section

8.6.3. Using Notifications with DbfScript

If the Notification was created with its event set to 'Script-Invoked Only' then the notification can only be raised through entering a command in script, either in an action or event.

This is a useful way to create custom notifications that are related to business conditions instead of simply whether a record has been updated.

To invoke a notification you should use the **notify** keyword described in section 20.5.

The following are some example of the **notify** keyword in action.

Notification of Significant Price Change

The following example allows users to subscribe to receive notification of a significant change in the price of an asset. This script would run on a Change event of the Price type:

```
// get out the last 2 prices, first is this one
var #lastPrice = GetNextRows(#input, 1)
if (Count(#lastPrice)>0) then
    set #lastPrice = ItemAt(#lastPrice, 1)
```

```

    If ((#lastPrice/Price div #input/Price) > 10) then
        notify of #input PriceChangeNotify "The price has changed more
        than 10%" at &path
    end if
end if

```

This script gets out the next row from the updating one (which, according to the default sort, should be the previous price). If there is a previous price, then it retrieves the price and compares it to the current price. If it exceeds 10% difference then it raises the notification. This will then send an email to all users who subscribed to the notification for that record.

Notification of a Void Invoice

This script runs on an action to void an invoice. The script voids the invoice and then sends out a notification.

This script would be entered on an action on an Invoice type.

```

set #input/State = "Void"
save #input
notify of #input VoidInvoice "The invoice has been voided"
    at &path

```

9. Creating Reports

9.1. Overview

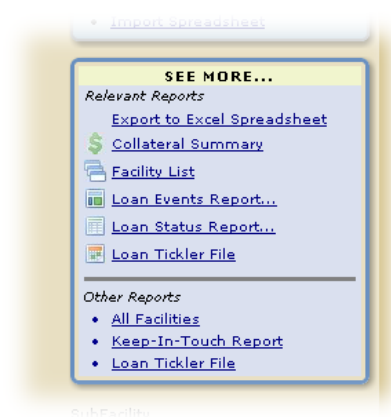
dbFramework provides a very easy way to create a data repository, with entry forms and display screens. But often enough a more detailed view of the data is required, one that aggregates, summarizes and otherwise processes the data, and this needs to be presented in a manner that is suitable for printing.

This is where the reporting features of dbFramework step in. dbFramework lets developers create reports in a couple of ways: either through an ad-hoc reporting interface where fields are selected in a point-and-click interface, or with absolute flexibility using a combination of Microsoft Word or HTML templates and embedded DbfScript commands.

Reports can be created either in HTML or in PDF format through a selection in the server configuration record.

Note: To generate reports in PDF format, Microsoft Word 2007 must be installed on the same machine that the DbfServer is running. Also the Microsoft Save-As-PDF plugin must be installed. dbFramework uses Microsoft Word to convert the HTML or Word template into a PDF file programmatically.

Once the report is created, it becomes accessible through the 'See More' list in the application screen.



There are two sections to the report list. The top section shows the reports that are related to the current selection in the application. The reports related to the current record along with reports related to parent records are shown in this list. If

a report for a parent record is selected, then that report runs as if the parent record were currently the focus on the screen.

The section below that, 'Other Reports', contains a list of the *global* reports that can be run from anywhere.

Both types of reports can either run immediately or require parameters input from the user before running. Reports that require parameters will appear with an ellipses (...) at the end of the report title.

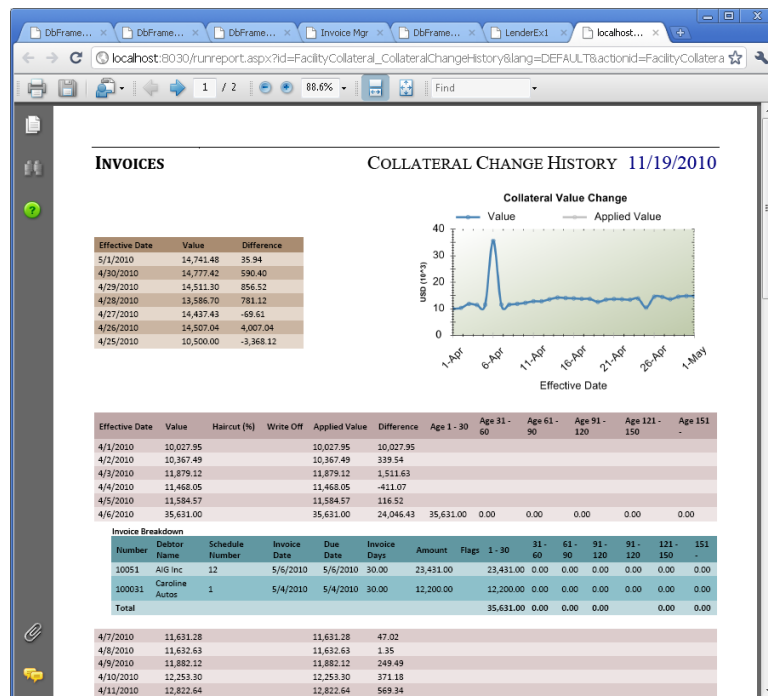
This section delves into creating these reports and exploring their capabilities.

9.2. Quick Reports

9.2.1. Overview

Quick Reports are the easiest way of creating reports, providing the ability to create simple or complex reports, from simple lists to multi-pane dashboards with multiple charts, summaries and lists. Quick Reports are also automatically localized should you introduce a new language into the application.

An example of a Quick Report as a PDF file is shown below:



The same Quick Report can also be shown as a custom view in the application window, providing dashboard like functionality with very little effort:

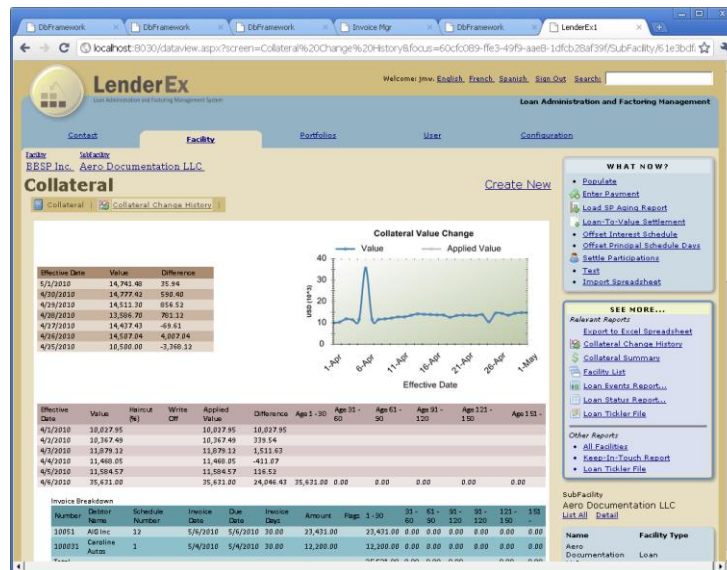


Figure 42: Quick Report viewed as a custom view

With a single click you can have a Quick Report user-subscribable. This provides an area for users to subscribe to the report, where the system will automatically generate the report on a specified schedule (every morning, for example), create a PDF file and then send it to the user in an email.

Reports can also be re-used. Any part of the report can re-use a section from another report somewhere else in the system, assuming both are working on the same kind of data. This allows you to create a whole asset library of report components that make generated new reports even simpler.

All of these aspects of report generation will be discussed in this section.

9.2.2. Quick Report Concepts

Creating a Quick Reports lets you focus on getting the data together, while letting dbFramework figure out how to display that data. You give it hints, such as whether to display data in a form layout or a tabular layout, and you specify the colors. Using a concept called 'horizontal slots' you can also segment the display into sections, and combined with its understanding of hierarchical data it can produce quite advanced layout with very little effort.

The closer your report requirements are to the data model you've chosen, the easier it will be to create the Quick Report. But that doesn't limit what Quick

Reports can do: there are very innovative ways to link together and navigate around the entire data model without having to write one line of code, or one query statement. The entire report creation experience can be a matter of point-and-click alone.

Before delving into the Quick Reports it's worth understanding some basic concepts to get a better grasp of how a Quick Report functions.

9.2.2.1. Sections

A 'section' in a Quick Report refers to a data query that *may* result in a new section being displayed on the screen. But that query result can also be combined into an existing section.

Consider this report output:

User ID	First Name	Last Name	Current Task	Task Status
fbell	Fred	Bell	Write Test Spec	Not Started
tsmith	Tim	Smith	Order Lunch	Started
Ccooper	Caroline	Cooper	Develop Marketing Plan	Not Started

Let's pretend this report runs on the user list. Whereas the first three columns take data from the user list – simple enough – the last two columns come from a task table. This may mean traversing down an 'Assigned Tasks' sub-field. So this would be a sub-section that is merged into the parent section. The merging is what adds the query result as new columns on the parent section, rather than creating a section in its own right.

A section represents a single step in a query navigation. So, for example, if you have to drilldown 6 levels in a hierarchy to get to the data you want to put into column X, you may have 6 sections, all nested. Each of those sections will be set to merge into the parent section.

Now let's consider what would happen if we make that sub-section for 'Assigned Tasks' have a Process Type of 'New Section':

User ID	First Name	Last Name
fbell	Fred	Bell
Current Task	Task Status	
Tsmith	Tim	Smith
Order Lunch	Started	
Ccooper	Caroline	Cooper
Develop Marketing Plan	Not Started	

You'll notice how, with that change, the task name and status now appear as a separate table underneath each row. This is what is meant by starting a new section.

A section can also have a 'Root' field left as blank, and this allows the parent query result to be used again. And also the 'Root' field can point to a link field, and this will take the query to wherever the link points to. Sub-sections of that can then point to sub-types in *that* location. The possibilities are endless.

9.2.3. Getting Started

To create a Quick Report there are two possible places to start.

1. If you wish to have a global report that can be run from anywhere, then you need to go to the Application record, then drilldown to 'Quick Reports'.
2. If you wish to have a report run from a specific place in the database, then you need to find the associated field definition in the data model, and then drilldown to 'Quick Reports' from there.



9.2.3.1. Creating a Quick Report

Once you have found the location of the Quick Reports, click Create New to add a new report.

The screenshot shows the 'Quick Reports' form in the dbFramework application. The form is titled 'Quick Reports' and has a 'Create' button at the bottom right. The form is divided into several sections:

- Title:** A text input field containing 'Test Report'.
- Enabled:** A dropdown menu set to 'No'.
- Icon:** A text input field with a dropdown arrow.
- Show Record Title:** A dropdown menu set to 'Yes'.
- Show Title:** A dropdown menu set to 'Yes'.
- Show Date:** A dropdown menu set to 'Yes'.
- Show Filters:** A dropdown menu set to 'Yes'.
- Layout:**
 - Horizontal Slots Per Page:** A text input field containing '1'.
 - Font Size (Points):** A text input field containing '8'.
 - Page Orientation:** A dropdown menu set to 'Portrait'.
 - Page Type:** A dropdown menu set to 'Letter'.
 - Allow Users to Subscribe:** A dropdown menu set to 'No'.
 - Custom View:** A dropdown menu set to 'None'.

Figure 43: Screenshot of a new Quick Report

The following fields can be entered:

Title

This is the title of the report as it will be shown in the reports, “See More”, list.

Enabled

This provides a quick way to disable report temporarily and remove it from the reports list.

Icon

Optionally lets you specify an icon that will be displayed alongside the report on the screen.

Show Record Title

Specifies whether the title of the currently selected record should be displayed at the top of the report.

Show Title

Specifies whether the title of the report itself should be displayed at the top of the report.

Show Date

Specifies whether the date the report was run should be displayed at the top of the report.

Show Filters

Specifies whether any filters (report parameters) entered by the user prior to running the report should be displayed at the top of the report.

Horizontal Slots Per Page

Explained in more detail in the next section, specifies the number of columns (referred to as 'horizontal slots') that will be utilized in the report layout. The default is 1, which means there will be only one column and essentially the layout will flow vertically.

Font Size (Points)

The default font size for the overall report.

Page Orientation

Specifies whether the report will be generated for a portrait (upright) or landscape (sideways) page orientation.

Page Type

Specifies the size of the page, of either Letter, Legal or A4. The default selection is letter.

Allow Users to Subscribe

Specifies whether additional structures will be created allowing for users to subscribe to this report. Once the user subscribes, the system will automatically generate and email the report to that user on the user-specified schedule.

Custom View

Specifies whether this report will also be incorporated into the application as a custom view.

- **None:** The report will not appear as a custom view.
- **Screen Option:** The report will appear as an optional custom view shown as a screen tab.
- **Main Screen:** The report will replace the default view in the screen, effectively becoming a dashboard.

9.2.3.2. Creating a Sub-Section

Every report must have at least one sub-section. A sub-section represents a list in the report, and is explained in more detail in the next section on concepts.

To create a sub-section, after creating the Quick Report record, click on the Sub-Sections drilldown and click 'Create New'.

The screenshot shows the 'Sub-Section' configuration form in the dbFramework Professional Edition. The form is divided into several sections:

- Internal Name:** A required text field.
- Heading Title (Optional):** A text field.
- Sub-Title (Optional):** A text field.
- Root Field:** A text field.
- Display Type:** A dropdown menu set to 'Tabular List'.
- Layout Variation:** A dropdown menu set to 'Normal'.
- Hierarchy:** A section with checkboxes for 'Recurs Through', 'Collect All, Deep', 'Flatten Hierarchy', and 'Add Index'.
- Query Details:** A section with a 'How to Process Results' dropdown set to 'New Section' and a 'Limit Rows' text field set to '500'.
- Layout:** A section with a 'Full Horizontal Width' checkbox checked, a 'Width (Percent)' text field set to '100', a 'Footer Node' dropdown set to 'User-Selectable', and a 'Horizontal Position' dropdown set to 'Left'.
- Re-Use Quick-Report:** A section with checkboxes for 'Field of Report to Re-Use', 'Quick-Report to Re-Use', and 'Re-Use Section'.

 A 'Create' button is located at the bottom right of the form. On the right side of the form, there is a sidebar with 'SEE MORE...' links for 'Relevant Reports', 'Quick Reports', and 'Other Reports'.

Although there looks like many things to enter, only the 'Internal Name' is required. The others can be left blank.

The following fields can be entered:

Internal Name

This is the name that you will use to reference the section. This name won't be displayed to the end-user.

Heading Title

Specifies an optional title that will display above the list.

Sub-Title

Specifies an optional sub-title that will display under the title.

Root Field

Optionally specify a nested type in your field list to traverse down. If left blank, uses the current section (either the parent, or the location where the report was

run from). This is discussed in greater detail in the next section on concepts. You can also select link fields, and the report will traverse down the link to wherever in the database the link is pointing.

Note: If changing the root field after it has been set up and used, all existing fields and subsections in the section will be deleted. This is because the root now refers to a different place, and the fields and subsections depend on that location to be valid.

Display Type

Specifies how this section will be presented in the report. The default is to display the data as a tabular list.

- **Tabular List**
The data is shown in a tabular list. The colors and layout can be overridden in Coloring and Default Renditions sections.
- **Form**
The data is shown as a vertical form layout, in two columns with the first column being the field title and the second column containing the value. See also Layout Variation to control how this is laid out.
- **Line Chart**
The data is shown as a graphical line chart. The first field in the section is used to seed a value for the X-Axis, and the remaining fields are plotted as Y-axis series. The coloring of the chart can be controlled by setting the Coloring section, and other chart options can be controlled in the Chart Options drilldown.
- **Pie Chart**
The data is shown as a graphical pie chart. The first field in the section is used to seed a value for the pie slice label, and the second field provides the value for the pie slice. The coloring of the chart can be controlled by setting the Coloring section, and other chart options can be controlled in the Chart Options drilldown.
- **Bar Chart**
The data is shown as a graphical bar chart. The first field in the section is used to seed a value for the X-Axis, and the remaining fields are plotted as bars on the Y-axis. The coloring of the chart can be controlled by setting the Coloring section, and other chart options can be controlled in the Chart Options drilldown.

Layout Variation

An optional list of variations in display format. If the 'Form' layout is selected, then the 'Compact' layout variation will display each field in the form beside the value, rather than aligned in a column.

Recurs Through

Because dbFramework centers on a hierarchical data model, the Quick Reports has the capability to easily incorporate hierarchical data into the reports. If the data being viewed is recursive (that is, it is self-referencing) then you can select the field that holds the drilldown to the same type to show a potentially unlimited hierarchy. This will be useful in presenting reports that have a folder and sub-folder like structure. This is explained in more details in the section on concepts.

Collect All Deep

For recursive hierarchical lists, provides a quick way to enable or disable the recursive behavior.

Links From

When the Root field refers to a foreign link field in another area of the database, and 'Links From' is set to 'Yes', retrieves all records in the database where that foreign link field links directly to the current record.

Unlike regular sections, a section with 'Links From' will produce a list of records that can be from different parts of the database and have entirely different paths.

Example: For example, imagine I had a 'Country' list and a 'Customer' list that contained references to countries. With this feature I could have a top-level section to list each Country entry, and then a sub-section of that with a Root field set to the Customer's 'Country Reference' field and the section's 'Links From' set to 'Yes'. When the report would run you would see all countries and a section underneath each country that lists all of the customers that are in that country.

Add Index

When 'Recurs Through' is set, indicates if an additional numeric 'Index' field will be added that describes the current level in the hierarchy.

Aggregate

Indicates if the data for the list should be aggregated, either summed, averaged or showing the minimum or maximum of a value. The aggregate function can be set for each field.

How To Process Results

Indicates if this level will be shown as its own list, or whether the results will be merged into the previous list as new column(s).

- **New Section**

The data from this subsection will be displayed as a new list, shown below the current row, slightly indented to show its relationship.

- **Merge Into Existing Section**

The data from this subsection will be merged with the current (parent) section. Each field in the section will be added as new columns or fields in the parent section. Each record in the result will result in a duplication of each parent record. Typically merged sections will only retrieve 1 record.

Condition

Filters that can be applied to retrieving this data. The format of the condition is a comma-separated list of predicates, in the form of “ field = value “.

The field must refer to a field that exists in the data that is being queried. It must match the name as it appears in the data model.

The Value can either a literal value, specified in double-quotes for a string, or a plain number. It can also be a field reference, which must be surrounded with percent signs.

The field reference in the value must use the ‘Internal Name’ of the field. You can also reference any field in the parent section (or sub-sections) so long as they are part of the same level (ie. merged). For example Date=%StartDate%. It can also reference fields that are in the filter page that is displayed when the report is run.

The value can also be substituted for “%NOW%” or %NOW-X%, where X is a number of days, or %NOW+X% where X is a number of days.

Note: When comparing values that have dropdown lists specified, make sure you compare to the *value* rather than the *title* of the dropdown list.

Limit Rows

Allows the number of records returned from the data query to be limited by a specific number. For example, entering 1 in this field will guarantee that at-most 1 record will be displayed or processed. The default is 500, restricting the maximum number of records to 500.

Fill Horizontal Width

Specifies if, when the section is displaying as a new list, whether that list will fill the entire width available to it, or whether it will just take up as much room as it needs to display.

Width (Percent)

If 'Fill Horizontal Width' is set to 'No', specifies the maximum width this section will take up.

Footer Note

Specifies optional text that will appear underneath the list.

Horizontal Position

Indicates how this section will be aligned within the area available to it.

Top Margin

Specifies optional spacing above the list, moving the section further down the page. Specified in lines based on the current font.

Slots to Span

Specifies how many horizontal slots (columns) this section should span if the overall report has more than 1 horizontal slot.

Page Break After

Indicates if each row in this section should cause a page break before the next row is displayed. Does not apply for the last row in the section. If the section is a chart, then a page break is inserted after the chart.

User Selected

Specifies whether the user will be able to select whether or not to include this section in the report.

Is Critical

If true, then this data is critical to the report. If no data is found for this section then running the report will fail with an error.

Field of Report to Re-Use

The next three fields are used when re-using an existing report. The existing report section is used instead of the currently defined section, and all processing

continues in that report's section. This entry box specifies the field/type that holds the report to re-use.

Quick-Report to Re-Use

Specifies the Quick Report in that field definition to re-use.

Re-Use Section

Specifies the section within the Quick Report to re-use. Using the '+' you can also drilldown to a sub-section.

9.2.3.3. Default and Title Rendition

Once you have created the section, you then have the option of configuring the way it will look. This is done through the 'Default Rendition' and 'Title Rendition' drilldowns, and the 'Coloring' drilldown described in the next section.

The Default Rendition controls the way the text looks inside the list. The Title Rendition is used to control how the heading text (column headers or column names in a form layout).

The following fields are available to edit:

Show Title

Specifies whether the column heading will be displayed.

Font Size (Em)

Specifies the size of the font in this section, relative to the font for the overall report. An 'Em' is a fraction of that font size, with 1 (or 0) being the same size, and 0.5 being half the size.

Bold

Specifies whether values in the list will appear in **bold font**.

Italics

Specifies whether values in the list will appear in *italics font*.

Small-Caps

Specifies whether values in the list will appear in SMALL CAPS.

Underline

Specifies whether values in the list will appear underlined.

Color (Hex)

This field is not used for Quick Reports. Rather the color should be specified in the Coloring drilldown.

Font Family

Overrides the name of the font family to use. The font must be installed on the server (if PDF reports are displayed) and on the client if the report is displayed in the browser as HTML.

Is Image?

Indicates if a field will be displayed as an image. Not used for Sections.

Image Width

If showing an image, specifies the width (in pixels) of the image. Not used for sections.

Show As Link

Indicates if this field will be displayed as a clickable URL link that will take the user to the associated record in the database.

Note: For this feature to work, the host name must be correctly filled-in in the configuration record of the application.

9.2.3.4. Coloring

The 'Coloring' drilldown allows you to override the default coloring for a list or form.

The following fields must be entered:

Show Border

Specifies whether to show a border around the list.

Horiz Grid Lines

Specifies whether to show horizontal grid lines.

Vertical Grid Lines

Specifies whether to show vertical grid lines.

Grid Color

Specifies the color of the grid lines.

Back Color

Specifies the general background color for the list.

Band Color

Specifies the alternate background color for bands in the list. If you don't want banding, then set this to be the same as the back color.

Header Color

Specifies the color for the column header.

Text Color

Specifies the general text color for the list.

9.2.3.5. Chart Options

When the section is displayed with a Display Type of a chart, then the Chart Options drilldown lets you configure specific chart-related options.

The following fields can be entered:

Chart Title

The title caption that is shown at the top of the chart.

Y Axis Title

The title for the Y-Axis. If this is not supplied, the Y-Axis is taken from the second field in the section.

Height Percent

The percentage of the height compared to the width, allowing you to control the aspect ratio.

Smooth Lines

Indicates whether to smooth the lines, so that instead of sharp edges they will have curves.

Thickness

The thickness of the series. Defaults to 2 pixels.

Show Data Points

Indicates if the data points in the series are highlighted.

Show Chart Shadow

Not currently used.

Show Line Shadow

Not currently used.

9.2.3.6. Fields

Once the section is complete, you can then add fields to the section.

There are two ways to add fields. The easiest way is using the Quick Entry at the bottom of the section record.

The screenshot shows a web application interface for managing sections. At the top, there's a 'Sub-Section' header with a 'Create New' link. Below it is a 'Test Section' record with various fields like 'Internal Name', 'Heading Title', 'Sub-Title', 'Root Field', 'Display Type', 'Layout Variation', 'Hierarchy', 'Query Details', 'Layout', and 'Re-Use QuickReport'. At the bottom, there's a 'Quick Entry (Alt+Q)' section with a 'Reference' input field and an 'OK' button. Below the 'Quick Entry' is a table with columns: 'New Title (Optional)', 'Reference', 'Calculation Formula', 'Visible Condition', 'Format', 'Sort', and 'Group By'. The table contains two rows: one with 'Name' and 'None' for Sort, and another with 'FacilityType' and 'None' for Sort.

New Title (Optional)	Reference	Calculation Formula	Visible Condition	Format	Sort	Group By
	Name				None	
	FacilityType				None	

Figure 44: Section record, showing the Quick Entry for adding fields at the bottom.

Entering from the Quick Entry allows you to quickly select a field to include in the report, click OK, and select another. Once entered you can still click on the field that was entered and edit it to customize it further.

The second method is to add the field through the form by clicking 'Create New'.

While just the field reference is required, the following remaining fields can be entered:

New Title (Optional)

Specifies an optional override of the title for this field. If this isn't supplied, then the title of the selected field reference will be used.

Reference

Select the field reference to use in the report. This field is required.

Calculation Formula

A DbfScript expression that uses the record data (passed in as #input) to derive the value to use. When a formula is used, the field Reference is still required to determine the type of the result. Any values from this level (and parent levels if merged) can be accessed using #input.

Note: When a calculation formula is specified, an Internal Name must also be provided so that the calculation result can be referenced.

Internal Name

This is an optional name that will be used to reference this field in formula.

Visible Condition

A simple condition that determines whether the data value will be displayed. Must be of the format field, operator and value. For example, EntryType=Payment. The field specified can either be a field in the data model, a calculated field or a field in another merged level.

Format

A standard custom format specifier used to format either numbers or dates. For example, “#,##0.00;(#,##0.00);0”.

Sort

Specifies whether this field will be used as a sort key to determine the order of the records. Multiple sort keys can be specified.

Group By

Specifies whether the record list will be grouped, and whether this field will be used as a group key. Multiple group keys can be specified.

Total Type

Specifies whether (and how) this field will be totaled at the end of the record list (and group).

Hide

Provides a quick way to hide a field so that it won't be included in the record list.

Override Position

A numeric index that allows you to provide a numeric index that overrides the order of the fields. When fields are merged together from multiple sections, this may be used to allow fields from one section to be displayed next to fields from a previous section.

Stack Vertically

Indicates if this field will be stacked, vertically, below the previous field if displayed in a tabular list.

Minimum Width

Indicates the minimum width, in pixels, that this column will fill.

Column Header Alignment

The alignment of the text in the column header.

Value Alignment

The alignment of the text in the data cell.

Filter Type

Indicates if the field can be used as part of a filter key when running the report. If selected then this field will be incorporated into a report parameters screen that will be displayed to the user prior to running the report. There are several different options for filtering:

- **No Filter**
This field will not be used as a filter.
- **Single Value**
The field will be presented as a single entry box for filtering by a single value.
- **Range**
The field will be presented with both a minimum and maximum value allowing the user to filter by a range.
- **Maximum**
The field will be presented with a single value that will be used to filter by the maximum of a field.
- **Minimum**
The field will be presented with a single value that will be used to filter by the minimum of a field.
- **Multiple Values**
The form will provide dynamic expansion allowing an unlimited number of values to be entered. Each value will be OR'd together to provide the filter.

Filter Prompt

Defines the title to display next to the filter field, if used.

Rendition

The rendition record is described in section 9.2.3.3 and lets you control the formatting of this field.

9.2.4. Report Summary

At any point while creating your report, or even to review existing reports, you can use the 'Report Summary' report. This is available on any Quick Report, and shows

an overall summary of the report structure, including all the sections and any fields selected in each section.

The 'Report Summary' is also available as a screen option, at the top of the report definition.

The screenshot displays the 'Quick Reports' interface in dbFramework. At the top, there are navigation tabs for 'Arithmetic', 'Model Types', and 'Nested Fields'. Below these, the breadcrumb path is 'LenderExi > Facility > Collateral'. The main heading is 'Quick Reports' with a 'Create New' link. A sub-header shows 'Quick Reports | Report Summary'. The 'Report Details' section indicates the title is 'Collateral Change History', horizontal slots per page are 2.00, and page orientation is Portrait. The 'Sections' table lists three sections: 'Collateral Summary', 'Change List Chart', and 'Change List'. Each section has associated fields with their internal names, new titles, references, calculation formulas, hide status, and filter types. On the right, there are two sidebars: 'WHAT NOW?' with links like 'Add Calculated Field' and 'Import Spreadsheet Structures', and 'SEE MORE...' with links like 'Export to Excel Spreadsheet' and 'Report Summary'. At the bottom right, there are sections for 'Nested Fields' and 'Types'.

Index	Internal Name	Heading Title (Optional)	Root Field	Display Type	How to Process Results	Condition	Limit Rows
1	Collateral Summary		Change	Tabular List	New Section		7.00
2	Change List Chart		Change	Line Chart	New Section		500.00
3	Change List		Change	Tabular List	New Section		500.00

This report can be very useful to quickly see the structure of a report to help you understand it or refresh your memory.

9.3. Automatic Schedule (Tickler) Reports

9.3.1. Overview

dbFramework provides an even easier way to create reports for values that involve due or event dates. Known as 'Tickler Files', these reports show schedules of upcoming events and due dates across the types that you specify.

dbFramework can create these reports for you. The quickest way is to just make a field of type 'Date/Time (Due Date)'. But you can also quickly create a custom tickler report: Just tell it which fields hold dates that are important, and what kind of record it refers to.

Not only does it automatically create a tickler report, but that report can be displayed as an alternative view in the application itself, and users can also

subscribe to that tickler report so they can have it appear in their inbox each morning.

The report also works hierarchically, traversing down sub-types that are of the same type as that selected.

The report includes special processing for past-due events, which it highlights in red at the top of the report, allowing you to keep on top of important information throughout your database with just a few clicks.



LOAN TICKLER FILE 11/19/2010

Location: Facility: ACME Fund

INTEREST DUE (PAST-DUE)

Date	Name	Amount	Days Late
9/22/2010	BBSP Inc. Facility: BBSP Inc.	54,051.00	58
10/9/2010	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	811.11	47
10/9/2010	JHL Trucking Facility: JHL Trucking	916.67	41
10/15/2010	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	969.83	35
10/20/2010	Envision Main Facility Facility: Envision Main Facility	884.44	30
11/2/2010	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	644.44	17
11/8/2010	JHL Trucking Facility: JHL Trucking	400.00	11
11/14/2010	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	426.67	5
11/18/2010	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	217.50	1
Total		48,780.34	

INTEREST DUE
Month: November 2010

Day	Name	Date	Amount	Days Left
24th	JHL Trucking Facility: JHL Trucking	11/24/2010	113.33	4
30th	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	11/30/2010	128.00	10
Total for November 2010			241.33	

Figure 45: An example tickler report automatically generated by dbFramework

The report shows one past-due section at the top of all past-due events across the system. Underneath that events are grouped by type and by month, with a column showing the date, number of days before the event is due, and any values (eg. amount) relevant to the entry.

These tickler reports automatically become user-subscribable, so that users can have these schedule reports come into their inbox each morning, letting them know exactly what needs to be done that day.

They also appear as a custom view screen option, letting you view this important information right within the web browser.

9.3.2. Creating a Tickler Report

9.3.2.1. The Quick Way

The fastest way to create a tickler report is to simply make one of the fields in your type a due date. To do this, you simply select a data type of 'Date (Due Date)'.

When dbFramework is publishing your application, it will see this due date field and then automatically create a tickler report. The tickler report will just contain records in that list, shown as a schedule covering 60 days. Users can also subscribe to that tickler report to have it delivered to them each day, and you can view it on the screen as an optional tab to click.

Note: You also need to define a title field in your type for the tickler file to be created. If you don't specify one field in your type with 'Is Title' set to 'Yes', the report will not be created.

9.3.2.2. Creating Your Own

You can also, just as easily, configure your own, more sophisticated, tickler reports.

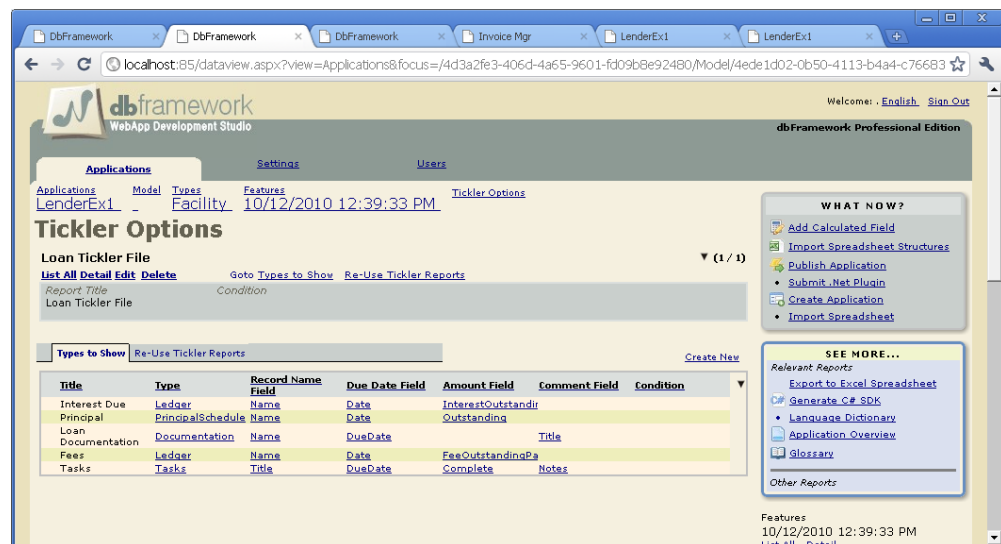


Figure 46: Editing a Tickler Report

The Tickler Report configuration is held in the Features drilldown from a field definition. You may have to create the Features drilldown before you can enter the Tickler Options. For more information on the 'Features' drilldown, see section 11.

Go to the Features record, and enable the option 'Tickler Report'.

The screenshot shows the 'Features' configuration page. It has a header with 'Features' and a dropdown for '(1 / 1)'. Below is a grid of 16 features, each with a label and a dropdown menu:

Live Updates	Show Calendar	Show Map	Support Audit Trail
No	No	No	No
Show Last-Modified/Create Dates	Reminders	Version Controlled	Ratable
No	No	No	No
Requires Review	Allow Comments	Support Forum	Supports RSS
No	No	No	No
Can Be Tagged?	Show Charts	User-Positionable	Tickler Report
No	No	No	No

The 'Tickler Report' dropdown is open, showing 'Yes' and 'No' options. An 'Update' button is at the bottom right.

After updating that, go to the Tickler Options drilldown and click Create New.

In this window you will only have to enter in 'Report Title'. Then click Create. Next drilldown to the 'Types to Show' drilldown, and click 'Create New'. The following fields can be entered:

Title

This is the title of the tickler report, as it will appear in the report list, and in the report.

Type

This is where you select the field at or below the current location in the data model that will be displayed in the report. A single tickler report can contain multiple, different types.

Record Name Field

The field to use for the name of the schedule entry in the tickler report, which can be anywhere in the data model hierarchy between the current type and the selected Type, above.

Due Date Field

The field to use for the Due Date, which must be a sub-field of the Type, as selected above.

Amount Field

An optional 'Amount' field, which goes into an Amount column in the report, to provide information, usually a value, related to the scheduled entry.

Comment Field

An optional 'Comment' field, which appears underneath the name of the schedule entry.

Condition

An optional condition that will be applied to the filter when querying for Type. See the 'Condition' field in section 9.2.3.2, on entering sub-sections in Quick Reports, for more details.

9.3.3. Using the Tickler Report

Tickler reports can be run from the reports list ('See More...') when the associated type is selected in the application. For tickler reports that are defined on a top-level type, the report will also show up as a global report, which can be run from anywhere.

In addition to being available as a report, all tickler reports are automatically created as custom views.

The screenshot shows the LenderEx web application interface. The main content area displays a 'Facility' custom view for 'BBSP Inc.' with two tables: 'INTEREST DUE (Past-Due)' and 'PRINCIPAL (Past-Due)'. The 'INTEREST DUE' table has columns for Date, Name, Amount, and Days Late. The 'PRINCIPAL' table has columns for Date, Name, Amount, and Days Late. The interface also includes a 'WHAT NOW?' sidebar with links to 'Enter Payment', 'Load SP Aging Report', 'Loan-To-Value Settlement', 'Offset Interest Schedule', 'Offset Principal Schedule Days', 'Settle Participations', 'Test', and 'Import Spreadsheet'. A 'SEE MORE...' section lists 'Relevant Reports' and 'Other Reports'.

Facility: BBSP Inc.			
INTEREST DUE (Past-Due)			
Date	Name	Amount	Days Late
9/22/2010	BBSP Inc.	54,051.00	59
Facility: BBSP Inc.			
Total		54,051.00	
PRINCIPAL (Past-Due)			
Date	Name	Amount	Days Late
4/11/2010	BBSP Inc.	-5,000,000.00	223
10/21/2010	BBSP Inc.	1,015,675.36	29
7:11:29 PM	Facility: BBSP Inc.		
10/22/2010	BBSP Inc.	5,689,473.86	28
9:16:33 PM	Facility: BBSP Inc.		
10/22/2010	BBSP Inc.	5,689,473.86	28
11:52:53 PM	Facility: BBSP Inc.		
10/22/2010	BBSP Inc.	5,689,473.86	28
11:55:48	Facility: BBSP Inc.		

Figure 47: Tickler Report as a custom view

Tickler Reports are also available for user-subscription. This means that any user can subscribe to the tickler report and have it delivered to their inbox each morning, at on any schedule they configure.

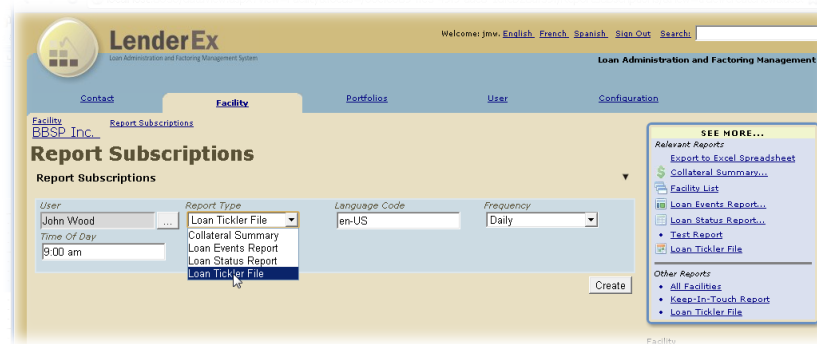


Figure 48: Configuring a report subscription for the tickler report

9.3.4. Anatomy of the Tickler Report

There are two main parts of the tickler report.

- The Past-Due section shows any events discovered by the tickler report that are older than today. These are separated by Type, and highlighted in red to bring them to the user's attention.
- The upcoming events section, split up by each Type that is defined in the report, and then grouped by month.

The Past-Due section shows all events in one list without grouping by month, with the actual date the event was due prominently displayed. An additional 'Days Late' column shows how many days late the event is, along with a total at the bottom to show the total for any amounts that were displayed. This provides a quick way to see the total amount of payments that are past-due, for example.

INTEREST DUE (PAST-DUE)			
Date	Name	Amount	Days Late
9/22/2010	BBSP Inc. Facility: BBSP Inc.	54,051.00	59
10/3/2010	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	811.11	48
10/9/2010	JHL Trucking Facility: JHL Trucking	916.67	42
10/15/2010	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	969.83	36
10/20/2010	Envision Main Facility Facility: Envision Main Facility	884.44	31
11/2/2010	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	644.44	18
11/8/2010	JHL Trucking Facility: JHL Trucking	400.00	12
11/14/2010	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	426.67	6

The upcoming events section shows all future events, up to 60 days ahead, separated by each Type that is defined, and then further grouped by month.

INTEREST DUE

Month: November 2010

Day	Name	Date	Amount	Days Left
24th	JHL Trucking Facility: JHL Trucking	11/24/2010	113.33	3
30th	Georgia Internetworking Sys Facility: Georgia Internetworking Sys	11/30/2010	128.00	9
Total for November 2010			241.33	

Month: December 2010

Day	Name	Date	Amount	Days Left
3rd	Caruna Sports Main Facility Facility: Caruna Sports Main Facility	12/3/2010	120.83	12
5th	Envision Main Facility Facility: Envision Main Facility	12/5/2010	120.00	14
11th	JHL Trucking Facility: JHL Trucking	12/11/2010	106.67	20

The first column shows the day of the month the event is due. The Name column shows the record title that was selected, along with the location of the record (the absolute path) below that, in gray. Optionally any comments will appear below that.

The next column shows the actual date the event is due. The Amount column shows the value of any Amount field that is selected. And the last column shows how many days are left until the event is due.

9.4. Custom Reports

9.4.1. Overview

Custom Reports are a different way to create reports in dbFramework using Microsoft Word or an HTML editor.

While Quick Reports and Tickler Reports provide an incredibly simple way to create great looking reports in next to no time, sometimes more flexibility is desired to create unusual reports, or perhaps layouts that are optimized for a specific purpose. To achieve this you can create a *Custom Report*.

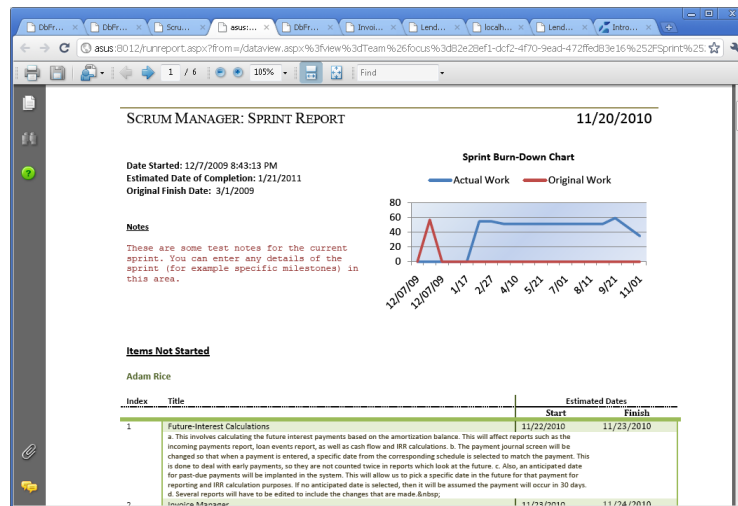


Figure 49: Sample custom report created using Microsoft Word

Custom Reports give you complete flexibility in how the report looks by allowing the entire page layout to be designed using a word processor, as something called a ‘template’.

There are two types of custom report templates that can be created:

- An HTML report template, that can be created using any HTML authoring tool, such as Microsoft Expression Web or Adobe’s DreamWeaver.
- A Word report template, that can be created using Microsoft Word 2007 (or higher).

The Word report template functionality has additional, specialized support for the Microsoft Charting engine, allowing all of the powerful features of the charting engine in Word to be utilized.

Both templates involve designing the report the way you wish the report to look, by embedding DbfScript commands to request and manipulate the data that will be displayed in the report. This works much the same way as designing custom views, as described in section 10, but with additional commands that are specific to manipulating report data and for utilizing the features of Microsoft Word and its charts.

Reports are generated on the server and are then converted to PDF format before being delivered to the client. The DbfScript commands run on the server, and the output of the commands determines how the text in the report template will be repeated to show the report output.

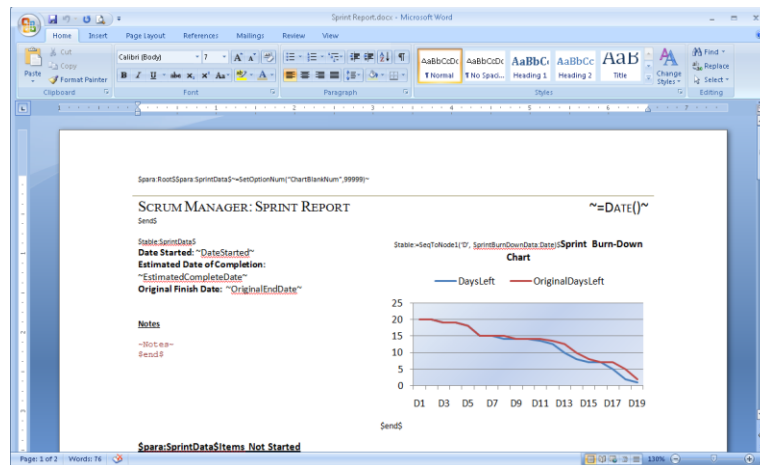


Figure 50: Screenshot of Microsoft Word, showing a report template

Note: The templates cannot be created in versions of Microsoft Word prior to 2007 because dbFramework utilizes the new open XML-based file format, known as the Office Open XML File Format. There are also other tools that use this file format, and these should also work as report designers, however only Microsoft Word is officially supported.

9.4.2. Creating a Custom Report: The Basics

This section will guide you through the stages of creating a custom report using Microsoft Word. The first step to creating the custom report is to open Microsoft Word with a blank document.

The document will look much like a regular Word document, but with special commands inserted that control how the text is processed in relation to the data it is intended to display.

\$para:Root\$\$para:SprintData\$

SCRUM MANAGER: SPRINT REPORT

Send\$

\$table:SprintData\$

Date Started: ~DateStarted~

Estimated Date of Completion:

~EstimatedCompleteDate~

Original Finish Date: ~OriginalEndDate~

It would be worth familiarizing oneself with the way in which custom views are created, because the techniques used are very similar. Please refer to section 10.2 for more details.

The entire input data is wrapped in a tag called 'Root', so one of the first tags in your Word template must be \$Root\$, and the last tag should be \$end:Root\$.

In order to properly align the data tags with different structures in Word, it's necessary to prefix tags with either **para** or **table**, followed by a colon and then the remainder of the tag. So, for example, if you were at the beginning of a paragraph and you want several paragraphs to be repeated for the tag 'Task' you would insert a command such as:

\$para:Task\$

At the end of the text you wish to repeat, you would insert a \$end:Task\$ command to signify the end of the Task tag.

If you had a table in your document, and you wish a specific row to be repeated for the tag Task, you would insert the text \$table:Task\$ into the table. For example:

Name	Status	Start Date	Due Date	Assigned To	% Complete
\$table:Task\$~Name~	~Status~	~StartDate~	~DueDate~	~AssignedTo~	~PercComplete~
\$end:Task\$					

It's important to use the correct tag prefix or either table or para throughout your document. If you do not then the resultant Word file may become corrupt and the report will not run.

To use a DbfScript formula, you should insert an equals sign before the DbfScript expression. For example:

```
~para:=Date()~
```

...would insert the current date.

There are some additional functions available in Word reports that are explained in the section 9.4.3.

9.4.3. Word-Specific Functions

9.4.3.1. *GroupBy*

The GroupBy function takes a parameter of the name of the field containing a list, a colon, and the name of the field to group by.

For example if the current context contained a field called TaskList, and we wanted to group by the field in TaskList called AssignedTo, and we wanted to show this in a table, we would use the command:

```
$table:=GroupBy(TaskList:AssignedTo)$
```

9.4.3.2. *Flatten*

The flatten command is used to convert self-referencing hierarchical data into a flat list that can be displayed in a single table.

The parameter to Flatten is the subfield containing the list, a colon, and the name of the field in that list that contains the self-reference.

For example, if we had a TaskList, and in that there was a SubTask field, we might use the following command:

```
$table:=Flatten(TaskList:SubTask)$
```

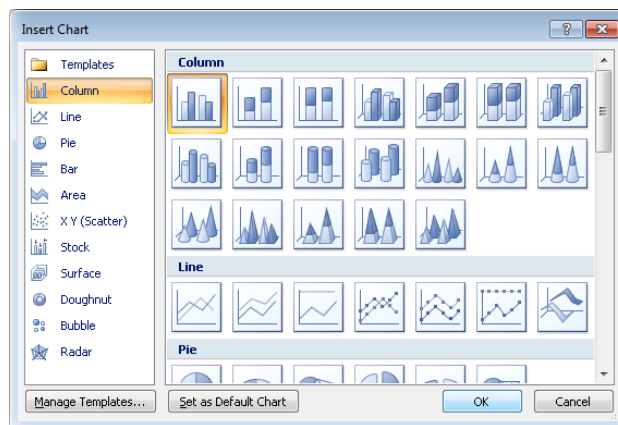
Once the hierarchy is flattened, an additional field named `_Index` is created, and this can be used to display the numeric index and depth of the record in the list.

9.4.4. Creating Charts

9.4.4.1. Line and Bar Charts

The Custom Reports feature has special support for creating charts using the built-in Microsoft Chart engine in Microsoft Office products.

To add a chart to your report template, you simply insert the report the way you would normally create a report.



Once the chart has been inserted, you will be taken to the data edit screen. In here we define the fields that will be displayed in the chart:

	A	B	C	D	E	F	G	H	I	J
1		DaysLeft	OriginalDaysLeft							
2	D1	20	20							
3	D2	20	20							
4	D3	19	19							
5	D4	19	19							
6	D5	18	18							
7	D6	15	15							
8	D7	15	15							
9	D8	14	15							
10	D9	14	14							
11	D10	14	14							
12	D11	13.5	14							
13	D12	12.5	13.5							
14	D13	10	12.5							
15	D14	8	10							

The fields that are used are defined at the top, as column headers. These must reflect the names of the fields in the list that you wish to display.

The row name, defined in the first column, must be prefixed with a special letter and then an incrementing number. The special letter will be discussed in a moment. Suffice to say any letter can be chosen.

Sample numbers can be entered in as the values of the chart. These values will be displayed in the chart in the template file, but when the report is run the values will be replaced with actual data.

Once the report is inserted, a special tag command needs to be inserted in the paragraph preceding the chart. The command, SeqToNode, converts the sequence (list) of records specified into a structure (a node, or record) that can be used for the chart.

As the parameter to the SeqToNode function you specify the special letter symbol (in our example it was D), followed by both the field containing the list, a colon and the field that will be used for the Y axis of the chart. For example:

```
para:=SeqToNode('D', TaskList:DateEntered)
```

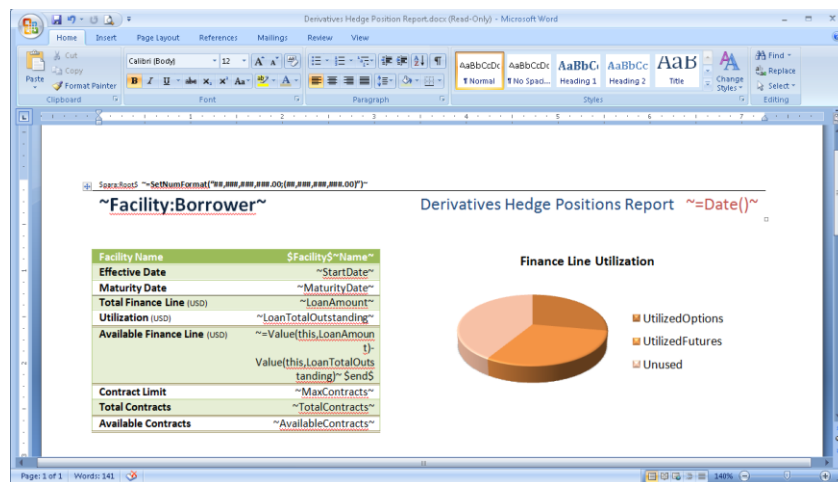
Underneath the chart, that tag must be closed with a \$end\$ tag.

The fields specified in the column head of the edit data spreadsheet must then correspond to the fields you wish to plot in the TaskList sub-field.

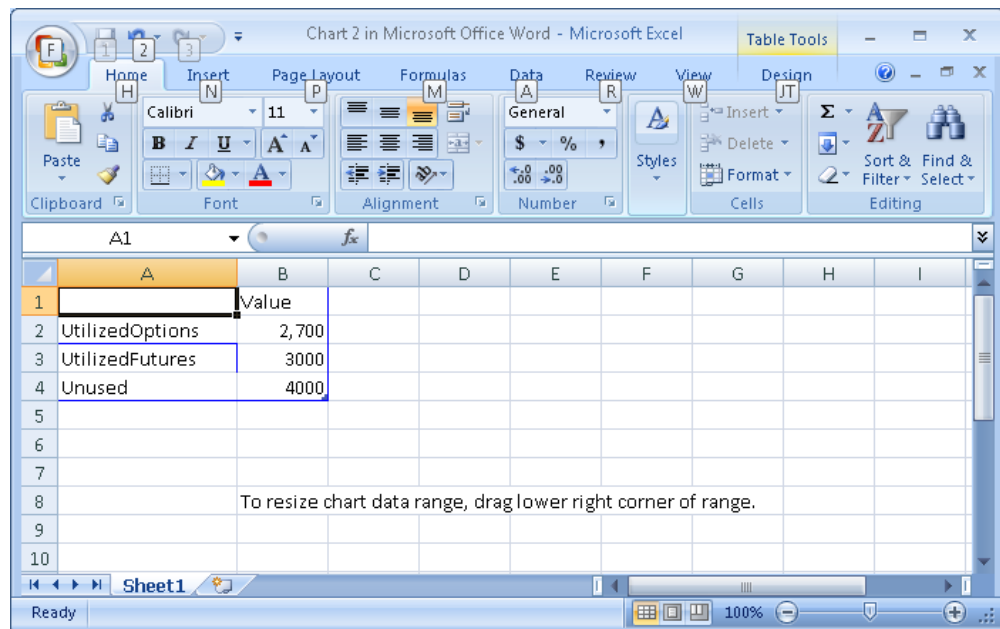
9.4.4.2. Pie Charts

Pie Charts are simpler to create than line charts. They assume that the fields you wish to represent in the pie chart are present in the context of the location of the chart within the report.

No special functions are required around the pie chart:



And the way in which the data is edited is simpler:



In this case the column header name is always going to be 'Value'. The row names reflect the field names you wish to show in the pie chart. The values can, as before, be any sample value you wish to show in the report.

9.4.5. Adding the Report to the System

Once the report template is created, you need to add the report into the system.

There are two places that the custom report can be added:

- For global reports that are accessible from anywhere, you add the report under the Application record, in the Custom Reports drilldown.
- The reports that are specific to a certain part of the data model, go to the specific field and then to the Custom Reports drilldown of that field definition.

Once you are at that list, click 'Create New' to add a new report.

Figure 51: Blank Custom Report screen

The following fields can be entered:

Title

This is the title of the report as it will appear in the reports list.

Icon

This is an optional graphical icon that will be displayed alongside the title where the report is made available.

Report File

Specifies the report template file, usually in Word 2007 (.docx) format, or an HTML file (.htm). Can also be a .txt file.

Input Type

Specifies the field definition in the data model that will be used to provide the report parameters screen, if any.

File Extension (Override)

Specifies any override for the output file format by specifying a new extension.

Editable Input

Specifies that the report input, specifies in Input Type, will be entered by the user.

Data Transform (Plugin)

Specifies the optional plugin that will be used to provide a data transform action. Data transform actions are discussed in the section below.

Data Transform Action

After selecting the plugin above, this is where the actual action is selected.

Query Path

For global reports, specifies the path where the data for the report will be retrieved.

Description

Provides an area to enter any description of the report. This is not shown to the user.

StringTable Base Path

Provides an optional location of an existing StringTable that will be re-used in this report. StringTables are discussed in the next few chapters.

Condition

Specifies a simple condition that must evaluate to True for the report to be available and displayed in the reports list.

Is Primary?

Indicates if the link to run this report will be displayed more prominently on the screen.

Allow Users to Subscribe?

Indicates if an area should be created allowing people to subscribe to this report, and have it generated and emailed to them on a specific schedule.

9.4.6. Configuring the Report's Data Query

As explained earlier, the purpose of the report is to take data and display it in a document. We'll now be discussing what this data is and where it will come from.

When the report runs, by default the current record is queried before being used in the report. Any short lists underneath the record are also downloaded and incorporated into the results automatically.

To incorporate additional, long lists that exist under the record, you will need to specify the filter to use on that list, to ensure that too much data isn't being downloaded.

To do this, you specify the filter in the 'Default Filters' drilldown. Then click 'Create New' to add a new filter:

The following fields can be entered:

Field

Specifies the field inside the subtype from the current record, which should be a nested type of a long list, that will be downloaded.

Operator

Specifies the operator of the condition that will be applied to the specified field.

Value

Specifies the value to compare against. Alternatively a 'Special Value' can be selected.

Special Value

Specifies an alternative way to determine the value that will be part of this condition. Can be one of the following:

- **Today**
For date fields, compares against today.

- **Today (Plus Days)**
For date fields, compares against today + days, where Value specifies the number of days to add from today.
- **Today (Minus Days)**
For date fields, compares against today - days, where Value specifies the number of days to deduct from today.
- **Retrieve All**
Retrieves all values in the list without any filter. This is the most common option to select.

9.4.7. Example

In this example we will be assuming the following structure:

1. Team
 - a. Name
 - b. TaskList
 - i. Title
 - ii. Status
 - iii. AssignedTo
 - iv. DateEntered
 - v. Log
 1. Date
 2. Text

In this structure, Team is a top level list of teams. Team has a Name. Inside the Team is a Task List, and each task contains a Title, Status, AssignedTo and DateEntered field. There is also a Log inside a task that contains a list of log entries each with a Date and Text for the log.

Because both TaskList and Log are nested types that are 'Long Lists', they won't be downloaded automatically before the report is run, so we will need to set up a Default Filter for them both.

In both cases we will need to have a Default Filter that is set to 'Retrieve All' because we want all of the records to display in the report. So we create two Default Filters with the following values:

1. First Default Filter:
 - a. Field = Team / TaskList
 - b. Special Value = Retrieve All
2. Second Default Filter

- a. Field = Team / TaskList / Log
- b. Special Value = Retrieve All

You would then create a template such as this:

```

$para:Root$
Team: ~Name~



| Date               | Title   | Status   | Assigned To  |
|--------------------|---------|----------|--------------|
| \$table:TaskList\$ | ~Title~ | ~Status~ | ~AssignedTo~ |
| ~Date~             |         |          |              |



| Log     |         |  |                  |
|---------|---------|--|------------------|
| \$Log\$ | ~Title~ |  | \$end:Log\$      |
| ~Date~  |         |  | \$end:TaskList\$ |



$end:Root$

```

The first couple of lines start the Root tag and output the team name. The table underneath that contains the Task List, with the appropriate fields.

Embedded within the Task List is a sub-table for the Log.

The output from this template might be something like:

```

Team: Development



| Date     | Title               | Status | Assigned To |
|----------|---------------------|--------|-------------|
| 4/5/2010 | Write Testing Guide | Open   | Sam Smith   |



| Log       |                                                 |         |         |
|-----------|-------------------------------------------------|---------|---------|
| 4/5/2010  | Handed to development manager to assign         |         |         |
| 4/7/2010  | Assigned to Sam, Jack is on vacation            |         |         |
| 4/10/2010 | Refactor XT Project                             | Started | Kim Tan |
| Log       |                                                 |         |         |
| 4/11/2010 | Need clarification on requirements              |         |         |
| 4/12/2010 | Talked to Sam, he's writing up the requirements |         |         |


```

Note: Be very careful to ensure that every start tag has an associated end tag. If the start and end tags are mismatched then the resultant report file may become corrupt.

9.5. Report Subscriptions

9.5.1. Overview

Report subscriptions allow users to have any report in the system generated automatically on a specific schedule, and have that report automatically emailed to the user at a certain time.

Quick Reports, Tickler Reports and Custom Reports all have a simple flag that needs to be set in order to enable report subscriptions. For Quick and Custom Reports, in the main record there is an option called 'Allow Users to Subscribe', and Tickler Reports are always enabled for subscription. There may also be other reports, created automatically by the system, that will also be setup for subscriptions.

9.5.2. Setting up a Report Subscription

Reports that are normally run from a particular record will have an additional drilldown called 'Report Subscriptions'.

Reports that are run globally (that is, from anywhere in the system) have their subscription area placed under the Configuration record, in a drilldown called 'Report Subscriptions'.

Once you are at the subscription list, click 'Create New'. You should see a form like this appear:

Report Subscriptions

Report Subscriptions

User: John Wood

Report Type: Keep-In-Touch Report

Language Code: en-US

Frequency: Daily

Time Of Day: 9am

Create

The following fields can be entered:

User

In this entry box you should select your user name, or the user name of the person who wishes to have the report emailed to them.

Report Type

Specify the report to which the user wants to subscribe. All reports that have subscriptions enabled for this area (whether a record or globally) will appear in this list.

Language Code

Specify the language in which the report will run. By default this is set to US English.

Frequency

Specify the frequency with which you want this report to be generated. The available options are:

- Daily
- Every Week
- Every 2 Weeks
- Every Month
- Every 3 Months
- Every 6 Months
- Every Year

Time Of Day

Specify the time of day you wish the report to be generated.

Once the report is scheduled to run, you will be notified the next time the report will run. This will usually be the next day.

9.5.3. Managing Report Subscriptions

All pending report subscriptions are managed in the 'Scheduled Reports' drilldown from the Configuration record.

These are scanned by the system every 15 minutes to check whether any new reports need to be generated.

In this list you will find all the reports that will be run in the future. An administrator can edit these scheduled reports and change the date the next report will run. This will be picked up the next time the scheduled reports are scanned.

9.5.4. Email Format

Once the report is generated, it will be attached to a new email that will be sent to the user's inbox. The subject line will specify where the report ran and the title of the associated record.

10. Creating Custom Views

10.1. Overview

While dbFramework creates sophisticated display screens, input forms and field editors automatically, in some cases you may need more control of how the screen looks and how the user interacts with the data. To allow for this, dbFramework enables the user to create additional screens to either complement or replace the default screens. These are referred to as Custom Views.

Custom Views, also referred to as ‘Screens’, appear either to replace the entire record or record list, or they appear as optional ‘tabs’ above the record or record list.

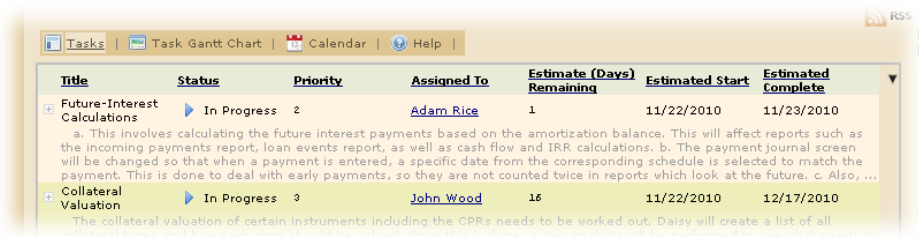


Figure 52: Showing multiple screen options above the record list.

There are a variety of ways to create custom views, but they can be split into either report views or HTML views.

Report views are created the same way that reports are created, which you can find out about in section 9. However, when creating the report you simply select ‘Screen Option’ in the ‘Custom View’ entry box in that form. This will automatically create a custom view that will display that report in-situ on the screen.

Because reports can be quite sophisticated and aggregate a lot of information into summaries and graphical charts, using reports for custom views is an easy and efficient way to create dashboards, which we’ll be discussing in this chapter.

The second and more flexible means to create a custom view is to design some custom HTML that will be displayed in the screen.

HTML is incredibly flexible, allowing for very dynamic layout and JavaScript to enable transitions, animations and powerful interactions. Using HTML you can also embed either Adobe Flash or Microsoft Silverlight or ActiveX components.

Depending on the browser you are targeting you may be able to use more modern HTML technology such as HTML 5, which allows for animated transitions and other graphical effects. The possibilities are endless.

The HTML code will utilize either the JSON API or server-side DbfScript API that dbFramework provides for every application, allowing you to easily retrieve any data from your application to display in your custom view.

Note: Any tickler reports you create will automatically become available as a custom view.

10.2. Custom Record View HTML

10.2.1. Designing the HTML

Note: You will need to have some basic understanding of HTML to create custom views in HTML. You will also have to be sensitive to HTML compatibility problems that exist between different browsers.

The way that an HTML page for a custom view is designed is identical to the way you would generate a custom report, however because the custom view is typically more interactive than a report you would normally print out, there will be some additional functions you can use to help the user interact with the data values.

To learn about creating custom report files, go to section 9.4.

An example of a custom view can be found in the Scrum Master sample application:

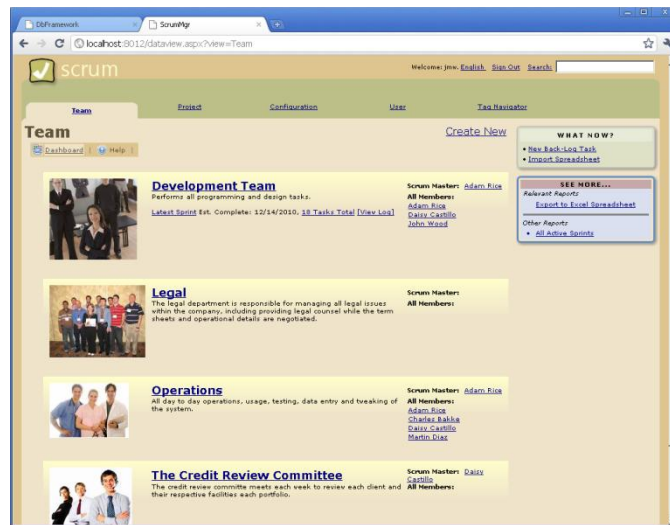


Figure 53: Example of a custom view HTML page

The data input to the report depends on the type of data the view is being used to display. For a Record view, the input will be record data itself. For a List view, the input will be record sequence. Both of these are accessed through the `$Root$` tag. The content between the `$Root$` tag and the corresponding `$end$` tag will be repeated for each record that is being displayed. Individual fields in the record can be accessed by putting the field name (not the title) in tilde characters (eg. `~Name~`).

Tip: If the value contains HTML that you want to integrate into the HTML page, you will need to prefix the name of the field with an ampersand (&). This disables formatting of the field and allows HTML tags to be displayed without modification.

Any subtypes, of records or sequences, can be expanded in the HTML by inserting the name of the subtype in dollar symbols (eg. `$Tasks$`). All of the HTML between the subtype command and the corresponding `$end$` command will be repeated for each of the records in the sequence. Any tags within those commands will be expanded according to the content of that record, and so fields (surrounded by tilde symbols) can be referenced that relate to the contained record.

In addition to subtypes, you can also put Boolean valued fields into dollar sign tags, and the text between those tags will only be expanded if that Boolean value evaluates to True. If the Boolean field name is prefixed with an explanation mark, the text will only be expanded if the value is False.

There are three special fields that can be used at the top level of the custom view template file:

~EditLink~

This is the URL to the edit page for the current record.

~DeleteLink~

This is the URL to the delete page for the current record.

~ListLink~

This is the URL to the record list page.

Functions can also be added by putting DbfScript in tilde characters, after an equals sign. Most of DbfScript can be used except for program control keywords (eg. if, loop, while). For example:

```
~=#input/Age + 20~
```

Although described in the section on creating custom reports, the following functions are particularly useful in creating interactive custom views:

FileLink (field)

This function enables you to create a URL link to a specific file in the database, specified as a path and a field. This file can be used to display in an image, or an audio tag for example. It can also be used to create a download link.

ViewLink(path)

This function creates a URL link to the specified XPath.

ViewForeignLink(link)

This function creates a URL link to the path that is held in the given foreign link value.

GetSmallIconHtml(iconID, backColor)

This function is useful for integrating icons into your custom view. The icon is rendered against the specified back color (the icons have an alpha-transparent background).

10.2.2. Enabling the Custom View

To create and enable the custom view, you will need to go to the field or type in the data model where you want the custom view to appear.

From there you should drilldown to the 'Custom Views' goto item. You can create any number of custom views for a given type.

Once you are viewing the custom view list click 'Create New'. Here's a rundown of the fields you'll need to enter:

Title

The title of the custom view, that is displayed in the tab that appears above the data record area.

Type

This defines when the custom view will be made visible.

- **Header:** This means that the HTML is displayed above the data record or data list but does not show any data from the data record. This can be used to display hints or other banner-like information.
- **Footer:** Similar to 'Header' but shows the information underneath the data record.
- **Side (View):** When the record is displayed in the parent view (underneath the reports list), this HTML will be used to display the information.
- **List (View):** When multiple records are listed (using List All), this HTML will be used to display the data.
- **Record (Edit):** When the data record is being edited, this HTML will be used to display the edit form.
- **Record (View):** When the data record is being displayed, this HTML will be used to display the record.

Enabled

Provides a means to enable or disable this view without having to delete it.

Icon

The optional icon that will be displayed next to the custom view option wherever it is displayed.

File

The actual HTML file that will be uploaded for this custom view.

Default

Indicates if this custom view will replace or complement the default view that was automatically created by dbFramework. If this is set to 'Yes' then the automatically generated custom view will not be visible, and only this custom view will be available.

Support Files

In addition, once the custom view record is created you will be able to upload additional, support files that can be referenced by the HTML. For example you may want to upload images that can be used in the custom view.

10.3. Using Quick Reports to Create Dashboards

You can also use Quick Reports to create custom views very quickly. Any Quick Report that you create can be set to appear as a custom view by setting the 'Custom View' option in the Quick Report to either 'Screen Option' or 'Main Screen'.

Because Quick Reports allow you to easily combine multiple lists and charts into one screen, you can use those reports to put together a dash board for an opening screen in your dbFramework application.

Note: More complex reports, or reports that incorporate graphs, may take a small amount of time to generate. This should be kept in mind if the report is used as a replacement view as it may slow down navigation when using the application.

10.4. Custom Field Viewers and Editors

Sometimes you don't want to create a custom view for the entire record or record list, but you just want to have one particular field display or behave differently to the others. You can achieve this in dbFramework by creating Custom Field Viewers.

A Custom Field Viewer comes in two flavors: the first is the viewer HTML which is used to render the field value in its entry box. The second is the editor HTML, which is displayed in a dropdown, or in-situ for fields that are of 'Text (Paragraphs)' data type.

You can see an example of a custom field viewer by looking at the icon selector or color selector in dbFramework. You can see an in-situ custom field viewer in action when you go to edit a script in dbFramework.

Unlike Custom Views, Custom Field Viewers are defined centrally and can be re-used across fields anywhere in the application.

To create a custom field viewer, first go to the Application record, then go to the drilldown 'Field Viewers'. Then click Create New.

The following fields will need to be entered:

Title

This is the title of the custom field viewer that will be displayed in the dropdown list when selecting the field viewer.

SubFolder Name

This is the subfolder from the application file folder where any support files will be stored.

Dropdown Width

For custom field editors that will show dropdowns, this indicates the width, in pixels, that the dropdown window will take up.

Dropdown Height

For custom field editors that will show dropdowns, this indicates the height, in pixels, that the dropdown window will take up.

Enabled

By disabling a custom field viewer, all fields using this viewer will revert to the standard viewer and editor.

View HTML

This is the snippet of HTML that will be used to display the field value when viewing the record. Any references to ~VALUE~ are replaced with the current field value, and any references to ~FIELD~ are replaced with the current field name.

Icon Selector ▼ (1 / 1)

Title: SubFolder Name: Dropdown Width: Dropdown Height:

Enabled: ☒ Yes

View HTML

```
<div style="position:relative; overflow: hidden; width: 16px; height: 16px">
</div> <script type="text/javascript"> var el = document.getElementById("iconview");
el.style.left = (-parseInt("~VALUE~".substring(0,4), 10)*16).toString() + "px";
el.style.top = (-parseInt("~VALUE~".substring(4,8), 10)*16).toString() + "px";
</script>
```

Editor HTML File

No file chosen

More sophisticated displays can be created by incorporating JavaScript into the HTML. Within the View HTML you can include JavaScript by inserting a `<script>` tag next to the value. By defining a `<div>` block, the script can access the div and use code to determine how the value should be displayed.

Editor HTML File

The editor HTML is provided in a separate file because the HTML for editors is usually more complex than that used to view the field value.

Identically to the view HTML, any references to `~VALUE~` are replaced with the current field value, and any references to `~FIELD~` are replaced with the current field name.

The `~FIELD~` value is important in the editor HTML because it is needed to set the value in the field once the user is done editing the value. A special function needs to be called to set the field value: `parent.setFormFieldValue`.

parent.setFormFieldValue (fieldName, display, value)

The fieldname would normally be entered as:

```
"~FIELD~"
```

The 'value' field should be the value you want the field to hold. The value for 'display' should be the same.

This function should be called when the user finishes editing, perhaps when they click a button or double click their selection. It will also close the dropdown window.

If the field editor isn't a dropdown list and is displayed in-situ, then that function may need to be called whenever the user makes any changes in the editor.

10.5. Skins

While you can easily create customized views for individual records or fields, sometimes you may want to change the way the entire application looks. You can do this using a feature of dbFramework called 'Skinning'.

One example of skinning is the mobile phone interface for your dbFramework applications. When dbFramework or its applications are accessed through a mobile phone, the interface is replaced with a significantly simpler and more primitive site that is compatible with the simple browser available on most phones, and also suitable for access by smaller screens that cannot accommodate the large space required by the regular interface.

dbFramework supports multiple skins which can be selected by a simple parameter in the URL. You could have a special skin designed that is optimized for the BlackBerry device, or you may have a special skin for users with special needs such as color-blindness. Alternatively you can replace the default skin that is used allowing you to have a completely different look for your entire application for all users.

For more information on creating new skins, see section 16.

11. Enterprise Features

11.1. Overview

The 'Enterprise Features' of dbFramework refer to the features that are useful, if not critical, to line-of-business applications. These features are focused on day-to-day operational requirements of businesses to quickly enable common workflow procedures, for example, add collaborative or other community oriented features, or perhaps show different charts with minimal effort. However many of the features could be useful for any type of application.

Most of the features in this category have been put into a special section under a field definition called 'Features'.

Each of these features provide a wealth of functionality with what is often just a single click. When designing the application, you can set these features on or off, and when you republish the application, the functionality will be made available automatically.

Also keep in mind that these features can be enabled *anywhere* in the data model, even in an area of the database that is buried several levels deep.

This section will go through these features and explain what they can do, how they work, and how they can be configured so you can make the most of them in your own line-of-business applications.

Warning! Once you have published your application with features enabled, and information was entered into the feature-related structures, when you switch that feature off and publish the application any data you entered relating to that feature will be deleted, and cannot be recovered unless the database is first backed up. Use the feature list with caution!

11.2. Approval Workflow

11.2.1. Overview

Workflow refers to a business process that has been implemented in a computer system. dbFramework's powerful workflow engine makes implementing even relatively sophisticated workflow a very simple process.

The most common workflow scenario involves routing submissions or change requests to a group for approval. This may be important for business-critical data, where you want to have another user or a superior verify the data before it goes into the system. Or perhaps the user is new and is in training, and they're not quite ready to take the reins.

This type of approval workflow can be enabled in your dbFramework application with just a few clicks.

Note: For a variation on this workflow, where everybody in a specific user group must vote on a change, and where the verdict is recorded with the document, consider using the Document Review workflow feature, described in section 11.13. The two workflow features can also be combined.

11.2.2. Scenario

Consider the following simple scenario. There is an office, Bill is the boss, and Ellie is the employee.

1. Bill doesn't want anything going into the system without a manager's approval.
2. Ellie needs to submit an invoice she received from a vendor
3. Ellie logs on to the system, clicks the 'Submit Document' and selects the invoice.
4. Once Ellie clicks 'Submit', a message appears saying that the operation requires approval before it can be committed.
5. The system sends out an email to all the managers in her department asking for approval, including Bill. The details entered by Ellie are copied into the email, along with the document itself.
6. One of the managers checks the invoice and approves it. The system sends an email to the originator (Ellie) saying that the request was approved.
7. The document now appears in the system.

11.2.3. Concepts

The approval workflow system in dbFramework is based on two areas: permissions and governance.

The fact that the approval workflow is permissioning-based means that some users may be subjected to approval, whereas others may be able to enter documents immediately. This all depends on how the permissions are configured for the user.

By default all users are excluded from the approval requirement. However this can be easily changed by simply editing the permissioning role assigned to the user.

The second area related to approval workflow is the governance. By this we say that a specific user group governs changes to a specific area of the data model and database. For example we could say that Human Resources governs all changes made to the 'Employee Vacations' list. Adding a governance is just a matter of adding a new record under the user group.

In addition to this, a governance can also be conditional. This is, in fact, how the workflow engine becomes very powerful: you can specify that a certain group requires their approval before a specific *type* of change is made to an area of the database. For example, if an expense greater than \$100 is submitted then this may require special approval by the department head.

Also multiple user groups can govern one area. Once anybody in one user group accepts the request, it will then be sent to the next user group for approval. If they deny the request the whole operation will be aborted and the user notified.

The combination of these two enables approval workflow. Any area of the application that the user doesn't have permission to access directly, and is governed by a specific user group, will be subject to approval before any change can be committed. The user interface for informing the user, or sending the emails, and providing forms to accept, decline or cancel a request, is all generated automatically by dbFramework without the requirement of any programming at all.



Figure 54: System message informing the user that the change must be approved

11.2.4. Configuring Approval Workflow

11.2.4.1. Updating Permissions

For more information on understanding how user roles and permissions are configured, please see section 11.24.

The security roles for the application are listed under the Application record, under 'Roles'. When editing a 'Role', create a Permission in that role. The available options will look something like this:

LenderEx1 Facility Workflow

Permissions [Create New](#)

Facility ▼ (1 / 1)

[List All](#) [Detail](#) [Edit](#) [Delete](#)

Field	Apply to Sub-Types	Specific XPath	Allow / Deny
Facility	Yes		Deny
Apply To...			
View	Update	Create	Delete
No	No	No	No
Owner Only	Approval Workflow Exclusion		
No	Yes		
Default Filter			
Filter Field	Value	Expression	

One of the options will be 'Approval Workflow Exclusion'. By default all users are excluded from approval workflow. To force approval workflow, this permission must be denied. So you must set the 'Allow / Deny' to 'Deny', and set the 'Approval Workflow Exclusion' to Yes (to apply the denial to that option).

Once the Role is created, and the application republished, you can go to the User record in the application and add that Role to the user.

11.2.4.2. *Setting Governance*

For more information on user groups, see section 11.24.

User groups are an optional structure that can be configured in any dbFramework application. They consist of a group of users under a title. A user can exist in more than one group. For example you may have groups for:

- Different departments
- Different ranks in the company, such as managers, new employees, senior or junior workers.
- Special cross-cutting groups, such as the disaster recovery team.

Under the user group, the drilldown 'Governance' allows a user group to be assigned governance over multiple areas of the data model.

To configure this, find the User Group (under Settings, User Groups), and then click on 'Workflow Governance'. Click 'Create New' to add a new area that will be governed by this user.

The screenshot shows a web-based form titled "Workflow Governance". The form is divided into three sections: "Path" (a large text area), "Condition" (a small text box), and "Notes" (a large text area). A "Create" button is located at the bottom right of the form.

The path that is entered here is in an XPath format, similar to the way you might refer to folders on your desktop computer. This concept was first introduced in section 6.14. The path entered here, though, is quite simple to formulate.

The start of the path must always be a forward slash ('/'). The next part is the name of the view, or the top-level field in your data model.

The governance affects all areas of the data model underneath the specified path. So, for example, if you specify /Facility and there are drilldowns from that type for

Documentation, then the governance will affect the Documentation as well as anything underneath that.

The path can also refer to specific records, rather than just general types. If the path contains a filter (known as a *predicate*), the governance will apply for that record only.

For example, the path can be:

```
/Team[Name="Development"]/Log
```

And this would make this user group only govern changes to the Log in the Development team, and not any other team.

Note: The filter can only apply to a single record at present.

Next is the **Condition**. The condition is an optional DbfScript expression that must return either true or false. If the condition is omitted, then the governance always applies. If the condition is specified, then the governance only applies if the condition expression evaluate to true.

The input to the expression is the record being changed. Therefore you may need to utilize the Exists function to determine if a specific field is present in the update before checking any values.

For example you may have a condition such as:

```
Exists(#input/Limit)
```

...and this would check whether the Limit field was present in the update. You could also have something that checks for specific values or ranges:

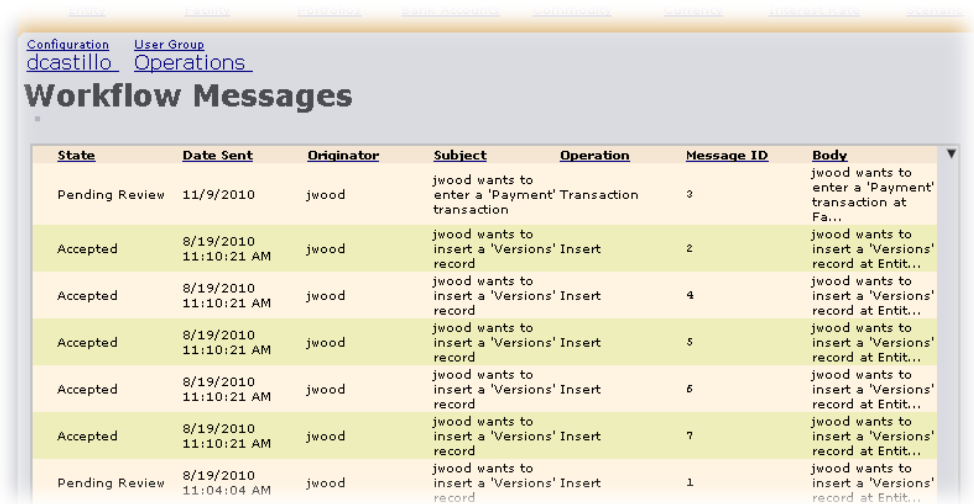
```
#input/Limit > 50000
```

Each workflow governance applies to all types underneath that type in the hierarchy. However, if a condition is specified then it will only apply to the type at the path specified.

Of course this will only enforce approval workflow if the user making the change is denied the approval exclusion permission, as discussed in the previous section.

11.2.5. Reviewing Workflow Messages

All requests that go through the approval system are stored in a special folder under each user group called 'Workflow Messages'.



The screenshot shows a web application interface with a navigation bar at the top containing links like 'Home', 'Users', 'Groups', 'Workflow Messages', 'Transactions', 'Reports', 'Administration', and 'Settings'. Below the navigation bar, there are links for 'Configuration' and 'User Group', with 'dcastillo' and 'Operations' selected. The main heading is 'Workflow Messages'. Below this is a table with the following data:

State	Date Sent	Originator	Subject	Operation	Message ID	Body
Pending Review	11/9/2010	jwood	jwood wants to enter a 'Payment' Transaction transaction		3	jwood wants to enter a 'Payment' transaction at Fa...
Accepted	8/19/2010 11:10:21 AM	jwood	jwood wants to insert a 'Versions' Insert record		2	jwood wants to insert a 'Versions' record at Entit...
Accepted	8/19/2010 11:10:21 AM	jwood	jwood wants to insert a 'Versions' Insert record		4	jwood wants to insert a 'Versions' record at Entit...
Accepted	8/19/2010 11:10:21 AM	jwood	jwood wants to insert a 'Versions' Insert record		5	jwood wants to insert a 'Versions' record at Entit...
Accepted	8/19/2010 11:10:21 AM	jwood	jwood wants to insert a 'Versions' Insert record		6	jwood wants to insert a 'Versions' record at Entit...
Accepted	8/19/2010 11:10:21 AM	jwood	jwood wants to insert a 'Versions' Insert record		7	jwood wants to insert a 'Versions' record at Entit...
Pending Review	8/19/2010 11:04:04 AM	jwood	jwood wants to insert a 'Versions' Insert record		1	jwood wants to insert a 'Versions' record at Entit...

Figure 55: List of workflow messages

This lists all of the requests that were sent, along with information on the request. The following fields are in the list:

- **State**
The state of the request, whether it is pending, it was accepted, rejected or canceled.
- **Date Sent**
The date the request was sent, which usually correlates to when the action was performed.
- **Originator**
The user name of the person who performed the original action causing the workflow request.
- **Subject**
A short description of the operation that was attempted.
- **Operation**
The type of operation, an Insert, Update, Delete or Transaction.
- **Message ID**
An ID that uniquely identifies this message. Messages can be retrieved by their ID using the Search capability (see section 14.11)
- **Body**
The full description of the action that was attempted. This will include

details of where the operation was occurring, and all of the information entered by the user when the operation was attempted.

- **User Notes**
An optional section for entering notes made by the user.
- **Accepted By**
The name of the user that accepted the request, if any.
- **Accepted Date**
The date the operation was accepted.
- **Reviewed By**
The name of the user that reviewed the request.
- **Last Emailed Date**
The date that the system sent an email regarding this request.

There is also a File Attachments drilldown from the message that contains any file attachments in the original request.

11.2.6. Approval Workflow Report

The list of approval workflow messages for the past 24 hours can be viewed in a report available from any user group called 'Approval Workflow'.

[illegible]

Figure 56: Daily Approval Workflow Report

This report provides much of the information held in each workflow message, in a format that is easily viewable and shows a clear chronology of events that have occurred over the past 24 hours.

By using report subscriptions, any user can subscribe to have this report automatically generated and emailed to their inbox each morning. The user receiving the report does not even have to be in that user group.

To configure this, go to the Settings tab of your application, go to Report Subscriptions, click Create New and add the user with the report 'Approval Workflow' selected. Because this report shows the past 24 hours of activity, it should be set up to generate daily. It may be best to have the report sent out at the beginning of each day so you can see all activity the prior day.

The same workflow report can also be viewed from within the user group screen, by clicking the 'Approval Workflow' screen tab at the top of the user group record.

LenderEx
Loan Administration and Factoring Management System

Welcome: jmw_workflow English French Spanish Sign Out Search

Loan Administration and Factoring Management

Contact Facility Portfolio User Configuration Wiki Test

Configuration asus:8030

User Group Create New

User Group Approval Workflow

Transactions for the past 24 hours.

Note: Only showing transactions that passed through the approval workflow system.

Date Requested	State	Request Description	Originator	Operation	Request ID	Reviewed By	Date Accepted
11/22/2010 8:30:22 PM	Pending Review	jmw_workflow wants to update a 'Facility' record	jmw_workflow	Update	3		
jmw_workflow wants to update a 'Facility' record at Facility: JHL Trucking with the following details: Description: This is a test loan that was set up for demonstration purposes. Last Modified: 11/22/2010 8:30:22 PM Last Modified By: jmw_workflow							
11/22/2010 8:30:53 PM	Pending Review	jmw_workflow wants to update a 'Facility' record	jmw_workflow	Update	3		
jmw_workflow wants to update a 'Facility' record at Facility: JHL Trucking with the following details: Facility Type: Loan Cash Loan: Yes Start Date: 6/27/2010 Maturity Date: 11/2/2013 Description: This is a test loan that was set up for demonstration purposes. Last Modified: 11/22/2010 8:30:53 PM Last Modified By: jmw_workflow							

Configuration asus:8030
List All Detail

WebServer Name: Email Server
asus:8030 mail.optonline.net
From Email: support@riaform.com

User List Workflow Messages Workflow Governance Report Subscriptions Create New

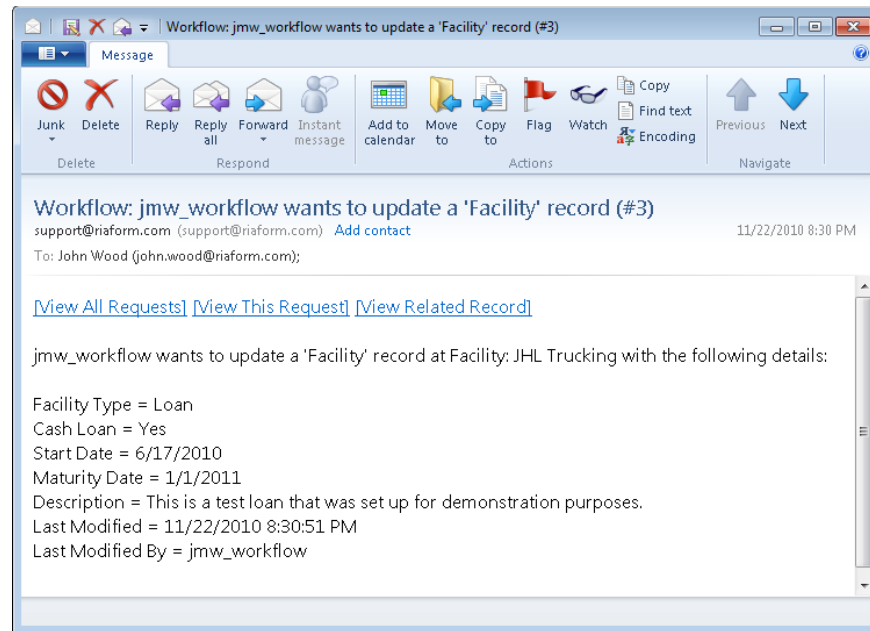
11.2.7. E-Mail Integration

Every time an approval is required, which is every time a new workflow message is created, the application automatically sends out an email to the user group to let them know that the request is pending.

Note: This function uses the email server configuration in the Config tab of the application in order to know how to send the email. If the Configuration record is not created in the application, EmailUser and EmailUsers will return an error. In addition, the user record must have an email address specified or they will be excluded from the email.

Any user in the group can respond to the pending request. The originator can also cancel the message, and this will send another email out to the group to inform them the request was canceled and should no longer be reviewed.

The email for a request will look something like this:



The top line consists of a number of links:

- **View All Requests**
This takes you to the Workflow Messages list for your user group. This may be useful if you know there are a lot of messages outstanding.
- **View This Request**
Takes the user to that specific request in the web browser, allowing them to accept or reject the request.
- **View Related Record**
For records being updated or deleted, this will take you to the original record before the change has been made.

Underneath these links is a statement specifying what the user is requesting and where in the database it was invoked.

Below that statement there will be a description of the data that is being updated. This data may be hierarchical and so may contain sub-records that will be displayed indented.

Also, any files that were attached to the original request will be attached to the email message for easy viewing both on and off-line.

Note: In this description of the record, certain values may not be displayed if they are deemed sensitive (password changes, or social security numbers). This is controlled based on whether the flag 'Hide Content' is set on the field being displayed.

11.2.8. Reviewing the Request

When clicking the link to view the request, you will see a screen similar to this:

The screenshot shows a web interface titled "Workflow Messages" with a timestamp of "11/22/2010 8:30:51 PM". At the top right, there are buttons for "Accept", "Reject", and "Cancel". Below the title, there are links for "List All Detail" and "Goto File Attachments". The main content area displays a message with the following details:

State	Date Sent	Originator	Subject
Pending Review	11/22/2010 8:30:51 PM	jmw_workflow	jmw_workflow wants to update a 'Facility' record

Below the table, the "Operation" is "Update" and the "Message ID" is "3". The "Body" of the message states: "jmw_workflow wants to update a 'Facility' record at Facility: JHL Trucking with the following details:"

- Facility Type = Loan
- Cash Loan = Yes
- Start Date = 6/17/2010
- Maturity Date = 1/1/2011
- Description = This is a test loan that was set up for demonstration purposes.
- Last Modified = 11/22/2010 8:30:51 PM
- Last Modified By = jmw_workflow

There is a section for "User Notes" with a table for tracking the request's status:

Accepted By	Accepted Date	Reviewed By	Last Emailed Date

At the bottom, there is a "Dates" section with a table:

Last Modified	Last Modified By	Created	Created By
		11/22/2010 8:30:51 PM	jmw_workflow

You can then click the 'Accept' button to accept the request and have it commit, or the 'Reject' button to decline the request, in which case you will also have to write an explanation which will be recorded against the request and included in an email to the originator.

The originator can view this request record and cancel their request at any time. Again an explanation must be given, and this will be recorded against the request and included in any emails or reports relating to the approval workflow.

Note: Even if the user is subject to approval workflow for that area, if no governing user group is found for the area being changed the change will **not** be subject to approval and will be applied to the database immediately.

11.2.9. Advanced Workflow Concepts

It's also possible to combine the approval workflow with other workflow features to produce more complex workflow processes.

For example, to trigger the workflow you can create an input form and simple script to update and save the record. Even though the operation originated from within a script, running that action will still result in the action being sent through

workflow so long as the user is subjected to workflow for any one of the records being updated.

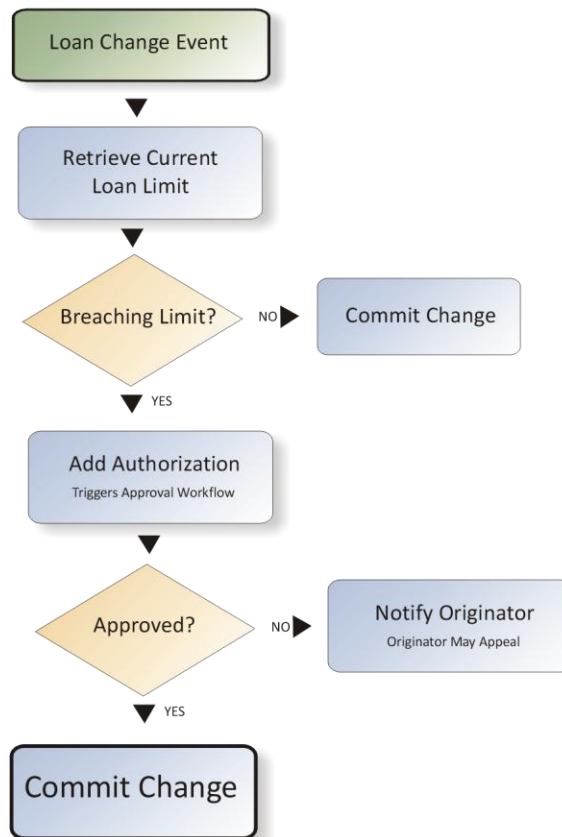


Figure 57: Example of combining features to create advanced workflow.

The same occurs if any synchronous events from the update touch records that fall inside an area requiring approval. In this case the entire transaction will be submitted for approval, even though only one part of the transaction, or a consequence of the transaction, requires approval.

By combining simple scripts and forms with the approval workflow configuration, many complex workflows can be modeled. In addition, the Document Review (section 11.13) and Data Entry Reminders (section 11.9) along with many other enterprise features and security restrictions can easily be combined to create a very powerful workflow process.

11.2.10. Example: Expense Approval

Let's take an example of an expense approval system.

The intention is to create a system that records expenses. However those expenses must be approved by the user's line manager. To achieve this we will only have to use two features:

1. Security restrictions to ensure users only have access to their department's expense records.
2. Approval workflow to ensure that nothing goes into that department's expense records without prior approval by both the line manager and accounting.

We first create the lists for the expenses. This can be a simple list that contains information such as:

- The employee
- A short name for the expense.
- A category, from a dropdown
- The amount
- Any taxes
- A project reference

We then create one of these lists for each department. We do this using the 'Existing Type' option when creating the lists, to re-use the expense report type.

Then we set up the users for the employees, along with a security Role for each of the departments we created. We'll have two roles – one that lets users edit the expenses freely, and another that enforces approval workflow. We assign the roles based on whether the user is a line manager for the department or an employee in that department.

Then we create the user group for each department's heads. In this we define the workflow governance by specifying the path to the expense report area. This ensures that these department heads will be needed to approve changes in that department expense list.

In addition to that we'll create a user group for Accounts, and have them govern all expense reports.

Each user is then given a link to their respective department's expense report list where they can enter their expense reports.

- They will only have access to enter an expense report in the area in which they have access.

- When they create an expense report, it will notify them that the department head will need to approve the request.
- Once the department head approves the request, they will be notified that the Accounting group will need to approve the request.
- Once that is approved, the originator will be notified and the expense report will be created in the expense list.

Finally, a report can be created using the Quick Reports that shows all expenses for the past week. By enabling user subscriptions, all interested users in each department can subscribe to this report to get a weekly list of expenses being claimed.

Taking this even further, we can have a workflow governance set up so that if an expense greater than \$1,000 is being claimed then the CFO must sign off on it. This is easy to set up. First we create a group that just the CFO belongs to. Then a workflow governance is created for that group with a special condition such as `#input/ExpenseAmount > 1000`.

Now, all regular expenses are approved by both the department head and accounting, but any large expenses will be approved by the department head, accounting and the CFO, before being entered into the expense.

With additional calculated fields to keep track of the budget for expenses, similar conditional governance can be set up to require approval under a specific event such as exceeding the monthly budget for expenses, or for a particular project.

So it should be clear how a pretty involved workflow can be set up without any programming whatsoever.

11.2.11. [Example: Vacation Request](#)

Another typical application of workflow is a vacation request.

In this example we have a central list of events that constitutes the company calendar. However employees don't have direct access to create entries here: rather they must go through approval workflow.

When the entry is made we request that the user puts in their department. A conditional governance is set up for each department, and an additional one for HR who will get to approve all entries in the calendar.

What's more, by putting a calculated field in the user record that totals all vacation taken by them this year, along with a field for their annual allowance, we can have

another workflow governance set up that has a condition of the user's total vacation days exceeding their allowance. Such a breach of allotted holiday allowance can be sent to the department head for authorization.

With some simple formula and the conditional workflow governance rules, a very sophisticated vacation request process can be implemented in very little time using the powerful features available in the dbFramework workflow engine.

11.2.12. Example: Content Management Workflow

Imagine we have a web content management system that maintains the pages that go into the corporate website. The company news page is also seen by clients and investors. The company is very particular about what is mentioned in the news so they have a procedure in place whereby everything published on the company news site has to go through several tiers of approval.

1. A member of the 'Editors' team must verify the post for spelling and grammatical errors.
2. Depending on the topic, the appropriate department head must check that the post is appropriate.
3. The marketing department must check that the post is in line with previous public statements made on the topic.
4. The legal department must approve the message to make sure nothing being said is confidential or introduces any possibility of legal liability.
5. Anything marked as 'Big News' has to also be approved by the company president.

This type of workflow is easy to set up in dbFramework applications.

First we assume that we have the area to post the news. This would be a simple list with entries for the title, the text, the associated department, a flag indicating whether it is 'Big News', and tags.

Then we create a security role that ensures that everybody is subject to approval workflow for that list.

Next we create the workflow governance rules:

- The 'Editors', 'Marketing' and 'Legal' teams must approve any change in the corporate news list. This is a simple governance with a path such as /CompanyNews.

- Next we point each department group to the company news with a condition based on the associated department selected in the news post. This will be a condition that checks for something like `#input/Department = "XYZ Product"`, that way only entries with that selected department will be subject to approval by that department group.
- Lastly we have a group specifically for the company president, and this will also be looking at the corporate news, with a condition such as `#input/IsBigNews`. If, and *only if*, that flag is set will the president's approval be required before the entry is posted. This ensures that no 'Big News' is made without the approval of the company president.

Once this governance has been configured, any change made to the 'Company News' list will require authorization from all of the above users. Each will review them sequentially, so that after one user group has approved the entry, it will go to the next.

Again, this type of workflow can be set up without any programming, and also without any design-time changes: All of this is configured within the application at run-time, just by changing the workflow governance records in the user groups section.

11.3. Tasks

11.3.1. Overview

The "Is A Task" option in the Features lets you indicate to dbFramework that you wish your list to also be treated as if it were a list of tasks.

A task is an item that can be worked on. It can be assigned to a user, can have priority, a progress meter, a due date and a status.

When you select this option, all of these fields are automatically added to the record structure. They can then be integrated into new reports, custom views and so on.

Note: When you turn off the 'Is A Task' feature, all of the task-related fields in the data model and any customization applied to them will be deleted.

11.3.2. Configuring Tasks

To add the task fields to your type, go to the list you wish to turn into tasks and go to the 'Features' drilldown, select 'Is A Task' and change it to 'Yes'. When you click Update, dbFramework will automatically add the task-related fields to the list.

The following fields will be added:

- **Status:** Indicates the status of the task. Can be 'In Progress', 'Not Started', 'Deferred', or 'Complete'.
- **Priority:** Indicates the relative priority of the item. Can be any number, but should be between 1 and 10. Fractions are allowed.
- **Assigned To:** The user to which this task has been assigned.
- **Started Date:** The date that work on the task was started. This is filled in automatically when the status changes to 'In Progress'.
- **Due Date:** Indicates when the task is due to be completed.
- **Completed Date:** Indicates when the task was completed. This is filled in automatically when the status changes to 'Complete'.
- **Estimated Days:** Indicates the estimated number of days this task will take to complete.
- **Actual Days:** Entered by the worker, indicates the actual number of days taken on this task.
- **Reminder Date:** A date that can optionally be entered to remind the worker when the task is due.
- **Contact Info:** Information on users who can be contacted for more information on this task.
- **Priority Justification:** If the priority is set to especially high or low, this can indicate the justification for that priority. For example, "The boss wants this done before the meeting on Friday."

The Status and Priority fields are also set as the sort keys by default. This can be changed after the fields are added, if you wish. This means that, by default, the tasks will be grouped by their status, and each group will be further sorted by priority (with the lowest numeric priority at the top). Also, any fields that come before the 'Status' field will take precedence in the sort-order.

11.3.3. Using the Tasks

For the most part, users are free to edit these task fields to the degree that they have permissions to do so.

However an action is added automatically to set a record as 'complete'. This can be taken out if the completion of a task is signaled via a different mechanism, or a custom script.

When a task is complete, it will be rendered as stricken-through, and in gray. This can also be customized: it is created as a rendition on the list type.

Name	Status	Priority	Progress
Rom/Cyn/Trad/Prog	In Progress		100
Assigned To	Started Date	Due Date	Completed Date
	12/14/2010 1:40:35 PM	12/14/2010	12/14/2010 1:40:19 PM
Estimated Days	Actual Days	Reminder Date	Contact Info
Priority Justification			
Notes			

In addition, because the Due Date of the task is set to be a 'Date (Due Date)' type, the system automatically creates a tickler report to show all items in this list in a schedule. The tickler report is automatically named according to the name of the type. The title field is used as the section title for the schedule, and any preview fields are added as comments.

If you do not want to have a tickler report created, or you want to create your own, perhaps one that groups the tasks by their assignee, simply update the type of the Due Date to be a general date instead.

11.4. Live Updates

In some cases data updates quite frequently, and the user would typically refresh the browser screen until they see the update they're looking for. This can be time consuming and cumbersome.

dbFramework provides real-time live updates of data for selected lists. These lists will update on the screen as they're updated from anywhere in the world connected to the same system. The direction of change in numeric values will be indicated by blue or red highlighting, and the values update without interfering with the normal page operation.

To achieve this, the browser sends a small request at a certain frequency to the server seeing if anything it is interested in has changed. When it does change, some browser-based code will find the appropriate spot in the page and update the page dynamically.

Because of the small overhead introduced on the server when live updates are enabled, it is not on by default. Instead each list you want to see update in real-time must be enabled in the dbFramework designer beforehand.

To enable live updates, go to the Features record of the list field definition. In there select 'Live Updates' and set it to 'Yes'.

Updates are displayed when the record is visible on the screen, either in detail, in a list, or in a sub-list that appears below a selected record.

All updates that appear on the screen are cumulative – that is, once they update they highlight and then that highlight stays on the screen until the page is refreshed. This is a feature that enables you to be away from your desk, and when you return get an immediate indication of what has changed since you were away.

The screenshot shows the 'Sprint' record for the 'Development Team'. The sprint was created on 12/8/2009 at 3:50:17 PM. It has a status of 'Current Sprint', is expanded to 83.5 days, and is estimated to be completed on 1/21/2011. The original end date is 3/1/2009. Below the sprint details is a list of tasks. The first task, 'Future Interest Calculations', is highlighted in green, indicating a title update. It is assigned to Adam Rice, has a priority of 2, and is in progress. The second task, 'Collateral Valuation', is assigned to John Wood, has a priority of 3, and is also in progress. The third task, 'Invoice Manager', is assigned to Adam Rice, has a priority of 5, and is in progress. The tasks list includes columns for Title, Status, Priority, Assigned To, Estimate (Days) Remaining, Estimated Start, and Estimated Complete.

Title	Status	Priority	Assigned To	Estimate (Days) Remaining	Estimated Start	Estimated Complete
Future Interest Calculations	In Progress	2	Adam Rice	1	11/22/2010	11/23/2010
Collateral Valuation	In Progress	3	John Wood	16	11/22/2010	12/17/2010
Invoice Manager	In Progress	5	Adam Rice	1	11/23/2010	11/24/2010

Figure 58: Showing the highlighted title update in the list below the sprint record.

11.5. Show Calendar

11.5.1. Overview

dbFramework provides a quick means to visualize date information in a calendar view, which appears as an alternate screen to the regular data list.

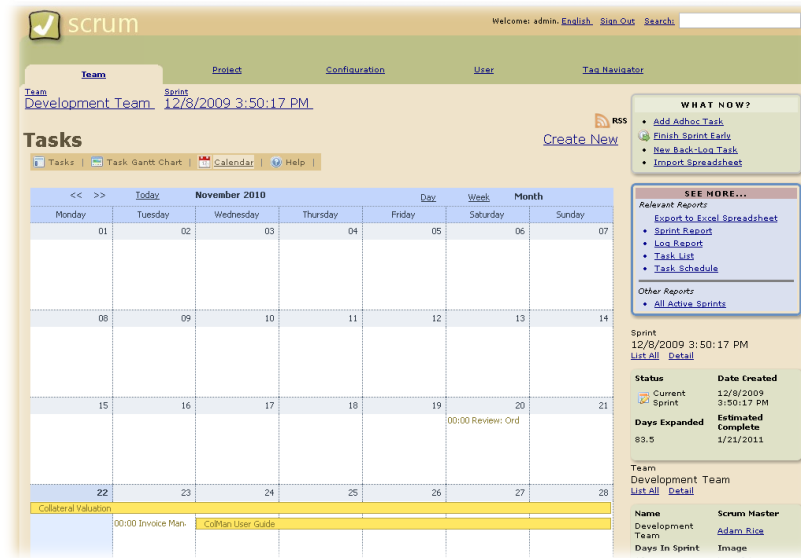


Figure 59: Example of an automatic calendar view.

Events that it discovers are displayed in the calendar in the appropriate month. These events may span a single day, based on a specified start and end date, and are shown visually spanning those days.

The calendar lets the user switch between day view, week view and month view, and provides simple controls for switching back and forth through the given period.

11.5.2. Configuring the Calendar

Go to the Features drilldown underneath the field definition in the list where the dates reside. On that record select 'Show Calendar' and change it to 'Yes'.

You must then drilldown to the 'Calendar Options' drilldown:

The screenshot shows the 'Calendar Options' form. At the top, there are navigation tabs: Applications, Model, Types, Features, and Calendar Options. The 'Features' tab is selected, showing the record 'LenderEx1 - Facility' with the date '11/22/2010 11:38:11 PM'. The main area is titled 'Calendar Options' and contains several fields: 'Date Start Field', 'Date End Field', 'Title Field', and 'Details Field'. There is also a 'Calendar Filter' field and a 'Maximum Events' field set to '500'. A 'Create' button is located at the bottom right of the form.

In this record there are a few fields you must enter:

Date Start Field

Here you select the field in your record that holds the *start* date.

Date End Field

Here you select the field in your record that holds the end date. If there is only one date field, then select the same, Start, field here.

Title Field

Specify the field that will hold the title of the event entry.

Details Field

Optionally specify a field that will hold the details of the event that will appear when the event is clicked.

Calendar Filter

Enter an optional condition to filter out specific events. The filter should be in the form of an XPath predicate, eg. Status="NotStarted".

Maximum Events

Specifies the limit of the number of events that will be displayed in the calendar. If the record list is sorted by date descending, then you can limit this based on the number of months you want to go back in the schedule.

Tip: Another feature of the calendar view is that you can query the data in iCal format so that it can be integrated into Microsoft Outlook and other calendar applications. Please see section 21.7 for more details.

11.6. Show Map

11.6.1. Overview

dbFramework provides a simple way to show geographic data on an interactive map. The map utilizes technology provided by Yahoo! Inc, which is now powered by Bing Maps produced by Microsoft Corporation.

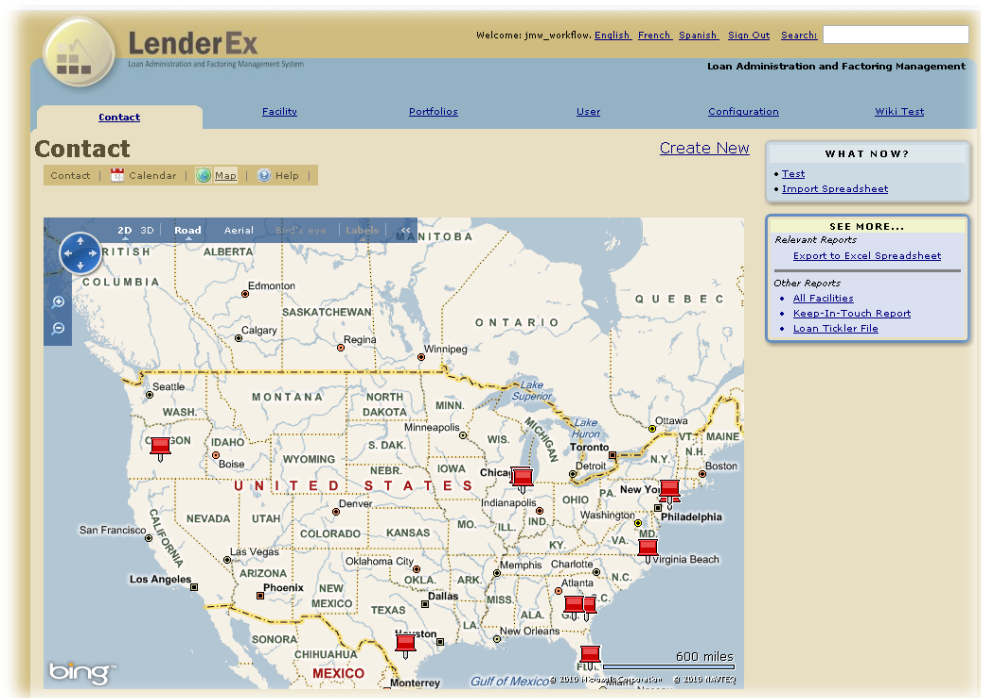


Figure 60: Example of a map visualization of geographic data.

Important: The use of Bing Maps in commercial applications may require a license from Microsoft. As of the date of writing, it appears that commercial applications that are not restricted by login authentication can use the maps for free. For more clarification please read the licensing agreement provided by Microsoft. A version should be available here: <http://www.microsoft.com/maps/product/terms.html>.

11.6.2. Concepts

The map views geographic data, which must be provided as an address of some form on the record within the list.

There is a two stage process involved in mapping data. The first is to produce a longitude and latitude from the address. To do this, dbFramework has the application use a geocoding service. The geocoding service takes the address and returns a longitude and latitude. These coordinates are then stored in special fields in the record.

The map itself is displayed as a custom view available in a screen tab above the record list, and will show records (as their titles and address) in the map once the coordinate information is available.

The coordinates can also be displayed by switching a configuration option.

11.6.3. Configuring the Map

To add a map to a list that contains address information, first go to the Features page and select 'Show Map' and change it to 'Yes'.

Then go to the 'Map Options' drilldown.

Map Options

AddressLine1

Show Long/Lat Fields: Yes

Address Field 1: AddressLine1

Address Field 2: AddressCity

Address Field 3: AddressState

Address Field 4: AddressZipCode

Address Field 5: AddressCountry

Update

The following fields should be entered:

Show Long/Lat Fields

This controls whether the actual longitude and latitude of the record is displayed in the record itself. If this is set to 'No' then only the point on the map will be displayed.

Note: There may be licensing restrictions relating to whether you are able to display the longitude and latitude data in your record without first getting permission from Google Inc.

Address Field(s)

Select as many address fields as available in your record. At the very least the zip code should be selected, and records may be in different countries.

Note: At least the zip code must be available.

Location Information	
Longitude	Latitude
-87.9116	42.1167

Figure 61: Showing the longitude and latitude of the address in the record.

11.7. Audit Trail

11.7.1. Overview

An audit trail is a record of changes made in the system.

The approval workflow feature provides a report showing requests to change records in areas where approval workflow is enabled, and this can prove to be a useful means to create an audit trail.

But without having workflow enabled, you can also have your dbFramework application keep a log of changes to a particular list, and to do this you simply enable Audit Trail on the field.

11.7.2. Configuring Audit Trail

There is very little that has to be done to enable the audit trail capability. Once in the Features record of the field definition, select 'Support Audit Trail' and change it to 'Yes'.

Then, when you republish, any change that you make to a record in that type will add to a log, stored in a drilldown called 'Audit Trail'. The log will describe precisely what changed in the record, along with a record of who changed it and when it was changed.

The screenshot shows a web application interface for a 'Contact' record. The record is for 'AFSL Commodity Research'. The interface includes a top navigation bar with 'Contact', 'Keep-In-Touch Report', and 'Help'. Below the record details, there is a section titled 'Audit Trail' which is currently selected in a tabbed interface. The audit trail table shows a single entry for a change made on 11/4/2010 at 12:27:52 PM by user 'lmw'. The change description is 'Notes = Commodity research used for prospective de...'. The record details include fields for Name, Address, Phone, Next Contact Date, and Location Information.

Change Date	Changes Made By	Changes
11/4/2010 12:27:52 PM	lmw	Notes = Commodity research used for prospective de...

Figure 62: Showing the audit trail list available at the bottom of the record display.



Figure 63: Showing the audit trail record created when a change is made.

Note also that the audit trail cannot be edited by the user in any way, providing a secure record of the changes.

11.8. Last-Modified Dates

While dbFramework always records the date that a record was last modified, who it was modified by, along with when and by whom the record was created, this information is kept hidden from the user by default.

To show these fields you can set a flag in the features record.

Go to the Features record, and select the option 'Show Last Modified / Create Dates'. Change this to 'Yes'.

Once you republish, you will notice a new group of fields in the record:

Caruna Sports test facility			
Dates			
Last Modified	Last Modified By	Created	Created By
11/23/2010 9:41:13 AM	jmw	6/21/2010 10:46:46 AM	admin

Figure 64: Showing the last modified and created dates

These change record dates don't just record when the user edits the record directly, but also if scripts modify the record indirectly, from an event of another record. In that case the username will refer to the user who *initiated* the operation.

11.9. Data Entry Reminders

11.9.1. Overview

There are not many businesses that don't revolve around the timely management of information. For example businesses may need the daily recording of warehouse values, or stock pricing, employee head counts, instrumentation values, car mileage and so on.

More often than not, the business depends on having this information entered into the database in a timely fashion. Yet keeping on top of that is challenging.

dbFramework provides a feature called 'Data Entry Reminders' that enables a special area in the database to specify how frequently values in that area of the database should be entered, along with the group responsible for entering them.

If that data isn't entered on the specified schedule, the dbFramework application will automatically send out emails, explaining what record is missing, and providing a link to the area of the database so it can be entered. These reminders will persist every 30 minutes until the data is entered.

Note: In the interest of sanity, the email reminders will cease after five hours, after which the entry for that day (or whatever period is configured) will be ignored. It is assumed that whomever was responsible for that entry, and their superiors, will notice all of the email reminders in their inbox and take the necessary action.

Incidentally, another way to keep track of scheduled events in the system is by using the Tickler Report features. These are discussed in more detail in section 11.21.

11.9.2. Configuring Data Entry Reminders

The data entry reminders can be created on any list where values are entered as new records. The reminder system checks to see if a record has been created on the day it is due, and if not will initiate the reminders.

11.9.2.1. *Designer*

When designing the application in dbFramework, you simply specify that reminders are enabled for the list. It's then up to the user of the application to set up the reminder frequency, time of day, and group responsible. That makes sense because we don't want to hard-code values like that into the application as they may change frequently.

The first step is to locate the field definition in the data model where you will want the data reminders to be made available. The field definition must be a nested type, with a List Type of long list. Short list works also, but assuming you'll be entering data in each day a long list makes the most sense here, to enable paging.

Once you have found the field, create the Features drilldown and in that select the option titled 'Reminders' and set it to 'Yes'. That's all you have to do to enable the data reminders, from this point you just republish and ask the users to set up the specific parameters for each reminder that is required.

Note: You can configure multiple drilldowns from a single record for data reminders. This will create several sections in the parent record, one for each drilldown.

11.9.2.2. *User*

The user must then locate the record that is one level up from the area where the records will normally be entered.

At the bottom of that record you should see something like this:

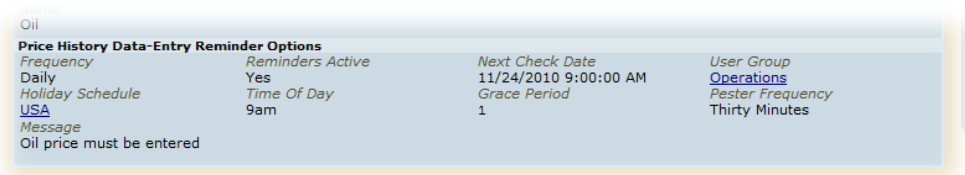


Figure 65: Data entry reminder options created by dbFramework

The following fields deserve some explanation:

Frequency

This is the frequency that the value record should be entered. The options are:

- Daily
- Weekly
- Monthly
- Quarterly
- Annually

Reminders Active

Provides a quick way to de-activate the reminders. Changing this record to 'Yes' schedules the data entry check.

Next Check Date

Specifies the next date and time that the system should check for the value record. This is only entered once, after that the system automatically updates this field.

User Group

Specifies the user group responsible for entering the values. All members of this group will be emailed when the data is missing.

Escalation Group

Specifies the optional user group that will receive messages of this reminder if the reminder is escalated.

Escalation Period

Specifies the number of hours the reminders will be sent before the message is escalated to the escalation group. If no escalation group is specified then this can be ignored.

Holiday Schedule

Optionally select a holiday schedule that will be used to check to see if the day the value is expected actually lies on a bank holiday. If it does then that entry will be skipped and scheduled for the next, non-holiday. This must also be selected if the values shouldn't be entered on weekends.

Time Of Day

Specifies the time of day that the value should be available. For example, specify "9am" to have the system begin checking for values at 9am.

Grace Period

Specifies an optional grace period, in hours. Only after the grace period has passed will email reminders be sent from the system. If the data is due at 5pm, for example, but you don't want reminders sent till the next morning then you would specify 16 hours to see the reminders the next day at 9am.

Pester Frequency

This specifies how frequently the email reminders will be sent.

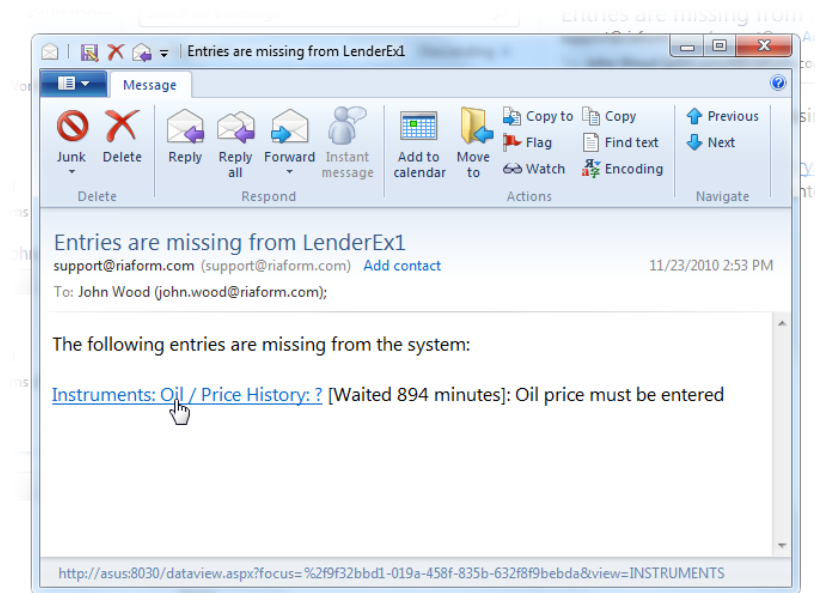
Message

An optional message that will be included in the email. For example: "Values are normally retrieved from www.blahblah.com".

11.9.3. Email Structure

The email that is sent from the system will contain all reminders combined into one email for a specific user group. This is to reduce inbox clutter.

The email subject will be something like “Entries are missing from AppName”, and the content will look something like this:



Each entry will consist of the location of the entry that is missing, the length of time that the system has waited for the entry, and any user message. The location will appear as a hyperlink into the system, letting the user click the link to take them directly to the place in the application where the record should be entered.

11.9.4. Escalation

If configured, the messages can be escalated to another group after a specified number of hours. This can be useful if the members of the group are not available, in a meeting for example, and so a second group can be made aware of the missing data and take necessary action.

Supervisors may want to have themselves in the escalation group, for example, but not the ordinary reminder group. This way they can be notified after there are actual problems and not just because of short administrative delay. Because the reminders are sent through email, the supervisor could receive the notification on his or her BlackBerry and make the necessary calls to ensure the data gets entered.

11.10. Version Control

11.10.1. Overview

The version control features of dbFramework provides several related features. It enables applications to:

- Store multiple versions of a record, and any files attached to that record, with a unique version number for each iteration.
- Maintain a history of those versions, including who changed what and when.
- Let users *lock* a file so that it can be edited by only one person at a time.
- Rollback to any version of the record on any date, along with any files attached to that record.

The version control features of dbFramework enable any application to offer document management features.

11.10.2. Concepts

Version control has several concepts with which you should become familiar:

- **Versions**
The version of a document or record is a fully changed entry that is different from a previous iteration. Each version has a 'version number' that is a number that increments each time a new change is committed.
- **Check Out**
Checking a record or document out involves locking that file so that no other user can edit it. Once checked out, the document or record is usually edited.
- **Check In**
Checking a document in involves submitting that document or record back to the system after the edits are complete. The new version of the record or document will become a new version in the versions history, and the record will be unlocked so another user can now perform edits.
- **Rollback**
Rolling back a version allows you to select an older version of a record or document and have that become the *current* version. This may be useful when you wish to undo an error that occurred in changing a record or document.

11.10.3. Configuring Version Control

The version control features can be enabled on any list of records. First locate the field definition in the data model where you wish version control to be enabled.

Once the field is selected, create the Features drilldown and then select the entry 'Version Controlled' and set it to 'Yes'.

Once the application is republished, go to the respective area in the application where you just enabled version control.

11.10.4. Using Version Control

There you will see something like this:

The screenshot shows a record titled 'Copper Research June 2010'. At the top right, there is a 'Check Out' button. Below the record title, there are links for 'List All', 'Detail', 'Edit', and 'Delete'. A 'Goto Version' link is also present. The record details include: Document Number 2, Title Copper Research June 2010, Date 6/20/2010, and Expiry Date 6/20/2011. A description 'Test Report.pdf' is listed under the 'File' section. The 'Version Control' section shows 'Version Status' as 'Checked Out By' and 'Checked Out Date' as 'Latest Revision'. At the bottom, there is a 'Version' tab and a table with columns: Version Number, Date Checked Out, Date Checked In, and Changes Made By.

This particular document has version control enabled. You'll notice several new additions.

Firstly, there is a 'Check Out' button at the top of the record. This enables the currently logged-in user to check out the record. This also checks out any files associates with the record, and locks them from being edited.

When the record is checked out the following message will appear:

The screenshot shows the 'Documents' section of the application. A yellow message box with an exclamation mark icon states: 'Check Out: The item has been checked out. While it is checked out you will be the only user who is able to make changes.' Below the message, the 'Documents' section is visible, showing the record 'Copper Research June 2010'. At the bottom right, there is a 'Check In' button.

...informing you that only the current user will be able to edit the record while it is checked out.

At this point the user who checked out the record won't notice any difference and can freely edit the record. Other users, however, will be given an error if they attempt to edit the record, explaining that the record is already checked out.

When the record is checked out, the button at the top of the screen will switch to 'Check In'. Clicking this will bring up a new form to fill in:

Check In

Check-In Details

Change Description

Updated the table of contents

Create

This form allows you to describe the changes that were made to the record. These will be included in the version entry for this record.

Once submitted, the lock is released on the record and it can be checked out by another user.

If the record is attempted to be edited while it is not checked out, an error will be displayed to the effect of: "This item is version controlled and cannot be edited without first being checked-out."

The new version of the record will then be seen in the version list at the bottom of the record, or by going to the Versions drilldown:

[AFSL Commodity Research](#) [Test record](#)

Version

1 (1 / 1)

Rollback To This Version

[List All Detail](#)

Version Number	Date Checked Out	Date Checked In	Changes Made By
1	11/23/2010 5:25:19 PM	11/23/2010 5:25:32 PM	imw

Comments
Added new title.

Details
Document Number = 4
Title = Test record
Date = 11/23/2010
Version Status = Checked Out
Checked Out By = jmw
Checked Out Date = 11/23/2010 5:25:19 PM
Latest Revision = 0

The version entry contains some key bits of information:

Version Number

This is a unique number identifying this version. It increments each time a document is checked in.

Date Checked Out

This is the date that this version was checked out.

Date Checked In

This is the date that this version was checked back in.

Changes Made By

This is a link to the user that checked out, made changes and checked in the record and associated files.

Comments

This holds any comments that were entered when the item was checked in.

Details

This field shows a detailed description of what was changed in the actual record between versions.

File1, File2, File3, File4, File5

These contain links to any files that were associated with the record.

On each version there will be a “Rollback to This Version” button. Clicking this enables you to revert the version of a record to an earlier date.

11.11. Wiki: Community Content

11.11.1. Overview

Products such as ‘Wikipedia’ have made the possibility of effective community-driven content a reality. A ‘wiki’ is just that: a means for users to add their own content easily. This is important because having users contribute to application content enriches the application and makes it more useful for everyone.

dbFramework offers a wiki capability that can be added to any aspect of the data model, providing an unlimited database of pages that allow users to add

discussions, knowledge, research or any important, collaborative content they wish to store.

Using a subset of the same wiki commands used by other wiki systems such as Wikipedia, the wiki editing experience should be simple and familiar.

11.11.2. Configuring Wiki Support

There are two ways to add wiki support to your application.

1. You can add a Wiki tab to the top of the application, providing a global wiki that can be accessed from anywhere, or
2. You can add a wiki to a specific record. Perhaps you want to have a wiki for a specific project, so you would enable Wiki support under the project record type.

To add a Wiki tab to your application, on the Application record locate the field “Enable Wiki” and set it to ‘Yes’ and republish.

To add a Wiki to a type in your application, first locate the area where you wish the Wiki to be available. Once you have located the field definition, go to the Features drilldown and set the ‘Supports Wiki’ to ‘Yes’. When you republish there will be a Wiki drilldown from that record.

11.11.3. Using the Wiki

The Wiki may either be available on a Tab, or on a specific record in the ‘Wiki’ drilldown.

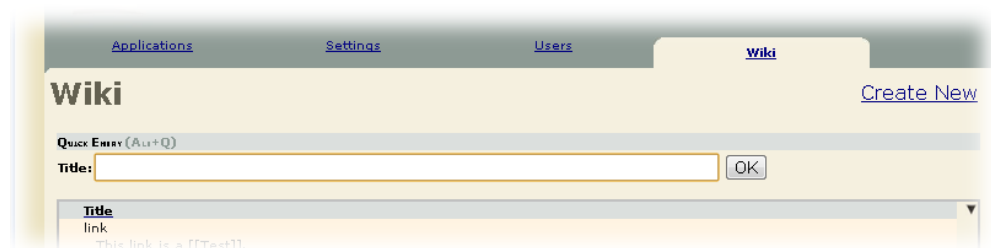


Figure 66: Wiki tab in the application

To create a Wiki entry, simply click ‘Create New’.

Editing the Wiki entry is as simple as entering in text. However, any topics you wish to become links should be entered surrounded by [[and]].

There are also additional commands you can use, which are described below.

For example, you may enter the following text:

```
The [[copper]] metal is a chemical element and [[ductile metal]] with
high thermal and electric conductivity.
```

Once entered into the system, this will link to, and create if not already existing, two topics for 'copper' and 'ductile metal'. The resultant text will look something like this:

```
The copper metal is a chemical element and ductile metal with high
thermal and electric conductivity.
```

Clicking those links will take you to those created topics, where you can edit and create more wiki entries.

There are additional commands that can be used to control the formatting of the output.

- Pressing ENTER twice at the of a line creates a paragraph.
- Surrounding a title with equals signs controls the heading level:
 - Putting text in single equals signs, eg. =Heading= will display the text in the top level heading format, with largest text and bold.
 - Putting text in double equals signs, eg. ==Heading==, will display the text in the second level heading format, with large text and bold font.
 - Putting text in triple equals signs, eg. ===Heading===, will display the text in the third level heading format, with medium size text and bold font.

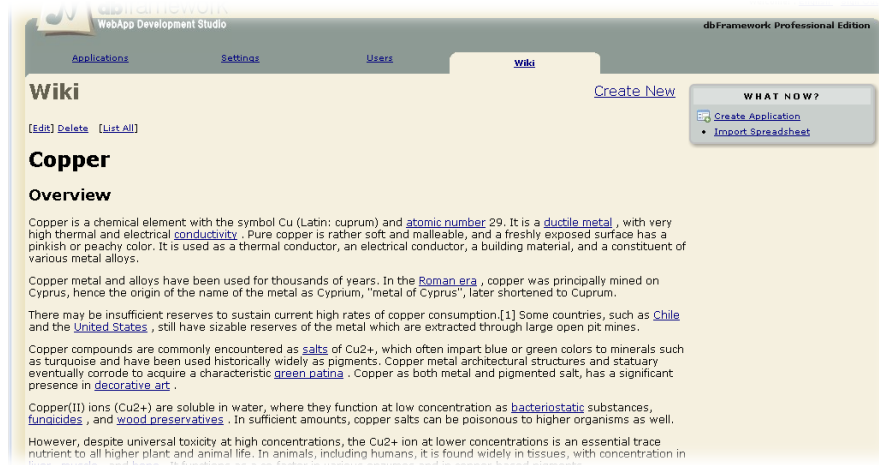


Figure 67: Example of a wiki entry in dbFramework

11.11.4. Comments

Wiki entries can also be commented upon, much in the same way that Wikipedia articles can also have comments, often by the editors or authors of the article.

To access the comments, drilldown to the Comments list which can be found at the bottom of the article.

For more information on commenting, see section 11.14.

11.12. Ratable

11.12.1. Overview

Another community-oriented feature is the ratings capability. Ratings allow users to vote on a particular record in the database. Each user can offer a single rating, and all of these ratings are averaged and displayed in the record as a single rating.

This provides a quick way to enable user feedback on the popularity of a particular entry. With multiple entries, it may enable quick decision making.

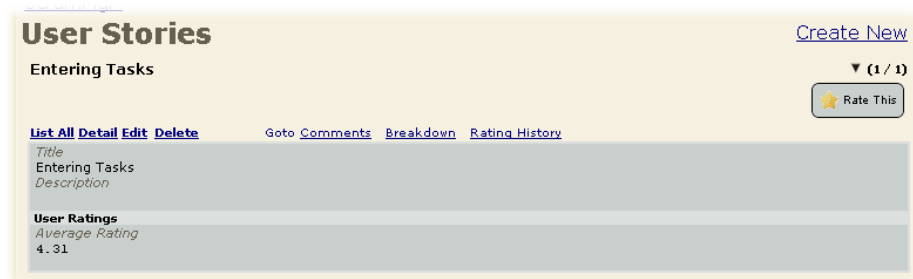
For example, in an acquisition database a number of possibilities may be entered for projectors that are available on the market. Users can then be encouraged to vote on which of those is deemed the best buy. After a couple of days of feedback the projector with the highest rating may be chosen as the product to purchase.

11.12.2. Configuring Ratings

Ratings can be provided on any list within the data model. Find the field definition that will hold the list, go to the Features record, find the 'Ratable' entry box and set that 'Yes'.

11.12.3. Using Ratings

After republishing, records in that list will have the following additional features:



- **A 'Rate This' button at the top of the record.**
This will be clicked by the user to enter their rating. The rating will be a number from 1 to 5. If the user has already rated, then they can replace their rating using this action also. By default all users are identified by their IP address.
- **A field group titled 'User Ratings', which holds the average rating.**
This is calculated based on the average of all the ratings that have been submitted, rounded to two decimal places.

The number of users who have rated can be viewed by drilling down to the 'Rating History'. While for the most part anonymous, the ratings are identified by the IP address of the machine from which the user entered their rating.

For feedback that isn't anonymous, it's recommended that a Comments section be enabled in addition to ratings. This is explained in section 11.14.

11.13. Document Review and Routing

11.13.1. Overview

While the Approval Workflow provides the ability to have documents or records reviewed by someone prior to being committed to the database, sometimes you wish to have a group of users review a record to garner feedback. This more *democratic* approach to document review is often called document routing.

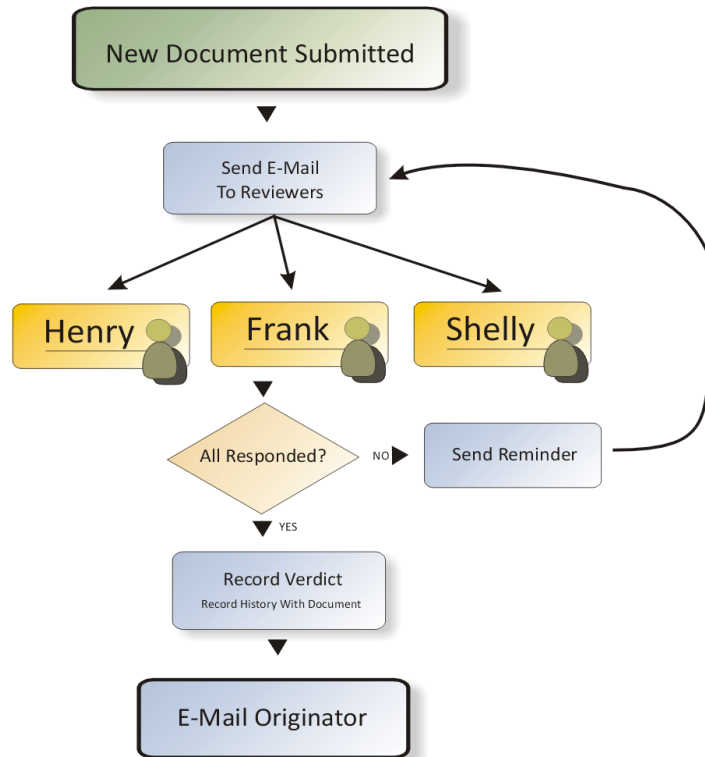


Figure 68: Document review workflow

The process of reviewing a document is as follows:

1. A new document is submitted.
2. E-Mail is sent to all members of the user group
3. The system waits for a response, and periodically emails members who have not responded to remind them.
4. Each member receives an email containing:
 - a. Details of the record content, along with any attachments
 - b. Links to enter a response
 - c. Link to see the verdict
5. Once all responses have been received, the system enters in a verdict against the original record. **The verdict is based on the majority opinion.**
6. The original group is notified of the verdict, along with anyone subscribing to updates on that record.

11.13.2. Configuring Document Routing

To enable document routing, find the area in the data model that defines the list where you wish to enable document routing.

Go to the Features record, then select 'Requires Review' and change it to 'Yes'. Once you republish you will notice some new features in a record created in the list.

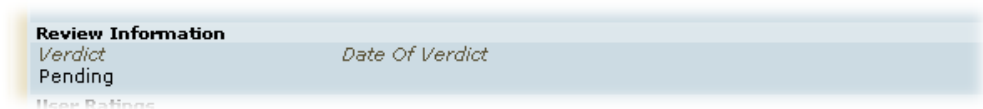
11.13.3. Using Document Routing

When you create a document in a list that has document review enabled, you will see this message appear:



This is to notify you that the review process has begun. At this point an email is sent to each member of the group, and an entry is made in a new drilldown from the record called 'Review History'.

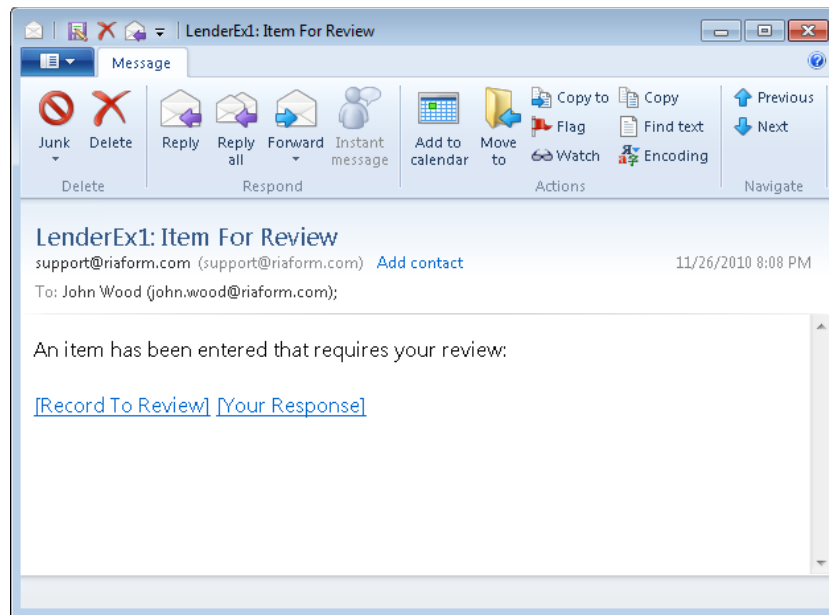
You will notice new fields on the record that will hold the verdict. At this point in the process the verdict will be 'Pending'.



Once the verdict is reached, the result will be entered here along with the date that the verdict was reached.

The group selected for the review process is determined by the workflow governance. Each user group can have any number of areas under its governance. The group governing the area of the database where the record was inserted will be selected for the review process. For more information on setting this up please refer to section 11.2.4.2.

The email that is sent to each user in the group will look something like this:



This email provides a link to the record that needs to be reviewed, along with a link to the specific area where that user's response should be recorded.

Clicking on the response record will bring up a record similar to this:

The following fields can be entered:

Reviewer

The reviewer is the link to the user performing the review. This shouldn't be changed.

Status

This is the response the user wishes to enter. By default it is set to 'Pending'. The system won't be satisfied until a response of Accept or Reject is entered.

Notes

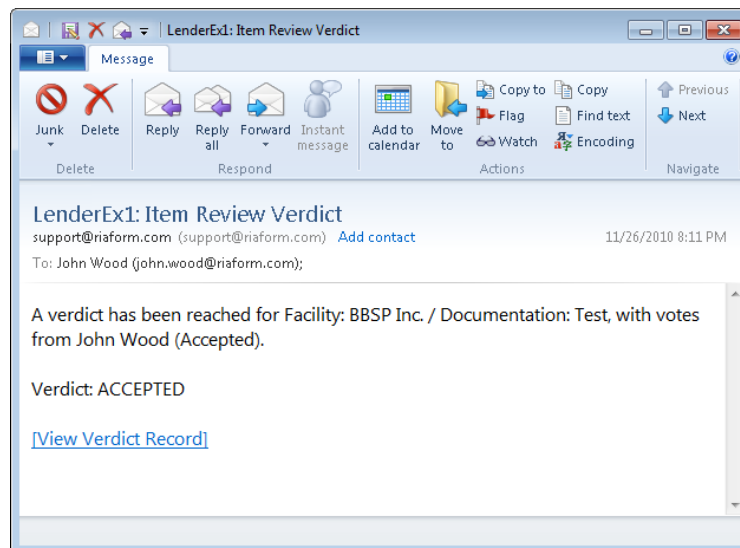
Any comments relating to the acceptance or rejection can be entered here so they can be seen by the originator.

If a user doesn't respond within 24 hours, another email will be sent to remind them that the review is outstanding.

Once all responses have been received, a verdict will be entered. This will be based on the majority rule.

Note: If the verdict results in a tie, then it is automatically entered as a rejection.

An email will be sent to each member of the group along with the originator (the user entering the record). The verdict email will look something like this:



The email provides a description of where the record is in the database, and details each users' response. The verdict is displayed prominently, along with a link to the record in the database.

11.13.4. Example: Product Ideas Database

An example of how this could be used might be with a database that manages ideas for a software product.

To promote openness, any user can submit written proposals for new features for a software product.

The list of ideas has document review configured, and the marketing team (who control the feature sets of products) is set to review anything placed in this list.

When a user writes out and posts an idea to the list, an email is sent to all members of the marketing group. The entry is placed in the list but the verdict field is set to 'Pending'.

Each member of the marketing group reviews the ideas and gives a thumbs up or down, along with any comments. Any members that do not review the item are reminded via email each day.

Finally, when all of the members have reviewed the entry and a verdict is established based on the majority opinion, the verdict is entered in the idea post, and an email is sent to all members of the group and the original author.

Additional features can be added to this setup.

1. A report can be created that filters on all accepted ideas and this can be emailed out each week to the marketing department and the development department.
2. Approval workflow can be configured so that before an idea gets set as 'Accepted' (which will occur when the last person makes their vote), an approval of the development manager will need to be made. This is configured in a workflow governance set to check the Verdict field being equal to 'Accepted' on the idea post. This is useful because it means the development manager is only asked about ideas that have reached the overall majority approval of the marketing department. If the development manager rejects the idea then the marketing department can decline and enter their development manager's opinion in the final vote.
3. A script can be created that checks for an approved idea and automatically creates a scheduled task for the development group in another list.

11.14. Comments

11.14.1. Overview

dbFramework provides many ways to enable users to enter their own opinions and feedback into the application. While ratings, wiki pages, approval workflow and forums provide more sophisticated means to entering feedback, more often than not you simply want a way for users to add a simple comment to a record.

This can be enabled anywhere in the database with a simple flag switch. This creates a 'Comments' drilldown from any record in your application, providing a space for an unlimited number of user comments to be added.

11.14.2. Configuring Comments

To add support for comments, locate the area in your data model where you wish the comments to be added. From that, go to the Features record, and select 'Allow Comments' and set it to 'Yes'.

Once the application is republished the Comments drilldown will appear.

11.14.3. Using Comments

Any user with permissions in that area can add comments. They simply go to the Comments drilldown and click 'Create New' to post a new comment.

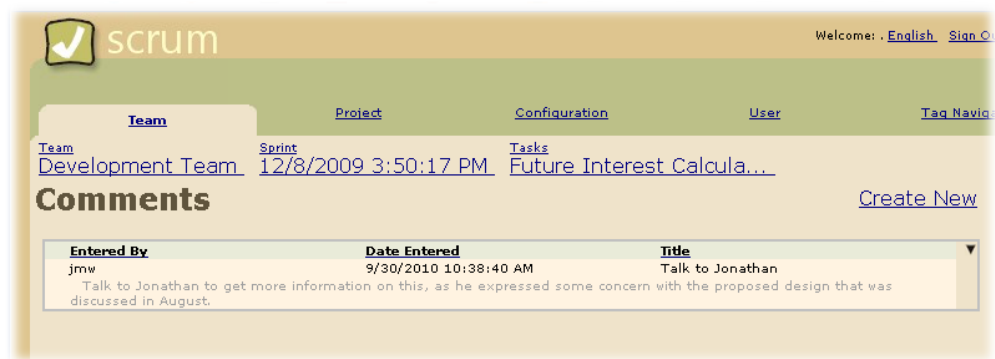


Figure 69: Showing a Comment entry

11.15. Forum (BBS)

11.15.1. Overview

While comments provide a way to add and maintain a simple dialog between several members, sometimes you want more structured conversation that is separated by subjects. For this dbFramework provides an in-built forum capability.

These forums would be familiar to anybody who has used a bulletin board system on a website. Forums can be made available and added to any record in the database.

For example you may have a list of products. For each product you can have a forum area to use for providing support services to users. Users can then use the forum to ask questions, offer tips and provide general peer-to-peer support.

11.15.2. Configuring Forum Support

Forums can be enabled on any record in the data model. The first step is to locate whereabouts in the model you wish the forums to be enabled. Then go to the Features record and click 'Supports Forums' and change it to 'Yes'.

Once you republish the forum capability will be available in a new drilldown from that record called 'Thread'.

11.15.3. Using the Forums

Once you are in the thread view, you will see something like this:

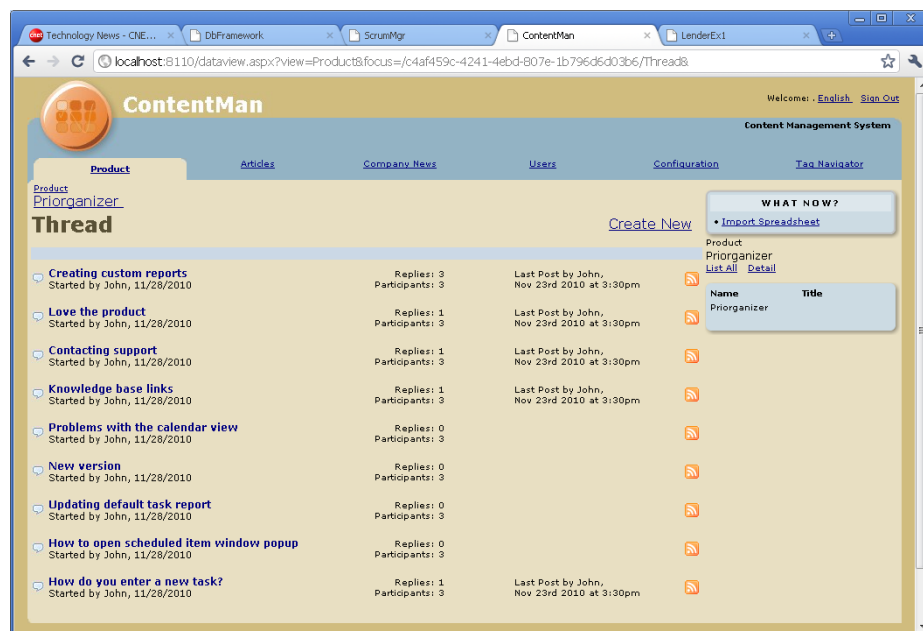


Figure 70: Showing a list of threads

The main list of threads has several columns that show useful information. The first column shows the title of the thread and who started it and when.

The next column provides information on the number of entries in the thread, and how many different users are involved.

The last column shows information on the last post in the thread, including who posted it and when.

At the end of each row is a link to the RSS feed for the thread. For more information on RSS please see section 11.16.

To post a new thread, click the 'Create New' . There you can enter in the title of the thread you wish to start.

You should then click on the newly created thread and post your first entry. To do this, click the 'Add Post' task. You will see an entry form such as this:

Add Post

Add Post

Text

Create

In the white box enter the text of your post and then click 'Create'.

Your post will be displayed with other posts in the thread in a format shown below:

Thread [Create New](#)

Creating custom reports
Started by John, 11/28/2010

Replies: 3
Participants: 3

Last Post by John,
Nov 23rd 2010 at 3:30pm

Posts [Create New](#)

#10
John
11/28/2010

I'm not that clear how I would do that. Can you provide more information?

#9
John
11/28/2010

The easiest way to do this is to create a custom HTML file.
To do this open up an HTML editor and open the template.htm file that's in the product folder.
It should be pretty clear how to add new columns.

#8
John
11/28/2010

How do I create a custom report?

Each post is shown with information on who posted, when it was posted, and the text of the post to the right. Each post also has a number (#10) which can be used with the Search feature to quickly locate posts.

11.16.Supports RSS

11.16.1. Overview

RSS (Really Simple Syndication) is a popular technology for tracking news and events from lots of different sources in one place. The 'RSS feed' is the source of these events, and the 'RSS Reader' watches these feeds for signs of updates. When updates occur they are aggregated into a simple list of notifications.

RSS technology saves people from having to constantly monitor a multitude of different data sources for updates.

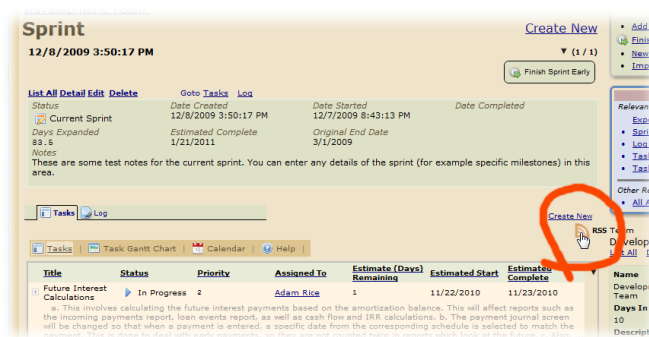
dbFramework has the ability to provide RSS feeds for any part of your application. RSS feeds can be generated for any record or any list. These can then be displayed in RSS readers much the same way as a news website would.

11.16.2. Configuring RSS

To enable RSS, go to the field definition in the data model where you wish the RSS feed to be provided. Then go to the Features record and select 'Supports RSS' and select 'Yes'. Once you republish the application the RSS support will be enabled.

11.16.3. Using RSS

Once RSS is enabled, you will see an RSS icon at the top-right of the record or list. Clicking this RSS icon will provide an option to subscribe to the feed. Alternatively you can add the URL provided by that button to any RSS reader.



Once added, the entries in that list will appear integrated into the screen, as shown below:

Figure 71: Showing the RSS feed link automatically displayed at the top of the list

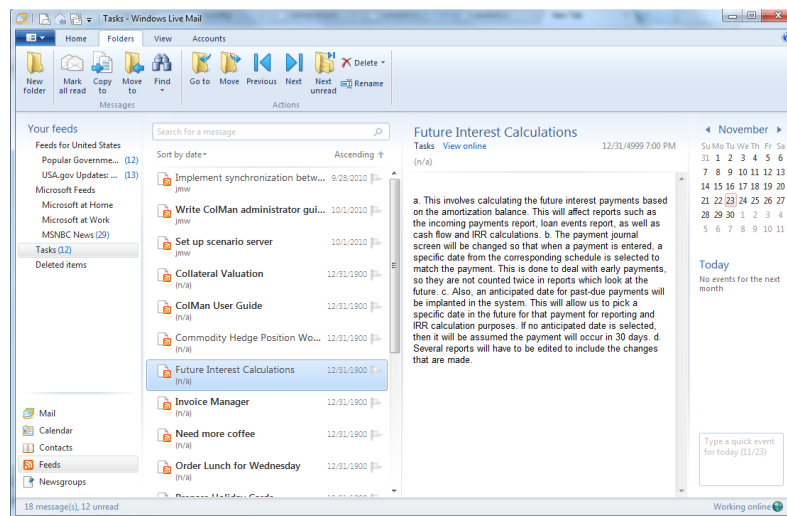


Figure 72: Showing the RSS feed in an RSS Reader or Mail program such as Outlook

The title of the entry will be taken from the field in the record that is marked as the Title field. The body text will be taken from any 'Text (Paragraphs)' field that has the 'Preview' flag added.

Note: RSS feeds will not work if a user login is **required** to access the system. This is because most RSS readers do not know how to login to an application. However the username and password can be provided in the URL, as discussed in section 21.6.2.2.

11.17. Tagging

11.17.1. Overview

Tagging is a way to quickly categorize items. A tag is a type of category. In this case any record in an application can be tagged by the user. They can select one or more tags to apply to the record, and then easily locate those records either through the general search or through the Tag Navigator.

Tagging is very useful because by selecting the same tag for multiple records lets you easily locate those records from that tag by simply drilling down the tag list. The Tag Navigator lets you see all tags and the list of related documents that have that tag assigned. You can also drilldown from a tag to other tags that have been selected together.

For example you may have a tag called 'Prospective Client'. You can tag all of your potential customers with this tag to easily locate them from the Tag Navigator. But some of your potential customers may also be large companies and some may be

small companies. So you might have an additional tag called 'Large Company', and tag some of your potential customers with this tag also.

You, using the Tag Navigator, you can easily locate all prospective clients by finding the tag called 'Prospective Client' and then looking at the list of 'Company' entries below that.

But you can also see all prospective clients that are also large companies by drilling-down from the 'Prospective Client' tag to the 'Large Company' tag. If you then look at the 'Company' list below that, you will only see companies that have *both* of those tags assigned to them.

11.17.2. Configuring the Tagging Engine

Enabling tagging is simply a matter of locating the type in the data model that you wish to have tagging available. Go to the Features record and select 'Can Be Tagged?' and change it to 'Yes'.

After updating that there are a few options can be configured, which are available in the 'Tag Options' drilldown.

Explanation of fields:

Tag List Location

This is an optional field to fill out. If specified, you can control where the tag navigator will be available. By default it's put at the top level, so it's global across the whole application. But you can also have a tag database just for one particular area of your data model. Just select a field somewhere that is a parent of the current location in the data model, and that's where the tag database will be made available.

Tag Selector Field Title

This lets you name the field used to select the tags. In most cases the tag selector will be called 'Tags', but it can be renamed if you prefer.

11.17.3. Using the Tagging Engine

Once enabled, the record on which you enabled tagging will have an additional drilldown called 'Tags':

But before selecting the tags, you need to create the tags that will be available. To do this find the Tag Navigator. This will either be a tab available at the top of the application, or it will be available at a certain location in the database, depending on what was selected in the Tag Options.

On the Tag Navigator, you can add tags using the Quick Entry at the top. Simply type in the name of the tag and press enter to add new ones.

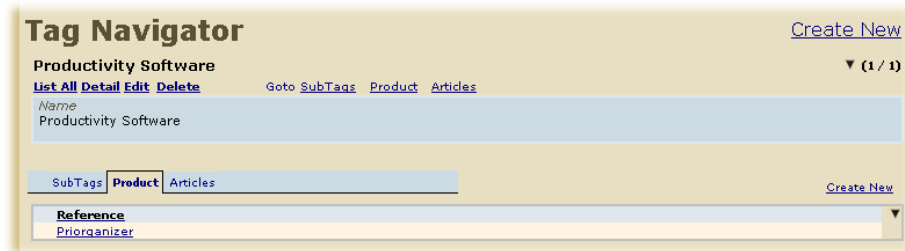
Once the available tags are entered, they can then be selected on records by the user.

The record where you enabled tagging will have an additional drilldown called 'Tags', as shown below:

Figure 73: Selecting a tag on a record

Here you can quickly use the Quick Entry in the 'Tags' drilldown shown below the record in order to assign new tags to the record. Any number of tags can be added.

After assigning the tags, you can then try out locating the record using the Tag Navigator. Go to the Tag Navigator and look at the tags you added. Then click on one of the tags to see the records associated with that tag.



Each area in your application where you enabled tagging will have its own drilldown tab shown below the tag, as shown above.

IN this example, the tag 'Productivity Software' has a drilldown called 'Product' and in that you can see the link to the 'Priorganizer' record. Clicking that link will take you directly to the Priorganizer product.

There is also always a 'SubTags' drilldown where you can see a list of tags in records that have the 'Productivity Software' tag also associated with it.

11.18. Quick Charts

11.18.1. Overview

There are two main ways to add charts to dbFramework applications:

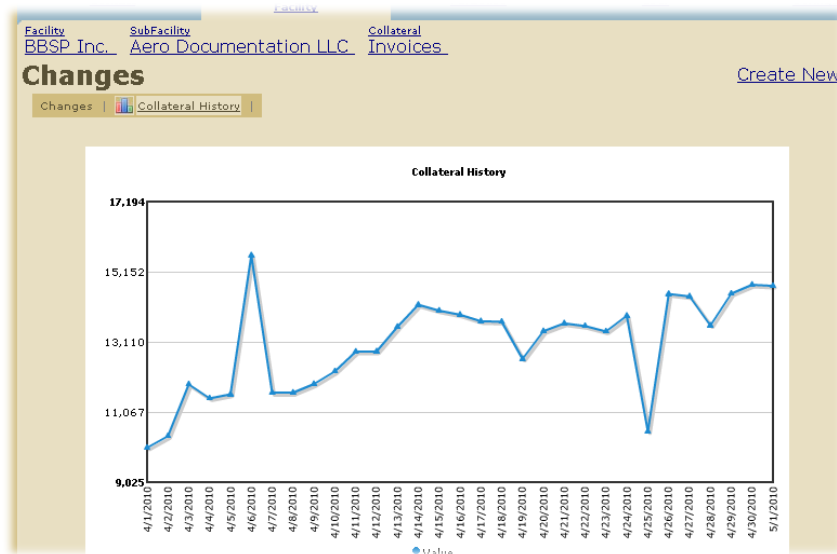
- Using the Quick Charts feature, where simple charts can be provided on any list.
- Using the Quick Reports feature, adding a chart section and enabling the 'Custom View' option to activate viewing of the Quick Report as a screen tab.

The Quick Charts feature will be explored here.

There are three types of charts that can be enabled:

11.18.1.1. *Line Charts*

Quick Line Charts can be enabled for any list that contains a unique date field and one or more numeric values.



11.18.1.2. *Configuring Line Charts*

To create a line chart, find the location in the data model where you wish the chart to be available. Then go to the Features record, and select 'Show Charts' and change it to 'Yes'.

Then drill down to the 'Chart Options' list and click 'Create New'.

Note: You can create multiple 'Chart Options' records for each chart you wish to display.

The following fields should be entered:

Chart Type

The type of chart to display. In this case select 'Line'.

Short Name

The short name for the chart. This will be displayed in the screen tab caption.

Caption

The longer title to show above the chart.

Sub-Caption

An optional sub-caption that will appear below the title.

X Axis Field

Select the unique field, typically a date field, to use as the value for the X Axis of the chart.

X Axis Formula

Optionally, instead of an X Axis Field you can enter in a formula, in DbfScript syntax, to calculate the value for the X Axis. Any field in the record can be accessed using the #input variable.

Y Axis Field

Select the field that will be used to seed the value for the chart, shown on the Y axis.

Y Axis Formula

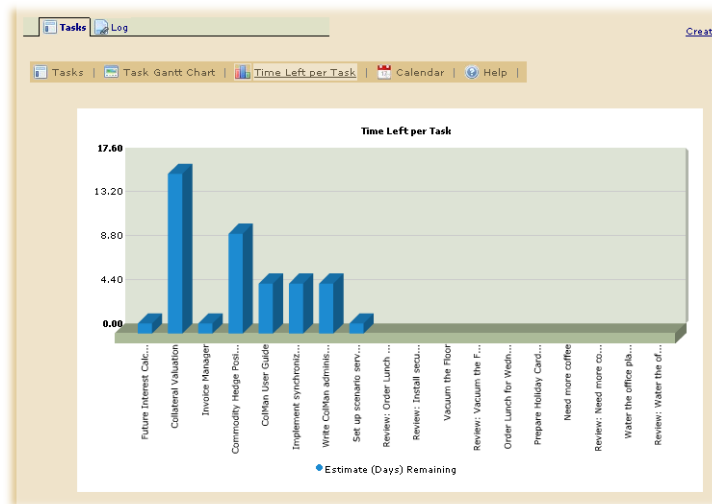
Optionally, instead of a Y Axis field, you can enter in a formula to calculate the value of the Y axis. Any field in the record can be accessed using the #input variable. For example you may wish to show the total of both 'Total Cost' plus 'Sales Tax'. So you would use a formula such as "#input/TotalCost + #input/SalesTax" in this field.

X Axis Aggregation

If there are duplicate values for the X axis, then this specifies how those are dealt with. By default, only the last entry of the group will be displayed. But you can also choose to show the 'Sum' (total) or the 'Average' (average) of the values in that group.

11.18.1.3. Bar Charts

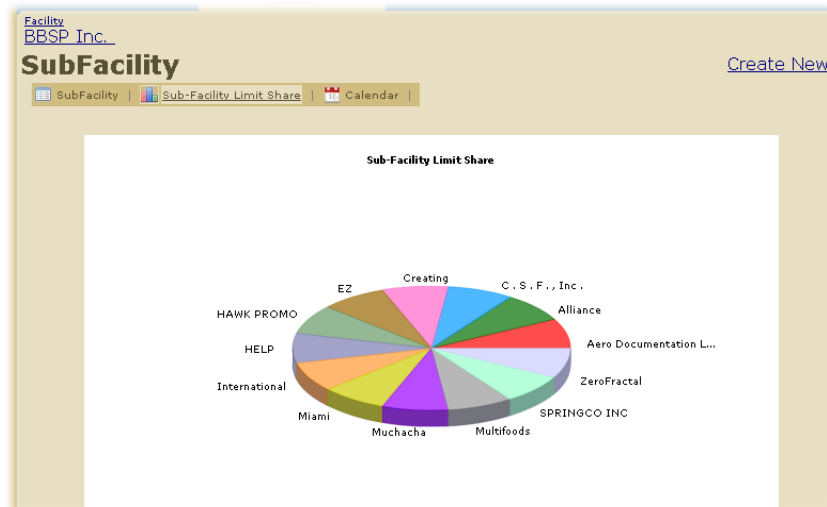
Bar charts work in a very similar fashion to line charts, except the X axis can be any unique field and does not have to be a date field, or sequential.



The fields are mostly the same as they were as described in the Line Chart section, except that the X Axis Field refers not to the X Axis of the line, but rather to each bar in the chart.

11.18.1.4. Pie Chart

The Pie Chart option allows for a quick visualization of proportions in a list.



The fields in the Chart Options are similar to those of the Line Chart, except that:

1. The X Axis field refers to the title of record that will be displayed as the title of the pie slice.

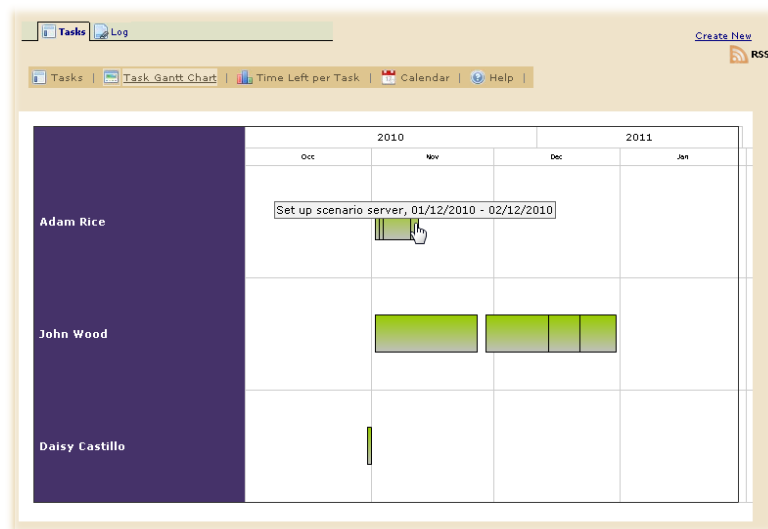
2. The Y Axis field refers to the value that will be visualized by the size of the pie slice. The percentage of the whole will be shown when the mouse cursor hovers over the pie slice of the respective record.

11.19. Gantt Charts

11.19.1. Overview and Use

Gantt charts provide a way to visualize time lines and time allocation among entities such as resources.

dbFramework provides a simple way to visualize data in a Gantt chart.



The Gantt chart shows records along with any two date fields in the record that represent a duration. That duration is then plotted out and separated by a specific category, allowing the user to compare the duration of those records and their allocation among categories visually. In addition tooltips are provided when the mouse cursor hovers over a time slot, showing details of the associated record for that time slot.

11.19.2. Configuring the Gantt Chart

To enable the Gantt chart view, first find the location in the data model where you wish the Gantt chart to display. Then go to the Features record and drill down to the 'Gantt Chart' record.

Click 'Create New' to enter in the configuration options.

Gantt Chart Options

Gantt Chart ▼ (1 / 1)

Name: Gantt Chart Caption: Task Gantt Chart

Fields

Name Field	Date Start Field	Date End Field	Category Field
Title	EstimatedStartDate	EstimatedCompleteDate	AssignedTo
Continuity Reference	Milestones	Milestones Date Field	Filter
			Status="Complete"

Update

The following fields should be entered:

Name

This is the name of the Gantt chart, as will be displayed in the screen tab.

Caption

This is the title that will appear above the Gantt chart.

Name Field

This is the field used to determine the title of a record shown in a time slot.

Date Start Field

Specifies the field used to determine the start date of the time slot.

Date End Field

Specifies the field used to determine the end date of the time slot.

Category Field

Specifies the field used to determine the category for the timeslot. This determines the values in the Y axis, showing the representation of the time allocation.

Continuity Reference

Optionally specifies a field that can link two timeslots together. For example, a 'Depends On' record reference could be used to show a link between two records visually.

Milestones

Specifies the field of a sublist that can show milestones.

Milestone Date Field

Specifies the field used to determine the date in the milestone sublist.

11.20. User Positioning

11.20.1. Overview

Quite often the order of a list of records will be based on the value of a specific field, such as the date the item was created, or its priority.

But sometimes, however, the order should be completely controlled by the user. Perhaps they want to define the order that payments will be made, or the order of seniority for team members.

dbFramework provides a quick way to make this possible. Once User Positioning is enabled, records in that list can be moved up or down, or from page to page, by simple using the clipboard drag and drop operations.

11.20.2. Configuring User Positioning

To enable user positioning, select the location in the data model that holds the list. The type must be a 'Nested Type', and have the List Type set to 'Short List' or 'Long List'.

Go to the Features record, select 'User Positioning' and change it to 'Yes'. When you republish the following will happen automatically:

- A hidden 'Position' field will be added. This will hold a numeric position of the record in the list.
- The default value for the Position field will be "CreatePosition()". This automatically defaults to the end of the list when an item is created.
- The list will be sorted by the Position field.

Note: No other sort fields are allowed. If you have an explicit sort on the list, it should be removed to allow the positioning to work correctly.

11.20.3. Using the User-Positioning

For the most part, the user positioning feature isn't immediately obvious when in use. To move items around, one must use the clipboard functions for cutting and dropping above or below another item.

1. Click on the record you wish to reposition.
2. Go to the clipboard menu and select 'Pick Up'.
3. Then click on the record that will be immediately above or below the record.

4. Go to the clipboard menu and select 'Drop Above Here' or 'Drop Below Here'.

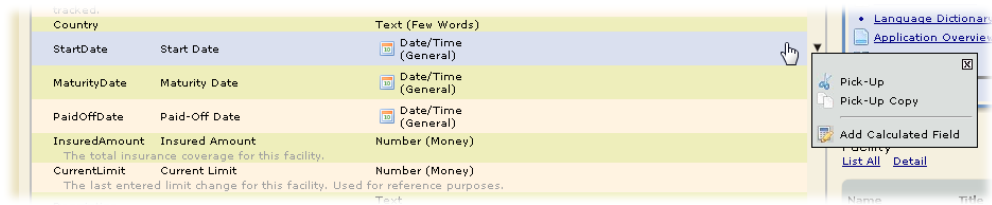


Figure 74: Showing the context menu at the end of each record

Performing this operation will move the record.

Internally this operation simply updates the Position field of the record. Because the list is sorted by the Position field, then updating the Position field effectively moves the record, so long as the default sort is active.

Tip: The new value for Position isn't important, but it's useful to know that no other records' Position values are updated when the record is dropped. Rather an average between the values of the Position field of the record immediately *above* and *below* that record is calculated and used.

11.21. Tickler Reports

Tickler reports are special reports that show future events, grouped by the date they occur. They're incredibly useful for keeping track of important dates across all of the data in your application.

To get more information on setting up tickler reports and their capabilities, refer to the Tickler Report chapter in section **Error! Reference source not found..**

11.22. Composite Lists

11.22.1. Overview

The 'Composite List' feature provides an easy way to have dbFramework maintain a single table that combines several date-based sub-lists and adds additional calculations. This feature is useful if you want to keep certain lists separate so they are more easily manageable, but having them in one list makes calculations significantly simpler. Using Composite Lists you can get the best of both worlds.

Similar to the way the Tickler Report works, the Composite List will pull together related events from several areas in the database and then perform special processing allows these disparate events to be used in calculations such as running totals.

An example of a use for a composite list is to calculate interest on a loan. For ease of management you may have decided to have one list that contains principal payments, another list that contains interest rate changes, and another that contains interest payments.

Using the Composite List feature you can now create another list that *combines* all of these events into one, single list. Now they are combined you can add calculations that work on *all* of these events. In our example we can add simple formula to calculate running interest accruals, which take into account changing interest rates and changes to the outstanding principal. We can also keep running balances.

Now they're in one list it becomes much easier to create reports that show meaningful data (such as the current balance), and also add this information to a Tickler Report. Also events can be added that process this unified data in more advanced ways possible than when the data was held in separate lists.

11.22.2. Concepts

The Composite List consists of multiple sources and calculation lists.

Each source represents a sub-list that you wish to integrate into the unified composite list target.

A source will consist of a root, where the list will be contained. Optionally an additional Sub-Field can be specified, if the list is two levels deeper than the parent of the target list.

Within the source will be a field list, and each of these will have a source field (within the sub-list) and a target (within your target list). These fields will just be copied directly into new rows in the target list.

As these records are unified into a single list, there will be an option whether to repeat the values of records when the next record is from a different list, or whether to zero the values. This is called 'Continuation' and can be managed on a per-field basis.

Then there are calculations. These are defined in the composite list and are calculated as the composite list target is being generated. The calculations can also reference previous records in the list.

11.22.3. Configuring the Composite List

The first step in creating a composite list is to create the actual list structure that you want to see stored in the output. This will be the field list that correspond to all of the fields you want to extract from the sub-lists.

Once you have created that list, which we'll refer to as the target list, you will then need to go to the Features record, and underneath that go to the Composite List record. Click 'Create New' to begin configuring the composite list:

Composite List
11/28/2010 11:12:55 PM ▼ (1 / 1)

Index Field: No ▼ Auto-Update: Yes ▼

Description: Generates the ledger from the principal schedule, interest schedule etc.

Update

There are several fields that can be entered:

Index Field

Indicates whether the field used to maintain the overall order of the target list is visible to the user.

Auto-Update

Specifies whether the list will automatically recreate whenever there is a change in one of the linked sub-lists. If this isn't set, you can trigger the recreation using DbfScript.

Description

This can be any description or notes you wish to enter about this composite list.

Once the Composite List record is created, you will then have to add Sources. To do this go to the Sources drilldown and click 'Create New'. The following screen will appear:

In here you will select the Root field, which will be the field definition that holds the sub-list you wish to add. The sub-list must be a sibling of the current type (a sub-type of the parent).

Optionally you can also specify a sub-field of that root. This allows you to access lists that are two levels deeper than the target's parent.

Next you add each field you want to extract from that list. To do this you can click 'Click to add a Field List record':

In each field list entry you will have to select the field within the source list that you wish to extract. Then you will select the target field of the field within the target (current) type. This defines where the data will go.

Finally you specify whether the values in this field will be continued for rows that come from a different list.

To understand the continuation better consider an example where you wish to combine Fees and Payments. If you have two fees and three payments, when those are combined into one list you wouldn't want the 'Fee Amount' to repeat for each of the 'Payment' entries. You may have a calculation for a 'Fees Outstanding' field, and this would be incorrect if the 'Fee Amount' was duplicated. So this field wouldn't be continued.

But if you have another sub-list, for example 'Discounts', you would want the discount percentage to continue until it was changed, so that it can be applied to all calculations following the instatement of the discount, so in this case the continuation would be set.

The next step is to create the calculations. To do this click the 'Calculations' drilldown from the Composite List record.

Applications: LenderEx1 | Model: Facility | Nested Fields: Ledger | Features: | Composite List: 11/28/2010 11:12:55 PM | Calculations

Calculations [Create New](#)

ActualPrinBal ▼ (1 / 11)
[Detail](#) [Edit](#) [Delete](#)
 TargetField: ActualPrinBal
 Formula: (#previous/ActualPrinBal + #current/ActualPrinPayment)
 Notes:

CollateralBalance ▼ (2 / 11)
[Detail](#) [Edit](#) [Delete](#)
 TargetField: CollateralBalance
 Formula: #previous/CollateralBalance + #current/CollateralValueDiff
 Notes:

Coverage ▼ (3 / 11)
[Detail](#) [Edit](#) [Delete](#)
 TargetField: Coverage
 Formula: IIf(#current/ActualPrinBal != 0, (#current/CollateralBalance div #current/ActualPrinBal) * -100, 0)
 Notes:

Figure 75: Example calculations

Then click 'Create New' to add a new calculation:

Calculations ▼ (1 / 1)

ActualPrinBal ▼ (1 / 1)
[Detail](#) [Edit](#) [Delete](#)
 TargetField: ActualPrinBal
 Formula: (#previous/ActualPrinBal + #current/ActualPrinPayment)
 Notes:

[Update](#)

The top 'Target Field' is where you specify the field that will hold the result of the calculation.

Underneath that is a box to place the formula. This formula will be in the typical DbfScript syntax (see section 19). You can utilize the variables #current to access

the current record, and #previous to access the previous record, as the list is being generated.

Finally at the bottom is an area to enter a description of the calculation.

11.23. Holiday Schedules

11.23.1. Overview

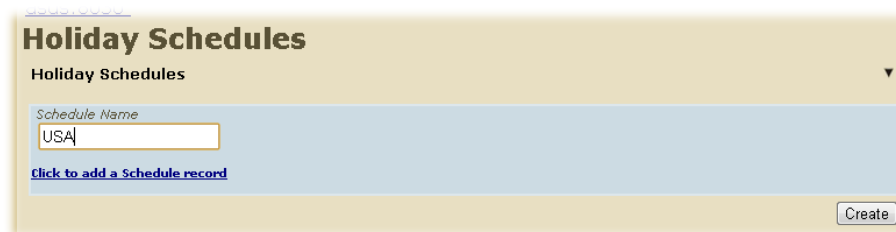
Because dbFramework has many features that involve communicating with users through email, or running certain operations on business days, it needs to understand which days are considered business days and which days are holidays. To maintain this information it provides an area in each application to store the holiday schedules.

Multiple holiday schedules can be stored, allowing the support of different countries for example, or perhaps even for different groups that work on different schedules.

11.23.2. Configuring the Holiday Schedule

The Holiday Schedule list is maintained in the Configuration tab, where you drilldown to the 'Holiday Schedule' list.

Once you are in that list click 'Create New' to add a new holiday schedule:



In this box you can enter the name of the holiday schedule, such as 'USA' for all American holidays. You can also create schedules that mix two different countries' schedules, such as 'USA & Brazil'.

Then click the 'Add a Schedule Record' link to add each holiday entry. You'll have to put each holiday date for each year, so this is something you may have to revisit from time to time.

Here you enter the holiday name, such as 'Memorial Day' or 'Christmas Day', followed by the date, which can also be selected from the dropdown date selector.

11.23.3. Holiday Schedules in Scripting

To add a holiday schedule selector to a form, you'll have to use the advanced link option. This is because the holiday schedule is a structure created when you publish the application, and isn't available to select before publishing.

Once you are in the 'Advanced' drilldown from a field, set the 'Advanced Link' field to be:

```
/StandingData/HolidaySchedules
```

Once published, this will then provide a way to select the holiday schedule to use in that form.

You can then easily integrate the defined holiday schedules with your DbfScript scripts.

From within the script you can use a special method called 'IsHoliday'. This takes parameters for the link to the selected holiday schedule, the date to check, and a boolean of whether or not to include weekends as a holiday. It returns TRUE if the specified date is a holiday according to that schedule.

For example, if a record had a holiday schedule selected in a field called HolidaySchedule, you could use some code like this:

```
if (IsHoliday(#input/HolidaySchedule, #date, true)) then
    message "Holiday", "The date is a holiday"
end if
```

11.24. What-If Server

One of the great advantages of using a hierarchical database is the support for the clipboard. When you copy and paste one record, you're also copying and pasting

everything that is contained within that record – all sub-records, and all file data within that.

This is especially useful for creating ‘What-If’ scenarios, which can be invaluable to making business decisions.

By taking a copy of a record, the system automatically gives the record a new name (usually putting a (1) at the end). You can then rename that record, for example, to include ‘Scenario 1’ at the end.

Once the record is copied, it is then disconnected from the other record. You are then free to modify that record any way you wish. All of the same reports and actions are available, for you to evaluate the impact of making those changes.

To completely isolate this record from the main system, a secondary server can be set up that mirrors all of the data on the primary server. This ensures the copied record is completely up to date, but guarantees that it won’t affect any of the normal daily operations.

For information on setting up a secondary scenario server, see section 14.5.

12. Help System

12.1. Overview

An integrated help system is an essential part of any application, both for helping new users learn to use the system, and also for reference purpose for more seasoned users.

dbFramework makes it an easy process to add a comprehensive help system to your applications.

In most development platforms adding help is a complicated process, involving using third-party tools and complex programming. However for dbFramework the help designer is in-built and can be added and edited right from within the browser.

The help system in dbFramework is dynamic and provides help pages that relate to the active context: for example if you're viewing a user record the help page available will tell you about the user record, how to edit the user record, what the fields mean and what actions and reports are available to you.

Developing the help system can be done piecemeal, with help pages added for specific aspects of the application when and if they're needed.



Figure 76: Example of a help page in dbFramework

12.2. Adding Help

Help pages can be added to any field in the data model . There are two types of help that can be added:

- Tooltips , which appear when the mouse cursor hovers over a field label, can be added by entering text into the description entry of the field definition.
- Help Pages, which appear in the main page and are available as a 'Help' tab above the data record, can be entered through the 'Help' drilldown from a field definition.

12.2.1. Entering Tooltips

To enter a tooltip, go to a field definition and enter the text into the 'Description' entry box. Once you republish the application you will notice the description appear as a floating 'tooltip' when the mouse hovers over the caption above the field entry box:



12.2.2. Entering Help Pages

To enter a help page, go to the field definition where you wish the help page appear. Click on the 'Help' drilldown and then click 'Create New'.

Help ▼ (1 / 1)

Title: Help Type: Language Code:

Text

B *I* U Font Size... Font Family... Font Format...

A Facility contains information on the credit line that is established for a particular client.
This is the facility tab where you can define each facility for each [/Contact:customer]}

The following fields can be entered:

Title

The title of the help page, allowing you to override the 'Help' title.

Help Type

Specifies whether the help is intended to be part of the user guide, or a more general how-to article.

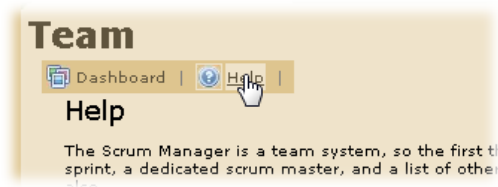
Language Code

Optional. Specifies the language code for this help article if not the default language. The language code should be specified in country-region format. Multiple help pages can be entered, one for each language code, to support multiple language applications.

Text

This is where the help page text is entered. The help page text is entered in rich-text with the ability to control formatting and layout.

Once the help page is saved, republish the application to see the help link at the top of the screen.



12.2.3. Linking Help Pages

You can also insert links in the help page that take you to the help page for another part of the application.

The format for this is:

```
[path:text]
```

Where 'path' and 'text' are replaced with the location in the application of the help topic, and the text that will be inserted into the help page. The help text will become a link that, when clicked, will take you to that help page.

To specify the location of the help text you're referring to, you use the familiar XPath format (described in section 6.15). For example, to refer to the Team top level item help topic you would use an XPath of /Team. To refer to the Task underneath a team, you would use an XPath of /Team/Task.

For example, to link to the task log's help page, you might use something like:

```
You can also add a [/Team/Task/Log:log] entry to the task.
```

In the help box this would display as:

```
You can also add a log entry to the task.
```

12.3. Using the Help System

Wherever the Help entries are available, you will see the 'Help' option available at the top of the screen as a tab beside any other custom views you have configured in that field.

Clicking the 'Help' tab will display the help text. Within the help text you may have links to help pages in other areas of the application. At any point you can click the 'Back' button in your browser to return to the previous screen.

13. Security

13.1. Overview

The security features of dbFramework allow you to control who has access to the application, which parts of the application they can access, and whether this access extends to viewing, editing, creating or deleting entries in that view.

13.2. Concepts

When you access a dbFramework application the user is logged on using a guest account. By default the guest account is enabled, but it can be disabled forcing the user to login through a setting on the Application record.

When the user logs in, they can then only use the application according to the restrictions defined for their user.

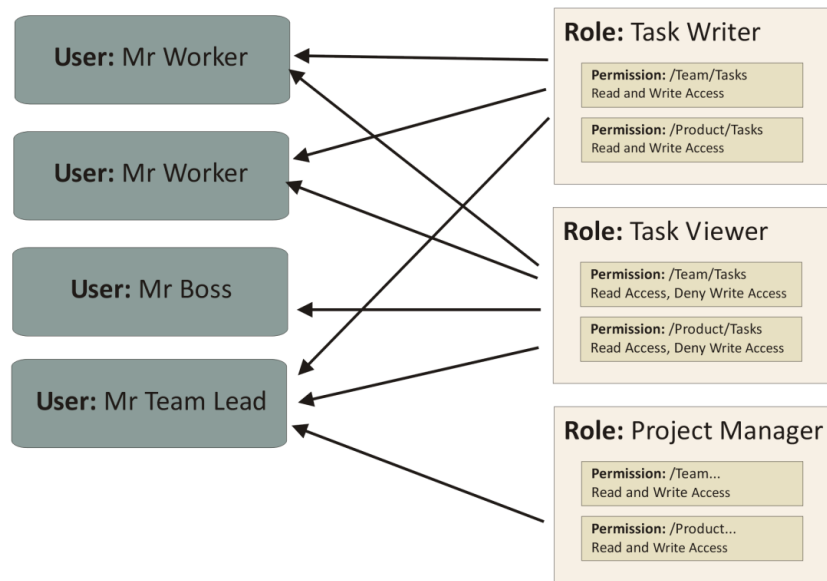
These restrictions are defined based on the selection of something called 'roles'. Each role has one or more *permissions* that describe how it can access a certain area in the application.

The roles are defined when designing the application in dbFramework. The users and their roles, however, are defined when using the application. That's because the roles typically are defined when you create the application, but the users and their permissions may change frequently.

Each security permission applies to a specific location in the data model, and applies one or more access restrictions.

These permissions can be hierarchical, so that they apply not just to the area where you apply it, but also every nested type below that in the data model hierarchy.

An example of the relationship between these concepts is depicted below:



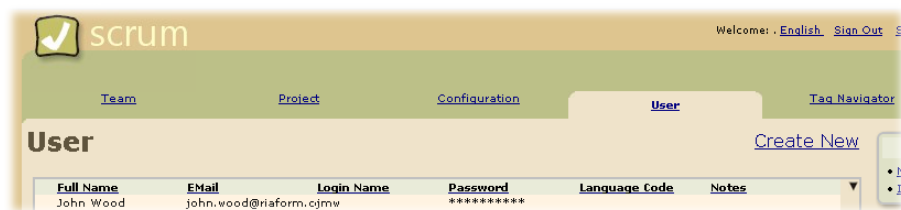
13.3. The Administrator User

Every dbFramework application has a special login called the 'admin' login. The 'admin' login has full access to everything in the entire application, so it should be guarded carefully. It is used to first access the application and set everything up, including the users (see the next section).

The username and password for this login is entered in the application record from within dbFramework. If you want to disable the admin user entirely, then just make the admin user name blank.

13.4. Creating Users

Except for the admin login, the login process validates the login name and password against the information held in the user table in the application itself. The list of users isn't changed at design time, it's managed entirely from within the application itself. All dbFramework applications have a user table, and this is accessed through the 'User' tab at the top of the application:



Once you are on the User tab, click 'Create New' to add a new user.

Note: It should go without saying that you will need to be logged-in with a user that has sufficient permissions to update the user table. The admin login is useful for this.

The user entry form will look something like this:

The screenshot shows a web form titled "User" with a sub-header "John Wood" and a dropdown menu showing "▼ (1 / 1)". The form contains several input fields: "Full Name" (containing "John Wood"), "Email" (containing "john.wood@riaform.com"), "Login Name" (containing "jmw"), and "Password" (empty). Below these is a "Language Code" field (empty) and a large "Notes" text area. An "Update" button is located at the bottom right of the form.

The following fields can be entered:

Full Name

This is the full name of the user, as it should be displayed in emails for example.

Email

Specifies the email address used to contact this user.

Login Name

This is the login name that the user will need to enter into the login page to access the application. The login name should not contain spaces.

Password

Specifies the password the user will enter to access the application.

Note: When this field is edited, the previous value is blanked out for security reasons. Also the value of this field is not transmitted in emails.

Language Code

Optionally specify the default language code this user prefers. This controls the default language that will be activated when this user logs into the application.

Notes

Optionally specify any notes about this user or specific login (eg. to indicate a temporary account).

13.5. Creating User Groups

User groups consist of a list of users that share some common responsibility or role within the system. Users can belong to multiple user groups, or they can belong to no user group at all.

User groups are used in the workflow operations such as the Approval Workflow (section 11.2) or Document Review (section 11.13).

How the user group is selected depends on the governance of the user group, and this is discussed in more detail in section 11.2.4.2.

To create a new user group, go to the Configuration tab, and in the configuration record drill down to 'User Group'. When you're there, click 'Create New':

The Name field holds the name of the user group, which should be fairly short but can contain spaces.

The Locked field is used for data synchronization and will be discussed in a later section.

You can then click the 'add a new User List record' to start adding the user references to the user group.

13.6. Creating a Role

As mentioned in the Concepts section, the security roles are defined in the designer and not in the application itself. The security roles are static parts of the application, and only the selection of roles in each user record should change frequently.

To create a new role, go to dbFramework, select your application record and drilldown to the 'Roles' list. There click 'Create New':

Roles

Roles

Name

Description

Permissions

Field Apply to Sub-Types Specific XPath Allow / Deny

View Update Create Delete

Owner Only Approval Workflow Exclusion

Default Filter

Filter Field Value Expression

[Click to add Permissions](#)

Create

The Name and Description of the role should describe the security role.

You can also set the 'Is Default' field. If you set this to 'Yes' then it will be added automatically to all users in the system, except those that are administrators.

Below that click 'add Permissions' to add and attach a permission to the role. Any number of permissions can be added to a role.

Each permission defines access to a specific area of the data model. The following fields can be entered:

Field

Specify the field definition within the data model that this permission applies to.

Apply to Sub-Types

Indicates whether this permission will also apply to all descendents of the selected field in the data model.

Specific XPath

An optional, advanced option. Allows an XPath to be specified instead of selecting the field. This allows for more advanced options, such as specifying something like `"*/TypeName"` to specify all areas of the data model using the specified type name.

You can also enter in paths of specific records in here, if the XPath to that record is known. This allows permissions to be set up that are specific to existing records in the database.

Allow / Deny

Each permission either enables or disables the specified access. This entry indicates whether this permission will allow (enable) or deny (disable) the given access.

For example this permission may deny deletes. In that case this will be set to 'Deny' and 'Delete' will be set to 'Yes', meaning that the permission affects deletes, and it should deny them.

View

Indicates if this permission action (or allowing or denying access) refers to viewing records.

Update

Indicates if this permission action (or allowing or denying access) refers to changing (updating) records.

Create

Indicates if this permission action (or allowing or denying access) refers to creating new records.

Delete

Indicates if this permission action (or allowing or denying access) refers to deleting existing records.

Owner Only

Indicates if this permission action (or allowing or denying access) refers to allowing write access to records the current user created. (Note: In dbFramework v1.0 this is not supported).

Approval Workflow Exclusion

Indicates if this permission refers to the approval workflow exclusion. By default all users are excluded from approval workflow, meaning they can freely change the database without having to get approval from another user. However by denying the approval workflow exclusion, this means that the user no longer has free access and all changes with this permission applied will be routed through the user

group that has workflow governance over the area being changed. If no group has governance, the change goes through directly.

Filter Field

Enables applying a read filter to all requests in this area of the data model, so that only results with this filter applied will ever be returned for users with this permission active. This can be used to restrict access to a specific user area, for example. The Filter Field specifies the field name that will be used in the filter that will be applied.

Value

If filtering is employed, specifies the value to filter with.

Expression

Currently only supports the command 'UserID', which gets the ID of the user record and applies that to the value.

13.7. Restricting Individual Record Access

While the security system described so far makes it easy to enable or disable access to specific *types* of data, we haven't described a way to restrict access to a *particular* record or list. We can do this using the filter option in a role.

For example let's say we wanted to create a security role that stopped users from viewing anything except their own user record. We can do this by creating a role with a filter set.

The name of the role could be 'My User Only'. We would set the 'Allow/Deny' to be 'Allow' and not set any of the commands (ie. Create, Update etc.). Instead we just set the Filter Field to be 'UserID'. We leave the Filter Value and put a Filter Expression of 'UserID'. This tells the system to use the currently-logged-in user's ID instead of a hard value.

After creating this role, publish and then select that role in the user with which you'll be testing (This is discussed in the next section).

Next, login as that user. You should now only be able to see your own user record and nobody else's. The same will apply for any reports that you run. Whenever a query is made to the Users list, it will enforce the specified filter. If you attempt to run a script that modifies another user record, that operation will fail saying it cannot find any other records to update.

Note: It would be a good idea to have

13.8. Assigning Roles to Users

Once the roles are created, they can then be attached to specific users. This is done in the application itself at runtime.

To do this, go to the user's record in the 'User' application tab and drilldown to 'Roles'. Then click 'Create New' to add a new role, select the role and click Update.

The screenshot shows the 'User' application interface for user 'John Wood'. At the top right is a 'Create New' link. Below the user name are tabs: 'List All', 'Detail', 'Edit', 'Delete', 'Goto', 'Availability Schedule', 'Skills', 'Work Log', and 'Roles'. The 'Roles' tab is selected. Below the tabs is a table with columns: Full Name, EMail, Login Name, and Password. The table contains one row for 'John Wood' with email 'john.wood@riaform.com' and login name 'jmw'. Below the table is a 'Quick Entry' section with a dropdown menu showing 'Admin' and 'Project Workflow'. An 'OK' button is next to the dropdown.

Alternatively you can select the role and click OK using the Quick Entry in the Roles tab below the user record.

Note: It's important to make sure you create a Role that disables access to the User list so that users cannot assign new roles to themselves.

13.9. Disabling All User Editing

Sometimes certain parts of the database should only be managed by the system, and users shouldn't be able to create, update or delete items in a list. An audit trail log is an example of this.

Rather than settings special permissions for everybody for that area, dbFramework provides the ability to disable certain aspects of editing for anyone using that area of the database.

This is controlled through the Flags drilldown of a field. The three flags, Disallow User Create, Disallow User Update and Disallow User Delete can be applied to stop users from creating new records, updating existing records and/or deleting existing records. Apply any combination of these flags to dictate what functionality will be available to your users.

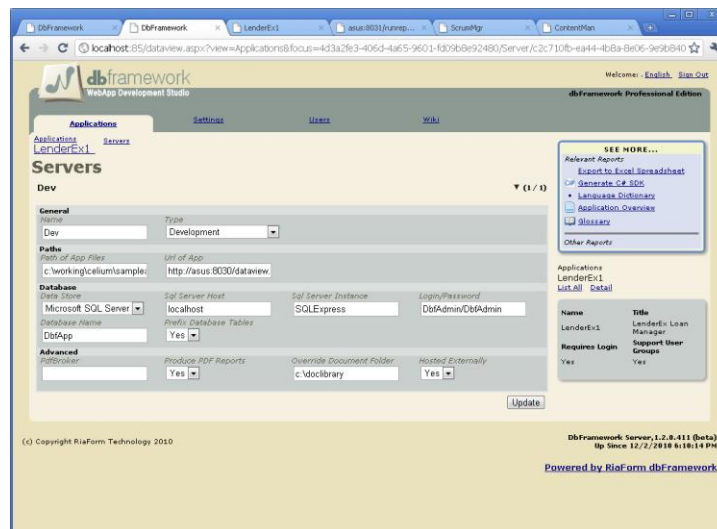
While this disables user-interface features, it still allows scripts to modify that section of the database. You may decide, for example, to enforce special forms for inserting, updating or even deleting items. You can then create special actions with their own input forms that have simple scripts to insert, update or delete records in that area.

14. Deployment

14.1. Creating Multiple Servers

dbFramework provides a number of features to facilitate the deployment of dbFramework applications.

Each application record has a Server 'GoTo' item that lets you define multiple deployments for a single application.



All applications will have a default entry added when the application was first created. Each server entry has the following fields:

- Name: This is the name of the server that will show up on the 'target' dropdown when you publish.
- Type: This specifies the type of the server, which can be one of the following values:
 - Production
 - Secondary Production
 - Development
(The first server of type 'Development' is selected by default if no target is selected during a publish)
 - Secondary Development
 - Testing (Enhanced)
(As well as creating a new server target, selecting this option will

add additional functionality to the server for facilitated testing.
This is discussed in more detail in the section on testing)

- Path of App Files: Specifies where the files for the application will be held.
- Url of App: Specifies the URL of the application once it is running. Also dictates the port number to use.
- Data Store: Specifies the type of data store, whether SQL Compact, SQL Server or XML Database.
- Sql Server Host: If using SQL Server, specifies the SQL Server host name. Can be (local) to use the local server with named pipe connections.
- Sql Server Instance: The instance name of the Sql Server, if any.
- Login/Password: The login and password to use when accessing the SQL Server. If left blank will use trusted Windows authentication to access the server.
- Database Name: The name of the database to store the data for this application.
- Prefix Database Tables: Whether to re-use an existing database but prefix tables with the name of the application, allowing multiple applications to share the same database.
- PdfBroker: Specifies the name of a server running the PdfBroker software to facilitate the generation of PDF files.
- Produce PDF Reports: If set to 'Yes' will generate PDF files from reports, otherwise will produce HTML reports.
- Override Document Folder: If specified, defines the folder in the system where documents will be stored.
- Hosted Externally: If set to 'Yes' then OfficeSuite Launcher will not start this application as it will be managed by Microsoft IIS or another external web server.

You can create any number of servers in this list, and multiple servers can be of the same type. Once each is published they will all be started by OfficeSuite Launcher. This has several uses, including creating backup systems, what-if scenario servers, testing servers, development servers and training servers.

Another powerful use is to create a Preview Server. This lets you roll out new features with a separate URL for people to test and provide feedback before you publish it on to the main site.

14.2. Publishing

When publishing, the parameter form that appears allows you to select the server to publish. This compiles the application and copies all of the changes onto the specified server.

If no 'Target Server' is specified, then the first 'Development' type server is selected for the application and that is used. Typically development servers are updated a lot more frequently than production servers.

Publishing the application increases the version number by 1. It should therefore be impossible to have the same version number on two servers. This is intentional to uniquely identify the specific configuration at the time of publishing.

14.3. Change Log and Notifications

Once this has finished publishing, an entry will be made in the Change Log drilldown from the Application. This entry will describe the time of the release, the type of release, the version of the application and the target that was selected.

Change Log

[Change Log](#) | [Calendar](#)

Release Date	ReleaseType	Version	Target	Release Notes
11/15/2010 1:04:18 PM	Final Release	1.2.0.318	Development	
11/15/2010 12:29:15 PM	Final Release	1.2.0.317	Development	
11/15/2010 10:01:08 AM	Final Release	1.2.0.316	Development	
11/14/2010 12:01:58 PM	Final Release	1.2.0.315	Development	
11/14/2010 11:59:37 AM	Final Release	1.2.0.314	Development	
11/12/2010 4:56:35 PM	Final Release	1.2.0.313	Development	
11/12/2010 4:47:59 PM	Final Release	1.2.0.312	Development	
11/12/2010 3:31:26 PM	Final Release	1.2.0.311	Development	
11/12/2010 2:58:16 PM	Final Release	1.2.0.310	Development	
11/12/2010 2:27:09 PM	Final Release	1.2.0.309	Development	
11/12/2010 1:56:04 PM	Final Release	1.2.0.308	Development	
11/12/2010 11:51:19 AM	Final Release	1.2.0.307	Development	
11/12/2010 9:45:37 AM	Final Release	1.2.0.306	Development	
11/12/2010 9:44:57 AM	Final Release	1.2.0.305	Development	
11/11/2010 11:00:51 PM	Final Release	1.2.0.304	Development	
11/11/2010 6:19:17 PM	Final Release	1.2.0.303	Development	
11/11/2010 3:39:27 PM	Final Release	1.2.0.302	Development	
11/11/2010 2:20:41 PM	Final Release	1.2.0.301	Development	
11/11/2010 2:18:41 PM	Final Release	1.2.0.300	Development	
11/11/2010 1:13:25 PM	Final Release	1.2.0.299	Development	

[Next Page >>](#)

dbFramework also allows for people to receive notifications about releases that are made. This can be particularly useful if you want to keep track of changes made to a production system.

To enable these notifications you need to go to the application record and find the 'Notifications' drilldown item.

In there, you can create a notification, select the user that will be notified and the type of notification (which should be 'New Release'). That user will then receive an email notification each time a publish is performed.

14.4. Feature Modules

14.4.1. Overview

One particularly powerful deployment feature in dbFramework is the 'feature modules' capability. This allows you to separate specific functionality in your application into different 'feature modules' or editions. Each server target can have any number of these 'feature modules' enabled or disabled, and that will determine the functionality that will be enabled in that particular application instance.

For example you may have a feature module called '**Professional Edition**'. For some special actions that you have created you may want these to only be available for the professional edition of the application. So you go to those actions, you drill down to the 'Required Modules' list, and add 'Professional Edition' under that. Then, in your server targets, you create both a Standard Edition entry and a Professional Edition entry. Out of these, only the Professional Edition entry will

have the 'Professional Edition' feature module selected. Once published, that functionality will then only be available in the Professional Edition server.

Another powerful use for this feature is to create a feature module for 'Work In Progress' features. As you are developing features you may want to indicate that these are part of the 'Work In Progress' feature module (WIP), and exclude that module from the production server target, ensuring that nothing half-implemented ends up in production.

14.4.2. Creating Feature Modules

To create a feature module, go to the Application record and drill down to the 'Feature Modules' list. Click 'Create New' and fill in the fields as follows:

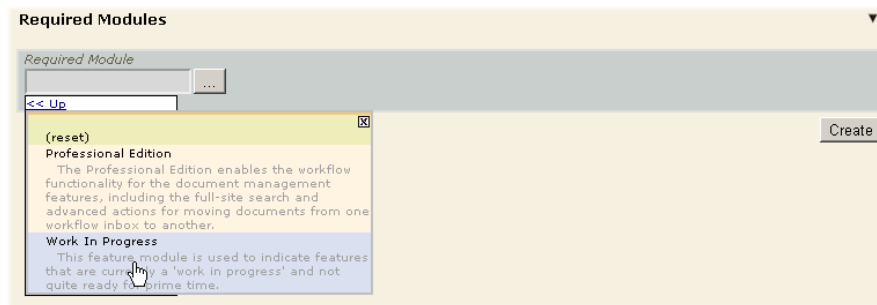
- **Code:** This is the short name for the feature. For example, for 'Work In Progress' you may enter WIP for the code.
- **Long Name:** This is the human-readable long name, for example 'Professional Edition'.
- **Description:** This is where you can enter in any amount of text describing the feature module.

14.4.3. Allocating Features

The next job is to allocate features to a specific feature module. You can do this in a number of places in the data model:

- Fields and types
- Actions
- Reports
- Events
- Scheduled Jobs

In each case the allocation is performed the same way. Find the drill down called 'Required Modules'. When in that, click 'Create New' and select the feature module that will be required for that field, action, report, event or scheduled job to be enabled.



Note: When a feature module is disabled and the application published, all data entered in the fields dependent on that feature module will be removed from the system. Please use with caution.

14.4.4. Selecting Feature Modules

Each server target can have any number of feature modules selected, or none at all. To configure each server target with specific feature modules first go to the Application record. Then drill down to Servers to select the specific server target you wish to update. Then drill down to 'Selected Modules'. This is where you will select the feature modules that will be enabled for that particular server.

For example, in the 'Development' server you may wish to include the 'Work In Progress' feature module.

14.5. Default Database

14.5.1. Overview

When a published application is first run, especially in a new environment, the database will be empty. It may sometimes be desirable to provide a default set of data, perhaps sample data, so the user has somewhere to start.

dbFramework offers the ability to provide this default data. It does this by taking a snapshot of the database in an existing application, storing this as an XML file, and attaching it to the application configuration. This XML file is then placed into the application folder when it is published. Then, when the application is started, if the database is empty it will load this XML file and reconstruct the database.

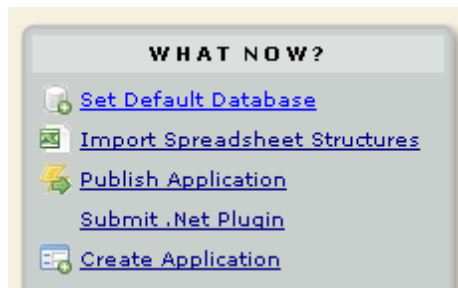
14.5.2. Adding the Default Database

The default database is configured on a per-server basis. Each server target, as specified in the 'Servers' dropdown from the 'Application' record, can have its own default database.

Before you create your default database, go to the running application and set up the data the way you wish it to be.

Note: Because the data is stored in an XML file, the database cannot be very large. For example you should make sure the number of records is under 1,000 maximum. You must also consider the size of any file attachments that are in the database, as these will be incorporated into the XML file.

Next, select the 'Server' entry that you wish to setup with a default database.



Next click the 'Set Default Database' link in the task list. This will bring up a form that will let you select the server of the running application from which you want the data to be taken. It does this by looking at the target URL of the selected server, and then asks that application to download all of its database into an XML file.

 A screenshot of the 'Set Default Database' form. The title 'Set Default Database' is at the top. Below it is a sub-header 'SetDefaultData'. The form contains three input fields: 'Application' (with a dropdown arrow), 'User Name', and 'Password'. Below the 'Application' field is a small dropdown menu showing '<< Up', '(reset)', and 'Test'. A 'Create' button is located at the bottom right of the form.

It then takes this resultant XML file and attaches it to the server. It places it in the 'Advanced' drilldown from the Server record, in a field called 'Default Database'. You can also edit this record directly and upload your own XML file.

Note that you can also enter in a user name and password. This may be required if login authentication is enabled on the server.

Now it is attached, the next time you publish the application using that server, it will put that XML file into a file called 'default.xml' in the Files folder of the server.

When the application is first run with an empty database, it will then load that XML file and recreate the database.

This is also useful if you distribute your application to others. This feature lets you easily add some sample data to your application. It can also be used to set up basic configuration information that will always be available when the application is used.

14.5.3. Keeping Data Private

When taking a snapshot of the database, you may not want all of the data to be included in the default database. Perhaps you also use the application and only want to make sure certain, private records are excluded.

dbFramework lets you hide certain records from the snapshot process. If you add a field to the record with the name '_Public' (note the underscore prefix), then setting this field to 'No' will tell the snapshot process to exclude that record and all its child records from the default database.

If the field is not present then the record will always be included.

14.5.4. Removing Sample Data

It may be likely that the user of the application will, at some point, want to remove all of the sample data that was entered. While they can easily delete the entries themselves, dbFramework adds a special action called 'Remove Sample Data' that can be run at any time in order to remove all of the entries that were imported from the default.xml file.

However, to allow the user to keep some entries that have been customized by them, this action will only remove entries that have a _Public field with a value of 'Yes'. This enables users to set that value to 'No', therefore keeping the record private, and stopping this action from removing that data.

14.5.5. Reset Sample Data

A second action that is added enables the user to reset their database to the sample data at any point after installation. If, for example, you update a new set of default data, this allows users to import that updated data without losing other data they may have entered.

This action first deletes all of the public entries that were imported from the sample data. It then imports the data from the default database and merges this with the existing data in the database.

14.6. Synchronizing Servers

14.6.1. Overview

It's often very useful to have several servers, as discussed in section 14.1, either to have a backup system, or perhaps so you can have a 'What-If' server that you can tinker with to evaluate different scenarios or proposals.

But having multiple servers isn't very useful without any data. Having to re-enter your data to each server would be tedious and prone to error.

dbFramework provides an easy solution to this: with just a couple of settings you can configure your servers to mirror one another. This means that any data entered on the primary server will automatically be entered into the secondary servers without any manual re-entering of data. And this also extends to the results of calculations or other script-entered data.

Technically this is achieved by having one server log all changes that are made to its system into a transaction file that exists in some area that all servers can access (such as a network share).

The secondary servers then monitor that file, and whenever information is added to the file the server will read it, apply the changes to its own database and remove that entry from the file.

This monitoring process occurs at a configurable frequency, usually every few minutes. Because it does not have to occur in real-time this also enables the possibility of having multiple servers synchronized over a wide area network, such as between two offices that are connected via a virtual network over the Internet.

14.6.2. Configuration

In your server configuration record, drilldown to the Advanced option:

The four options underneath the title ‘Synchronization’ are what we’ll be looking at:

Feed From File

This is where you enter the name of the file that the server will read from. It will monitor this file and commit any changes that are written to it. You will set this only on the server that will be receiving the updates.

Feed To File

This is where you enter the name of the file that the server will write to. It will write any changes committed to the database to this file, expecting another server to periodically pick up the changes and delete the file.

Feed From Frequency

This is the number of minutes between each time it checks the ‘Feed From’ file.

Feed File Max Days

This is the maximum number of days to keep in the ‘Feed To’ file. This is just a safeguard in case the ‘feed from’ server stops reading the file.

14.6.3. Locking Records

You may want to be more selective about which parts of the database are synchronized, and which are left alone. dbFramework allows this to be controlled by the user by the introduction of a special field called ‘_Locked’.

By adding the _Locked field to any type, you provide the opportunity for the user to indicate that this particular record, and any data underneath it, cannot be overwritten by the system during a synchronize.

Some areas of the database have this available automatically, such as the user group area. This is because you normally wouldn’t want approval workflow messages to be sent to the secondary server because they never occurred there.

But you are free to add the `_Locked` field to anywhere in the database.

14.6.4. Technical Details

The transaction log file must be completely accessible to both servers.

In addition, the 'Feed To' file name can consist of multiple filenames, each separated by a semi-colon. This will write the same information to all of those files. This allows multiple 'Feed From' servers to be set up. They each need their own file because they delete that file once they read all of the transactions for that period.

The transactions are stored in XML format, with special codes inserted to separate the transactions.

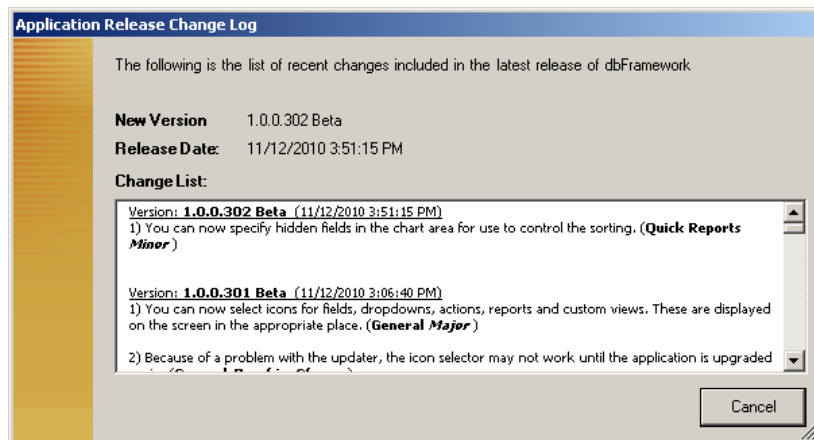
Keep in mind that file data is also stored in this file, encoded into base 64 format. These files can cause the XML file to get quite large. Also keep in mind that if the file is too large and the server memory is fragmented you may receive an 'Out of Memory' error. You should periodically monitor the `systemlog.aspx` file of the servers to check for any errors such as these.

14.7. Upgrading

Whenever an application is republished, the first time the newly published application is accessed dbFramework analyzes the database at hand to determine what will need to be changed to run the new application updates. It does this all automatically without any intervention required by the user. It also performs the upgrade once, and it won't need to be run again until more changes are made and the application republished. During this time the application will display a 'Please Wait' caption at the top of the browser screen, and will automatically reload the page when the server upgrade is complete.

The dbFramework system itself often has updates available, and updating the system is free and will usually have a positive effect on your applications with the introduction of new functionality, fixes to existing problems, or performance enhancements.

All upgrades to dbFramework are performed in the OfficeSuite Launcher application. This usually happens automatically, as the system scans for updates every few hours. When an update is found, the OfficeSuite Launcher application will show a window similar to this:



This window describes the changes that have recently been made to the application, but the upgrade window will also offer the option of downloading and installing the upgrade automatically. Once the intent to upgrade is confirmed, the download occurs in the background, with progress displayed in the task bar. Once downloaded the upgrade is installed and you will be asked to restart the server.

Note: You can click 'No' to ignore this update. If this update is ignored, you'll have a chance to upgrade again the next time an update is published. You can also click 'Cancel' which means 'Not right now'. After a few hours have passed it will ask you again if you wish to upgrade.

Note: Updates are cumulative, so each update incorporates the changes from the previous updates.

After OfficeSuite Launcher is restarted the recent changes will be enabled across all of your applications. Also, occasionally you will need to republish your applications to see the changes.

Always read the list of changes that go into the release, and keep in mind that you may have skipped a couple of releases so look at all the changes that have occurred since your current version of dbFramework. There may sometimes be instructions in the change list that need to be followed, and sometimes new features will include a description of how to use that feature.

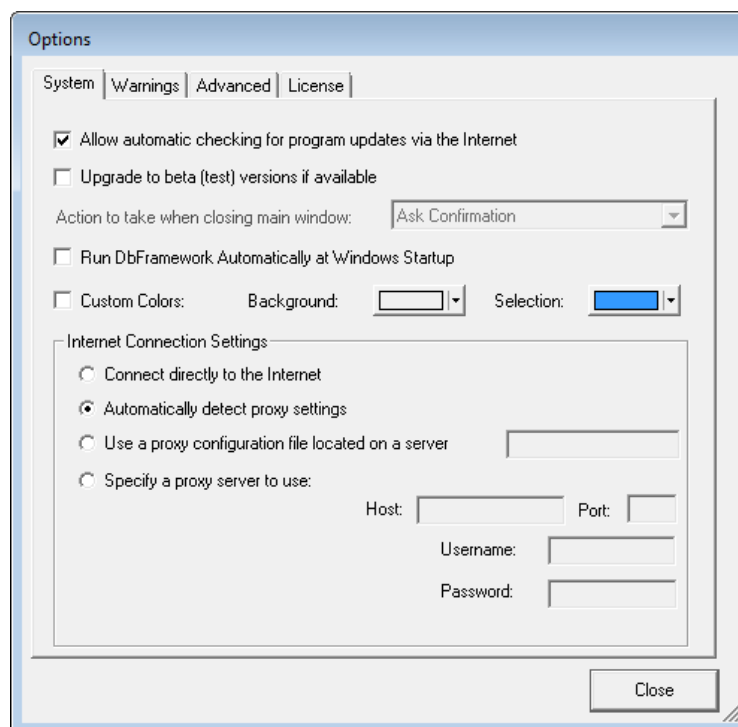
If you do miss the change list, you can always retrieve it at any time by going to Tools / View Recent Updates.

14.8. Beta Testing

dbFramework updates frequently, but for those willing to try out new features early, that may not yet be complete, or that may not yet be fully tested, dbFramework can be configured to upgrade to something called beta-test versions. This means that dbFramework will upgrade more frequently, but may upgrade to versions that are not always reliable.

Important! Please keep in mind that beta versions should never be installed onto production environments where the system is depended upon for business critical operations. If you wish to try out new beta versions, please install them on a separate system that can be used independently of the usual business operations.

If you wish to take this risk and enable upgrading to beta-test versions, then go to the Options in the dbFramework Application Monitor application and select 'Upgrade to Beta (Test) Versions if Available'.



Note: If you already have a beta version installed, then dbFramework will automatically download beta versions until the final version is made available. After that only non-beta, release versions will be installed.

14.9. Using DbfServer Independently

While OfficeSuite Launcher will run the servers automatically, you can also manually start a server from a command prompt using the DbfServer.exe utility.

This small utility starts up a specific application from the command line.

```
dbFramework Application Hosting Server v1.0 (c) 2010 RiaForm Technology
Usage: dbfserver server-folder [app-folder portNumber, ...]
```

The first parameter is the folder location of dbFramework. The second parameter is the application folder (where the app.ini file resides), and the third parameter is the port number to use.

Once the server starts, you can close the server by typing CLOSE and pressing ENTER.

Important: OfficeSuite Launcher closes all DbfServer processes on the machine at startup. For this reason you cannot use DbfServer independently and then use OfficeSuite Launcher.

Note: DbfServer must be run with elevated permissions.

14.10. Using dbFramework with Microsoft IIS

14.10.1. Installing IIS

Microsoft Internet Information Services is the name given to Microsoft's web server technology. Depending on your version of Windows this may or may not be installed already on your machine. To install it manually, follow these directions:

- In Windows Vista, open Control Panel and then click Programs and Features.
- In the right pane, click Turn Windows features on or off. The Windows Features form will open.
- Select the Internet Information Services check box.
- Double-click (or expand) Web Management Tools, double-click IIS 6 Management Compatibility, and then select the IIS 6 Metabase and IIS 6 Configuration Compatibility check box.

- Double-click (or expand) World Wide Web Services, double-click Application Development Features, and then select the ASP.NET check box.

• **Note:** The related options that are necessary for Web application development will automatically be selected.

- Click OK to start the IIS and ASP.NET installation process.

14.10.2. Configuring dbFramework

dbFramework comes with an embedded web server to makes it easy to get started with your applications. However this web server, while capable of serving a fairly large number of users, lacks the sophisticated functionality of a professional web server. To use features such as unattended operation, web farms, secure sockets and certificates, to name just a few, you will need to use dbFramework with Microsoft Internet Information Server, which comes with Windows Server.

Luckily dbFramework supports Microsoft IIS with just a simple download called the dbFramework Connectivity Pack, which can be installed by contacting RiaForm Support.

This download will contain a number of DLL files that will need to be placed in the application's bin folder. And, to make the transition easier, dbFramework's folder structure reflects the structure required by Microsoft IIS. You will simply need to point IIS to the application folder. You will also need to enable ASP.Net, and also enable 32 bit (under the setting 'Enable 32 bit') in the Application Pool settings.

After that, a web.config file will need to be created with some special entries.

In the configuration / system.web entry of that file, you will need to insert the following XML:

```
<httpHandlers>
  <add verb="*" path="*.aspx" type="DbfTest.DbfHandler, App_Code"
/>
</httpHandlers>
```

The installation package will also contain an application called PdfBroker, which can be used to make it easier to generate PDF files from the IIS service. For more information on configuring this tool, please contact RiaForm Support.

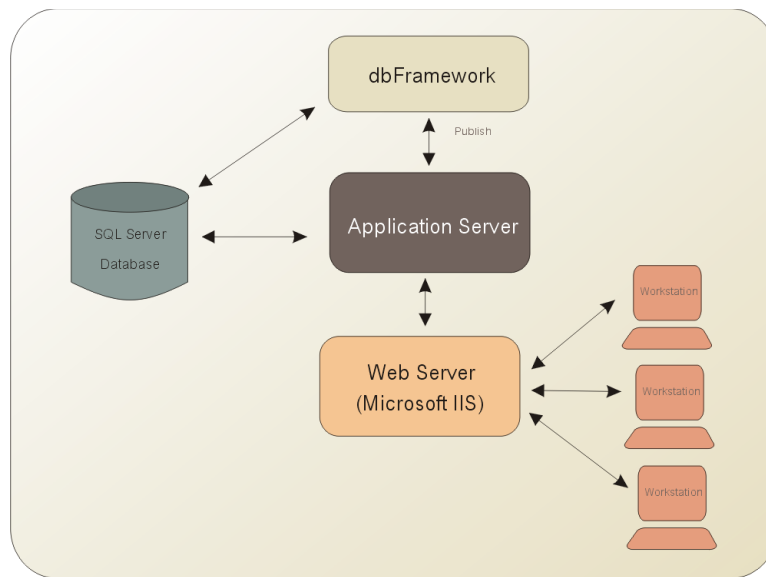


Figure 77: Configuration when using a dbFramework application with IIS

14.10.3. Troubleshooting

It's also very important that you enable IIS for x86 (32 bit) modules. By default on 64 bit machines this will NOT be enabled. You may see an error saying something about a 'Bad image' when accessing the dbFramework application. To enable it, In the IIS management application, go to the Application Pool settings, got to Advanced Settings, find the 'Enable 32 bit' option and change it to True.

There may also be a problem where IIS is configured to deny the use of HTTP handlers, a technology used by dbFramework to manage the functionality of the web application. This setting is configured in the system32/INetSvr/Config folder.

It's also possible that ASP.Net isn't installed on your machine. This can be accessed in Control Panel, in Programs and Features. Expand the IIS option and enable ASP.Net from that list.

Finally, firewall settings may stop remote machines from accessing the server. While DbfServer automatically configured your firewall to allow access, IIS will not do this and you will have to adjust your firewall settings manually.

14.11. Internet Hosting

dbFramework applications can use Microsoft SQL Server for data storage and can be hosted with Microsoft IIS – both of which are technologies used by Internet hosting companies.

This means it's possible to deploy a dbFramework application to a web-host. There are three types of web hosts that are prevalent in the industry:

14.11.1. Dedicated hosting

In this configuration, the dbFramework application is deployed to a server machine that is dedicated to running the application. This is the most desirable configuration because you have complete control over the resources available to your application.

When outsourced this option is often the most expensive of them all. Of course you can always just house the server yourself, but then you must consider the bandwidth requirements you may face if you have hundreds or thousands of users. The bandwidth to support that many users is, itself, often quite expensive.

14.11.2. Shared hosting

In this configuration, the dbFramework application is running beside many other web applications. The database is often also shared among many users. A typical shared hosting application will be installed on a server that may have upwards of 300 other application installed and running.

In a shared hosting environment special consideration should be given to the reduced resources available to the application. This includes processing power and available memory.

In particular memory may be far more fragmented, which means that some operations that require large memory allocations (such as uploading large files, or using the admin tools download options) may result in an 'Out of Memory' error.

Suffice to say, shared hosting is often the lowest cost and most economic of the three deployment options. They often rely on people not using their service very often, and so use the low cost to spread out among many fewer clients than those who have accounts. There are many services that offer both SQL Server database and hosting for just \$5 per month.

14.11.3. Cloud and Virtualized Servers

The third option is known as grid, cloud or virtualization. In this configuration the server you are using is not a real machine, and instead is a *virtual* machine running on potentially many different devices that are all specially interconnected.

This technology is far more dynamic than the above two options, and available processing power is usually guaranteed and easily expandable. Memory availability is also dynamic and can be expanded through simple configuration. This allows the server to scale faster and far more easily because no hardware changes are required to expand the capabilities of the server.

The cost of cloud computing is charged based on usage. It's usually charged per hour of usage and will be charged based on the amount of CPU and memory you wish the application to have available to it.

For applications that aren't used very often, it can be quite affordable. However in that case you may want to consider shared hosting. For applications that need guaranteed response time and availability, and whose user base is dynamic and may grow with little advance notice, cloud computing may be the best option.

In addition, cloud computing usually offers more advanced security than shared hosting can. Servers are protected from other clients, and have a number of failover mechanisms that protect the applications from hardware faults.

14.11.4. Configuration

Each of these deployment options are similar in that you will find a vendor for each that provides Microsoft IIS hosting and Microsoft SQL Server database provisions. Usually you will be given a folder in which to place the application files, and a web-based utility for IIS management (such as activating the folder as a web application).

For more information on setting up IIS to provide the web hosting, see the previous section 14.10.

The database configuration will depend on the particular host. Usually they will give you the machine name of the SQL Server that will be accessible to the IIS machine hosting the application. This should be entered into the 'Server' name in the server record of the application in dbFramework.

Sometimes they may also provide the database name specifically for your account. You can enter this in the area for the database, but make sure you select the 'Prefix tables' option to enable multiple applications to be managed in the same database.

14.11.5. Licensing

Special consideration should be given to licensing. The same restrictions apply to the number of users that can be created in a dbFramework application that is hosted over the Internet that to one running on a local network.

There are no restrictions on the number of physical users accessing the application, but only on the number of unique logins in the application.

For more information on licensing please see section 3.1.

15. Searching

15.1. Overview

The unprecedented success of Google can be attributed to one thing: that providing the ability to find information effectively is more important than the quantity of even the quality of the information held. Search has become the entryway to the Internet.

So it goes without saying that search has become central to our everyday experience using a computer, and people have come to expect search capability in their applications.

dbFramework utilizes an embedded version of Apache's Lucene search engine to provide these search capabilities throughout each application. Lucene is a powerful and fast search engine that is used by many leading websites including Wikipedia.

15.2. Background

15.2.1. Categorizing Documents

Traditionally documents are stored according to a specific category. These categories are often hierarchical. The most common form of categorizing documents is the use of folders in a filing system, and dbFramework makes this incredibly easy to achieve.

For example a document called 'John Smith Review' may be stored under a folder called 'Corporate / Human Resources / Employee Reviews', where the '/' denotes a hierarchical path separator. The folders in this case represent categories that are structured hierarchically.

Once categorized, other users can then locate that document by determining which category to find that document. For many documents this may be obvious. For example, presented with a list of top level categories, in this example as folders relating to each department name, this employee review for John Smith would obviously be stored under the folder named Human Resources. They would then expand that folder category and see sub-folders which represent sub-categories, where they would obviously select 'Employee Reviews' because the document clearly relates to reviews and not, for example, hiring. They would repeat this navigation until they find the document for which they were looking.

For some documents, however, the process of selecting a category is not so clear cut. For example some documents may involve two or more departments. So which department is chosen? One possible solution is to create a special top level entry for combination of departments, and break it down from there, to duplicate the entries, or to simply choose one and hope that people will find it.

However that is not the only possible category that can be chosen, and because of the variety of options available a certain degree of ambiguity arises in the categorizing process. This ambiguity also gives rise to inconsistencies, as different people have different ideas about which category to choose. At best it becomes difficult to find a document, but at worst this results in documents being put into two or more different categories by different people, some containing more recent versions of the document than others. This mistake can be very costly as outdated versions of the document may be found through what would generally be considered legitimate means.

So it should be clear that a rigid structure of categories can sometimes break down. Faced with this possibility, two different approaches can be taken:

1. The category structure that is adopted by the company is more formally and definitively managed. It would be created by business specialists in the company and reviewed frequently according to specific procedures. A concerted effort would be made to communicate these categories to all personnel in the company to ensure all are in conformance. Periodic reviews would be conducted of all the documents in the system to ensure they are properly filed.
2. Alternatively, categories may be abandoned in favor of search. Instead of relying on the categorizing of documents, documents are placed into any folder and discovery of those documents is left to the capability of search technology. The document is then found according to key words that are known to appear within that document title or contents.

The latter option, search, has come into favor in recent years, especially as search technology has improved and digital communications has become the norm for business activity.

However, as will be discussed, neither of these approaches in themselves are a panacea for document filing. The most optimal approach consists in a hybrid solution that incorporates the best of both worlds: the discipline of a well considered, comprehensive yet succinct category structure and the convenience of a search engine to provide quick access to documents that may otherwise be ambiguous to find.

Luckily dbFramework makes it very easy to implement this hybrid approach right out of the box without any extra programming.

15.2.2. Discovering Documents

15.2.2.1. *Search Engines are Not Intelligent*

As described in the glossary, a Search Engine is an application that indexes documents according to terms used in those documents, building an index that can then be used to locate documents that reference the words that are sought.

It should be stressed, however, that Search Engines are not intelligent.

Finding the most optimal means of indexing a file is the Holy Grail for many companies, and many different approaches have been patented by companies where search technology is seen as a most valuable asset. The industry is competing vigorously to lead in the market of providing search capability to the vastness of information available on the Internet and ordinary businesses. In part this has fueled the fast development of search technology which has benefited the enterprise search market tremendously.

However, while these search algorithms have been greatly refined and optimized, so long as the computer cannot understand the document, the process of indexing will never be perfect.

Indexing a document involves not just a matter of extracting words out of the document to add to an index, rather the words are given an importance-rating. For example, if the word 'sugar' appears in a document in headers rather than just in the body text, then it will be given a higher importance rating. The same goes for whether words appear near the beginning of the document (e.g. in the introduction) and how frequently words appear. Also if words generally related to sugar appear in the document (for example, cane) then that might also improve the importance-rating of any instances of the word sugar.

But for all the clever techniques that are used to determine this importance rating, at best what it arrives at represents a very poor approximation and often it's only luck or accident that leads one to find the document that is required.

So, for example, with search technology one cannot guarantee at all that a document would be classified as a employee handbook document just because the document mentions the words employee and handbook. That kind of categorizing

can only be determined by someone who understands the document and the context in which it was written.

Another important consideration is that, with thousands of documents, many documents will be competing to arrive at the top of the search results and be seen by the user. In the eyes of a search engine the only way to differentiate two or more documents is by unique, distinct words that appear in those documents and not in others, and that these words are included in the search terms. One simply cannot guarantee that documents would include enough unique words to make them sufficiently easy to locate using these search algorithms.

Lastly, the topic of confidentiality is an important consideration. There is no means to ensure that any search technology would be capable of discerning between confidential and public emails or attachments thereof. While many companies may strive for a certain degree of transparency in its operations, there is a certain limit to that, especially when the correspondence is dealing with personnel issues that have no place in search results made by any employee.

Therefore it must be understood that despite the maturity of search technology, these techniques certainly do not offer a replacement for an expert opinion on classification of documents or the significance of certain words in a document.

15.2.2.2. Scanned Documents

Not all documents contain machine-readable text. While Word documents, spreadsheets, emails and some PDF documents consist of text, some documents that are scanned produce image files, and not textual documents. Computers cannot 'read' images the way humans do, instead they rely on having access to something that was typed into the computer. Ordinary scanning results in a file that is akin to taking a photo of the document, and because the computer cannot read the image the resultant file cannot be searched.

A technique known as OCR, or Optical Character Recognition, can be used to convert a file that was scanned as an image into a format that can be read by a computer. This technology analyzes the image to find words that it can recognize, and then effectively 'types' those words into a document that is stored with (or replaces) the original image.

OCR is far from perfect, and the reliability depends greatly on the quality of the original scanned document and the consistency of the writing in the document. Many computer based techniques to recognize characters have problems with ambiguities, much as humans do when reading documents.

Not only is OCR problematic in terms of reliability, it is also a slow process. To convert a multi-page document into computer readable format can often take several minutes. It also has to be done manually by a human as it is difficult to set this up to work in even a facilitated manner. Therefore it must be accepted that not all documents can be indexed effectively, at least not without a considerable investment in time and labor.

15.2.2.3. Don't Forget Proper Classification

One problem with search technology is that it can lead to a reliance on these unreliable search algorithms at the cost of properly classifying a document or record.

The result can be even more problematic than a poorly designed category structure or the ambiguities of categorizing certain documents. In some cases the documents may not be discoverable at all because the content does not contain sufficiently unique words to differentiate it from other documents, and without any human categorization the documents would effectively disappear into the ether.

15.2.2.4. Search Indexing Isn't Always Instant

Even the most advanced search technology will take some time to index files. Depending on the way the search engine is set up, it may take minutes for the search engine to update its index with changes made to documents or newly submitted documents. This means that documents submitted by someone may not show up in search results for a certain amount of time after the fact, which may become misleading and lead to mistakes.

15.2.2.5. Searching Requires Some Skill

As anyone who uses the Internet frequently would know, there is undeniably some skill involved in learning how to use a search engine. While at a basic level one can simply enter in 'sugar industry report' in a search engine and assume the computer will be intelligent enough to know how to find the document, as we have discussed the reality is quite the opposite.

Search technology allows the use of complex expressions to clarify the intent of the search and narrow-down the results. For example you can indicate which words must be included in results, and which words might be included in results, and even which words should definitely not be included in the results. One can combine words into terms that must appear next to each other, and specify which words should be close to each other, and so on.

Understanding how to use a search engine effectively is certainly a skill that can be learnt, but many users will probably never master. However, navigating a category hierarchy, on the other hand, is straightforward enough that (given enough time) any user can probably locate the appropriate folder for a document.

15.2.2.6. Tagging: The Hybrid Approach

As such, searching really shouldn't be the primary approach to discovering a document. Any document that is submitted to, or stored in, a system must be first classified according to a well documented and thoroughly vetted category structure. Only after the document is classified and stored should the search engine be able to index the file so that search can be used as a complementary means of discovering a document.

One such hybrid approach that has become more popular in recent years is called tagging. Tagging is a way to let the business expert provide a subjective opinion as to the indexing of a document, rather than just leaving this up to the search engine. Certain words known as tags are explicitly associated with a document, and these words are given an importance-rating higher than any other word that appears in the document. This tagging helps the search engine provide more reliable and meaningful results to the user. Tagging is easy to set up in dbFramework and this is discussed in more detail in section 11.17.

In some ways having a hierarchical categories structure is a form of tagging because then certain words (the categories, or folder names) are associated with the document and automatically given a higher importance-rating than simply the words in the document.

However folders are not sufficient in themselves. Sometimes when folders are used to categorize the documents there may not be an opportunity to provide all the tags that could be used to describe that document more effectively, simply because doing so may make the category hierarchy unwieldy. So to complement the folder names, providing the ability to both place the document in a category folder and assign a certain number of tags to the document helps towards generating the most optimal search results, while also maintaining a disciplined categorization of the documents.

So it's generally recommend to adopt an approach that allows both folder names and tags to be assigned to documents, and using the search engine to complement the discovering of these documents using both the categories and tag names.

15.3. Configuring Search in dbFramework

Enabling search in your dbFramework application is a simple process: just go to the application record, find the 'Enable Search' entry box and change it to 'Yes'.

Once you republish the application, the search index will be built. This may take a certain amount of time depending on the size of the data in the application. If there are lots of file attachments the process can take a bit longer than if there are only data records.

All information in the application is indexed. Unlike traditional search engines, only the data itself is indexed and not the entire web page used to display that data. That has the benefit of keeping the index free of clutter and therefore improving the chance of finding the most relevant document.

After the search index is created, it will be maintained incrementally: that is, every time a record is changed, or a new document attached, the search index will be updated with just that change and not rebuilt from scratch. This can take a few seconds to a few minutes depending on the load on the server.

15.4. Using Search

Once the search is enabled, the search screen can be accessed by clicking the 'Search' link at the top-right of every page:



Here you can click in the white box and enter a search term. Pressing ENTER will run the search and display the search results.

Alternatively you can click the 'Search' text itself and be taken to a screen that provides the ability to either search for a specific term, or jump directly to a record based on an existing record number.

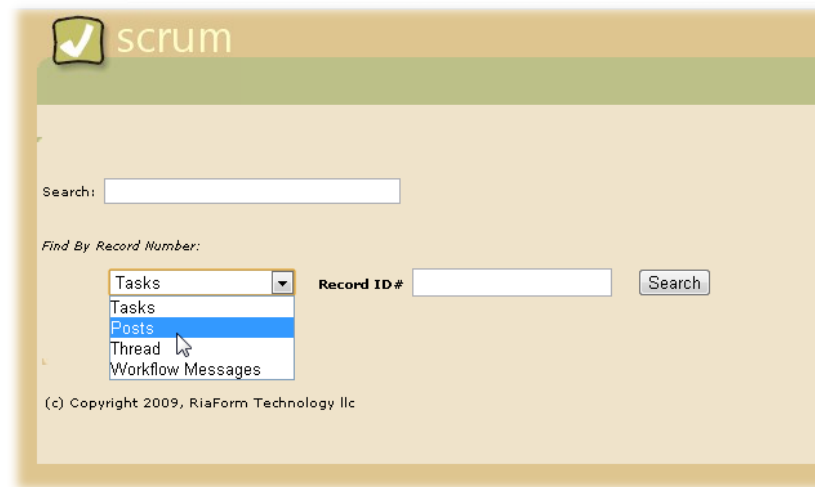


Figure 78: Advanced search page

In the advanced search page, you can select the area of the application and enter in a record number to jump directly to that record. This is a very efficient way to locate records if you have the record number noted down on the printed document, for example.

If instead you enter a search term, you will be presented with the search results page:

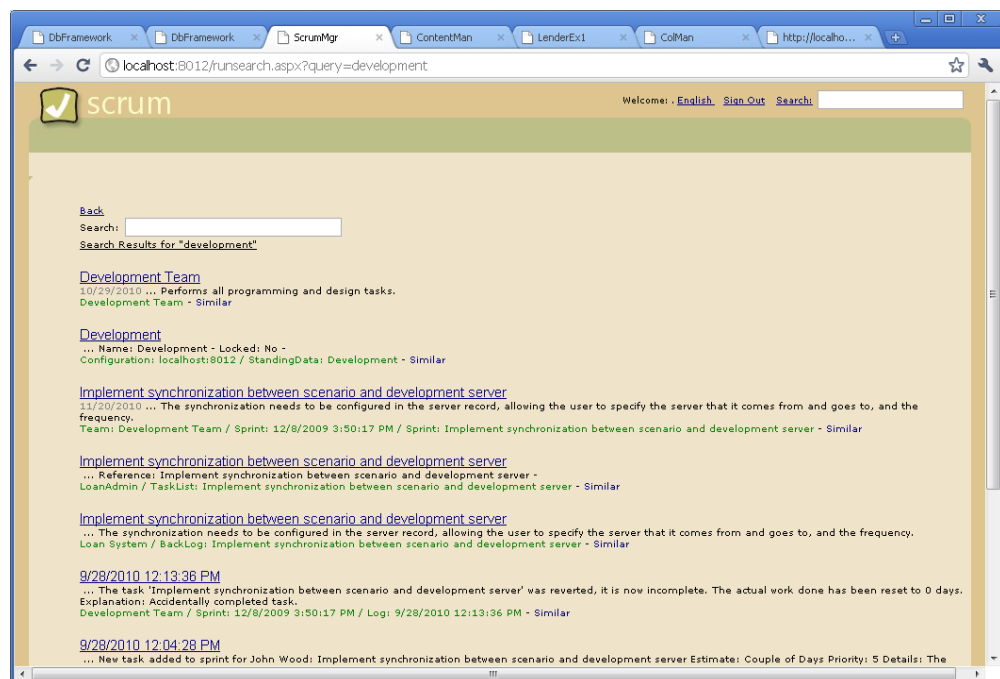


Figure 79: Showing search results

Each search result displays the title of the record, and underneath that the date the record was last modified, followed by an extract of the text within that record. Any text marked with the 'Preview' flag will appear here.

Below that is the path of the location to that record in the application.

You can also click a 'Similar' link beside that which will show records that have the same terms used in the title of that record.

15.5. Search Ranking

dbFramework automatically ranks the results based on three criterion:

1. Records that were modified more recently will appear closer to the top.
2. Records near the top level of the database will appear closer to the top.
3. Terms that appear in the record title will appear closer to the top.

15.6. Security

It's important to note that the user can only search within the records that they have access to view. Any records that are returned from the search that the user cannot access for security reasons will be omitted from the result.

There are two levels of security being applied here:

- On the one hand, a check is made to see whether any single result from the search is accessible to the user based on the roles with which they are configured. If they are not allowed to see that record then it is skipped and the next record in the result is reviewed.
- Secondly, any result that is clicked on will be checked again to see if the user has appropriate rights to view that record. This security is also applied when links to a record are sent to a different user to ensure that they have access to that record.

This is a vast improvement on the security you would get by using a third-party search engine. In those all records available in the database will be indexed because the crawler will view the application as a website without any concern for individual access restrictions. Because dbFramework understands the database and the way users have been granted access, it can be far more diligent about ensuring that the search results do not breach the access rights granted to them.

15.7. Installing IFilter Plugins

dbFramework tries to also index any file attachments that have been uploaded into the application's database. However sometimes it does not know how to read the file. For example many file formats are proprietary and cannot be understood without the need for proprietary tools.

When this happens, vendors typically provide something called an IFilter plugin. This tool is often downloadable for free, and will help the search engine extract text out of specific types of files. This text is then entered into the search index.

A typical example of this is the Adobe PDF file format: Adobe provide a free IFilter plugin that allows search engines to open and extract text out of any PDF file. There are also IFilter plugins for Microsoft Office applications and many other commercial software packages. The best option is to look on the Internet to find the IFilter plugins you will need.

So long as the IFilter plugin is properly registered with the system, dbFramework will attempt to find and load it. Please check the documentation that comes with your specific IFilter plugins to see how they should be registered and configured.

15.8. Query Syntax

The general syntax for entering queries is simple, and works very similarly to Google and other leading search engines: you can just enter a list of words that can appear in the records you are searching for.

But dbFramework provides a number of additional syntax options for you to narrow down your search results.

15.8.1. Terms

A query is broken up into terms and operators. There are two types of terms: Single Terms and Phrases.

A Single Term is a single word such as "test" or "hello".

A Phrase is a group of words surrounded by double quotes such as "hello dolly".

Multiple terms can be combined together with Boolean operators to form a more complex query (see below).

15.8.2. Wildcard Searches

A wildcard search allows the user to find words with parts that are not known, or that can be replaced. A question mark is used to indicate a single character that can match any character. An asterisk is used to indicate a part of a word that can match any number of characters.

For example, the search term:

```
P?per
```

...will match both "Paper" and "Piper".

The search term:

```
Can*
```

...will match words such as "can", "candle", "canoe", or "canister".

Note: You cannot use wildcards as the first letter of a search term.

15.8.3. Fuzzy Searches

A fuzzy search is similar to a wildcard search, but it finds all similar sounding words. For example, the term:

```
Roam~
```

...will find roams, foam, roaming and so on.

15.8.4. Proximity Searches

dbFramework also supports finding words that are within a specific distance from one another. To perform a proximity search use the tilde, "~", symbol at the end of a search phrase. For example to search for "apache" and "jakarta" within 10 words of each other in a document use the search:

```
"jakarta apache"~10
```

15.8.5. Range Searches

Range searches allow you to search for records within a specific range of values. This currently works with modified dates of records. To perform a search for records that have been modified between specific dates, you can enter a search such as:

```
Modified:[YYYYMMDD TO YYYYMMDD]
```

...where YYYY is the year, MM the month number and DD the day of the month.

15.8.6. Boosting a Term

While results are returned based on relevancy rules, such as whether a specific term appears multiple times in a document, you can also force certain terms to be treated with priority in your search. To do this you use the caret character in your search, followed by a boost index. The higher the boost factor, the higher priority that term has in the search.

For example:

```
Invoice^4 CSDN
```

Will search for both invoice and CSDN, but will treat Invoice as far more important than CSDN when looking for records.

15.8.7. Boolean Operators

Boolean operators allow terms to be combined through logic operators. dbFramework's search engine supports AND, "+", OR, NOT and "-" as Boolean operators (note: all Boolean operators must be ALL CAPS).

The OR operator is the default conjunction operator. This means that if there is no Boolean operator between two terms, the OR operator is used. The OR operator links two terms and finds a matching document if either of the terms exist in a document. This is equivalent to a union using sets.

To search for documents that contain either "jakarta apache" or just "jakarta" use the query:

```
"jakarta apache" jakarta
```

or

```
"jakarta apache" OR Jakarta
```

The AND operator matches documents where both terms exist anywhere in the text of a single document. This is equivalent to an intersection using sets. The symbol && can be used in place of the word AND.

To search for documents that contain "jakarta apache" and "Apache Lucene" use the query:

```
"jakarta apache" AND "Apache Lucene"
```

The "+" or required operator requires that the term after the "+" symbol exist somewhere in a the field of a single document.

To search for documents that must contain "jakarta" and may contain "lucene" use the query:

```
+jakarta lucene
```

The NOT operator excludes documents that contain the term after NOT. This is equivalent to a difference using sets. The symbol ! can be used in place of the word NOT.

To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" NOT "Apache Lucene"
```

Note: The NOT operator cannot be used with just one term. For example, the following search will return no results:

```
NOT "jakarta apache"
```

The "-" or prohibit operator excludes documents that contain the term after the "-" symbol.

To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" -"Apache Lucene"
```

15.8.8. Grouping

The dbFramework search engine supports using parentheses to group clauses to form sub queries. This can be very useful if you want to control the boolean logic for a query.

To search for either "jakarta" or "apache" and "website" use the query:

```
(jakarta OR apache) AND website
```

This eliminates any confusion and makes sure you that website must exist and either term jakarta or apache may exist.

15.8.9. Special Characters

dbFramework's search engine supports escaping special characters that are part of the query syntax. The current list special characters are

```
+ - && | ! ( ) { } [ ] ^ " ~ * ? : \
```

To escape these character use the \ before the character. For example to search for (1+1):2 use the query:

```
\ (1\+1\) \:2
```

16. Skinning

16.1. Overview

The term 'skinning' refers to changing the way the application appears and is interacted with without actually changing the functionality of the application.

Skinning an application is something that the user normally does in order to customize the look of the application in a way that differs to the standard look provided by the application vendor.

dbFramework allows both the dbFramework designer and any applications created with dbFramework to be skinned. These skins can be provided as selectable alternatives or replacements for the default look of the application.

Each skin is provided as an HTML template file. Any content that can be rendered by the target browser can be utilized in the HTML template to customize the way the entire page, or any portion thereof.

Note: This is different from the 'custom views' that we discussed earlier. The skin affects the entire site, whereas a custom view is just a replacement view for a specific record or field displayed *within* the page.

16.2. Platforms

Skins are most often used to provide pages that are optimized for platform-specific access. For example, dbFramework comes with an alternate skin for access by primitive web browsers on mobile phones such as the BlackBerry.



Figure 80: dbFramework application running on a simple cell phone.

But other alternative platforms can also be supported. For example an Adobe Flash interface can be provided, or perhaps a Microsoft Silverlight interface that is optimized for Windows Phones. Special support could also be provided for HTML5-based browsers, utilizing the enhanced display features of that platform such as transitions and 3D.

Because of the flexibility of the skinning feature, you can even use Skins to support new data access protocols, such as OData. The output is by no means limited to HTML and can actually return any text-based document.

To indicate to the application that you wish to view the content from a different platform, a special code is added to the URL:

`http://localhost:85/dataview.aspx?view=Applications&focus=/4d3a2fe3-406d-4a65-9601-fd09b8e92480&platform=mob`

By adding the '**platform**' URL value we can specify the name (code) of the platform to enable for this page view. This platform will be maintained throughout navigation of the application site.

By default the 'mob' platform comes with the application and provides access via an HTML interface that is optimized for small screens and basic web browsers such as those found on mobile phones.

But any additional platforms can easily be added, just by creating an HTML file, without any programming. This process will be discussed in this section.

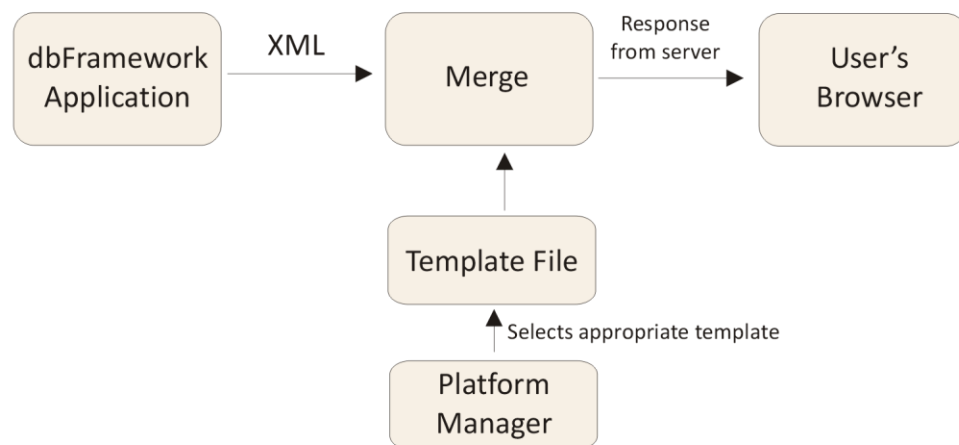
16.3. Understanding the Process

Each time a page is requested from a dbFramework application, it first generates data structures containing the raw information that is used on the screen. This contains the record being viewed, but also includes things like the list of actions and reports relevant to the record, the navigator content that shows where you are, the 'goto' list and so on. All of these are in raw data structures.

You can view these structures by appending '&xml=true' to the end of any URL for accessing dbFramework. This will return an XML representation of the raw data structures, telling dbFramework to bypass the remaining steps.

In the next step, the application loads the appropriate template file. This is an HTML text file that contains special markup that says how it utilizes that data to format it in a way that is appropriate to display on the screen.

Finally, it merges together the raw data structures and the template file, to create an HTML file that can be sent back to the user for displaying in their web-browser.



This three- step process happens automatically every time you request a page.

By default the main page template that is loaded is called 'dbf.htm', and this is held in the 'Files' subfolder of the application.

However, if a different platform is selected, then a different template file is loaded. The name of the template is based on the name of the platform and the type of template. If you have an alternative for the "dbf.htm" file, and the platform is called, for example, 'Silverlight', then the file will be called p_silverlight_dbf.htm.

Every platform file is prefixed with 'p_', followed by the name of the platform, and then an underscore and the name of the template file.

16.4. Adding a New Skin

To add a new skin to an application, go to the Application record and then drilldown to 'Skins'. Click 'Create New' to add a new skin.

The screenshot shows a web form titled "Skins". At the top, there are two input fields labeled "Name" and "Platform ID". Below these is a section titled "Template Files" which contains a list of template types: Master, Styles, Record, Data List, Dropdown, Error, Login, Search Input, Search Results, and Notes. Each type has a "Choose File" button and the text "No file chosen". At the bottom right of the form is a "Create" button.

Figure 81: Creating a new skin

The new skin form allows you to upload the template files for various aspects of your dbFramework application. The first two fields let you put in a full name (eg. 'Flash Interface', and then a platform ID which should be short and must not contain spaces. The platform ID is what you set the platform value to in the URL.

The template file fields are described below:

- **Master:** The master template holds the overall site design, including the logo at the top and the area for the actions and reports on the side. Look at the 'dbf.htm' file for an example.
- **Styles:** This is the CSS styles file that will be used by the templates. Look at the 'styles.css' file for an example.
- **Record:** This is the template for viewing or editing an individual record. Look at rectemplate.htm for an example. Note: The record template is only

used if you are using the default master template, as this contains an include tag to incorporate the record template.

- **Data List:** This is the template for viewing a list of records. Look at `datalist.inc` for an example. Note: The data list template is only used if you are using the default master template, as this contains an include tag to incorporate the data list template.
- **Dropdown:** This is the template that is used to display dropdown lists, such as when you are selecting a record from another part of the database. Look at `dropdown.htm` for an example.
- **Error:** This is the template that is used to display errors. Look at `error.htm` for an example.
- **Login:** this is the template that is used for logging into the system. Look at `login.htm` for an example.
- **Search Input:** This is the template that is used for entering search terms and locating records by their ID. Look at `searchinput.htm` for an example.
- **Search Results:** This is the template that is used to display search results. Look at `srchres.htm` for an example.

There is also a section for entering any notes related to this skin. These are not displayed to the user.

Any one of these files can be uploaded, but the rest can be left blank and the default will be used. This may be useful, for example, if you only want to replace the dropdown template but not the other items. You can also just replace the master page and leave all of the records, dropdowns and list templates as their default files. For consistency it may be better to replace all of the files however.

16.5. Creating a Template

The template merging process uses the same system as the custom view feature to merge data with the HTML.

For comprehensive details on how this works, please refer to section 10. A few things should be kept in mind that are specific to creating skin templates:

1. The overall file must be wrapped in a `$Root$` and `$end$` tag.
2. The data that is merged with the file depends on the specific template file being replaced.

There is also one additional command, 'include', that can be used to help componentize a web page into several files. This is used, for example, to split up the record and data list templates into separate files.

The format of this command is:

```
~#include:filename~
```

...where 'filename' is the name of the file to include. This will then import that file and insert it into that location in the template, processing it in exactly the same context.

16.6. The Data Structures

The following section describes the data structures that can be referenced in the templates for each type of template that you can provide.

The structure is described in a table, with line-indentation indicating a parent-child relationship. So, for example, the 'Navigator' entry has a child, 'ListTitle'. This means you would access this in your template as:

```
$Navigator$ Title: ~ListTitle~ $end:Navigator$
```

...which will repeat 'ListTitle' for each Navigator entry.

Only the basic structures are described, covering the essential items required to create a basic template. For information on other fields that may be available, append the 'xml=true' to the end of the URL of the screen you wish to re-template and analyze the XML response.

Note: The data structures described in this section are subject to change in future versions. This may not always be documented immediately. If you find that a template stops working after an upgrade, consider contacting RiaForm support for assistance. For safety always test your templates on a development system after upgrading before rolling the release out to a production server.

16.6.1. The Master Template

The Master template refers to the dataview page. It encompasses the record template and the data list template, which render the 'Records' list shown below but are held in separate files for manageability.

AppTitle	The title of the application
appName	The name of the application
appDescription	The description of the application
appCopyright	The copyright notice of the application
version	The current version of the application
ListTitle	The title of the data list in focus
Views	Information pertaining to the available views.
ViewList	
Selected	A Boolean that indicates if this view is selected.
Link	The URL to select this view.
Title	The title of the view.
Navigator	Information pertaining to the current focus within the database. Maintained as a list of path items.
ListTitle	The title of the list at this index of the path.
RowTitle	The title of the record at this index in the path.
LinkLink	The URL to the list at this index in the path
RowLink	The URL to the record at this index in the path.
Warning	Information pertaining to any active system-level warning messages.
WarningText	The text of the warning message.
InfoPop	Information pertaining to any active application-level messages.
Message	The application-level message that should be conveyed to the user.
DataListOrAutoExpand	Contains information on the actual data being viewed.
CreateURL	The URL to invoke the create-new action for this list.
Columns	
Title1 – Title7	The title of the first 7 columns in the list
ID1 – ID7	The internal ID of the first 7 columns in the list
Records	The list of records being viewed.

FormData	HTML that is specific to a form required for the current edit action.
Header	Metadata about the record
RowTitle	The title of the record
ShowThisLink	The URL to show detail for this record
EditLink	The URL to edit this record
DeleteLink	The URL to delete this record
InEdit	Indicates if the record is being edited
ChildViews	A list of drilldowns, or 'GoTo' items
Title	The title of the 'GoTo' item
Link	The link to expand the 'GoTo' item
HasPreview	Indicates if this record has some preview text.
PreviewText	The preview text for this record.
RowLines	A list of lines to display in this record, for a form-based layout.
Col1 – Col4	The title of the four columns in this row line.
Val1 – Val4	The value of the four columns in this row line.
AllValues	A dynamic list of values in this record
Expansion	Contains information about the link to drilldown (the '+' icon)
HasChildren	Whether this record has drilldown children
Link	The URL to the drilldown
Attrs1 – 9	The CSS attributes for the values
ShowDetailLink	The URL to this record
Stripe	Whether this record is striped
Title	The record title
Val1 – Val9	The values in this row
SectionFooter	Any additional HTML to display at the end of the record.
Actions	The list of actions pertaining to the current focus.
ActionList	
ActionTitle	The title of the action.
ActionURL	The URL to invoke the action.
Reports	Information pertaining to the reports available for this record
ContextReports	The reports specific to this record

ReportURL	The URL to invoke this report.
ReportTitle	The title of the report
GlobalReports	The reports available throughout the application.
ReportURL	The URL to invoke this report.
ReportTitle	The title of the report
HasPages	Indicates if this list has multiple pages.
HasFirstPageLink	Indicates if this list has a first page.
FirstPageLink	The link to the first page.
HasNextPageLink	Indicates if this list has a next page.
NextPageLink	The link to the next page.
HasPrevPageLink	Indicates if this list has a previous page.
PrevPageLink	The link to the previous page.
searchLink	The link to the full-text search page.
IsLongList	Indicates if the data being viewed is a tabular (long list), or form-based (non-long list).

16.6.2. The Dropdown Template

The dropdown template refers to the window that appears when the ellipses button is clicked besides a field that shows a dropdown to select foreign-link data.

UpLink	The URL that moves the focus to the parent
HasPages	Indicates if this list has multiple pages.
HasPrevPageLink	Indicates if this list has a previous page link
HasNextPageLink	Indicates if this list has a next page link
PrevPageLink	The URL link to the previous page.
NextPageLink	The URL link to the next page.
FirstPageLink	The URL link to the first page.
DataList	Information on the data in the dropdown
Records	The list of rows to display in the dropdown. From this point the structures are the same as the 'Records' section in the master template.

16.6.3. The Login Page Template

The login page refers to the screen that is displayed when users are required to provide login information.

LabelUserName	The text of the user name prompt.
LabelPassword	The text of the password prompt.
LabelInvalidLogin	Any login failure text that may be displayed after a failed login attempt.
bad	Indicates a failed login attempt.
LabelLogin	The text of the login button.
from	The URL of the page accessed prior to the login request.

The login page must contain a form that sends login information to the server. The information must be posted to a page called 'authenticate.aspx'. The following fields are required in the form:

- USERNAME: The login name.
- PASSWORD: The password, in plain text.
- from: The URL-encoded value of the URL of the page accessed prior to the login request.

If the login is successful, the response will redirect to the value of the 'from' parameter. If not then the login page will be returned with the error held in the LabelInvalidLogin value.

Note: For security purposes, the website should employ SSL security so that passwords are not sent in plain text.

16.6.4. The Error Page Template

The error page refers to the screen that is displayed when an internal error occurs.

LabelInternalErrorMessage	The error message title label.
LabelErrorMessage	The error message label.
LabelBackToContinue	The text explaining to go back to continue.
TITLE	The error message title.
LabelTechnicalInfo	The label for any technical

	information.
TECHINFO	The technical information.

16.7. Replacing Skins in dbFramework Itself

You can also add skins that are used in dbFramework. While you would not do this using web-interface itself, you can add the files it would normally create right into the dbFramework's Files folder. For example you can create a p_test_dbf.htm file into the Files folder, and then entering '&platform=test' to any dbFramework URL will then load that file as the master template.

Providing detailed instructions on accomplishing this is beyond the scope of this guide, but refer to the example files specified in the previous section to get an idea of which filenames to use.

Note: You cannot overwrite the default templates used in dbFramework. If you do this then they will be replaced on the next upgrade.

17. Localization

17.1. Overview

With an increasingly globalized economy, it's no longer possible to guarantee that any one software application will be needed in purely one language. A big part of usability and user-reach is to have that application display screens, reports, navigation links and error messages in the language with which the user is most familiar.

Luckily dbFramework provides extensive support for providing an application in multiple languages, known as localizing the application.

dbFramework's localization features are innovative and extremely quick and simple to configure and use.

Unlike other applications, dbFramework applications allow you to switch language on the fly from any screen, without having to restart the application.



Figure 82: Language selection in an application

Once the languages are entered into the system, links are automatically displayed at the top of the application for switching the language. These links can be clicked on from any screen and the entire screen switches language on the fly.

Note that not only the terms change, but also any cultural, locale-specific conventions are also adopted. For example, number formats may change and dates will be displayed in a format that is normal in that culture.

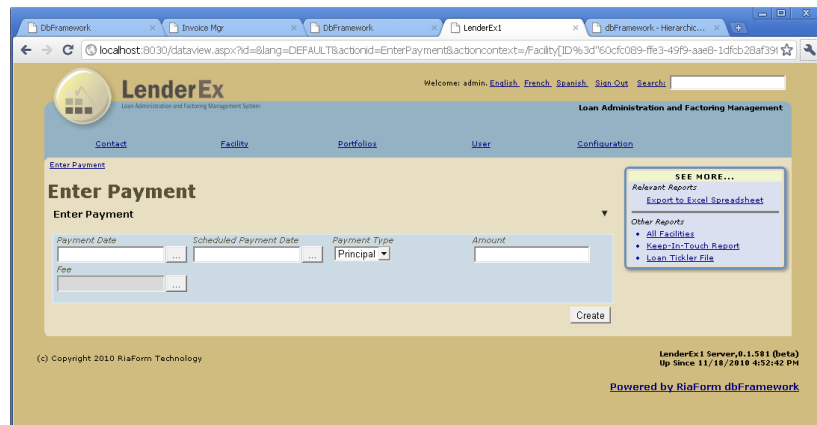


Figure 83: Original, English-language screen

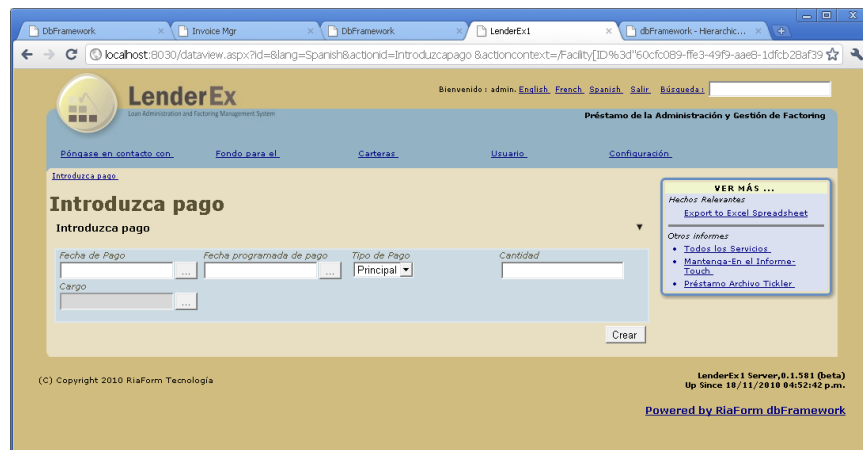


Figure 84: Same screen after clicking the Spanish language option

17.2. Adding a New Language

To add a new language to an application, the first step is to create the dictionary file. The dictionary file consists of a list of all the terms used in an application, along with a suitable translation for the specific language you are targeting.

There are two ways to create the language dictionary file:

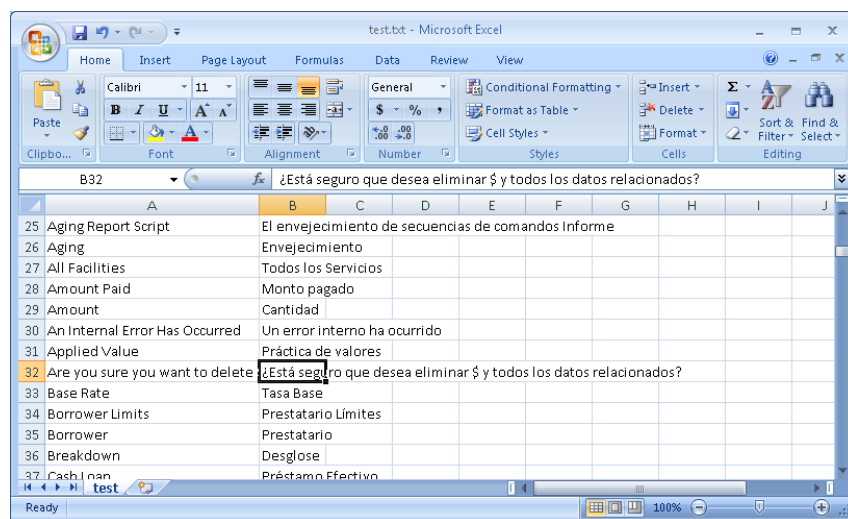
- Create a new empty dictionary file.
- Create a language update file.

The new, empty dictionary file is what you will be doing when you want to add a new language to your application. If you want to update an existing dictionary file with any new terms that were added to the application, then you will create the language update file.

To create a new, empty dictionary file simply click on the 'Language Dictionary' link under the 'See More' reports list. This will bring up a file that you can save. The first column will contain the English language (or entered language) term, and the second column will be used to enter in the appropriate translation.

To update an existing language file, go to the specific language entry under the Application's Languages drilldown, and then click on the 'Language File Update' link in the 'See More' reports list. This will create a similar file to the one above, but the existing translations will be filled-in already. Any new terms will have a blank entry for the translation.

You should then load the file into Microsoft Excel or another spreadsheet to make it easier to edit.



Once you have entered in all of the new terms, save the file as a text file (tab-separated).

This file can also be sent to anyone familiar with the specific language and application terms. That user doesn't need to have or use dbFramework and just edit the file using a common spreadsheet application.

Next go to the Application record, and then go to the Languages drilldown. Click 'Create New' to create a new language entry.

The screenshot shows a web form titled "Languages". It has a light yellow header bar with the title and a small downward arrow. Below the header, there are three input fields. The first is labeled "Culture ID" and contains the text "es-MX". The second is labeled "Title" and contains the text "Spanish". The third is labeled "Translation File" and has a "Choose File" button next to it, with the text "No file chosen" to its right. At the bottom right of the form is a "Create" button.

The following fields are required:

Culture ID

The culture ID is a special ID that is used to designate to the system which language and locale to which you are referring. This code is based on RFC 4646 and is a combination of an ISO 639 two-letter lowercase culture code associated with a language and an ISO 3166 two-letter uppercase subculture code associated with a country or region.

This code is used by the system to determine how to display numbers and dates. If this isn't important, you can enter en-US for American English formats.

A list of common culture IDs can be found in section 24.1.

Title

The title is the name of the language. This language will appear at the top right of the screen. It's customary to enter the language title in the language itself, e.g. Espanol rather than Spanish.

Translation File

This is where you upload the actual dictionary file that you edited. Click the button to choose the file, which must be in a tab-separated text file format which can easily be exported from Microsoft Excel.

Once the file is uploaded, the language will be available the next time the application is published.

18. Collaborative Design

18.1. Overview

One of the huge advantages of using dbFramework to develop applications is that it's a collaborative development environment, in which the process of development can be shared between any number of developers in parallel. This is a vast improvement on the fragmented development that usually occurs with application programming.

But beyond developing, dbFramework also provides a number of features for designing an application in collaboration with others. This design process occurs before the application creation is started. It helps to get everyone on the same page and understand the requirements for the application. The design process usually involves the users and management along with those developing the application.

There are two areas in dbFramework designated for this purpose: User Stories and the Glossary.

18.2. User Stories

The concept of 'user stories' is a common paradigm in agile development.

As described in Wikipedia:

A user story is one or more sentences in the everyday or business language of the user that captures what the user wants to achieve. User stories are used with Agile software development methodologies for the basis of what features that can be implemented. Each user story is limited, so it fits on a small paper note card—usually a 3×5 inches card—to ensure that it does not grow too large. The user stories should be written by the customers for a software project and are their main instrument to influence the development of the software.

User stories are a quick way of handling customer requirements without having to elaborate vast formalized requirement documents and without performing overloaded administrative tasks related to maintaining them. The intention of the user story is to be able to respond faster and with less overhead to rapidly changing real-world requirements.

dbFramework provides an area under the Application record for entering user-stories, and even allows these to be organized hierarchically, and for users to discuss the user-story that is on the screen with integrated real-time chat.

To create a user-story, go to the Application record, and drill down to 'User Stories'. Then click 'Create New'.

You have the option of either entering a user story in here in its entirety, or you can create additional breakdown records underneath the user story to break it down into parts.

An example of a user story might be something like:

Title: "New Search Feature"

Details: "The user has a link in the corner labeled 'Search' and a text box. They click in the text box and enter in the search term. When they click enter it runs the search across the whole site and displays the results in a new page."

Once the user story is entered, click on the 'Details' link of the user story to bring that user story into focus.



You'll then notice that, at the bottom of the screen, there is an area for comments. You can enter comments into this and any other users who are viewing the same user story will see those comments appear in real-time, much like how Twitter works.

This makes for a great way to meet about and collaboratively refine user-stories as a critical part of the design process.

18.3. Glossary

In any one industry there are often several terms that have the same meaning. In the commercial finance industry, for example, what is a facility to one person is called a credit line to another.

Although difficult to achieve, using consistent terminology is critical to the success of an application. When terminology changes from one screen to another, simply because different people developed those screens, users get confused, usability suffers and training expenses increase.

To make it easier to manage and agree on the terminology used in an application, dbFramework provides a glossary database that can be edited collaboratively.

You can start editing the glossary by going to the Application record and then drilling down to 'Glossary' under that. Click 'Create New' or use the Quick Entry bar at the top to enter your term and definition.

Once you are done you can view the glossary either by running the Glossary report (under 'See More'), or by clicking the Glossary view in an application.

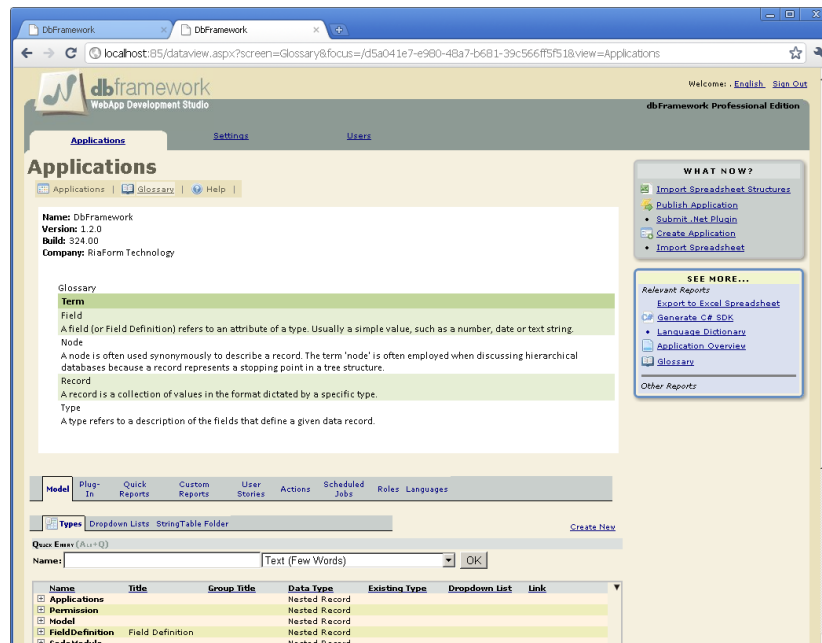


Figure 85: View showing the application glossary

19. Automated Testing

19.1. Overview

Once an application is created and in use, the goal should be to maintain and improve that product, and that will undoubtedly involve putting new versions of the software to replace the current version.

But there is a risk every time software is updated that a change will inadvertently break some existing functionality. Any useful system will get to a degree of complexity where this is unavoidable.

To mitigate the risk of this occurring, developers usually employ automated testing tools that facilitate the process of ensuring that new versions of software maintain existing functionality.

While most development systems require third party testing tools, dbFramework comes integrated with a set of testing tools that allow you to automate a great deal of testing in your application, providing a significant tool for keeping your solutions stable and effective as they're developed.

19.2. Concepts

19.2.1. Basics

The purpose of testing is to verify that the system functions as you would expect. Rather than telling dbFramework what you would expect, it presumes that the application is functioning the way you want right after it was developed. So in some ways it expects you to manually test the functionality to ensure it's working before you start setting up your tests.

Setting up the tests simply involves *recording* interactions and behavior of your application. It does this behind the scenes – all you have to do is use the application while it's recording, and make sure you *use* the bits of your application you want to include in the automated testing.

Once you're done recording, the information the system collected will be stored in something called a 'Test Group'. You can then play that back at any time, or even have it scheduled to be played back on a certain frequency.

19.2.2. Things to Keep In Mind

It's almost as simple as that. But there are a few caveats we'll discuss in this section.

Firstly, because of the way session management works in Microsoft IIS, it's recommended that you use the OfficeSuite Launcher application's built-in web server to host your testing system. This system keeps sessions alive for longer, and it's important that the session is kept alive because it stores some critical information relating to the currently recording or running test.

It's also critical that only one person use the test system at a time while tests are being recorded or played back. The automated testing environment is not a multi-user system as tests can interfere with one another. The testing system is for testing business logic and expected functionality, not for testing stress or performance in multi-user conditions.

The next point worth making is that there are some variables that occur when interacting with the application, that change each time you run them. These variables are difficult to verify in the future because they'll certainly be different, and so what does it compare it against?

One example of this is the internal record IDs that are created. These are called GUIDs, they're long codes and they change each time they're generated. So if you create a record in your test, then when you play that test back, the record ID will be different so it cannot know if it worked correctly because it doesn't know what to compare it with.

To work around this dbFramework tries to match up the records based on the order in which the events occur. Because events generally trigger in the same order each time, this is a pretty reliable way to identify records even if the internal IDs of the records have changed.

This doesn't work, however, if your application uses asynchronous events (see section 8.4), because these often happen in parallel with other events, and the event that gets to go first cannot easily be determined or reproduced reliably.

One way to work around this is to use a pause to wait for asynchronous or idle events to complete. The testing system provides an action for this that you can use if you know that asynchronous or idle events will occur.

The other problematic area is dates. If records have values that default to the current date, then each time the test runs this will be different. That will cause the

test to fail, because the values of the fields are then different, and other calculations that are based on that value will also differ.

To work around the date problem, there are two features provided:

1. The testing system provides a function to actually change the system date on the server. If default dates will be a problem then you should change the date at the beginning of the test so that each time it runs all default dates will be roughly equal.
2. When dates are compared from the test case to the actual result, the seconds are ignored. That's because it will be very difficult, if not impossible, to have the seconds match up each time it runs – simply because the test runs at a different speed than the user originally recording the test.

Lastly, in order to simplify the requirements of the configuration of the database prior to running the test, each time you record a test dbFramework will take a snapshot of your entire database. When the test is re-run, that database snapshot will be re-instated. That means all existing data in the database is deleted and replaced with the database as it existed when the test was recorded.

This makes the test more reliable, but it does mean that all the data in the database will be lost each time the test runs. That means you need to run the tests in a *different* database than the one you normally use. We'll discuss setting this up in the next section.

There are also some limitations on this snapshot and re-instating of the database: the entire database is stored in the test group record, and this is limited in size. Generally it's a good idea to only have a handful of records in the database when you record the test, otherwise you may see an 'Out of Memory' error, which means it ran out of room to store the database in the test record.

Note: If you do notice this error when recording a test, consider restarting the server. Because of problems with fragmented memory, restarting the server application (OfficeSuite Launcher) can reorganize memory and provide more contiguous memory slots to store the database snapshot.

It's also worth mentioning that, while the test is running, all scheduled jobs are disabled. That's to make sure that actions occurring in the scheduled jobs don't interfere with the tests. If you specifically want to test code that is in a scheduled job, you should move that into an action that is visible only in the test server, then run that action when recording the test.

With these few caveats in mind, the dbFramework testing system still provides a great deal of useful assistance in maintaining the functioning and quality of your system.

19.3. Configuring the Testing System

The testing system is integrated into the resultant dbFramework application, after it is published. However it's not available in every application – only applications that have a Server Type of "Testing (Enhanced)". This means the system is enhanced to provide testing capability.

To enable this, go to your Application record in dbFramework. Then go to the Servers drilldown. Create a new Server entry for your testing system. You can also copy and paste your existing Server entry, but make sure:

1. You give the server entry a suitable name, which should include 'Testing' in the title.
2. You select 'Testing (Enhanced)' in the Type dropdown.
3. You change the name of the database. Change the 'Prefix Database Tables' to 'No' and enter a new name into 'Database Name', which should generally include the term 'Testing' for differentiation.
4. Update the 'URL of App' to have a new port number. The existing port number comes after the host, so whereas it may say <http://localhost:8030> you may want to change that to be <http://localhost:8031> so that it doesn't interfere with the existing web server instance.
5. Update the 'Path of App Files' to point to a new folder, but still under the dbFramework's application folder, in the Apps subfolder. The name of the folder should generally match the name of the application, with the addition of 'Testing' for differentiation.
6. Drill down to 'Advanced' and set the 'Override Title' and 'Override Description' to include the term 'Testing', again for differentiation.'

Figure 86: Setting up a new server entry

Also, remember to include any 'Feature Modules' you have selected in your regular server target, to ensure the same functionality is enabled in both systems.

Once your server is setup, you'll be ready to publish.

To do this, click Publish and select the name of your new server entry from the 'Target' dropdown list:

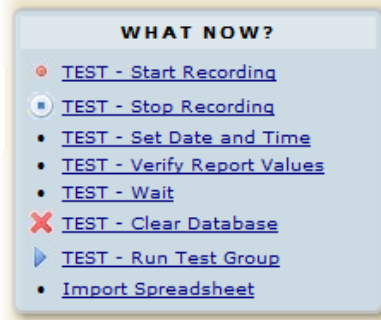
This will create the application files and start up the web server pointing to your new application.

Initially the application's data will be empty. **If it is NOT empty then something is wrong and you must be pointing at an existing database!** If so, go back to the bit about editing the server entry and make sure you've set up the database correctly.

Note: Make especially sure you don't run the **"TEST – Clear Database"** action unless you're absolutely sure you're using the test application's database!

19.4. Testing Features

Once the server is running, you'll notice a new tab at the top of the screen called 'Tests', along with a number of new actions in the 'What Now' list.



All of the features related to testing are prefixed with "TEST" so you can differentiate them from the rest of the regular actions in your application. Each of these actions is explained in more detail below:

TEST – Start Recording

Starts recording your actions and the system's response, ready to save them as a new test. Also takes a snapshot of the database and stores it in memory.

TEST – Stop Recording

Stops recording the previously recorded actions and responses and asks for a name for the tests, which it then stores in the Tests tab. Also stores the database snapshot that was taken when the test recording begun.

TEST – Set Date and Time

Changes the system date and time on the server to that specified. This is used to have each test replay use the same default dates in fields so that values and general behavior can be compared more reliably.

Note: Make sure you reset the date to the correct date and time when you are done!

TEST - Verify Report Values

Checks for the appearance of the specified values in the latest run report, in the order specified, in the output from the report. Also lets you indicate the maximum amount of seconds it should take to run the report. If the values are not found, if

there was an error running the report, or if it takes longer than anticipated, the test will fail.

TEST – Wait

Has the system pause for the given number of seconds. This is useful to wait for any idle or other background events to complete before continuing.

TEST – Clear Database

Deletes everything in the application’s database except for the users and the tests.

Use with utmost caution! This should be run at the beginning of each test so that the database is in a known, consistent state.

TEST – Run Test Group

Runs a previously recorded test, and monitors the response and behavior of each action, verifying it against the recorded response and behavior to determine if the test succeeded or failed.

In addition to these actions, there is a new ‘Test’ tab at the top of the application. This is where all of the tests and logs of the test-runs are recorded.

19.5. Recording Tests

To begin recording a test, you should first clear out the database so that it is empty and then add in the records that you need to run that test. Because of space limitations (the database is stored in the test itself), please limit the number of records you create for the test.

To clear out the database click on the “**TEST – Clear Database**” action. You will have to accept a couple of warnings so you don’t click this type of thing accidentally.

Once the database is cleared out you can begin adding your records. You do this BEFORE recording. When you hit the action to start recording dbFramework will take a snapshot of all of the records in the database that you added.

Before you start recording, first plan what it is you want to test.

- Consider what data you will need for your test, and think about the values you will use.
- Consider any operation that raises events or has calculated fields that will need testing.

- Think of reports that you can run, and what kind of data you will expect to see in the report.
- Also consider cases of deleting records and re-running reports to see how things change.
- If possible don't perform any actions that require email interaction, such as approval workflow or document review. If human interaction is required the test won't be able to be run automatically.
- It shouldn't matter if notification emails are sent, however. Make sure people are aware that the notification is just a test (you can usually do this by changing the system's application title (in the target server entry's advanced / override title option), as this is incorporated into the email from line).

Planning these things ahead of time makes the testing easier to perform.

When you are ready, click '**TEST – Start Recording**'. You don't have to enter a name at this point.

Note: It doesn't matter how long you take to run the test, when the test runs it executes everything as fast as it can, regardless of how long it originally took to be entered.

You'll then see an indication that your actions are being recorded through a caption at the top of the screen:

Recording: You are currently recording a test case. Click Stop Recording to stop and save the current test case.

This indicator will remain on the screen for as long as the recording is active. When you finish recording it will hide.

At this point you're free to do whatever you please in the application. You can insert records, change records, delete records. Also don't forget to perform actions that you *know* will fail as this is part of the test: If it doesn't display an error when the test is re-run then the test will fail.

Note, though, that only database changes will be recorded. Other operations, like viewing lists or running searches, won't be recorded or verified in the test case.

When you're done recording, click "TEST – Stop Recording".

This is where you enter the title of your test. For example if you were testing the totaling of invoices, then enter 'Invoice Totaling Test' in the title. You can also create test groups in a hierarchy, using the 'Test' tab, and select the group in which the test will reside in the 'Group to Store' dropdown.

Once you submit that, the red indication at the top of the screen will disappear and you will be able to continue using the application without any further recording. The test will now be available to run.

19.6. Playing Back Tests

You can play back a test at any time. But keep in mind that running a test will delete all the data currently in your database and reinstate the database to the state it was in when the test was recorded.

To play back a test, click "TEST – Run Test Group".

This is where you select the test to run. You can also enter some notes that will be recorded in the test case's log, for future reference.

Once you submit that form, the test will run immediately and a few seconds later you will see the outcome.

If the test succeeds you will see a notice like this:



Successful: The test group 'A Simple Test' ran successfully without error.

You can then go to the 'Test Groups' tab and see the Log entry underneath the test you created, which records when the test was run and the outcome for each individual step.

If, however, the test failed, you'll see something like this:



FAILED: The test group 'Test 4' failed.

This means that when the same action was executed, the response or behavior of the application was different than what was recorded. When tests fail we have to figure out what precisely went wrong, which is what we'll focus on in the next section.

19.7. When Tests Fail

You can review the latest test run for a test case at any time. Just go to the 'Test Groups' tab, find the test group that you ran, and click the report called 'Test Run Details', or click the custom view link above the record:

TEST RUN DETAILS 12/2/2010

Title: Test 5

Date Test Was Run: 12/2/2010 5:18:16 PM

Time	Action	Successful	Result Text
12/2/2010 5:18:16 PM	Transaction	Yes	
Action:			
Action: Insert			
Path: Portfolios			
Name = plop			
Events:			
12/2/2010 5:18:16 PM	Transaction	Yes	
Action:			
Action: Insert			
Path: Portfolios			
Name = test			

This provides a list of all the actions that occurred, along with whereabouts in the database they occurred, and a list of the events that occurred as a consequence. It also displays whether the test was a success, and any explanation (Result Text) for tests that failed.

Using this information you should be able to determine what went wrong with the test if one failed. For example it may say that an insert event was expected but didn't occur. Perhaps this is because you previously had an event handler that caused a different insert to occur, but that is no longer functioning.

Keep in mind that a test might also fail because the test relied on data that comes in to the system externally and so cannot be reproduced.

19.8. Testing Reports

19.8.1. Overview

The dbFramework testing system has special support for testing the output of Quick Reports. It tests three things:

1. That the report runs at all. If the report doesn't exist anymore, or if it produces an error, then the test will fail.
2. That the report runs within a certain amount of time. If the report takes too long to run it will cause the test to fail.
3. That the report contains certain, key values.

There are a couple of things to keep in mind, though:

1. The report testing does not verify the layout of the report, however.
2. You cannot currently test Word-based, Custom Reports.

Note: It's important that the report you choose to test does not make any changes to the database. Generally it shouldn't unless you have some custom code that does this. If the report generation changes the database the test will fail.

19.8.2. Running a Report Test

To test a report, while recording your test go to the location where you wish to run the report. Then simply run the report.

Once the report has run, click the special testing action titled "TEST – Verify Report Values". This brings up a form like this:

Recording: You are currently recording a test case. Click Stop Recording to stop and save the current test case.

TEST - Verify Report Values

TEST - Verify Report Values ▼

Maximum Seconds To Run	Test Value 1	Test Value 2	Test Value 3
Test Value 4	Test Value 5	Test Value 6	

Create

Note: Usually it doesn't matter how long a period between running the report and running this action.

The first field is the maximum seconds that the report can take before an error should be raised. For simple reports that do not show much data, you should put something reasonable such as 30 seconds. Please keep in mind that some factors outside of dbFramework's control make the time it takes to run reports quite variable.

The values for Test Value (1 to 6) are where you enter in values you think should appear in the report. You should enter these in based on the order in which they should appear. You do not have to enter all of the values. If you need more values then run this action two or more times for each group of values.

If the value cannot be found then an error will be raised. When you're recording the test make sure that you enter values you know will definitely be found. When the test is replayed then if these values are not present in the report the test will fail.

19.9. Having Tests Run Automatically

You can also configured your dbFramework application to run the tests every day at a certain time. Then you can have the test result report detailing whether the test was a success or failure generated and sent to your inbox each morning. This provides a great way to get assurance that new defects are not being introduced into the system, and if they are they can be fixed far more easily than if they were introduced a week or more before.

To set up a test to run automatically, find the test you wish to run automatically in the 'Test Groups' tab. Then click Edit:

The screenshot shows a web form for configuring a test. At the top, there is a 'Title' field with the text 'Test Wiki capability'. Below this is a 'Scheduling' section. It contains a 'Run Automatically' dropdown menu set to 'Yes' and a 'Time Of Day' text field containing '8:00am'. There is also a large 'Notes' text area which is currently empty. At the bottom right of the form is an 'Update' button.

Now change the 'Run Automatically' to 'Yes', and enter the time of day you wish the test to run in the 'Time of Day' field. Then click Update.

This tells dbFramework to make sure that test runs once a day, after the specified time, in this example 8am.

Keep in mind that you should not have tests run at the same time. Because each test reinstates the database to a specific set of records, and the behavior of the application that is being verified relies upon not having tests interfere with one another, running more than one test concurrently is not possible.

You can also have the application send you the test result in an email. To do this go to the 'Report Subscriptions' drilldown from the test, and click Create New:

The screenshot shows a form titled 'Report Subscriptions'. It has several fields: 'User' (a text field with a dropdown arrow), 'Report Type' (a dropdown menu set to 'Test Run Details'), 'Language Code' (a text field containing 'en-US'), and 'Frequency' (a dropdown menu set to 'Daily'). There is also a 'Time Of Day' text field which is empty. At the bottom right is a 'Create' button.

Here you should select 'Test Run Details' as the report type, and select the frequency you wish to see the test result, along with the time of day. Make sure the time of day is a reasonable duration later than when the test was automatically run.

19.10. Testing Performance

The dbFramework testing is to verify functionality of the application. It doesn't test the performance or stability of the application under stress. To check this kind of thing takes a different approach to testing that can be quite complex to set up.

However, luckily because dbFramework is web based there are plenty of third-party applications that can test a stressful load on a web server, simulating tens, hundreds or thousands of simultaneous users.

A few examples of commercial products that can help test performance of your system under stress, none of which are tested or endorsed by RiaForm Technology, are listed below:

<http://www.webperformanceinc.com/>

<http://www.proxy-sniffer.com/>

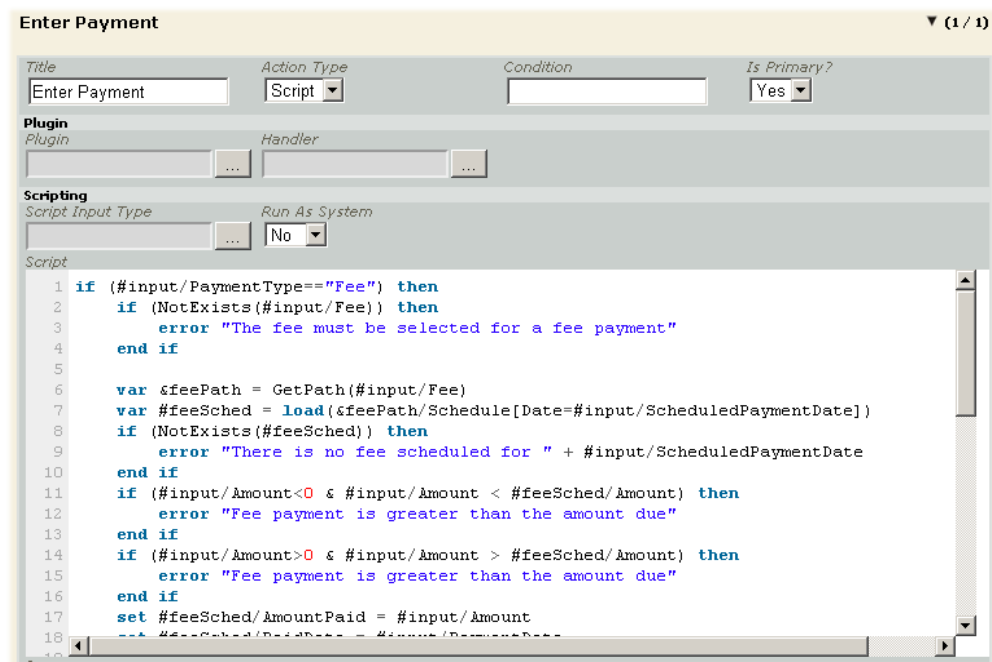
<http://www.nrgglobal.com/load-testing-tools/quality-assurance-and-load-testing-apploder.html>

<http://www.neotys.com/>

20. DbfScript Reference

20.1. Introduction

dbFramework provides a built-in scripting language that enables a wide variety of functionality to be implemented right inside the web-browser without having to use any external, complex programming environments.



DbfScript was designed to be simple to use. The syntax of the language utilizes the easy readability of popular languages such as Visual Basic with the freedom provided by the web-based JavaScript language. The result is a comprehensive and flexible language that should be sufficient for building business logic for quite complex systems.

dbFramework provides a built-in code editor which features syntax coloring and automatic indentation, features usually found in desktop integrated development environments, right inside your browser window.

20.2. Terminology

The following terminology is useful to remember when reading the reference guide and using DbfScript:

Node: A node is the same as a record in the database.

Path: The path refers to a string that represents the location in the database.

File object: this is the value of a field that is set to 'File' data type that contains file data. This data is stored in something called a File Object that can only be accessed using the functions defined in the File Functions section.

Sequence: This is a variable that contains a list of records.

20.3. Variables

var #variable = value

Declares a new variable. All record or value variables must start with a pound (#) symbol. All path variables must start with an ampersand (&). Variables can be declared twice, but cannot be used until declared. An initial value for the variable should be defined.

set #variable = value

Sets a new value to an existing variable.

#variable/path/path

If #variable is a record, lets you access child records using path notation.

If a part of the path refers to a field containing a foreign link (see section 6.14), the referenced record is automatically retrieved from the database, and subsequent fields will access values in that record.

&root/path/path

Allows access to any path in the database, from the root path.

&path/path[field=value]

A predicate on the path is surrounded by square brackets. The predicate specifies a filter in the path, which may be used when loading data from the database.

20.4. Program Control

if (condition) then

...[statements]...

else

...[statements2]...

end if

Runs [statements] if the condition is true. If 'else' is specified, runs [statements2] if condition is false.

while (condition)

...[statements]...

end while

Repeats [statements] while (condition) is true.

loop through (list) using #variable

...[statements]

end loop

For each element of (list), repeats [statements], setting #variable to the current element.

exit script

Exits the script without executing any further statements.

run task name at path [with node]

Runs the action called 'name' at 'path' in the database, optionally passing in 'node' that will contain the input parameters.

more time

Usually scripts must run within 120 seconds. This statement extends the amount of time allowed for this script to run by another 120 seconds. This configurable limitation is added as a protection against defective code such as infinite loops.

20.5. Errors and Notification

log "message"

Outputs a message to the system log. This message is not presented to the user but is intended to convey diagnostic information for the developer or administrator.

error "message"

Raises an error, displaying the specified message. Execution of the script aborts.

message “title”, “message”

Sets the message that will be displayed to the user when the operation completes.

Note: The message is only displayed if the script is running synchronously from an event or action. Messages won't be displayed from scheduled jobs, asynchronous actions or events or throttled events. Instead, 'log' should be used to place the message in the system log.

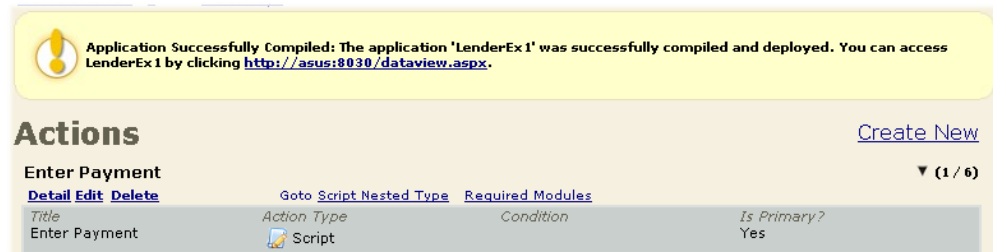


Figure 87: Example of an application 'message'

notify of [node] notification-name “message” at &path

Sends a configured notification (added through the Notifications drilldown from a field definition). The specified node, in square-brackets, is included in the email. Optionally a message can be specified, in parentheses after the notification name. The notification is sent via an email to all users who subscribe to that notification.

warning “message”

Stops execution and displays a warning message to the user. The user then has the option of accepting the warning and retrying the operation. If the warning was accepted by the user, the statement has no effect and execution continues.

Note: Warning messages have no effect if the warning is raised in an event that is executed asynchronously. However, warnings raised in synchronous events will be displayed to the user and will have to be accepted for the operation to execute.

Also: Warnings should be provided if it relates to a threat to something affecting the *user* of the system. If the warning relates to an action that might affect something more important, you should consider configuring conditional approval workflow instead. See section 11.2. Rather than asking the user, this would ask a supervisor for authorization first.

20.6. Reports

RunReport(path, name)

RunReportWithInput(path, name, node)

Runs the report called 'name' at 'path' in the database. Returns a file object containing the report output (see File Namespace). If using RunReportWithInput, pass in the parameter input to the report.

20.7. Database Access

new record for &path

Returns a new, empty record instance that is ready to be inserted into the specified path in the database.

record (field = value, field = value, ...) for &path

Constructs a new record that is ready to be inserted into the specified path in the database. The field and value pairs indicate the field values to set in the record. A value can also be a 'record' statement to construct a hierarchical record structure. When a record is nested within a value, it is not necessary to specify the 'for' suffix.

new list for &path

Returns a new list instance that is ready to be inserted into the specified path.

load (&path)

Loads the specified path in the database, and returns a list containing that data.

load (&path with [fieldlist]; [sortlist]; [top]; [files]; [range]; [links])

A load statement with more options, as defined below:

fieldlist: fields (field1, field2 [count | total | average | file])

Defines the list of fields to retrieve. An optional aggregate function is specified after the field name.

sortlist: sort (field1 ascending, field2 descending)

Defines the sort criterion to use when retrieving data. Multiple fields can be specified, and each field can be sorted ascending or descending.

top: top 4

Retrieves the top 4 (or any number) records from the database, based on either the default or specified sort criteria.

files: *files*

Retrieves any file data in the record.

range: *range 1 to 10*

Retrieves only the records of the range 1 to 10 (or any specified range).

links: links from &foreignLinkField

Retrieves back links. Retrieves records throughout the database that point to the specified load path. Retrieves only those records that have the field specified in the path 'foreignLinkField' pointing to this record. See also the 'PointersTo' function.

Note: Only the path of the nodes is retrieved. You can access this using the PathFromNode function. You must then load the actual data using another loan statement.

PointersTo(recordPath, type, foreignField)

Returns a list of paths that link to the record specified by recordPath. Returns only those paths pointing to the record that match 'type' (specified as a path), and are selected in the field specified in 'foreignField'.

For example, if I have a task with an 'AssignedTo' field that links to a user, then I might use code on the user record such as:

```
loop through PointersTo(&path, &root/Task, "AssignedTo") as &task
  log DescribePath(&task)
end loop
```

...to loop through all of the tasks that are assigned to this particular user.

This function can be very useful when propagating updates between linked records, or for reporting purposes.

save [all] #variable

Saves any changes made to the variable, which must be previously loaded from the database. Optionally saves all of the data in the record again (if 'all' is specified), otherwise just saves the changes that were made.

insert #variable at &path

Inserts the specified record or list (held in #variable) into the database at the location specified by path.

insert #variable into #list

Inserts the specified record or list into the existing list specified in #list.

delete at &path

Deletes the record(s) from the database at the path specified.

Note: All child records are also deleted, there is no confirmation, and no way to undo the delete. Use with caution!

remove #variable from #list

Removes the specified record (in variable) from the specified list, that is stored in memory.

Item(sequence, index)

Returns the Nth (index) record in the given sequence list.

sync path1 with path2 linking field1 and field2 (new as #new, deleted as #deleted, updated as #updated)

Synchronizes two parts of the database, as specified in path1 and path2, using the values in field1 of path1 and field2 of path2 to link the records.

Any records that are missing from path2 will be placed into a list called #new. Any records that are missing from path1 will be placed into a list called #deleted, and any records that are different will be placed into a list called #updated.

The synchronization operation performs no changes to the database. Rather, the records in the lists should be inserted, saved or deleted to apply the synchronization changes.

The modified dates in the records are utilized to determine whether a record has been updated.

Top(sequence, count)

For a sequence object, returns only the top 'count' number of records in that list.

Sort(sequence, field, ascending)

Sorts the given sequence object by the specified field. If ascending parameter is 'false' then the list will be sorted by descending values.

NewList()

Returns a new, empty sequence.

NewRecord()

Returns a new, empty record.

20.8. General Functions

call function()

When a function is invoked that does not return any value, it should be prefixed with 'call'.

return #variable

When returning values from functions (or code that is Eval'd), this indicates the value that will be returned to the calling function.

Eval (script, record, path)

Runs the provided text as a script, with the specified record becoming the #input variable, and the path becoming the &path variable.

setByName node, field, value

Sets the field specified in the parameter 'field' to equal the value specified in the parameter 'value' on the specified node. The 'field' parameter can be held in a variable.

Merge (list, path)

Merges the specified list of nodes into the given path in the database. Each node in the list can contain an additional field, '_MergeStrategy' that can be either 'None', 'ForceAdd' which forces the record to add and never update, 'NeverUpdate' which

allows child records of the node to be merged but will never update fields in the given record if it already exists, or 'Delete' to delete the record that matches.

LoadSpreadsheet(text)

Imports the specified text as a tab-separated spreadsheet. The top line is a list of header columns that must be either the name of the title of the corresponding field, and subsequent rows contain the data. Returns the list translated from the data.

RecreateCompositeList(path, field)

Recreates the composite list defined at the specified path (Path + field).

NodeToXml(record)

Creates an XML textual representation of the specified record.

NodeFromXml(record, text)

Creates a node from the specified XML text, using record as a template.

Nothing()

Returns a null value.

Nolink()

Returns a null value suitable for a foreign link field.

UserRef()

Returns a reference to the user path for the currently logged-in user.

DescribeNode(record)

Returns a human-readable textual representation of the specified record.

HtmlNode(record)

Returns the same as DescribeNode but in HTML format.

DescribePath(path)

Returns a human-readable, textual representation of the specified path.

NewGuid()

Returns a new globally-unique identifier.

Exists(value)

Returns True if the value contains data, or False if the value is null.

NotExists(value)

Returns False if the value contains data, or True if the value is null.

Substring(string, start, length)

Returns a substring of the specified string, starting at character position 'start', for 'length' characters.

SubstringRight(string, length)

Returns a substring of the specified string, starting at the far left of the string and returning 'length' characters.

Contains(text, substring)

Returns true if the substring is contained within the text.

GetGoverningUsersFor(path)

Returns a list of user area records for the users governing the specified path for workflow purposes.

ViewLink(path)

Returns the URL to the specified path in the database.

ViewLinkWithHost(path)

Returns the URL to the specified path in the database, and the URL includes the web server host name as specified in the application's configuration record.

GetSmallIconHtml(iconID, backColor)

Returns some HTML to display the small icon for the specified identifier, with the specified background color. The background color should be provided in hexadecimal (RGB format). The icon ID should be the ID of the icon as displayed when selecting icons in dbFramework.

MakeLink(display, path)

Returns a link string for the given display text and path, suitable for setting a foreign-link variable.

ViewForeignLink(value)

Returns a URL to the specified path, specified in a foreign link value, in the database.

EmailUser(userRecord, subject, body, fileAttachment)

Sends an email to the user specified by the userRecord, using the subject, body and an optional file attachment.

Note: This function uses the email server configuration in the Config tab of the application in order to know how to send the email. If the Configuration record is not created in the application, EmailUser and EmailUsers will return an error. In addition, the userRecord must have an email address specified or they will be excluded from the email.

EmailUsers(userList, subject, body, fileAttachment)

Works the same as EmailUser but sends an email to a list of users as specified in userList. Each user will be incorporated into the To line of a single email.

FormatDouble(date, formatting)

Returns a textual representation of the number using the specified standard formatting string.

Note: Information on the options for formatting codes can be found at:
<http://msdn.microsoft.com/en-us/library/0c899ak8.aspx>

Double(text)

Interprets the specified text string as a double (numeric) type.

Round(value)

Rounds the given value to the next whole number.

Trunc(value)

Truncates the given value to the whole number.

FileLink(path)

Returns a URL to the path in the database that must point to a field of file type in a record.

GetTitle(record)

Returns the title of the specified record.

GetMapDesc(record)

Returns the address for the record that has Map Features enabled.

Count(path)

Returns the number of records at the given location in the database, or in the given list variable.

Sum(path)

Returns the total of the path in the database. The last field of the path must be the field to total.

Average(path)

Returns the average of the path in the database. The last field of the path must be the field to average.

IPAddress()

Returns the IP address of the user's machine.

IIf(condition, result1, result2)

Chooses either result1 or result2 depending on the value of condition.

ConvertToString(object)

Converts the given variable value to a string representation.

Random(min, max, decPlaces)

Returns a random number between min and max, rounded to the specified decimal places.

Numbers(start, end, steps)

Returns a list of numbers starting at 'start', ending at 'end', jumping 'steps' for each entry. Can be used with loops.

StringList(text, delimiter)

Returns a sequence of words contained in text when split using the specified delimiter.

ValidateScript(script)

Checks whether the given DbfScript is valid.

AddComma(value)

If the value is not empty, returns a string representation of the value with an appended comma also. Used for creating text lists.

CreateInputForAction(actionName, path)

Creates a record that will be used for the input of an action (actionName) found at the specified path, and returns the new record.

20.9. Path Functions

ExtractPath(path1, path2)

Returns a subset of the specified path1 that resembles the path specified in path2.

Parent(path)

Returns the parent path of the specified path.

ClearPredicates(path)

Returns a path with all predicates of the last path item removed.

AddPredicate(path, predicate)

Returns a new path with an additional predicate (eg. Age>30).

GetPath(link)

Returns the actual path held in a foreign link value.

ForeignLinkDisplay(link)

Returns the display value of a foreign link value. This is the value that is displayed on the screen, which may differ to the title of the record that the link points to.

ForeignLinkURL(link)

Returns a URL that links to the record linked to by this link value.

GetPathString(path)

Returns a text version of the given path variable. Whereas DescribePath will provide a human-readable version of the path, this returns only a system-readable version.

PathForNode(node) – or- PathForRecord(node)

Returns the path where this record was originally located.

URLSafe(text)

Returns a version of the specified text that is safe for inclusion in a URL.

DecodeURL(text)

Returns the original version of the specified text that was previously encoded to be safe for inclusion in a URL.

20.10.Date/Time Functions

Today()

Returns the date of the current day, without any time portion.

DateOnly(date)

Returns the date portion of the specified date and time.

SetTime(date, time)

Returns a date with the specified time string appended.

AddDays(date, days)

Adds 'days' number of days to the specified date.

AddMonths(date, months)

Adds 'months' number of months to the specified date.

AddMinutes(date, minutes)

Adds 'minutes' number of minutes to the specified date/time.

AddSeconds(date, seconds)

Adds 'seconds' number of seconds to the specified date/time.

Today()

Returns the date portion of today's date.

DateOnly(date)

Returns the date portion of the specified date.

Year(date)

Returns the year portion of the specified date.

Month(date)

Returns the month portion of the specified date.

Day(date)

Returns the day of the month of the specified date.

DaysBetween(date1, date2)

Returns the number of days between the two dates.

SecondsBetween(date1, date2)

Returns the number of seconds between the two dates.

FormatDate(date, formatString)

Returns a textual representation of the date using the specified standard formatting string.

Note: Information on the available formatting codes can be found at:

<http://msdn.microsoft.com/en-us/library/az4se3k1.aspx>, and:

<http://msdn.microsoft.com/en-us/library/8kb3ddd4.aspx>

Date(text)

Interprets the specified text string as a date and returns the date.

Now()

Returns the current date and time.

IsHoliday(path, date, includeWeekends)

Returns True if the holiday schedule, indicated in path, contains a holiday entry for the specified date. If includeWeekends is True, will also return True if the specified date is a weekend.

NullDate()

Returns a value that is used to indicate a null, or unset, date.

DateStr(date)

Returns a string representation of the given date.

GetDate(text)

Returns a date value for the given string.

20.11. File Functions

file:Load(file)

Loads the specified file at the given absolute path, which must be local to the dbFramework server. Returns a string variable of the file content.

Note: This function should not be used as a permanent solution. Creating dependencies on file locations of the server makes it difficult to relocate the server. These files may be removed or touched by other users, thus making the server less stable and less reliable. Rather the file should be stored in the database where it is protected, and accessed using the load keyword.

GetFileText(file)

Given the specified file object, extracts the text in that file.

GetFileName(file)

Given the specified file object, extracts the original name of the file.

20.12. Tokenizer Namespace

The tokenizer namespace deals with splitting files into tokens, typically for data-scraping from data files or websites.

Note: The following functions must be prefixed with "tok:".

Load(text)

Loads the specified text into the tokenizer engine.

LoadEx(text, indicateLines, letterChars)

Provides the same functionality as Load, but with additional options. 'IndicateLines' can be True or False, and indicates whether to include NEWLINE constants where there are new lines in the file. 'LetterChars' is a string list of characters that should be included in tokens, and not separated.

LoadUrl (url)

Loads a web page from the specified URL into the tokenizer engine.

GetBookmark ()

Returns a bookmark that represents the current location of the tokenizer cursor.

SetBookmark(bookmark)

Sets current cursor location to be the specified bookmark.

GetText(bookmark1, bookmark2)

Extracts the text between bookmark1 and bookmark2.

Closer(token1, token2)

Returns True if token1 is closer than token2 from the current cursor position.

EOF()

Returns True if the end of the file has been reached.

Exists (token)

Returns True if the specified token exists from the current cursor position onwards.

SkipTo(token)

Moves the cursor beyond the next occurrence of the specified token.

SkipBackTo(token)

Moves the cursor before the previous occurrence of the specified token.

Peek()

Returns the next token without moving the cursor position.

Next()

Returns the next token, and moves the cursor on one.

BackOne()

Moves the current cursor location back one.

NextNum()

Returns the next token(s) and interprets it as a number, moving the cursor to the position directly following.

NextDate()

Returns the next token(s) and interprets it as a date, of the format MM/DD/YYYY, moving the cursor to the position directly following.

NextTo(token)

Returns all tokens from the current cursor position to the location before the specified token.

MustBe(token)

Checks to see if the next token is the specified token. If it is not, an error is raised and executed stops. If it is, moves the cursor position to the following token.

IsNext(token)

Returns True if the next token is the one specified, or False if it is not.

21. Connectivity

21.1. Overview

While dbFramework provides an end-to-end development system that does not require any third-party tools to create sophisticated applications, it's sometimes desirable to take advantage of the wealth of tools and utilities provided by other systems, libraries and platforms.

dbFramework provides a variety of means to integrate with third-party systems. The most comprehensive of these is the .Net SDK, but there are also APIs built-in that allow access to your dbFramework applications using Microsoft Excel, XML, Javascript's JSON format, iCal calendar and RSS feeds.

These programming interfaces enable such things as:

- Easily creating business logic in different languages and platforms.
- Integrating dbFramework applications with other systems, including:
 - Displaying application data in spreadsheets
 - Invoking actions in applications when events occur in other systems.
 - Invoking actions in other systems when something occurs in a dbFramework application.
 - Synchronizing data between two systems.

21.2. Emailing Page Links

Because all records in a dbFramework application have a unique web address (URL), it's quite easy to simply copy-and-paste this URL into an email to send to another user to bring it to their attention. These URLs contain enough information to get to that record, and it will always be accessible so long as the record exists in the database.

When sending these links, some consideration should be given as to whether the recipient will have sufficient privileges to access the record. When the recipient receives the email and clicks on the link, they may be asked to first login to the system. After they login they will be taken to the page in the link.

However, if the user does not have access to view that record, they will get an error displayed such as 'Access Denied'.

Any other restrictions placed upon them, such as their ability to modify, create or delete records, will also be applied once they login. Suffice to say: don't expect the recipient's experience to precisely match your own.

21.3. Spreadsheet Import/Export

21.3.1. Importing Spreadsheet Structures

If you have an existing spreadsheet that contains a list that you simply want to turn into a database, dbFramework offers a very simple mechanism to achieve that.

From within dbFramework, there will be a task called 'Import Spreadsheet Structures'. Click this and you'll be provided with a place to paste your spreadsheet data.

Once you submit that, dbFramework will analyze the data that you pasted and determine the fields to add to a new table that will be able to hold all of that data. It will examine the data in each column to determine the data type.

The resultant tables can then be modified to suit your specific needs.

Once the table is created you can then publish the application and use the 'Import Spreadsheet' task to actually import the data from the same spreadsheet. This is covered in the next few sections.

21.3.2. Exporting

Every screen in your dbFramework application can easily be exported to Microsoft Excel or any other spreadsheet by clicking an automatically-added report called 'Export to Excel Spreadsheet'.

This report will export all of the data in the currently viewed list, up to the first 500 rows. To get data beyond that, you must add a &page=2 (or whichever page you are on) to the URL to see the next 500.

The export feature does not retrieve child data, just a flat list of the data available at the current location.

Note: The data that is exported is in .TSV format (tab separated values). This can be opened in any spreadsheet application. However Excel may show a warning that the file data is not in its native format. Just ignore this warning to open the file.

21.3.3. Importing

Data can also be imported from a spreadsheet just as easily. From anywhere in the system there is a special action called 'Import Spreadsheet'. This lets you import data in the same format into the list.

Data can either be stored in a file and selected through the button at the top, or can be pasted directly into the large text area at the bottom.

The format of the data is identical to that of the exported data, with the first row being the header (field names) and subsequent rows being the data itself.

21.3.4. Importing Hierarchical Data

However, the Import Spreadsheet feature also supports importing of hierarchical data using a special format. If the first column header in the imported data starts with a forward slash, then it indicates to the system that some rows should be placed into child records.

In this case you specify multiple headers for each area in the database you wish to specify data. Each header starts with the relative path of where that data will go. Once the headers are specified, you leave a blank line before you start the actual data.

In the actual data the first column indicates the level at which that row will be inserted.

It's easier to explain this with an example. If, for example, you wanted to import data into a contact and sub-contacts, you might use the following data:

/	Name
/SubContacts	Name

/	RiaForm Technology
/SubContacts	Mr Contact
/SubContacts	Mr Other Contact

The first line indicates the columns for the root (this would happen in a Contacts list), and we have just one column called 'Name'.

The second line indicates the columns for the child type, 'SubContacts'. This also has a column called 'Name', but it could be any kind of column name.

The third line is an empty line to indicate that the header section is complete.

The next line is the start of the actual data. The first column must indicate the relative location of the row, relative to the previous row. In the case of the first line it's the top level contact.

Under that we have a subcontact, so the path shows a relative location of /SubContacts. The same thing happens in the next line. This inserts two sub-contacts under the top contact.

21.4. Excel Add-In

21.4.1. Overview

The Import and Export feature allows static pages of data to be transferred between systems. But you can also integrate your dbFramework application's live, dynamic data into spreadsheets.

Using a tool that can be requested from RiaForm (see contact section), dbFramework applications can easily be integrated *into* spreadsheets created with Microsoft Excel version 2003 onwards. The tool is known as an RTD Server (Real-Time Data Server) and provides a special formula syntax that can be used to query data from any dbFramework application.

What's more, the data that is queried is kept up-to-date in real-time so that if the data changes, that change will be reflected in the spreadsheet almost immediately.

This live, real-time connectivity between the spreadsheet and application works even if the dbFramework application is running on another machine in another country over the Internet.

21.4.2. Installing DbfRtdServer

Before you get started, you will need to request the DbfRtdServer.dll file from RiaForm (see contact page at back of book).

Once you have that file, you will need to register it. To do this run the following command from the command line prompt, ensuring you are in the folder where the DbfRtdServer file resides:

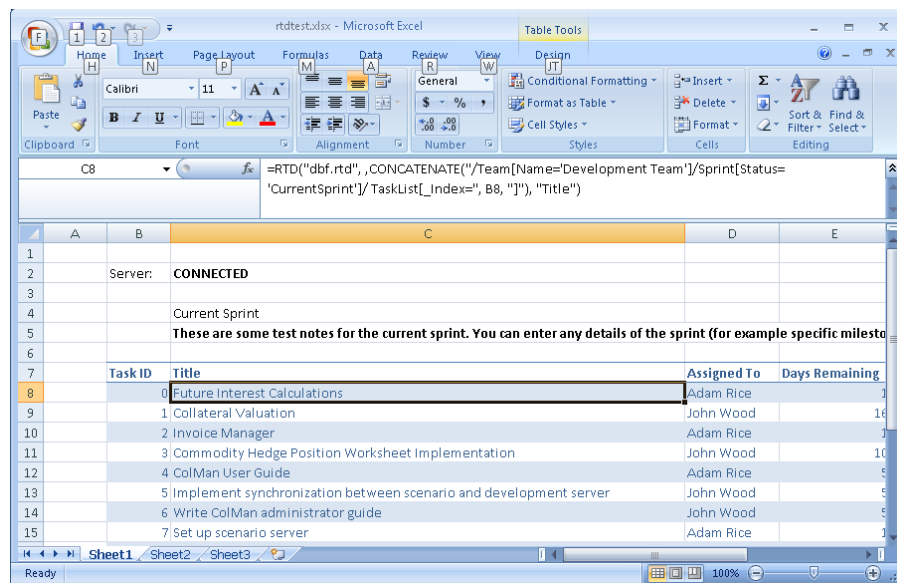
```
\Windows\Microsoft.Net\v2.0.50727\RegAsm.exe DbfRtdServer.dll
```

Assuming your Windows installation is in the \Windows folder, and you have .Net v2 installed, this will register the RTD add-in with the system and make it available to Excel.

If you have difficulty with this step, please contact RiaForm Support.

21.4.3. Using the RTD Server

Open Microsoft Excel and create a new spreadsheet.



To use the dbFramework RTD server, you will need to use special formulae. There are two types of formula you will use:

- A formula command to locate and login to the application
- A formula command to retrieve data

Every spreadsheet must have one (and only one) login formula that will establish the connection with the dbFramework application.

The formula will look like this:

```
=RTD("dbf.rtd", , "LOGIN", "host:port", "user", "password")
```

The 'host:port' should be the name of the server and the port on which the application is running. The 'user' and 'password' should refer to the username and password with which you wish to login to the application.

After you run that command the response should be either 'CONNECTED' or 'Access Denied', if the user cannot be found in the system or the password does not match. There may also be other errors if the system cannot connect to the specified server.

Note: If your application has no login requirement, leave the user and password as empty strings.

Once connected, the next formula will be to actually retrieve some data.

Note: Even if the system does not connect successfully, formula in the spreadsheet that had previously run will still show their last retrieved value.

This formula to retrieve data is just as simple. Here is the structure:

```
=RTD("dbf.rtd", , "path", "field")
```

In this formula, path is the XPath that points to the location in the database to retrieve, and 'field' refers to the field in that record to put into the cell.

Here's an example:

```
=RTD("dbf.rtd", , "/Team[Name='Development  
Team']/Sprint[Status='CurrentSprint']", "Notes")
```

Notice how, in the path, the double quotes that are normally used in predicates are replaced with single quotes.

You can also have the path point to a list and retrieve a record by its ordinal position (based on the default sort order). To do this use a special predicate field called `_Index`. For example:

```
=RTD("dbf.rtd", , "/Team[_Index=1]", "Name")
```

This would retrieve the first team entry. Putting `_Index=2` would retrieve the second entry, and so on.

You can also integrate values from other cells in your path selectors. For example:

```
=RTD("dbf.rtd", , CONCATENATE("/Team[Name='Development  
Team']/Sprint[Status='CurrentSprint']/ TaskList[_Index=", B8, "]),  
"Title")
```

Here we're using the `CONCATENATE` function to add two strings together, and referencing the cell B8 which will hold an index. This lets us specify the row number so we can repeat the formula to retrieve the entire list.

21.4.4. Advanced Usage

It's also possible to combine the Excel RTD server plugin with VBA (Visual Basic for Applications) scripting in order to invoke actions in the dbFramework application. You could do this by using the XML API. To invoke the actions through the web interface you would need to use a special object called "MSXML".

An example of creating this object and configuring it to post data to a server is as follows:

```
Set objHTTP = CreateObject("MSXML2.ServerXMLHTTP")  
URL = "http://server/apiput.aspx"  
objHTTP.Open "POST", URL, False  
objHTTP.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE  
6.0; Windows NT 5.0)"  
objHTTP.send "<post data>"
```

The post data should be in the format specified in the XML API.

The best part of this: once you update the database, the cells that reflect that data will update automatically in real-time, without the need for any refreshing.

21.5. Using the .Net SDK

21.5.1. Overview

dbFramework provides comprehensive support for using third party languages and technologies through the .Net interface. Any application created with dbFramework can be enhanced by creating a .Net plug-in DLL. The plug-in DLL will have access to all services and data provided in the application through first-class types, enabling features such as compile-time syntax checking and errors along with intellisense dropdowns and statement completion. Yet the plugin can also access any other service available to it – including third-party web services, libraries, development tools and debuggers.

21.5.2. Prerequisites

The following setup is required:

- Microsoft .Net 2.0 runtime
- Suitable code editor, such as
 - Microsoft Visual Studio 2005 or higher (The free Express version works fine). C# is preferred.
 - Notepad
- dbFramework Connectivity DLL Pack. This is available on request from RiaForm Support (see contact page at back of book).

21.5.3. Setting up Your Project

21.5.3.1. *Creating the Project*

This section will assume you're using Visual Studio 2005 or later, with the C# programming language.

To get started, create a new project 'Class Library'.

Then add references to the DLLs required by the dbFramework project that come with the dbFramework Connectivity DLL Pack:

- Saxdaapi.dll
- Xaml.dll
- Radeo.dll
- CeliumApp.dll

Keep these files in a central location so they can easily be updated if new releases are made available.

21.5.3.2. Build Settings

It's very important that you set the project settings to compile to the 32 bit platform. At present dbFramework runs as a 32bit process, so if the module is compiled to 64 bits then the system won't recognize the module and it cannot be used.

To change the platform, go to Build / Configuration Manager, and create a new solution platform with a target platform of x86.

When you compile, make sure that is the active solution platform. Also note that the output folder may be in an x86 folder.

21.5.3.3. Generating the SDK

dbFramework can create a whole SDK especially to access your application. The SDK consists of a number of classes that can be used to interact with all of the features of your application from querying, to updating data, running transactions and running reports.

To generate the SDK, go to your application in dbFramework, and locate the report called 'Generate C# SDK'. Right click on this and have it save the target. Select a file called 'SDK.CS' and save this into your project folder.

Once generated, go to Visual Studio and insert this file into your project. If your project isn't using C#, then follow the steps in section 21.5.15.

Once the SDK is inserted into your project, and the steps for setting up the project above were followed, try building the project to make sure everything is configured correctly.

Note: The SDK can be regenerated at any time and the SDK.CS file overwritten. After overwriting it, rebuild the project and resubmit the DLL to dbFramework.

By having a generated SDK attached as a reference in your project, you will see all of the types in your application available as first-class citizens of your code. You will see intellisense dropdowns for each of the types and fields you defined in your application, you will get code-completion to make coding faster, and will see compile time errors when you attempt to access fields in your database incorrectly. All these features go a long way towards making coding far more efficient.

21.5.3.4. *Attaching the Module to your dbFramework Application*

Once the module is compiled, you can then attach it to your dbFramework application so that it can be used.

To do this, go to your application record in dbFramework, and click the task labeled 'Submit .Net Module'.

Submit .Net Plugin

Code Module Submission

DLL File
 No file chosen

Symbol File
 No file chosen

Description

Here you select the DLL file that was output by the compilation process by clicking the DLL file button.

You can optionally also select the symbol file. This is the PDB file in the same folder, and contains debugging information about the module that lets you trace through the program using a debugger. This process will be discussed in more detail at the end of this chapter.

Once you submit the DLL file the system looks for methods in the module that can be used in your dbFramework application. These will automatically become available in screens such as the event editor. Once these are selected the application will have to be republished before the changes take effect.

You can submit the module any number of times and it will just overwrite any entries that are already in the application.

21.5.4. Concepts

21.5.4.1. *Service Orientation*

dbFramework uses an architectural model known as 'Service Orientation'. This means that each function in the application is considered a service that can run on any data with which it is provided. This differs to 'Object Orientation' where functions and data are tightly bound to one another. This loose binding between

business logic and data solves many problems in application design, especially in scalability (ie. the ability for an application to grow in size), but also simplifies the design, allowing a great deal of functionality to be created without writing any code at all.

For efficiency, and because .Net languages are all object-oriented, these services are created as methods on classes, but the classes are not used to hold data.

Important! Do not store any data in the objects created in your module. As tempting as it may be, there is absolutely no guarantee that data will be available each time the service is called, and doing this will impede the ability of your application to scale to multiple servers. Always use the application itself to store data, via the functions in the SDK.

Instead each class has a reference to a 'service coordinator' that lets it interact with the system. These interactions allow the services to:

- Retrieve data from the database
- Update, insert or delete data from the database
- Run other services
- Run reports

These four aspects are sufficient to access all of the functionality in your application. Because .Net languages also have access to the operating system, other platform features and third-party libraries, there really is no limit to how this limitless functionality can be integrated with your application to enrich the features available to your users.

21.5.4.2. *System Interaction*

Each class that holds services must have a public field defined of type AppObjects. Conventionally this is called 'App', as it allows access to the application through various objects created in the SDK.

```
public AppObjects App;
```

When the service is used, this App field is set *automatically* by the system. Every interaction with the application will be done through this App instance. Access to the application should not be attempted through any other means.

All data types used in your service should be accessed through the namespace generated by the SDK. This namespace is named based on the short company

name and the application name. For example, if the short company name was ABCCorp, and the application name was 'InvoiceMgr' then the namespace will be ABCCorp.InvoiceMgr. In this namespace you will find all of the classes that represent each element in your data model.

When data is retrieved from the application, it is stored in objects called records, or sometimes nodes (reflecting its origin as a tree-based hierarchical data store). The result can either be a single record or a list of records, known as a 'Sequence'. Both the node and the sequence will be of types that are defined in the SDK namespace.

All data is accessed using something called a Path. This path is the hierarchical address that uniquely refers to a specific node in the database. The SDK provides many different functions for manipulating paths letting you freely navigate through the database.

21.5.5. Creating a Service

21.5.5.1. Overview

A service provided by your module is simply a method in a class. The only differentiating signature of your method are the parameters of the method and an attribute that must be attached.

The attribute is put above the method signature, for example:

```
[DbfMethod (DbfMethodTypes.Action) ]
public void QuickApplication(model.ActionInputTypesNewApp newApp)
```

The **DbfMethod** attribute must be applied to any method you wish to be recognized as a service by dbFramework. The parameter to that attribute indicates what type of service you are creating. There are several options:

- **Action**
An action is the most common type of service. This takes a path and a node and performs some change in the database. It is commonly attached to events, tasks and scheduled jobs.
- **Calculator**
A calculator can be used to take one set of data, modify or supplement it, and return a different set of data. It is commonly used when generating reports.

- **Layout**
A layout service provides a list of columns to display on the screen. It allows business logic to determine the layout of the form, and captions of fields, when editing or displaying records.
- **Preparer**
A preparer service provides default values for fields in a form when it is being used to create a new record.
- **Query**
A query service returns data that is typically used in a report.
- **Verb**
A verb is a special type of action that takes no parameters.

Each of these types of services will still interact with the application in the same way. But the signature (ie. the parameters and return value) of each method may differ.

You are free to choose any name for your method, but it should reflect what the service does in order to avoid confusion.

21.5.5.2. Parameters

There are four types of parameters that can be declared in a service:

1. **A Path**
A descendent of the BasePath class, each declared in the model.PathBuilder namespace. An object that holds an address to a specific location in the database, and provides functions that can be used to manipulate the path.
2. **A Node**
A descendent of the BaseNode class, each declared in the model namespace.
3. **A Sequence**
A descendent of the BaseSequence class, each declared in the model namespace, suffixed with the name 'Sequence'.
4. **A Warning Manager**
A special object of type 'WarningManager'. This is used for raising warnings to the user, and determining which warnings have previously been accepted by the user.

Whichever of these parameters are declared on your service will be set automatically by the system when the service is invoked.

However there may be times when requesting one of these is actually invalid. For example a verb cannot have a path because it is run globally.

Each of these will be discussed in the next section(s).

21.5.6. Paths

All path classes have a suffix of 'Path'.

There is a path class created for each type in the data model that it points to. For example if I wanted to have a path to a task the path object may be an instance of `model.PathBuilder.TaskPath`.

You can also create a path using the `model.Root.Path` variable, which provides a path for the root of the database hierarchy. If you had a list of Teams at the top-level of your application, you could get to that by using `model.Root.Path.Teams`.

Each node in the path is actually a method that returns a new path given a specific predicate. This predicate, which is a filter, takes the form of a field, an operator (such as '=' or '>=') and a value.

The field the predicate requires must be a field of the type at that point in the path. For example, if the predicate was for `model.Root.Path.Teams`, the field would have to be a field in the Teams type.

Fields for a specific type can be found in `model.YourType.Fields` list, where `YourType` is the name of the type.

For example:

```
model.Root.Path.Teams(model.Teams.Fields.Name, "Development")
```

In this example we are creating a path to a team with the name 'Development'. Notice how, if you omit the equality comparison operator, it will default to equality.

While this is a method that returns a new path instance, you can combine these method calls on one line to create more advanced paths. For example consider this:

```
Model.Root.Path.Teams(model.Teams.Fields.Name,
    "Development").Tasks(model.Tasks.Fields.Complete, false).Log()
```

This statement will return a path that points to the log for incomplete tasks in the Development team. The actual XPath will be:

```
/Teams[Name="Development"]/Tasks[Complete = False]/Log
```

There are several methods being called here. The first method call to Teams returns the path, /Teams[Name="Development"]. The second method is on *that* path, which provides access to the tasks. The 'Tasks' method call returns a specific task path, which then provides access to the Log path.

Multiple filters can be applied to a field in the path. You do this using the .Where method. If you have two filters that reference the same field, they will be 'OR'd together. If there are different fields, they will be 'AND'd together.

For example, consider the following path:

```
Model.Root.Path.Teams(model.Teams.Fields.Name,
    "Development").Tasks(model.Tasks.Fields.Complete,
    false).Where(model.Tasks.Fields.Priority, "<", 5)
```

This adds a filter Tasks for both Complete being false, and Priority being less than 5.

Also consider this:

```
Model.Root.Path.Teams(model.Teams.Fields.Name,
    "Development").Tasks(model.Tasks.Fields.Status,
    "Waiting").Where(model.Tasks.Fields.Status, "NotStarted")
```

This adds two filters to Status. This will create a filter where the status is either Waiting OR NotStarted.

Also note that dropdown lists in your data model are created as enumerators in the SDK. This is discussed in the dropdown list section.

You can create variables that can hold a path to any type in your data model. For example, consider this variable declaration:

```
model.PathBuilder.TasksPath taskPath;
```

This declares a path that points to the Tasks type. This can hold any path so long as it addresses a record that is of the Tasks type.

Note: The use of the ID field is very important in constructing paths. Some functions such as Insert, Delete and Update require that you only use IDs in the part of the paths up to the last path element. The ID uniquely references the specific record.

A path instance provides a number of other functions that can be used to manipulate that path.

ClearPredicates

Removes any existing filters at the end of the path.

Clone

Returns a new instance of the path that is no longer associated with the existing path.

Equals (BasePath path)

Returns true if the passed-in path is equal to the current path, even if the two paths are different instances.

ExtractPath<type>(App, path)

Using generics, looks for the last item in the path that is of the specified type. For example, let's say you had a structure such as a list of Teams, containing a list of Tasks. If you were passed a path to a task, but wanted to get a path to the team, you could use something like this:

```
taskPath.ExtractPath<model.PathBuilder.TeamPath>(App, taskPath)
```

This would take the path to a task (taskPath) and extract from that path the path to the team (TeamPath).

GetPath

Returns the XPath string for this path instance.

ParentPath

A field that holds the XPath of the parent of this path.

SetPath

Takes a string XPath and sets the path used internally in this path instance. Must point to the correct type.

Where

Takes a field, optional comparison operator, and value and adds this as a predicate on the path. Returns a new path with that predicate set. Can be used to combine multiple filters on a path. This is useful if you have an existing path instance but wish to apply a new predicate to it.

21.5.7. Dropdown Lists

All dropdown lists, whether defined in the 'Dropdown Lists' drilldown from the Model record, or in the drilldown from a field record, will be created as enumerators in the SDK. This lets you use values for enumerators with compilation checking.

The compilation checking is very useful to detect errors in your code if the model changes. If, for example, you change the Task state 'Deferred' to 'ReAssigned', then you will get compilation errors for each line in your code that still uses 'Deferred'. If you just referenced 'Deferred' as a literal string, the compiler wouldn't catch these errors and the problem may not appear until the product is in use in production.

Each enumerator is defined in the top level of the namespace. For example a dropdown list called 'TaskStates' will be referenced as:

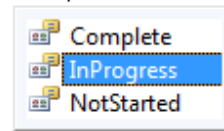
```
model.TaskStates
```

If you had a variable called 'task' that was of type 'model.Tasks', and that had a 'State' field, you might use a line like this to change the state:

```
task.State = model.TaskStates.Complete;
```

You will also notice the dropdown in the code editor that gives you the options for each value in this dropdown list:

```
task.Status = TaskStatus.
```



```
TaskStatus TaskStatus.InProgress
```

Check all children of this task to see what their status is. B

In addition to this, any comments associated with the dropdown items will appear in the tooltip beside the dropdown entry.

21.5.8. Nodes

A node holds the data for a single record of a specific type in the data model. Each type in the data model has its own class to hold data. The name of the class is the same name as the type.

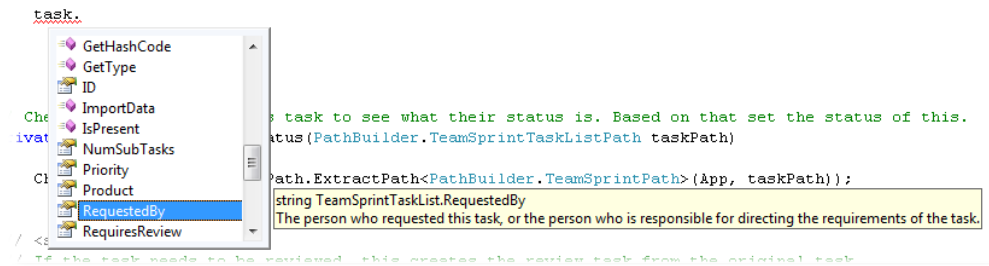
Note: Type names are based on the hierarchy. If you have a top-level field called Team, which contains a nested-type called Tasks, the type name for the tasks will be TeamTasks. It combines the names in the hierarchy to derive the type name. You can override the type name that is used in the Advanced drilldown from a type.

This class is similar to a regular class in C# in that it has properties that reflect the values. You can read and change values in the record. Changing data doesn't affect the database, however, it just changes an in-memory cache of that data. You would then commit that data to the database by calling a service to update data at a specific path. All data that is retrieved from the database is then disconnected from it so that it can be modified offline.

To access a field in your type, you just use the field name as a property of the class. For example, to access the title field of a task record held in a variable called 'task', you might use the following expression:

```
task.Title
```

You'll notice that in Visual Studio, as soon as you press the period after task you will see a dropdown list of the fields available in that record:



Each of these fields will have a description displayed in a tooltip, as seen above, that reflects the Description value you entered for that field in your data model. This can be very useful to convey the meaning of fields to the developer.

As mentioned previously, updating a field to a new value just changes the value of the field in the cached, disconnected node. It won't update the database automatically. This kind of 'disconnected' architecture is central to 'Service Orientation' and makes scaling the application far easier.

To change a field, you just assign a new value to it:

```
task.Title = "Some New Title";
```

To actually commit that to the database, you will pass the 'task' variable into an Update service, which will be discussed in the next section.

To create a new, empty record of a specific type, perhaps to insert into the database, you can use the 'new' keyword. The constructor for the type takes the AppObjects class instance as a parameter.

For example, to create a new task record, you can use code such as:

```
model.Task someTask = new model.Task(App)
```

You can then set fields on the 'someTask' variable, and when you're done you can insert that into the database.

All node instances have a set of methods that can be used to interact with the data:

ExportData

Creates an IDataSourceNode instance of the class that can be used like a dictionary class for examining the content of the record.

GetChanges

Creates a new instance of the node which just contains the changes to the node since the last time GetChanges was run. This is an important method to use when updating the database to ensure that you are only changing the fields that have been modified, and not every single field. It is important for performance reasons and also for any audit trails.

ID

Every node has an ID property that contains a unique GUID to identify the record.

ImportData

Takes the IDDataSourceNode instance and imports that into the current record.

IsPresent

Returns true if the specified field is set in the record.

Schema

Provides access to the Schema instance that describes the entire data model. A detailed explanation of the Schema is beyond the scope of this manual.

SchemaElementName

Indicates the name of the type within the Schema instance used to describe this record. Pass SchemaElementName into the Schema's FindElement method to get the ComplexType which describes all the fields in this record. A detailed explanation of the Schema is beyond the scope of this manual.

Services

Returns an object that provides access to all of the services (actions and reports) available for this record. These correlate to the 'What Now' and 'See More' lists in the application. These are discussed more in a later section.

21.5.9. Sequences

A sequence holds a list of nodes that are all of the same type. A sequence can also be empty.

For each node class there is also a sequence class, and this has the same name but with a 'Sequence' suffix. So, for example, if the task class is called model.Task, then the sequence for that will be called model.TaskSequence.

Sequences are either retrieved from the database, where they are also disconnected from the database so they can be modified, or they are created anew so that they can be filled up.

To create a new sequence, you use the same mechanism as you did for creating a node. For example:

```
model.TaskSequence someTaskList = new model.TaskSequence(App)
```

There are then several methods you can use that are specific to each sequence:

Add

Adds the specific node instance to the sequence. Once the node is added, a copy of the node is put into the list so changes to the original will no longer affect the one in the list.

AddArray

Adds an existing list (array) to this sequence.

AddRange

Adds an existing sequence to this sequence.

Clear

Removes all items from the sequence.

Count

A property that returns the number of items in the sequence.

Find(field, value)

Looks for the specified value in the specified field in the current sequence. Returns the first occurrence.

GetChanges

Returns only the nodes in the sequence that have been changed since the last call to GetChanges.

GetNodeType

Returns the .Net type instance of the class that represents the node held in the sequence.

IndexOf(node)

Returns the relative index (1 based) of the node within the sequence, or -1 if the node does not exist.

Insert(position, node)

Inserts the specified node instance at the specified position (1 based).

Remove(node)

Removes the specified node from the sequence, if it exists.

**Replace(position, node) -- or --
Replace(field, value, node)**

Finds the node either based on position or matching the given field and value, and replaces it with the specified node.

Sort(ascDesc, field)

Sorts the list. The ascDesc parameter should be 'true' to sort ascending, or 'false' to sort descending. The specified field is used as the sort key.

Schema

Provides access to the Schema instance that describes the entire data model. A detailed explanation of the Schema is beyond the scope of this manual.

SchemaElementName

Indicates the name of the type within the Schema instance used to describe this sequence. Pass SchemaElementName into the Schema's FindElement method to get the ComplexType which describes all the fields in this record. A detailed explanation of the Schema is beyond the scope of this manual.

21.5.10. Retrieving Data

Retrieving data from the database is done through services available on the App variable that must be declared in every class that contains services (see section 21.5.5).

There are two separate methods to query the database for each type in your data model. One retrieves a single node from the database, and the other retrieves a sequence.

Both of these methods start with the word 'Select'. This is then followed by either the name of the type (to retrieve a sequence), or the word 'Single' followed by the type, to retrieve a single node.

Both of these methods take the path instance of the location in the database you wish to retrieve. Optionally they take a parameter that provides additional information about the query you wish to run. This may include information such as the sort order of the result, the specific fields to request, whether to retrieve file data, and whether to query hierarchically.

To access the services for retrieving data, use the `App.Queries.SqlDatabase` object. In this object is a list of methods for retrieving data from every type in the database, along with methods to update and delete that data which we'll be discussing in the next section.

To retrieve a list of tasks, for example, you may use something like this:

```
App.Queries.SqlDatabase.SelectTask(taskPath)
```

This would then return a `model.TaskSequence` instance with the result of the query inserted as nodes in that sequence.

If you wanted to retrieve just a single task, you would use something like:

```
App.Queries.SqlDatabase.SelectSingleTask(taskPath)
```

And this would return a single `model.Task` node.

You can also pass in a `QueryRequest` instance to this function in order to specify more advanced querying options. The `QueryRequest` class is specific to each type, but most of the properties of this class are the same for each.

The following properties are available:

AddField

Lets you specify that you want only this field to be retrieved in the query, or any other fields specified in a call to `AddFied`. Here you can also specify an aggregate type, which causes the query to return a single row with values of that field aggregated as specified.

AddSort

Lets you specify the sort order of the results. An additional sort key is specific each time this method is called.

DefaultLock

Lets you specify the database locking strategy to use when requesting data. The default is RowLock, which means only a single row will be locked at a time. The LockHintsByType property allows this locking to be controlled on a per-type basis.

DownloadFileData

Setting this property to TRUE means that all file data associated with this record will be downloaded and made available in the appropriate fields.

RowsToReturn

Indicates the number of rows that should be returned in this query. Used for paging.

StartingAtRow

Indicates the response's row number that the query should start at. Used for paging.

TimeSeriesConfig

Indicates the configuration to use when converting the query result into a time series (which spans a specific number of days, with data for every day).

For example, to query the total number of days that the task list was expanded, based on information in a log, you may use code such as this:

```
TeamSprintLog.QueryRequest qr = new TeamSprintLog.QueryRequest();
qr.AddField(TeamSprintLog.Fields.ExpansionDays, AggregateType.Total);
TeamSprintLog totals =
App.Queries.SqlDatabase.SelectSingleTeamSprintLog(teamSprintPath.Log(TeamSprintLog.Fields.Date, ">", sprint.DateStarted).GetPath(), qr);
```

21.5.11. Updating Data

There is only one method for updating data, and this can be found at App.Queries.SqlDatabase.Update. This method has overrides for each type in the data model.

To use this function you pass in the path instance where the data resides that you're updating, and the node containing the changes you wish to update. Only one record can be updated at a time.

When passing in the node with the changes, you should use the `.GetChanges()` method on the node to return only the fields that have actually been changed. This reduces the amount of work the server has to perform, and also allows accurate logging of what had changed.

For example, to retrieve a task, change the state and update it, you might use the following code:

```
TeamSprintTaskList task =
App.Queries.SqlDatabase.SelectSingleTeamSprintTaskList(tPath);
task.Status = TaskStatus.Complete;
App.Queries.Update(tPath, task.GetChanges())
```

21.5.12. Inserting Data

To insert data, there is also only one method to use. This has overrides for each type in the data model.

The Insert function is available at `App.Queries.SqlDatabase.Insert`. It takes a path and a node, and works much the same way as the Update method.

The path, however, must point to a sequence – that is, it must not have a predicate at the end. The path, therefore, refers to the parent of the area in the database where the node will be stored.

Here is some example code for inserting a log into a sprint:

```
TeamSprintLog log = new TeamSprintLog(App);
log.Date = txn.EffectiveDate;
log.Title = "Task Progressed";
log.EntryType = LogEntryType.Progress;
App.Queries.SqlDatabase.Insert(sprintPath.Log(), log);
```

Some fields will be defaulted during the insert process. To retrieve these you must query the record after you insert it.

Note: if you need to know the ID of the record that you're inserting without having to try to locate it, just set the ID explicitly before inserting it. You can set the ID to `Guid.NewGuid().ToString()`.

21.5.13. Deleting Data

There is a separate delete method for each type in the data model. This is found in `App.Queries.SqlDatabase`, and is named 'Delete' followed by the type name.

The Delete method simply takes a path to delete. It will then delete everything that matches that path, and everything contained within that hierarchy.

Caution! The Delete method has no warnings or confirmations. It will delete everything you specify and there is no way to undo the operation. It will delete everything below that item in the database hierarchy. While you may think you're simply deleting one record, you could be deleting thousands of records underneath that record. Use with absolute caution!

Here is an example statement that will delete a specific log record entry:

```
App.Queries.SqlDatabase.DeleteTeamSprintLog(sprintPath.Log(TeamSprintLog.
Fields.ID, log.ID));
```

21.5.14. Services

Every type in the data model has a list of services that can be invoked. These correlate to the tasks and reports defined against that type in `dbFramework`.

To access these services, use the `Services` method of the node, passing in a reference to the `AppObjects` object.

Depending on how you have your application configured, this can return any number of methods, but they will be visible with intellisense in the editor:

RunReport

The `RunReport` functions allow you to run a report that is configured for the specific record. The report will be returned as a byte array that can be output to a file, or attached to an email.

Actions

Other methods returned will reflect the task actions available for that record. These may take inputs which can be provided as specific nodes.

21.5.15. Adding Member Functions

While the SDK generates classes for each type in the data model, and we discussed earlier how service orientation usually dictates that we keep the actions and data separate architecturally, the SDK does allow for member functions to be added to types, so that you can extend the classes generated in the SDK with your own methods.

To do this the SDK utilizes a technology known as ‘partial classes’. This allows you to declare the same type name as the data model type (with the same namespace declaration), and use the keyword ‘partial’ to add your own methods.

Keep in mind two things, however:

- You cannot add fields that store data, as these will be lost as the record is created for each result from the database.
- This is only available for C# projects because the partial keyword requires that the source code part of your project.

21.5.16. Using Visual Basic.Net and other .Net Languages

While the SDK that is generated by dbFramework is in C#, you can actually use the SDK with any .Net language. To do this you will need to compile the SDK into a DLL first, and then reference that DLL along with the dbFramework Connectivity DLL pack in your .Net project. Any language can be used, including Visual Basic.Net, Pascal, C++, Fortran, Java, LISP, JavaScript, Modula-2, Perl, PHP, SmallTalk, Prolog, COBOL, FORTRAN and TCL.

To compile the SDK down to a DLL, you can use the following command from the folder where the dbFramework Connectivity DLL pack and SDK .CS file resides:

```
\windows\microsoft.net\Framework\v2.0.50727\csc /out:sdk.dll /t:library
/o+ sdk.cs /reference: saxdaapi1.dll, xaml.dll, radeo.dll, celiumapp.dll
```

This assumes that Windows is installed in the \Windows folder, and that the SDK was saved as a .CS file, and that you are using Microsoft .Net version 2. Change the parameters to fit your precise configuration.

This then creates a SDK.DLL file that you can reference in any .Net language.

21.5.17. Debugging

So long as you submit the PDB file of your project, it is quite easy to debug your module. Simply locate the process (running as DbfServer.exe – you may have to show the command line parameters in the task manager to locate the specific process for your application), check the PID number, then use Visual Studio to attach to the process with that PID.

You can then load the source file into the Visual Studio code editor, and set breakpoints where you wish the debugger to stop.

Note: Edit and Continue is not currently available when debugging dbFramework modules.

21.6. Using the XML API

21.6.1. Overview

XML is a popular format to structure data and requests in web-based applications. It enables a vast number of languages and platforms to access the data in your application.

Familiarity with the XML format will be assumed.

There are two ways to access a dbFramework application programmatically using XML.

Firstly, every screen in your application can be retrieved in XML format by appending an “&xml=true” at the end of the URL. The data that is retrieved can then be consumed by an application in any way it sees fit. In fact this very approach is used internally by your application to generate the HTML pages. This approach is called the session-based protocol.

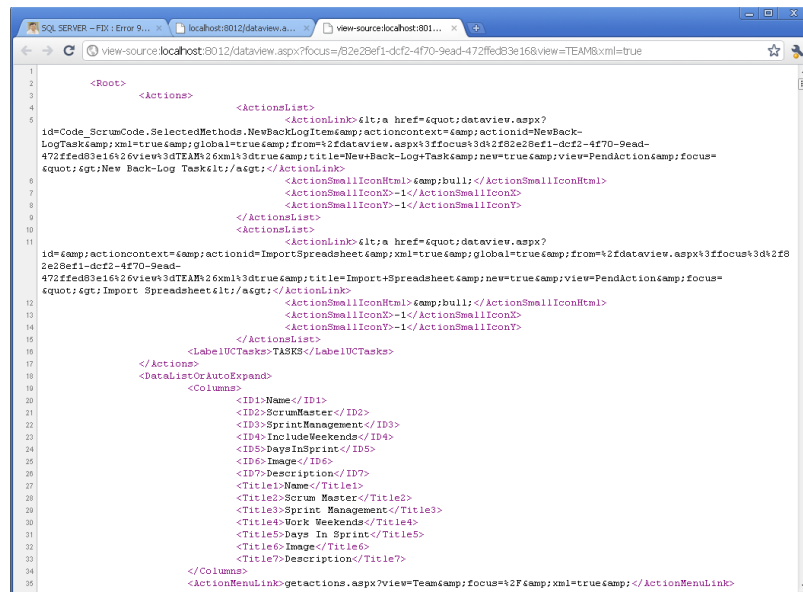


Figure 88: A regular application page rendered as XML

The second approach is to use some special stateless XML-based commands to retrieve and update data in the application. This doesn't require any session to be created. We'll be discussing both mechanisms in this section.

In this section whenever we refer to <http://servername>, the servername should be replaced with the host name of your server.

21.6.2. Logging In

21.6.2.1. Session-Based

You can either login once to create an authenticated session, or you can you're your username and encoded password with each request.

If the server does not require logging in, then this section can be skipped.

To create a session and perform a login, you will first need to request a page from the server. This is how you retrieve the session ID for your new session.

Request <http://servername/dataview.aspx?xml=true> and you should get back a page, probably redirected to the login page, but with a new cookie (in the Set-Cookies header) called Sess. Record this ID as you will need to send it with each request.

Perform an HTTP POST with the following form-encoded data:

```

USERNAME=user
PASSWORD=password
from=dataview.aspx

```

Set 'user' to be the username, and 'password' to be the plain-text password, set the Sess cookie, and post this data to <http://servername/authenticate.aspx?xml=true>. If the request was not successful you will receive a redirect URL that will include bad=true in the address, any other response should be considered a success.

21.6.2.2. *Stateless*

The stateless, sessionless approach involves sending the authentication data with each request to the server.

This will be included in the description of each API entry, but can be ignored if instead you are opting to create a session and pass the session in each time.

Alternatively the username and password can be provided as part of the URL, instead of putting the data into the POST payload. The parameter names are USERNAME and PASSWORD, and the value the password should be a base64 encoded version of the password.

21.6.3. Querying Data

Any data in the database can be easily queried and the result returned in XML format.

These requests will use HTTP GET and can easily be tested from within the browser by entering the command into the URL.

<http://servername/apiget.aspx?path=<path>&format=xml>

...where <path> is replaced with the XPath of the area in the database to query.

If a session has not been established, you should also specify USERNAME and PASSWORD parameters in the URL, both in capital letters, to specify the login name and the base64-encoded password, respectively.

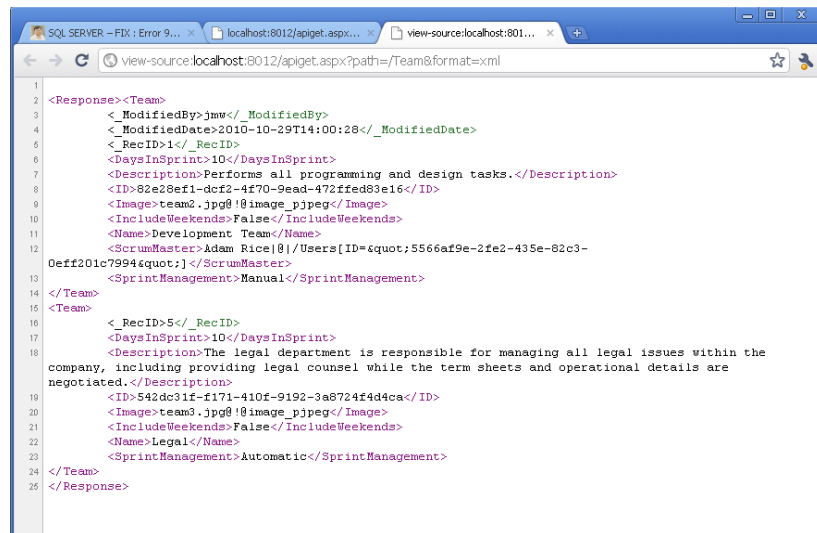


Figure 89: Example of an XML query

There are also additional parameters that can be specified in the URL, which are described below:

mapping=oldfield,newfield,oldfield,newfield...

This parameter allows you to map field names. The oldfield refers to the field name returned from the server, and newfield refers to the field name you wish to see in the result. Any number of these can be supplied, comma separated.

fields=field1,field2,field3...

This parameter allows you to specify which fields you wish to be returned. This can reduce network bandwidth and server performance, especially when requesting a lot of data. If this parameter is omitted all fields are requested.

21.6.4. Inserting Records

To insert a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```

Username=<user>
Password=<password>
Path=<path>
ActionID=insert
Xmldata=<xml>

```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

Note: The password is encoded to stop casual snooping of the login information, but it is not particularly secure. If possible use HTTPS to access the server.

The <path> variable is the XPath to the location in the database where you wish to insert the record.

Finally, the <xml> entry will be the XML of the record you wish to insert.

For example:

```
Username=test
Password=<base64 encoded password>
Path=/Reports[Name="My Reports"]/History
ActionID = insert
Xmldata= <base-64-encoded version of this data>
<Root>
<ReportID>46135</ReportID>
<Title>This Is My Report</Title>
</Root>
```

21.6.5. Updating Records

To update a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```
Username=<user>
Password=<password>
Path=<path>
ActionID=update
Xmldata=<xml>
```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

The <path> variable is the XPath to the location in the database where you wish to update the record.

Finally, the <xml> entry will be the XML of the changes you wish to apply to the record. Only the fields you wish to modify should be provided.

For example:

```
Username=test
Password=<base64 encoded password>
Path=/Reports[Name="My Reports"]/History[ReportID="46135"]
```

```

ActionID = update
Xmldata= <base-64-encoded version of this data>
<Root>
<Title>This Is A Different Title</Title>
</Root>

```

21.6.6. Deleting a Record

To delete a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```

Username=<user>
Password=<password>
Path=<path>
ActionID=delete

```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

The <path> entry is the XPath to the location in the database where you wish to delete the record.

Note: The API provides no warning or confirmation of deletes. The record and all data below that record will be deleted and this operation cannot be undone. Please use with caution!

For example:

```

Username=test
Password=<base64 encoded password>
Path=/Reports[Name="My Reports"]/History[ReportID="46135"]
ActionID = delete

```

21.6.7. File Data

You can also include file data in your records. To do this, wherever a field is of the 'File' type, enter in XML such as this:

```

<SomeFileField>
  <fileName>SomeFile</fileName>
  <contentType>application/pdf</contentType>
  <fileData>data</ fileData >
</SomeFileField>

```

In this example, SomeFile would be the name of the file to insert, application/pdf should be changed to the HTTP content type of the data, and fileData should be the base-64-encoded data of the file.

21.7. Querying iCal Data

iCal is a popular format for communicating calendar data. Many calendar applications, such as Microsoft Outlook, have the capability of importing iCal data to integrate those dates into the calendar.

Any dbFramework application that contains date information can expose that information in the iCal format automatically.

To have a particular part of the database exposed in iCal format, you must enable the Calendar Feature as described in section 11.5. This tells dbFramework which fields are to be used for the start and end date for the event.

Once that is configured, you can then run a special query to retrieve the iCal file:

<http://servername/ical.aspx?path=<path>&from=<fromdate>&to=<todate>>

Here the <path> should be the XPath to the location in the database containing the event data. The fromdate and todateries should define the range of dates you wish to include. The date can be in the local format.

dbFramework also supports a more recent format called iCalX, which is similar to the iCal protocol but in XML format. The same query can be used as above, but the page name would be icalx instead of ical.

21.8. Using the JSON API

JSON is a popular format for communicating with servers from JavaScript within web browsers. However it can also be used from applications, and there are a variety of libraries that make this very easy from many high level languages.

The protocol for accessing operations in your dbFramework application using JSON is very similar to the way in which you interact using XML. The difference is really in the format of the data records that you send, as these will be in JSON format instead of XML.

Familiarity with the JSON format is assumed.

21.8.1. Logging In

JSON utilizes the stateless, sessionless approach to logging in, involving sending the authentication data with each request to the server.

This will be included in the description of each API entry, but can be ignored if instead you are opting to create a session and pass the session in each time.

21.8.2. Querying Data

Any data in the database can be easily queried and the result returned in JSON format.

These requests will use HTTP GET and can easily be tested from within the browser by entering the command into the URL.

`http://servername/apiget.aspx?path=<path>&format=json`

...where <path> is replaced with the XPath of the area in the database to query.

If a session has not been established, you should also specify USERNAME and PASSWORD parameters in the URL, both in capital letters, to specify the login name and the base64-encoded password, respectively.

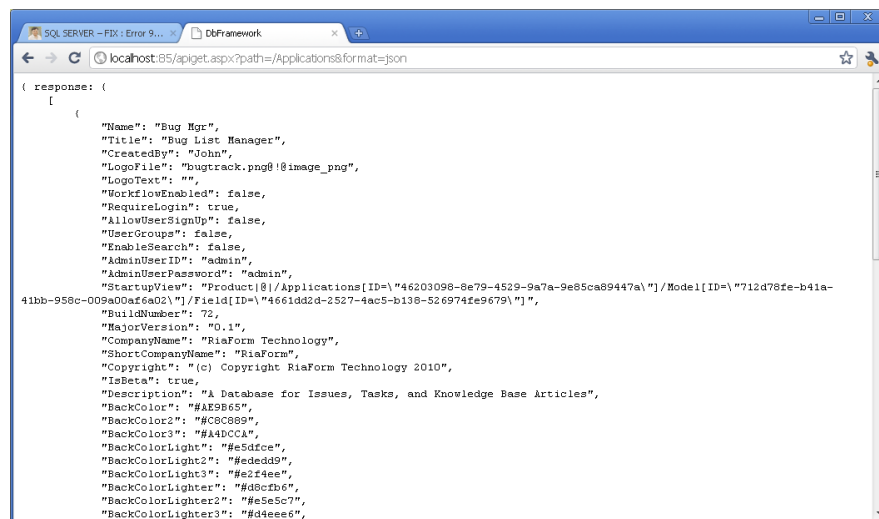


Figure 90: Example of a JSON query

There are also additional parameters that can be specified in the URL, which are described below:

mapping=oldfield,newfield,oldfield,newfield...

This parameter allows you to map field names. The oldfield refers to the field name returned from the server, and newfield refers to the field name you wish to see in the result. Any number of these can be supplied, comma separated.

fields=field1,field2,field3...

This parameter allows you to specify which fields you wish to be returned. This can reduce network bandwidth and server performance, especially when requesting a lot of data. If this parameter is omitted all fields are requested.

21.8.3. Inserting Records

To insert a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```
username=<user>
password=<password>
path=<path>
actionID=insert
json=<json>
```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

Note: The password is encoded to stop casual snooping of the login information, but it is not particularly secure. If possible use HTTPS to access the server.

The <path> variable is the XPath to the location in the database where you wish to insert the record.

Finally, the <json> entry will be the JSON of the record you wish to insert.

For example:

```
username=test
password=<base64 encoded password>
path=/Reports[Name="My Reports"]/History
actionID = insert
json= <base-64-encoded version of this data>
{
  ReportID: "46135",
  Title: "This Is My Report"
}
```

21.8.4. Updating Records

To update a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```
username=<user>
password=<password>
path=<path>
actionID=update
json=<json>
```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

The <path> variable is the XPath to the location in the database where you wish to update the record.

Finally, the <json> entry will be the JSON of the changes you wish to apply to the record. Only the fields you wish to modify should be provided.

For example:

```
username=test
password=<base64 encoded password>
path=/Reports[Name="My Reports"]/History[ReportID="46135"]
actionID = update
json= <base-64-encoded version of this data>
{
  Title: "This Is A Different Title"
}
```

21.8.5. Deleting a Record

To delete a record, you will need to HTTP POST the following information to <http://servername/apiput.aspx>

```
username=<user>
password=<password>
path=<path>
actionID=delete
```

The <user> entry should be a valid username, and <password> should be a base-64 encoded version of the password.

The <path> entry is the XPath to the location in the database where you wish to delete the record.

Note: The API provides no warning or confirmation of deletes. The record and all data below that record will be deleted and this operation cannot be undone. Please use with caution!

For example:

```
username=test
password=<base64 encoded password>
path=/Reports[Name="My Reports"]/History[ReportID="46135"]
actionID = delete
```

21.8.6. File Data

You can also include file data in your records. To do this, wherever a field is of the 'File' type, enter in JSON such as this:

In JSON format the request would be something like:

```
PDFFile: { "fileName": "test.pdf", "contentType":
"application/pdf", "fileData": "***F***" }
```

In this example, test.pdf would be the name of the file to insert, application/pdf should be changed to the HTTP content type of the data, and ***F*** should be the base-64-encoded data of the file.

22. Maintenance

22.1. Backing up Your Application

22.1.1. Overview

It's very important to back up your application configuration as often as possible. Data corruption or data loss is always a real possibility, and the only way to protect yourself against such disasters would be to back up your data.

There are two ways to do this:

1. If using MS SQL Compact, you can locate the database file and copy that to a safe location.
2. If using SQL Server, back up the SQL Server database.
3. For any database, you can use the admintools export functionality described in section 23.1.

22.1.1.1. *SQL Compact*

For any dbFramework application, the database is held in the Files subfolder of the application. The database file is called **data.sdf**. The size of the database file will depend on the amount of data in the application.

There are two databases that will need to be backed up. The first is the application configuration that is held in the dbFramework database. This is in the dbFramework/Files folder.

The second database is the data of the actual application(s). This will be in the application's Files subfolder.

Each of these databases can simply be copied using either Windows Explorer or the DOS prompt using the copy command.

You can also automate the backup of the database by creating a simple batch file that will copy the files, and have this configured to run automatically using the Windows Task Scheduler.

22.1.1.2. *SQL Server*

There are two SQL Server databases that will need to be backed up: The first is the DbfDev database. This stores the data on the application configuration that you

create in dbFramework. The second is the DbfApp database which stores the actual dbFramework application data. By default DbfApp contains multiple applications' data, and each table is prefixed with the application name.

22.1.2. Backing up SQL Server

There are two option for backing up SQL Server. The easiest is to use the SQL Management Studio, which can be downloaded from Microsoft. This has menu options and wizards that guide you through the backup process.

In order to automate the backing up of the database on a certain schedule, each night for example, you will need to download a utility called sqlcmd.exe which can be executed from a batch file. The sqlcmd download can be found in the SQL Server Feature Pack.

Once installed, you can run the following command to backup DbfDev:

```
sqlcmd -S (local)\SQLEXPRESS -E -Q "BACKUP DATABASE [DbfDev] TO DISK=
N'C:\temp\dbfdev.bak' WITH FORMAT"
```

This will backup the database to the c:\temp\dbfdev.bak file. You can modify that to suit your own configuration.

To backup the application data, use the following command:

```
sqlcmd -S (local)\SQLEXPRESS -E -Q "BACKUP DATABASE [DbfApp] TO DISK=
N'C:\temp\dbfapp.bak' WITH FORMAT"
```

This command will backup all of the applications' data.

You can automate this by putting these commands into a batch file (.bat), and then setting up a scheduled task to run that batch file each night.

23. Advanced Topics

23.1. Admin Tools

Although not directly available from the user-interface, there are several advanced features that are available through a special URL.

For any page in a dbFramework application, by replacing the `dataview.aspx` part of the URL to be `admintools.aspx`, you will be taken to a page that provides a few special features.

23.1.1. Exporting to XML

There will be a 'Download XML' link on the screen that downloads everything from the current record downwards to an XML file. You should right click on this link and select 'Save As...' to save the XML to a file.

Note: This feature will not work if the data exceeds approximately 120Mb, or less depending on the amount of contiguous (un-fragmented) memory available on the system at the time.

23.1.2. Importing XML

Underneath the 'Upload XML' label there is a button to select an XML file. After selecting the XML file you can click Upload to upload that file into the system.

This is an easy way to transfer data from one system to another. However, several limitations should be noted:

- The operation will fail if the file is too large, usually if the file is more than 100Mb in size.
- File data is not transferred. This should be transferred separately.
- The IDs of records are retained between export and import, making this unsuitable for *copying* data from one part of the database to another. However moving the data is fine.
- Events are not raised on the data as it is imported. This may make the database inconsistent.

23.1.3. DbfScript Command Window

This command window allows you to enter any DbfScript commands and they will be run as a script. The &path variable is set up to point to the current location in the database.

An additional command, 'log', can be used to output results to the screen.

This is a very useful feature for diagnostics purposes, or to test out and develop new actions in the application itself rather than in dbFramework.

23.2. System Log

The system log contains information on background processes and errors that occur asynchronously. You can view the system log by going to the /systemlog.aspx page. The log consists of multiple lines listed chronologically. Often the messages are cropped to fit on the screen and can be expanded by clicking the ellipses.

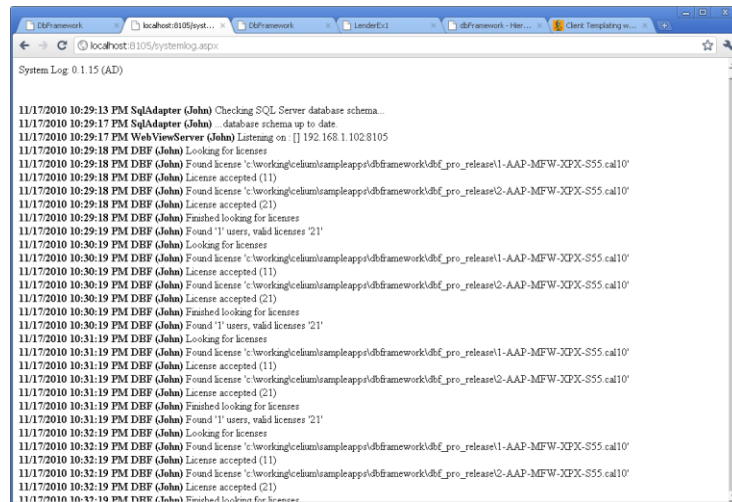


Figure 91: System log screen

23.3. About DbfServer.exe

The DbfServer process runs and manages the running of a single application. This keeps the application protected both from unintentional access from other applications, and also protected from failures in other applications.

OfficeSuite Launcher runs an instance of DbfServer for each application it needs to manage. You will have one instance of DbfServer running in your Task Manager for

each application, including one for dbFramework, which itself is a dbFramework application. It is normal for DbfServer to use 10Mb upwards of private memory.

When the application is republished the DbfServer process is restarted. DbfServer watches for file changes in the application's folder to determine if it needs to restart. When it does, it launches itself and waits for itself to close.

If the DbfServer process refuses to close, which may be the case if using third-party tools that won't shut-down cleanly, and also in some situations beyond dbFramework's control, then it will force the process to terminate after a certain amount of time (approximately 5 seconds). Transactions will be cleanly aborted and disk caches and connections should be closed and flushed, but there is still some possibility for data loss in this situation. For this reason it may be preferable to republish a server when it is not in use, or to have people stop using the server before it is restarted.

23.4. In-Memory XML Database

By default dbFramework uses Microsoft SQL Compact database (Express) or SQL Server (Professional) as its data store. You can, however, have dbFramework use an XML file instead. This stores all data in an XML file in the application's Files folder as a file called 'data.xml'.

The XML database is considerably faster than Microsoft SQL Compact or SQL Server, and so it may be suitable for systems that require faster transaction processing.

This file is stored in memory most of the time, and is saved to disk every couple of minutes.

However, several important limitations should be noted:

- The XML database is not fully transactional. Transactions cannot be rolled back.
- Because changes are stored in memory they are not guaranteed to be committed to disk, if the application crashes or the computer is switched off, those changes will be lost.
- Tagging does not currently work with the XML database.
- The size of the database is limited to the amount of available memory, and is also limited by memory fragmentation. The fragmentation issue may mean that databases larger than 100Mb may result in an out of memory

exception being raised at startup. As such, the In-Memory XML database option is really only recommended for smaller databases and very specific scenarios where performance is the priority.

To enable the XML database, go to the server configuration record for the application. You can do this by going to the application record, drilldown to the selected server target, and change the 'Data Store' entry box to be 'XML Database'. You will then have to republish.

Note: Any data previously held in a SQL Server database will not be transferred to the XML database. You can, however, use the admin-tools export and import to move your data over.

23.5. Database Concurrency and Deadlocks

When dbFramework applications utilize MS SQL Compact or MS SQL Server, every operation runs in the context of a transaction. Any errors that occur will ensure that the entire transaction is rolled-back and no changes are committed to the database. This also applies to updating the search index and to sending any real-time notifications.

While the transaction is in progress, all affected records will be locked by the database. Row-level locking is applied (instead of the more typical page-based locking), but this can be overridden when updating records with scripting or through the .Net SDK.

Because dbFramework is multi-threaded, it's possible that two scripts that are running in parallel may cause a deadlock situation to occur. This occurs when two separate threads are dependent on each others' records, causing them to hang indefinitely. dbFramework has special support for detecting a deadlock situation and will take necessary remedial action. To do this it will fail one of the transactions, releasing the locks and allowing the other to proceed. It will then retry the application after a random amount of time (under 2 seconds). If the deadlock repeats more than 50 times the operation will fail and an error will be displayed.

To reduce the likelihood of deadlocks occurring, consider the following:

1. Reduce the number of asynchronous events. Because these launch on separate threads, the probability of multiple operations running in parallel increases, and this increases the likelihood that they will be competing to lock the same records.

2. Reduce the amount of time spent in code. Optimize your event handling code to ensure that it runs as fast as possible. Avoid looping through lots of records unless necessary.
3. If using SQL Server, make sure that the OfficeSuite Launcher (or other hosting application) has optimal access to the SQL Server database. If possible both the dbFramework hosting application and the SQL Server database on the same machine so that database operations utilize the inter-process communication mechanism (IPC), rather than TCP/IP. The faster the application can access the database, the less time it will take and the less opportunity will exist for excessive lock retention and deadlocks.
4. If you are using Microsoft SQL Server, consider switching on **snapshot isolation**. This is a relatively new locking strategy that uses row versioning instead of holding locks. Because each change creates a new version of the record, two transactions updating the same record will not conflict, and therefore it is not possible for a deadlock situation to occur. However this strategy does incur an overhead in terms of storage, and also in terms of update performance because of the additional time required to create the record version. It can also impede query performance because specific versions of records need to be found in the query. A comprehensive study of the pros and cons of using snapshot isolation, and how to enable it, is beyond the scope of this document. More information should be sought from online Microsoft SQL Server resources.

23.6. Database Schema Upgrading

At startup the dbFramework application will compare the current schema of the database with the required schema, create a script to update the database schema and then run that script.

This update of the database schema will run automatically if you are upgrading to a newer version, and even if you are downgrading to an older version. Simply running the older version will automatically downgrade the database to the version and schema it expects.

Note: If you are downgrading to an older version, the database schema update may involve deleting some columns and tables. If you are concerned about data loss before downgrading, please back up the content of your database.

This operation occurs when the application is first run after a publish, and while the text 'Please Wait' is displayed on the screen. The scripts can sometimes be quite complex but this rarely causes any significant delay at startup, or any problems.

If you are using Microsoft SQL Server, because this server runs as a separate process there may be some times when this operation fails. If this occurs then a different screen will be displayed instead of the application screen, and it will call on the user to contact the system administrator.

If this does occur there are two possible options to pursue:

- Review the recent changes to the data model. Undo those changes and republish to see if the system now starts.
- Contact RiaForm Support for assistance.

In either case it's recommended that you contact RiaForm Support to notify them of the problem so it can be remedied or handled better in a future release.

23.7. SQL Table Structures

23.7.1. Overview

While not always recommended, it is possible to connect directly to the Microsoft SQL Server or SQL Compact database to access the data stored from your dbFramework application directly.

You may need to do this, for example, to:

- Integrate your dbFramework application with other SQL-based systems. dbFramework can create tables in an existing database that can make it possible to combine your dbFramework application's data with other applications that use SQL Server.
- Use third-party reporting tools that require a direct connection to a SQL Server.
- For backup purposes, to copy the data from your database to another medium.
- To quickly update data in tables when using DbfScript isn't possible or very efficient.
- For access to the data in your application where performance of queries is critical.
- For writing business logic in Transact-SQL (T-SQL) in the form of stored procedures. This may be necessary where lots of data is accessed or joined in ways that is inefficient to accomplish using the mechanisms provided by dbFramework.

- For creating or using other special-purpose SQL features such as triggers or SQL CLR.

For this reason this manual details the current format used by dbFramework for storing information in SQL Server and SQL Compact.

Note! While future versions of dbFramework will likely use a similar structure to that described here, there is no guarantee at all that the format will be maintained. There is no guarantee that any code you write that depends on these structures will work with future releases. This is especially true for any use made of path index structures, which were created to improve performance and will likely be tweaked in the future to make dbFramework applications perform even better. To protect your code, keep any dependencies on the structures as minimal as possible.

23.7.2. Table Structures

The table structures reflect the structure created for your application in your data model.

However, because your data model is hierarchical, and SQL does not have support for hierarchies, certain provisions are made to encode this hierarchical structure in regular flat table names.

dbFramework creates a table for each type in your data model. The name of the table will reflect whereabouts in the data model the type is defined. If, for example, you have a type called 'Item' that is within a type called 'Invoice', the resultant table name will be InvoiceItem. The parent of the type is always merged together to form a simple table name.

In addition, each field in your type will have a corresponding field in the table. The type of the field will, for the most part, reflect what you selected as the data type for each field, but there has to be some translation again because dbFramework offers different data types than those the SQL database provides.

The translation of each data type is described in this section.

23.7.2.1. *Text (Few Words)*

This will be stored as an nvarchar(255). The maximum number of characters in the string is 255. If text larger than that is entered, an error will be displayed.

23.7.2.2. *Text (Paragraphs)*

This will be stored as an ntext type. This is a memo field so there is no limit to the size of the text.

23.7.2.3. *Number (Floating Point)*

This will be stored as a float type. While a float type can store a very wide range of numbers, the accuracy is sometimes compromised. Use the 'money' type to ensure accuracy.

23.7.2.4. *Number (Money)*

The money type is stored as a decimal data type.

23.7.2.5. *Number (Integer)*

The integer type is stored as an int type.

23.7.2.6. *Date / Time (General) and Date / Time (Due Date)*

All date types are stored as a datetime type. If the time is omitted, then midnight is stored.

23.7.2.7. *Boolean*

The Boolean type is stored as a bit field. True is represented as 1 and False is represented as 0.

23.7.2.8. *Counter*

The counter type is stored as an int field. The value is set to @@identity. There can only be one counter field per table.

23.7.2.9. *Link Fields*

Any fields that have a foreign link configured will be stored as two fields in the table. The first field will hold a special string that contains the display value (to display to the user) and the XPath, combined into one string.

The second field will have a _RecID suffix. This will contain the record ID (_RecID field) of the record in the table that this foreign link points to. This can be used in creating SQL joins.

23.7.3. Special Fields

In addition to the fields in your data model, some additional fields will be added to each table, each of which serve a special purpose.

23.7.3.1. *_RecID*

The *_RecID* field holds the unique internal record ID associated with each record. Generally this number is not seen by the user. However it may be seen if a counter field exists on the table.

The *RecID* field value is calculated from the @@identity formula.

23.7.3.2. *Parent_* Fields*

A number of fields will be added that start with the name 'Parent_'. Each of these holds the record ID of the record that is the parent of this record. There is one field for each type that *can* be a parent of this record. Because some types are shared among types, there may be multiple parent fields for each table. The parent fields that are not used will be NULL.

These fields are useful for creating complex queries using INNER JOINs.

23.7.3.3. *_Modified and _Created Fields*

Every table has fields to hold the created date and created by user, along with the date and user who performed the last modification to the record. These are held in the *_CreatedDate*, *_CreatedBy*, *_ModifiedDate* and *_ModifiedBy* fields respectively.

23.7.3.4. *UNQ_* Fields*

In order to guarantee uniqueness among records that have unique fields set, certain constraints are added to the table schema. Because records may belong to a number of lists in the hierarchy, and you only want the fields to be unique in that *particular* location in the hierarchy, and given the one table holds all records across all lists in the whole hierarchy, the uniqueness cannot be set on the fields alone as there will be many duplicates in different places of the hierarchy.

To solve this problem the constraint adds in one *Parent_* field for each constraint. This ensures that it's unique for just that one list.

However this exposes a limitation in SQL Server and SQL Compact: For records that are in one list, the remainder *Parent_** fields have NULL values, and these are considered duplicates. To fix this problem we synthesize one *UNQ_** field for each

parent that is guaranteed to hold a value and won't be NULL. This is then used in the constraint to ensure unique key violations cannot be applied to the database.

23.7.3.5. *ID field*

Every table has an ID field. This is held in a string field but is always equal to a GUID (Globally-Unique Identifier).

GUIDs are used throughout the system to allow records to be created from multiple servers using synchronization without any problems with duplicate rows. This makes it easier to have a wide variety of configurations and removes potential deployment bottlenecks.

23.7.4. Path Index Tables

For each table in the data model there will also be a table with the same name but prefixed with PX__. This table holds some special data used by the system to locate records in the database hierarchy.

Warning: Out of all the tables created, the path index tables are the most likely to change in the future. If at all possible, don't depend on the values in these tables in your own applications or SQL statements.

Each of these tables holds the same fields. They reflect the location of each record in the database according to their path. The field names and GUIDs for each record are stored as Fn and Kn fields (where n is a number between 0 and 5). For paths longer than 6 fields, additional records are created and they are linked together using the Parent field. The size of the path is identified by the Size field.

The corresponding record in the actual table is identified using the NodeRecID field, which corresponds to the _RecID field in the actual table.

As stated in the warning above, it's best not to concern yourself with these tables as they are used internally for performance improvements and are subject to change in the future. Instead you should use the Parent_* fields and perform regular SQL joins between tables to evaluate your hierarchical path.

23.7.5. Schema Table

There is also a table called _CelSystem. This is used to hold internal structures used by dbFramework to determine how the database needs to be upgraded each time it is published.

This table is internal and shouldn't be used.

23.7.6. Session Table

In order to maintain session information (such as the user login), information about individual sessions is stored in a special table called `_SessionInfo`.

This table is internal and shouldn't be used.

23.7.7. File Data

File data in dbFramework applications is not stored in the database. Instead it is stored in a special file and folder structure in the document folder specified in the server record.

23.8. Overriding the Server Port Number

The port number used to access an application is specified in the server record configuration. Go to an application record, drilldown to the Server list. Select the server you want to change and update the 'Url of App' to have a different port number. Once you re-publish the application it will be running on a the new port number.

You can also update the port number that dbFramework uses. By default dbFramework uses port 85, but you can update the app.ini file in the application folder (usually in `\Program Files\dbFramework\Apps\dbFramework`), and change the port entry to point to a different port number. You will then have to restart the OfficeSuite Launcher to see the changes take effect.

24. References

24.1. Culture ID Reference

The following is a partial list of culture IDs that are used when adding languages to dbFramework.

Code	Language – Country/Region	Code	Language – Country/Region
af	Afrikaans	hu-HU	Hungarian – Hungary
af-ZA	Afrikaans – South Africa	is	Icelandic
sq	Albanian	is-IS	Icelandic – Iceland
sq-AL	Albanian – Albania	id	Indonesian
ar	Arabic	id-ID	Indonesian – Indonesia
ar-DZ	Arabic – Algeria	it	Italian
ar-BH	Arabic – Bahrain	it-IT	Italian – Italy
ar-EG	Arabic – Egypt	it-CH	Italian – Switzerland
ar-IQ	Arabic – Iraq	ja	Japanese
ar-JO	Arabic – Jordan	ja-JP	Japanese – Japan
ar-KW	Arabic – Kuwait	kn	Kannada
ar-LB	Arabic – Lebanon	kn-IN	Kannada – India
ar-LY	Arabic – Libya	kk	Kazakh
ar-MA	Arabic – Morocco	kk-KZ	Kazakh – Kazakhstan
ar-OM	Arabic – Oman	kok	Konkani
ar-QA	Arabic – Qatar	kok-IN	Konkani – India
ar-SA	Arabic – Saudi Arabia	ko	Korean
ar-SY	Arabic – Syria	ko-KR	Korean – Korea
ar-TN	Arabic – Tunisia	ky	Kyrgyz
ar-AE	Arabic – United Arab Emirates	ky-KG	Kyrgyz – Kyrgyzstan
ar-YE	Arabic – Yemen	lv	Latvian
hy	Armenian	lv-LV	Latvian – Latvia
hy-AM	Armenian – Armenia	lt	Lithuanian
az	Azeri	lt-LT	Lithuanian – Lithuania
az-AZ-Cyrl	Azeri (Cyrillic) – Azerbaijan	mk	Macedonian

az-AZ-Latn	Azeri (Latin) – Azerbaijan	mk-MK	Macedonian – Former Yugoslav Republic of Macedonia
eu	Basque	ms	Malay
eu-ES	Basque – Basque	ms-BN	Malay – Brunei
be	Belarusian	ms-MY	Malay – Malaysia
be-BY	Belarusian – Belarus	mr	Marathi
bg	Bulgarian	mr-IN	Marathi – India
bg-BG	Bulgarian – Bulgaria	mn	Mongolian
ca	Catalan	mn-MN	Mongolian – Mongolia
ca-ES	Catalan – Catalan	no	Norwegian
zh-HK	Chinese – Hong Kong SAR	nb-NO	Norwegian (Bokmål) – Norway
zh-MO	Chinese – Macao SAR	nn-NO	Norwegian (Nynorsk) – Norway
zh-CN	Chinese – China	pl	Polish
zh-CHS	Chinese (Simplified)	pl-PL	Polish – Poland
zh-SG	Chinese – Singapore	pt	Portuguese
zh-TW	Chinese – Taiwan	pt-BR	Portuguese – Brazil
zh-CHT	Chinese (Traditional)	pt-PT	Portuguese – Portugal
hr	Croatian	pa	Punjabi
hr-HR	Croatian – Croatia	pa-IN	Punjabi – India
cs	Czech	ro	Romanian
cs-CZ	Czech – Czech Republic	ro-RO	Romanian – Romania
da	Danish	ru	Russian
da-DK	Danish – Denmark	ru-RU	Russian – Russia
div	Dhivehi	sa	Sanskrit
div-MV	Dhivehi – Maldives	sa-IN	Sanskrit – India
nl	Dutch	sr-SP-Cyrl	Serbian (Cyrillic) – Serbia
nl-BE	Dutch – Belgium	sr-SP-Latn	Serbian (Latin) – Serbia
nl-NL	Dutch – The Netherlands	sk	Slovak
en	English	sk-SK	Slovak – Slovakia
en-AU	English – Australia	sl	Slovenian
en-BZ	English – Belize	sl-SI	Slovenian – Slovenia

en-CA	English – Canada	es	Spanish
en-CB	English – Caribbean	es-AR	Spanish – Argentina
en-IE	English – Ireland	es-BO	Spanish – Bolivia
en-JM	English – Jamaica	es-CL	Spanish – Chile
en-NZ	English – New Zealand	es-CO	Spanish – Colombia
en-PH	English – Philippines	es-CR	Spanish – Costa Rica
en-ZA	English – South Africa	es-DO	Spanish – Dominican Republic
en-TT	English – Trinidad and Tobago	es-EC	Spanish – Ecuador
en-GB	English – United Kingdom	es-SV	Spanish – El Salvador
en-US	English – United States	es-GT	Spanish – Guatemala
en-ZW	English – Zimbabwe	es-HN	Spanish – Honduras
et	Estonian	es-MX	Spanish – Mexico
et-EE	Estonian – Estonia	es-NI	Spanish – Nicaragua
fo	Faroese	es-PA	Spanish – Panama
fo-FO	Faroese – Faroe Islands	es-PY	Spanish – Paraguay
fa	Farsi	es-PE	Spanish – Peru
fa-IR	Farsi – Iran	es-PR	Spanish – Puerto Rico
fi	Finnish	es-ES	Spanish – Spain
fi-FI	Finnish – Finland	es-UY	Spanish – Uruguay
fr	French	es-VE	Spanish – Venezuela
fr-BE	French – Belgium	sw	Swahili
fr-CA	French – Canada	sw-KE	Swahili – Kenya
fr-FR	French – France	sv	Swedish
fr-LU	French – Luxembourg	sv-FI	Swedish – Finland
fr-MC	French – Monaco	sv-SE	Swedish – Sweden
fr-CH	French – Switzerland	syr	Syriac
gl	Galician	syr-SY	Syriac – Syria
gl-ES	Galician – Galician	ta	Tamil
ka	Georgian	ta-IN	Tamil – India
ka-GE	Georgian – Georgia	tt	Tatar
de	German	tt-RU	Tatar – Russia
de-AT	German – Austria	te	Telugu

de-DE	German – Germany	te-IN	Telugu – India
de-LI	German – Liechtenstein	th	Thai
de-LU	German – Luxembourg	th-TH	Thai – Thailand
de-CH	German – Switzerland	tr	Turkish
el	Greek	tr-TR	Turkish – Turkey
el-GR	Greek – Greece	uk	Ukrainian
gu	Gujarati	uk-UA	Ukrainian – Ukraine
gu-IN	Gujarati – India	ur	Urdu
he	Hebrew	ur-PK	Urdu – Pakistan
he-IL	Hebrew – Israel	uz	Uzbek
hi	Hindi	uz-UZ-Cyrl	Uzbek (Cyrillic) – Uzbekistan
hi-IN	Hindi – India	uz-UZ-Latn	Uzbek (Latin) – Uzbekistan
hu	Hungarian	vi	Vietnamese

24.2. Icon Reference

The following is a guide to locating an icon in the standard icons shipped with dbFramework.



24.3. Contacting RiaForm Technology

The best way to contact RiaForm Technology is through email at support @ riaform.com.

Keep in mind that RiaForm support will only be able to help you with technical problems related to installing and basic usage of the system, but cannot provide support for the development of particular projects or advanced, complex problems without first establishing a support contract. For more information mention this in your email to support.